Journal of Machine Learning Research 11 (2010) 2935-2972 Submitted 7/09; Revised 7/10; Published 10/10

Tree Decomposition for Large-Scale SVM Problems

Fu Chang FCHANG@IIS.SINICA.EDU.TW
Chien-Yang Guo ASDGUO@IIS.SINICA.EDU.TW
Xiao-Rong Lin ECLIPSEX527@IIS.SINICA.EDU.TW
Chi-Jen Lu CJLU@IIS.SINICA.EDU.TW
Institute of Information Science
Academia Sinica
Taipei, Taiwan

Editor: Alexander Smola

Abstract

To handle problems created by large data sets, we propose a method that uses a decision tree to decompose a given data space and train SVMs on the decomposed regions. Although there are other means of decomposing a data space, we show that the decision tree has several merits for large-scale SVM training. First, it can classify some data points by its own means, thereby reducing the cost of SVM training for the remaining data points. Second, it is efficient in determining the parameter values that maximize the validation accuracy, which helps maintain good test accuracy. Third, the tree decomposition method can derive a generalization error bound for the classifier. For data sets whose size can be handled by current non-linear, or kernel-based, SVM training techniques, the proposed method can speed up the training by a factor of thousands, and still achieve comparable test accuracy.

Keywords: binary tree, generalization error ïbound, margin-based theory, pattern classification, tree decomposition, support vector machine, VC theory

1. Introduction

Support vector machines (SVMs) have proven very effective for solving pattern classification problems (Cortes and Vapnik, 1995; Vapnik, 1995). Because of the growing trend to apply them to various domains of interest, including bioinformatics, computer vision, data mining and knowledge discovery, the size of training data sets continues to grow at a rapid rate. At the same time, there is an ongoing effort to speed up the SVM training. One approach, called the *numerical technique* in this paper, seeks efficient solutions to SVM optimization problems.

Well-known numerical methods for solving dual optimization problems include sequential minimal optimization (SMO) (Platt, 1998) and SVM$^{\text{light}}$ (Joachims, 1998). Both methods break a large problem into a series of small problems in order to reduce the amount of memory required for computation. SMO, in particular, has proven superior to similar methods, such as the projected conjugated gradient "chunking" algorithm (Burges, 1998) and Osuna's algorithm (Osuna et al., 1997). For solving dual problems, there are now many new and faster methods, including LASVM (Bordes et al., 2005), maximum-gain working set selection (Glasmachers and Igle, 2006), SVM$^{\text{perf}}$ (Joachims, 2006), LaRank (Bordes et al., 2007), Pegasos (Shalev-Shwartz et al., 2007), bundle methods (Smola et al., 2007), and LIBLINEAR (Fan et al., 2008).

A new direction that has attracted increasing interest in recent years uses the stochastic gradient descent (SGD) technique to solve large-scale SVM problems. The advantage of SGD is that it implements an online learning process that converges to an optimal solution in one examination of the training samples. The above-mentioned LASVM, LaRank and Pegasos algorithms apply SGD to dual optimization problems. There are also algorithms that apply SGD to primal optimization problems, for example, NORMA (Kivinen et al., 2004) and SGD-QN (Bordes et al., 2009).

In addition to the methods for solving dual or primal problems, a number of approaches for solving large SVMs have been proposed, including core vector machines (Tsang et al., 2005) and OCAS (Franc and Sonnenburg, 2008). Readers may refer to a useful paper by Menon (2009) for more details of the numerical methods.

Another type of approach, called *data-reduction* in this paper, reduces a large training data set to one or several small data sets. If only one reduced set is obtained, we call the method *single-set reduction* (SSR); and if multiple reduced sets are obtained, we call the method *multiple-set reduction* (MSR). In the latter case, SVM training is conducted on each of the reduced sets and all the SVMs are combined into a final classifier.

We review MSR methods first. Perhaps the simplest MSR method is bagging (Breiman, 1996). It employs a number of down-sampled data sets to train SVMs, which jointly classify a test object based on majority vote. The boosting method (Schapire, 1990; Schapire and Singer, 2000) trains SVMs in a sequential manner, and the training of a particular SVM is dependent on the training and performance of previously trained SVMs. The divide-and-combine strategy (Rida et al., 1999) decomposes an input space into possibly overlapping regions, assigns each region a local predictor, and combines the local predictors to derive a global solution to the prediction problem. The Bayesian committee machine (Tresp, 2000) partitions a large data set into smaller ones, and the SVMs trained on the reduced sets jointly define the posteriori probabilities of the classes into which test objects are categorized. The method proposed by Collobert et al. (2002) divides a set of input samples into smaller subsets, assigns each subset a local expert, and forms a loop to re-assign samples to local experts according to how well the experts perform. The cascade SVM method (Graf et al., 2004) also splits a large data set into smaller sets and extracts support vectors (SVs) from each of them. The resulting SVs are combined and filtered in a cascade of SVMs. A few passes through the cascade ensures that the optimal solution is found.

On the SSR side of the data-reduction approach, the squashing method (Pavlov et al., 2000) uses a likelihood-based squashing technique to obtain a reduced data set, and then trains linear SVMs on that set. The sparse greedy approximation method (Smola and Schölkopf, 2000) constructs a compressed representation of the design matrix involved in the QP problem; while information vector machines (Lawrence et al., 2003) use a sparse Gaussian process to select training samples using criteria based on information-theoretic principles. Clustering-based SVM (Yu et al., 2003) applies a hierarchical clustering algorithm to obtain a reduced data set, which is used to train SVMs. The concept boundary detection (CBD) method (Panda et al., 2006) prepares nearest-neighbor lists as training samples, and uses a special down-sampling technique to extract the data points that lie close to class boundaries. This method can find a single set of near-boundary points for all class pairs. In contrast, many other methods that use SVMs to analyze training samples have to find different reduced sets for different class pairs, since SVMs can only work on one class pair at a time. For more details of data-reduction methods proposed up to 2001, readers may refer to Tresp (2001).

Finally, we remark that the numerical and data reduction approaches, instead of competing, can actually complement each other's functions. The data reduction approach must train SVMs on reduced data sets, so having an efficient numerical method to perform the task would certainly be useful. The numerical approach, on the other hand, could benefit by using an efficient data reduction method to reduce its computational burden.

In this paper, we propose a method that decomposes a large data set into a number of smaller ones and trains SVMs on each of them. This approach can reduce the total training time because the time complexity of training an SVM is in the order of n^2, where n is the number of training samples (Platt, 1998; Joachims, 1998) . If each smaller problem deals with σ samples, then the complexity of solving all the problems is in the order of $(n/\sigma) \times \sigma^2 = n\sigma$, which is much smaller than n^2 if n is significantly higher than σ. Decomposing a large problem into smaller problems has the added benefit of reducing the number of SVs in each of the resultant SVMs. Since a test sample is classified by only one of these SVMs, the decomposition strategy reduces the time required for the testing process in which the number of SVs dominates the complexity of the computation. One additional benefit of the decomposition approach is the ease of using multi-core/parallel/distributed computing for further speedup, since the SVM problems associated with the decomposed regions can be parallelized idealistically.

The proposed method can be categorized as an MSR method. However, it differs from other MSR methods in that it uses a *decision tree* to obtain multiple reduced data sets, whereas other methods use non-supervised clustering (Rida et al., 1999), random sampling (Breiman, 1996), or random partition (Tresp, 2000; Collobert et al., 2002; Graf et al., 2004). We thus call our method a *decision-tree support vector machine* (DTSVM) and the resultant classifier a DTSVM classifier.

A decision tree decomposes a data space recursively into smaller regions. In terms of the ways the regions are formed, a decision tree can be classified into three types: axis-parallel, oblique and Voronoi types. In the axis-parallel type, the regions are bounded by hyperplanes represented as $x_i = c$, where x_i is a feature and c is a real number (Breiman et al., 1984; Quinlan, 1986). In the oblique type, the regions are bounded by hyperplanes represented as $\sum \alpha_i x_i = c$, where α_i are real numbers (Murthy et al., 1994; Bennett and Blue, 1998; Wu et al., 1999; Bennett et al., 2000). In the Voronoi type, the regions are formed as Voronoi cells by way of various clustering techniques (for a survey, see Dattatreya and Kanal, 1985).

In this paper, we take an axis-parallel decision tree as our decomposition scheme because of its speed in both the training and testing phases. The other two types of decision trees can certainly be used as decomposition schemes, but their computational cost is significantly higher than that of the axis-parallel type. Without conducting a tradeoff study, it is rather difficult to determine whether the additional cost would yield a noteworthy benefit; therefore, we have decided not to adopt them at this stage.

A number of studies have attempted to combine decision trees and SVMs. Some of the methods were designed to improve the classification accuracy (e.g., Bennett and Blue, 1998; Wu et al., 1999; Bennett et al., 2000; Ramaswamy, 2006; Tibshirani and Hastie, 2007); while others were designed to speed up the SVM testing process (e.g., Platt et al., 2000; Sahbi and Geman, 2006; Sun et al., 2007). To the best of our knowledge, using a decision tree to speed up the training of multiclass SVMs has not been proposed previously.

Using a decision tree as a decomposition scheme can yield following benefits when dealing with large-scale SVM problems. First, the decision tree may decompose the data space so that certain decomposed regions become homogeneous; that is, they contain samples of the same labels. Then,

in the testing phase, when a data point flows to a homogeneous region, we simply classify it in terms of the common label of that region. This alleviates the burden of SVM training, which is only conducted in heterogeneous regions. In fact, our experiments revealed that, for certain data sets, more than 90% of the training samples reside in homogeneous regions; thus, the decision tree method saves an enormous amount of time when training SVMs. Random partition, on the other hand, cannot produce such an effect, since random pooling of a set of samples can hardly create a homogeneous data set due to the independent sampling operation.

Another benefit of using the decision tree is the convenience it provides when searching for all the relevant parameter values to maximize the solution's validation accuracy, which in turn helps maintain good test accuracy rates. The goal of the DTSVM method is to attain comparable validation accuracy while consuming less time than training SVMs on full data sets. To achieve our objective, we found that it is important to control the size σ of the tree-decomposed regions as well as the SVM-parameter values. For some data sets, σ could be set at 1,500; but for other data sets, it had to be set at a larger value. Thus, in the DTSVM method, σ is an additional parameter to the usual SVM-parameters. Other MSR methods do not attempt to search for the optimal size of decomposed regions. Such searches are particularly easy under the DTSVM method because a decision tree is constructed in a recursive manner; hence, obtaining a tree with a larger size of σ does not require the reconstruction of a decision tree corresponding to that size of σ.

Using a decision tree also reduces the cost of searching for the optimal values of SVM-parameters. Searching for these values is important, but it takes a tremendous amount of time, especially when training non-linear SVMs. To the best of our knowledge, no data-reduction method has attempted to reduce the cost of this operation. Our strategy involves training SVMs with all combinations of SVM-parameter values, but *only* for decomposed regions with an initial σ-level. The optimal values of the SVM-parameters obtained at this level are not necessarily the same as those obtained at higher levels. However, we observe that the best values for a higher level are usually among the top-ranked values for the initial level. Therefore, when we want to train SVMs for a higher σ-level, we only train them with the top-ranked values obtained for the initial level. Given the n^2-complexity of SVM training, restricting the full search of SVM-parameter values to regions with the initial σ-level certainly reduces the SVM training time. In fact, our experiments showed that such savings were possible even when the optimal σ-level was higher than that of the full size data set.

Although the decision tree method may not be the only way to achieve the above benefits for large-scale SVM problems, its effect can be understood in theory and a generalization error bound can be derived for the DTSVM classifier. The bound is the sum of two terms: the first term dominates in magnitude and is associated with SVM training; and the second term is associated with tree training. Our experimental results show that the numerical value of the dominant term is as small as, or of the same order of magnitude as, its counterpart in the generalization error bound for SVM training conducted on the whole data set. This finding constitutes indirect evidence of the efficacy of tree decomposition for large-scale SVM problems.

Finally, we remark that it is possible to have multiple decompositions of the same data space with multiple trees. These trees can be obtained by using a randomized, rather than the optimal, split point at each tree node (Dietterich, 2000). By so doing, we train SVMs on all the decomposed regions and classify the test data based on a majority vote strategy. We have actually studied the effect of such multiple decompositions. In terms of test accuracy, multiple decompositions are not as effective as searching for the optimal σ-level of decomposed regions in a single decision tree. In fact, under the latter search scheme, introducing multiple decompositions does not lead to

any significant improvement. Therefore, to avoid unnecessary complications, we only consider the decomposition of a data space by a single tree in this paper.

In the experimental study, we divided each data set into training, validation and test components. We then used the training component to build DTSVM classifiers, the validation component to determine the optimal parameters, and the test component to measure the test accuracy. We adopted two types of SVM training: one-against-one (1A1) (Knerr et al., 1990) and one-against-others (1AO) (Bottou et al., 1994) . Furthermore, we built non-linear SVMs on the data sets. When evaluating the DTSVM method, we found it could train DTSVM classifiers that achieved comparable test accuracy rates to those of SVM classifiers. For seven medium-size data sets, in which the largest number of sample was 494K and the largest number of feature was 62K, DTSVM achieved speedup factors between 4 and 3,691 for 1A1 training, and between 29 and 5,775 for 1AO training. Moreover, DTSVM achieved much higher speedup factors than several data reduction methods and numerical methods. To demonstrate that DTSVM can train classifiers efficiently for larger data sets, we applied it to four large-size data sets in which the largest number of samples was 4.9M and the largest number of features was 16.6M. For all the data sets, DTSVM could complete 1A1 training and 1AO training within 18.25 hours. Note that the training time included the time required to build a decision tree, the time to train SVMs on all the leaves, and the time to search for the optimal parameters.

The remainder of this paper is organized as follows. In Section 2, we describe the DTSVM method. Section 3 details the experimental results. In Section 4, we provide theoretical results for the DTSVM method. Section 5 contains some concluding remarks.

2. The DTSVM Method

In this section, we describe the decision tree that we use as the decomposition scheme, and discuss the training process for the DTSVM method. An implementation of the DTSVM method is available at

http://ocrwks11.iis.sinica.edu.tw/dar/Download/WebPages/DTSVM.htm

2.1 The Decision Tree

For the decomposition scheme, we adopt CART (Breiman et al., 1984) or a binary C4.5 scheme (Quinlan, 1986) that allows two child nodes to grow from each node that is not a leaf. Using a C4.5 scheme that allows multiple child nodes is feasible; however, we do not consider it in this paper, since a binary C4.5 performs the job rather well for us.

To grow a binary tree, we follow a recursive process, whereby each training sample flowing to a node is sent to its left-hand or right-hand child node. At a given node E, a certain feature f_E of the training samples flowing to E is compared with a certain value v_E so that all samples with $f_E < v_E$ are sent to the left-hand child node, and the remaining samples are sent to the right-hand child node. The values of f_E and v_E are determined as follows. First, the *split point* v_f of each feature f is calculated by

$$v_f = \arg\max_v IR(f, v),$$

where

$$IR(f,v) = I(S) - \frac{|S_{f<v}|}{|S|} I(S_{f<v}) - \frac{|S_{f\geq v}|}{|S|} I(S_{f\geq v}),$$

S is the set of all samples flowing to E; $S_{f<v}$ consists of the elements of S with $f < v$; $S_{f\geq v} = S \backslash S_{f<v}$; $|X|$ is the size of any data set X; and $I(X)$ is the impurity of X. The impurity function used in our experiments is the entropy measure, defined as

$$I(\mathrm{S}) = -\sum_{y} p(S_y) \log p(S_y),$$

where $p(S_y)$ is the proportion of S's samples whose label is y. Then,

$$f_E = \arg\max_{f} IR(f, v_f),$$

and v_E is taken as the split point of f_E.

We stop splitting a node E when one of the following conditions is satisfied: (i) the number of samples that flow to E is smaller than a *ceiling size* σ; or (ii) when $IR(f,v) = 0$ for all f and v at E. The value of σ in the first condition is determined in a data-driven fashion, which we describe in Section 2.2. The second condition occurs mainly in the following cases. (a) All the samples that flow to E are homogeneous; or (b) a subset of them is homogeneous and the remaining samples, although carrying different labels, are identical to some members of the homogeneous subset. There are other possible cases for the second condition, but their occurrence is extremely rare. If we want to split E in these cases, we can choose the following split point to minimize the difference between $|S_{f<v}|$ and $|S_{f\geq v}|$, that is,

$$v_f = \arg\min_{v} \left| |S_{f<v}| - |S_{f\geq v}| \right|.$$

After growing a tree, we train an SVM on each of its leaves, using samples that flow to each leaf as training data (Figure 1). The values of the SVM parameters are also determined in a data-driven fashion. A tree and all SVMs associated with its leaves constitute a DTSVM classifier, as shown in Figure 1. In the training phase, all the SVMs in a DTSVM classifier are trained with the same parameter values. We explain how the optimal values are obtained in Section 2.2. In the validation/testing process, we first input a given validation/test object $\mathbf{x}$ to the tree. If $\mathbf{x}$ reaches a leaf that contains homogeneous samples, we classify $\mathbf{x}$ as the label of those samples; otherwise, we classify it with the SVM associated with that leaf.

2.2 The DTSVM Training Process

Given a training and validation component, we build a DTSVM classifier on the training component and determine its optimal parameter values with the help of the validation component. The parameters associated with a DTSVM classifier are: (i) σ, the ceiling size of the decision tree; and (ii) the SVM parameters. Their optimal values are determined in the following manner.

We begin by training a binary tree with an initial ceiling size σ_0, and then train SVMs on the leaves with SVM-parameters $\theta \in \Theta$, where Θ is the set of all possible SVM-parameter values whose effects we want to evaluate. Note that we express θ in boldface to indicate that it may consist of more than one parameter. Let $v(\sigma_0, \theta)$ be the validation accuracy of the resultant DTSVM classifier.

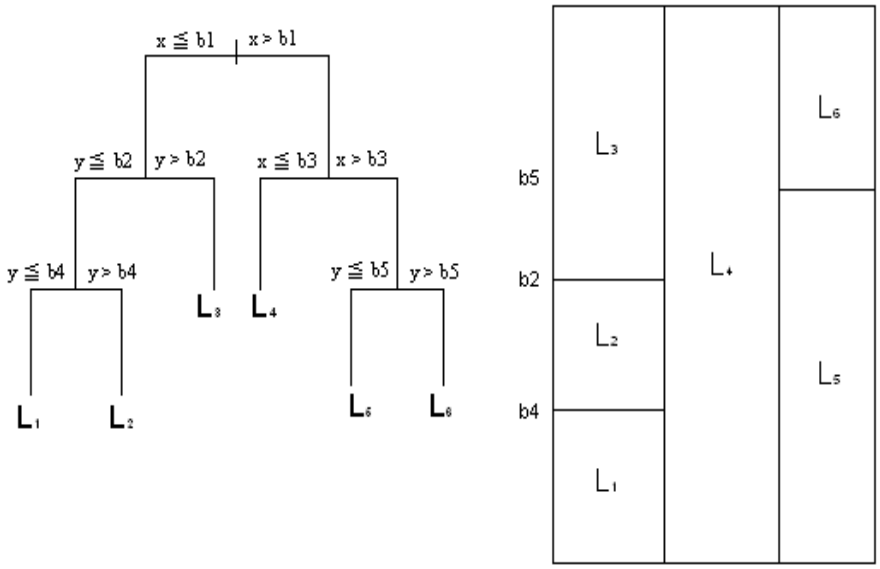


Figure 1: The architecture of a DTSVM classifier: a tree and all its leaves (L1 to L6) are produced and SVMs are trained on the leaves.

Next, we want to construct DTSVM classifiers with larger ceiling sizes, but we only train their associated SVMs with k top-ranked θ. To do this, we rank θ in descending order of $v(\sigma_0, \theta)$. Let Θ_k be the set that consists of k top-ranked θ.

More specifically, we implement the following sub-process, denoted as *SubProcess*(θ), for each $\theta \in \Theta_k$.

1. Set $t = 0$ and get the binary tree with the ceiling size σ_0.

2. Increase t by 1 and set $\sigma_t = 4 \times \sigma_{t-1}$. Modify the tree with ceiling size σ_{t-1} to obtain a tree with ceiling size σ_t. This is done by moving from the root towards the leaves and retaining each node whose parent's size is greater than σ_t. Then, train SVMs on the leaves with SVM-parameters θ. Let $v(\sigma_t, \theta)$ be the validation accuracy of the resultant DTSVM classifier.

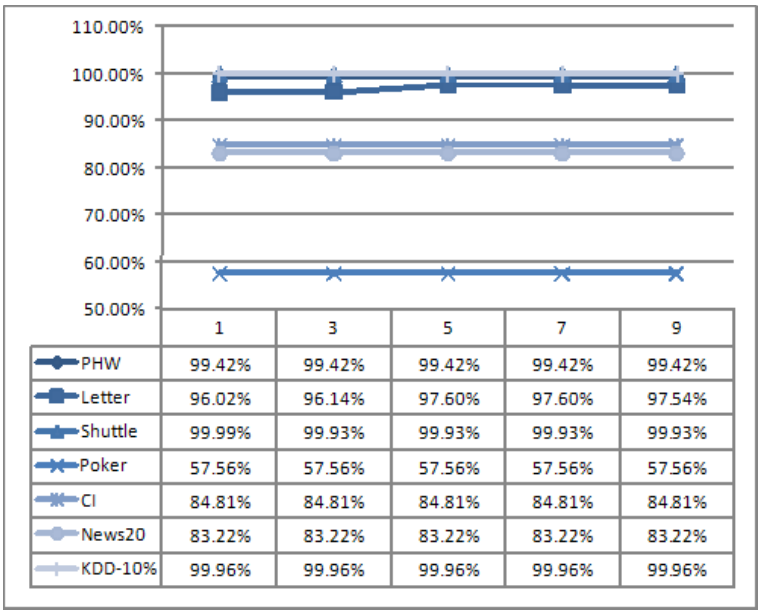
3. If $v(\sigma_t, \theta) - v(\sigma_{t-1}, \theta) \geq 0.5\%$ and σ_t is less than the size of the training component, proceed to step 2.

4. Let $\sigma(\theta) = \sigma_{t-1}$ if $v(\sigma_t, \theta) - v(\sigma_{t-1}, \theta) < 0.5\%$ or $\sigma(\theta) = \sigma_t$ if σ_t is greater than or equal to the size of the training component.

When we have completed *SubProcess*(θ) for all $\theta \in \Theta_k$, we define

$$\theta_{opt} = \arg\max_{\theta \in \Theta_k} v(\sigma(\theta), \theta) \text{ and } \sigma_{opt} = \sigma(\theta_{opt}).$$

We then output the DTSVM classifier with the SVM-parameter θ_{opt} and the ceiling size σ_{opt}.



	1	3	5	7	9
PHW	99.42%	99.42%	99.42%	99.42%	99.42%
Letter	96.02%	96.14%	97.60%	97.60%	97.54%
Shuttle	99.99%	99.93%	99.93%	99.93%	99.93%
Poker	57.56%	57.56%	57.56%	57.56%	57.56%
CI	84.81%	84.81%	84.81%	84.81%	84.81%
News20	83.22%	83.22%	83.22%	83.22%	83.22%
KDD-10%	99.96%	99.96%	99.96%	99.96%	99.96%

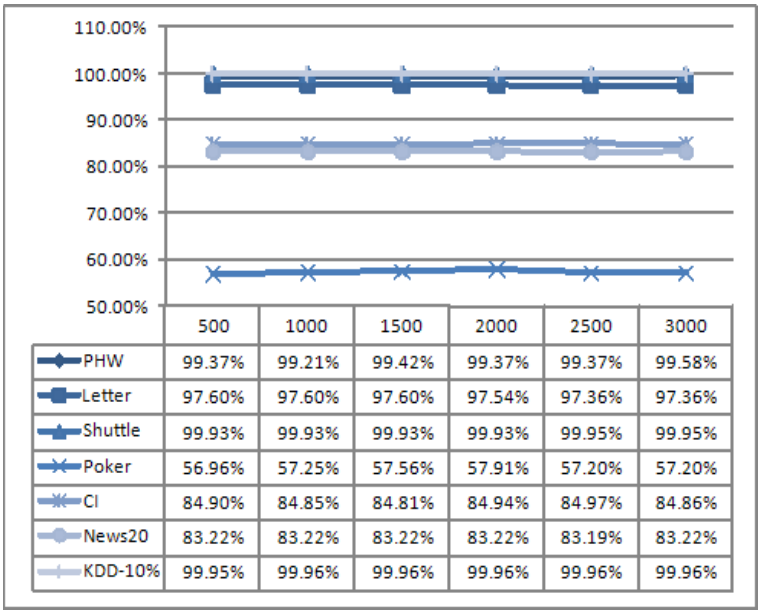
Figure 2: The test accuracy rates obtained by DTSVM on the seven data sets when σ_0 = 1,500 and k = 1, 3, 5, 7 and 9.

Note that, in each *SubProcess*(θ), we set σ_t as quadruple the size (rather than double the size) of σ_{t-1} for two reasons. First, quadrupling the size produces more significant differences between $v(\sigma_t, \theta)$ and $v(\sigma_{t-1}, \theta)$, especially when t is small. This means that if a *SubProcess* terminates at a small t, there is less risk of a low validation accuracy rate. Second, quadrupling the size enables the training process to progress at a faster pace. This means that if a *SubProcess* terminates at a large t, it moves more rapidly towards that end of the process.

The initial ceiling size σ_0 (=1,500) and the number k (=5) of the top-ranked parameters are set heuristically. To observe how these settings impact the test accuracy, we first fix σ_0 at 1,500 and vary k from 1 to 9 at a step size of 2; we then plot the test accuracy rates obtained by DTSVM on the seven data sets whose details are shown in Table 1. Figure 2 shows that any value of k is good for all the data sets except "Letter", while k = 5 or 7 is particularly good for "Letter". Moreover, setting the right value of k improves the test accuracy of "Letter" significantly. Next, we fix k at 5 and vary σ_0 from 500 to 3,000 with a step size of 500. As shown in Figure 3, varying the value of σ_0 does *not* affect the test accuracy of any data set significantly.

3. Experimental Results

In this section, we describe the data sets used in the experiments and the methods that we compared. We then present and discuss the experimental results.



	500	1000	1500	2000	2500	3000
PHW	99.37%	99.21%	99.42%	99.37%	99.37%	99.58%
Letter	97.60%	97.60%	97.60%	97.54%	97.36%	97.36%
Shuttle	99.93%	99.93%	99.93%	99.93%	99.95%	99.95%
Poker	56.96%	57.25%	57.56%	57.91%	57.20%	57.20%
CI	84.90%	84.85%	84.81%	84.94%	84.97%	84.86%
News20	83.22%	83.22%	83.22%	83.22%	83.19%	83.22%
KDD-10%	99.95%	99.96%	99.96%	99.96%	99.96%	99.96%

Figure 3: The test accuracy rates obtained by DTSVM on the seven data sets when $k = 5$, and $\sigma_0 =$ 500, 1,000, 1,500, 2,000, 2,500 and 3,000.

3.1 The Data Sets

In the experiments, we divided the data sets into two groups. The first group was used to evaluate the efficiency of DTSVM and some alternative methods in terms of speeding up SVM training. The second group was used to verify that the DTSVM method could handle much larger data sets, for which most of the alternative methods required an excessive amount of time to complete the training process. The first group comprised seven medium-size data sets, ranging from 10K to 494K in size, as shown in Table 1. Most of the data sets have less than 50 features, but the "News20" has 62,060 features. The second group comprised four large-size data sets, ranging from 240K to 4,898K in size, as shown in the Table 2. The "Webspam" data set is not very large in terms of the number of samples (240K), but the number of features is more than 16M; thus, we consider it a large-size data set. All the data sets were obtained from UPI repository (Newman et al., 1998), with the following two exceptions: "PPI", which was used in a protein-protein interaction study (Tseng et al., 2010), and "Webspam", which was obtained from

http://www.cc.gatech.edu/projects/doi/WebbSpamCorpus.html

Note that the actual "Poker" data set in the repository contains 1 million samples; however, we only used its training component in our experiments.

We randomly divided each data set into six parts of approximately equal size, and used four parts as the training component, one part as the validation component, and the remaining part as the test component. The DTSVM classifiers were trained on the training and validation components, as described in Section 2.2. On completion of the training process, we applied the output DTSVM classifier to the corresponding test data set to obtain the test accuracy rate. All the data sets, divided into training, validation and test components, are available on the following website.

Data set	No. of Labels	No. of Samples	No. of Features
Pen Hand Written (PHW)	10	10,992	16
Letter	26	20,000	16
Shuttle	7	58,000	9
Poker	10	25,010	10
Census Income (CI)	2	45,222	14
News20	20	19,927	62,060
KDD CUP 10% (KDD-10%)	5	494,021	41

Table 1: The medium-size data sets used in our experiments.

Data set	No. of Labels	No. of Samples	No. of Features
Forest	7	581,012	54
PPI	2	1,249,814	14
KDD-full	5	4,898,431	41
Webspam	2	240,000	16,609,143

Table 2: The large-size data sets used in our experiments.

http://ocrwks11.iis.sinica.edu.tw/dar/Download/DataSets/DTSVM/datasets.htm

In each data set, we normalized all the feature values to a real number between 0 and 1. We did this by transforming each value v of feature f into $(v - f_{min})/(f_{max} - f_{min})$, where f_{max} and f_{min} are the maximum and minimum values of f respectively.

We only studied non-linear SVMs in our experiments. Moreover, we used the RBF kernel function to measure the similarity between vectors. As a result, we had two SVM parameters: the penalty factor C, whose values were taken from $\Phi = \{10^a : a = -1, 0, \ldots, 5\}$; and the γ parameter in the RBF function, whose values were taken from $\Psi = \{10^b : b = -4, -3, \ldots, 4\}$. Thus, the set of all SVM parameter values was $\Theta = \Phi \times \Psi$, which comprised 63 pairs of values for (C, γ).

SVM training is implemented under the 1A1 and 1AO approaches. When the 1A1 approach is used, there are $n(n-1)/2$ classifiers, where n is the number of labels. Each classifier assigns one of two possible labels to a given validation/test sample. We use all the classifiers to classify a given validation/test sample $\mathbf{x}$, based on a majority vote. Note that a more efficient technique (Platt et al., 2000) that only requires n classifiers can be used in the validation/testing procedure. However, we adopt Knerr et al.'s (1990) technique, which requires $n(n-1)/2$ classifiers, because we are only interested in the relative, rather than the absolute, performance of the methods compared in our experiments. When the 1AO approach is used, there are n decision functions, each of which is associated with a label. We assign $\mathbf{x}$ the label associated with the decision function that yields the highest functional value.

3.2 Methods Compared

The following methods are compared in our experiments.

CART. CART (Breiman et al., 1984) is similar to the decomposition scheme used in DTSVM, but it differs in terms of the stop and classification criteria. In the training phase, CART stops splitting a node when $IR(f,v) = 0$ for all features f and their values v. In the testing phase, it classifies a test sample $\mathbf{x}$ by the label shared by the majority of samples residing at the leaf to which $\mathbf{x}$ flows. Although CART is not designed for speeding up SVMs, it serves here as a benchmark for DTSVM. If CART performs as well as DTSVM in every respect, then there is no need for DTSVM, since CART runs much faster than DTSVM in both the training and testing phases.

RDSVM. RDSVM (randomized SVM) is an alternative to DTSVM that differs from DTSVM in the way it decomposes a data space. In the training phase, when a size σ is given, DTSVM randomly assigns a training sample to one of d subsets, where d is the smallest integer that is greater than or equal to n/σ and n is the number of training samples. RDSVM uses the same procedure as DTSVM to search for the optimal parameters. In the testing phase, RDSVM randomly assigns a test sample to a subset and classifies it according to the SVM associated with that subset.

Bagging. When implementing bagging (Breiman, 1996), we created a number of SVMs for each $\theta \in \Theta$. Each SVM was trained on 1,500 training samples chosen at random. For each θ, the training was conducted sequentially. We stopped at the first m so that the validation accuracy rate of m SVMs did not exceed that of $m-1$ SVMs by more than 0.5%.

CBD. The training process of CBD (Panda et al., 2006) comprises two steps: finding a reduced set, and training an SVM on that set for each $\theta \in \Theta$. The first part involves finding the k-nearest neighbors of each training sample and deriving the reduced data set via a down-sampling technique. Following Panda et al. (2006), we set k at 100. When searching for the 100 nearest neighbors of each training sample $\mathbf{x}$, we keep the current list of 100 nearest neighbors of $\mathbf{x}$. For another training sample $\mathbf{z}$, let $d(\mathbf{x},\mathbf{z})$ be the distance between $\mathbf{x}$ and $\mathbf{z}$. We need to compare this distance with $d(\mathbf{x},\mathbf{w})$, where $\mathbf{w}$ is on the current list and has the largest distance with $\mathbf{x}$. Since the squared distance is the sum of the squared feature differences, we can speed up the comparison by computing the partial sum of $d^2(\mathbf{x},\mathbf{z})$. When this partial sum exceeds $d^2(\mathbf{x},\mathbf{w})$, we stop the comparison and exclude $\mathbf{z}$ from the current list of $\mathbf{x}$.

LIBSVM. LIBSVM (Fan et al., 2005) is now the most widely used software for training and testing SVMs. We take it as the baseline in our experiments; thus, the speedup factor is 1 by assumption. If a compared method is faster in training than LIBSVM, it has a speedup factor above 1.

LASVM. LASVM (Bordes et al., 2005) is a method that solves a dual optimization problem by way of a stochastic gradient descent method that converges to an optimal solution in one examination of the training samples.

LIBLINEAR. LIBLINEAR (Fan et al., 2008) is a fast version of training and testing *linear* SVMs. Its training speed is comparable to, or even faster than, that of Pegasos (Shalev-Shwartz et al., 2007) and SVM$^{\text{perf}}$ (Joachims, 2006). Since LIBLINEAR is *not* a method for speeding up non-linear SVMs, we only include it in our experiments for large-size data sets, for which bagging, CBD, LIBSVM and LASVM take too long to complete the training process. When we train linear SVMs, the values of the penalty factor C are taken from $\Phi = \{10^a : a = -1, 0, \ldots, 5\}$, which comprises 7 real numbers. Furthermore, we require the discriminant function of the classifier to include a bias term.

Among the above methods, DTSVM, RTSVM, bagging and CBD are data reduction methods, while LIBSVM, LASVM and LIBLIEAR are numerical methods.

3.3 Results on Medium-Size Data Sets

The results of applying seven methods, CART, DTSVM, RDSVM, bagging, CBD, LIBSVM and LASVM, to the seven medium-size data sets are shown in Figures 4-8 for 1A1 training, and in Figures 9-13 for 1AO training. In all SVM training sessions, except for LASVM, we used the LIBSVM software (Fan et al., 2005). We adopted all default options of the software, except the parameter values, which we specified in Section 3.1.

Figure 4 and Figure 9 show the training times of the seven methods. The training time of each method comprises the time required to obtain reduced data sets if it is a data reduction method, the time to train all SVMs and the time to search for optimal parameters; however, the time required to input or output data is *not* included. The computation for all the medium-size data sets was performed on an Intel Xeon CPU 3.2 GHz with a 2GB RAM, while that for all the large-size data sets was performed on a Quad-Core Intel Xeon X5365 3.0GHz CPU and 32GB RAM.

Figure 5 and Figure 10 show the *speedup factors* of all the methods except LIBSVM, where the speedup factor of a method $\mathcal{M}$ is computed as LIBSVM's training time divided by $\mathcal{M}$'s training time.

Figure 6 and Figure 11 show the test accuracy rates of the four compared methods. Note that the DTSVM test accuracy is that of the DTSVM classifier with the ceiling size σ_{opt} and SVM-parameters θ_{opt}. When classifying a test sample with SVMs, the most time-consuming part is computing a decision function, whose complexity can be measured in terms of how many SVs are encountered in the classification. Therefore, we use the "*number of encountered support vectors*" (NESV) as a measure of the time-complexity of the test process. NESV is defined as the number of SVs contained in the decision function used to classify a test sample. When a DTSVM or an RDSVM classifier is used, the NESV is associated with the leaf that the test sample flows to. Thus, in the two cases, NESV is the *average* number of SVs encountered by a test sample.

Figure 7 and Figure 12 list the NESVs of the four methods; while Figure 8 and Figure 13 show the *NESV ratios* of all the methods except LIBSVM and CART, where the NESV ratio of a method $\mathcal{M}$ is computed as LIBSVM's NESV divided by $\mathcal{M}$'s NESV.

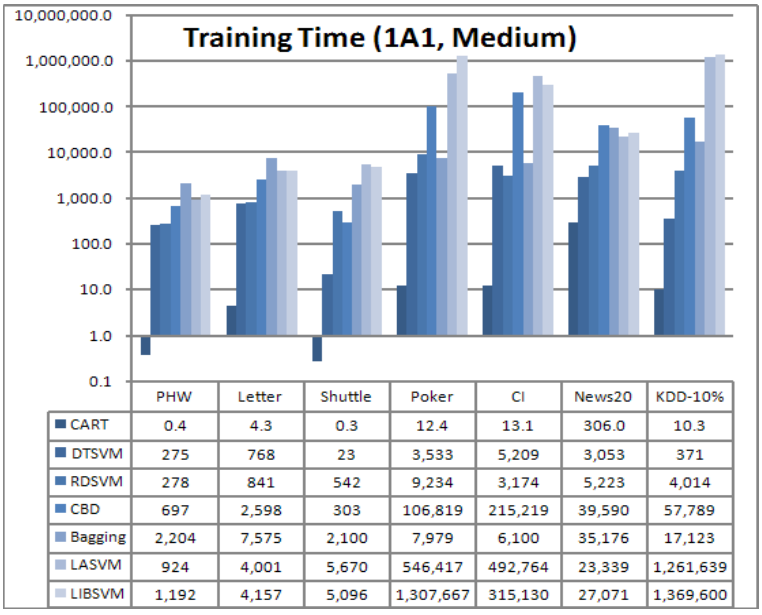
We now summarize the results shown in Figures 4 to 13.

1. There is no doubt that CART was extremely fast in training, but its test accuracy was poor, except on the "Shuttle" and "KDD-10%" data sets, where its accuracy matched the best of all the other methods.

2. In terms of training time, DTSVM outperformed all the other methods, except CART; and in terms of test accuracy and NESV, DSTVM outperformed or performed comparably to all the other methods. It also achieved very large speedup factors and NESV ratios on "Shuttle", "Poker", "CI" and "KDD-10%".

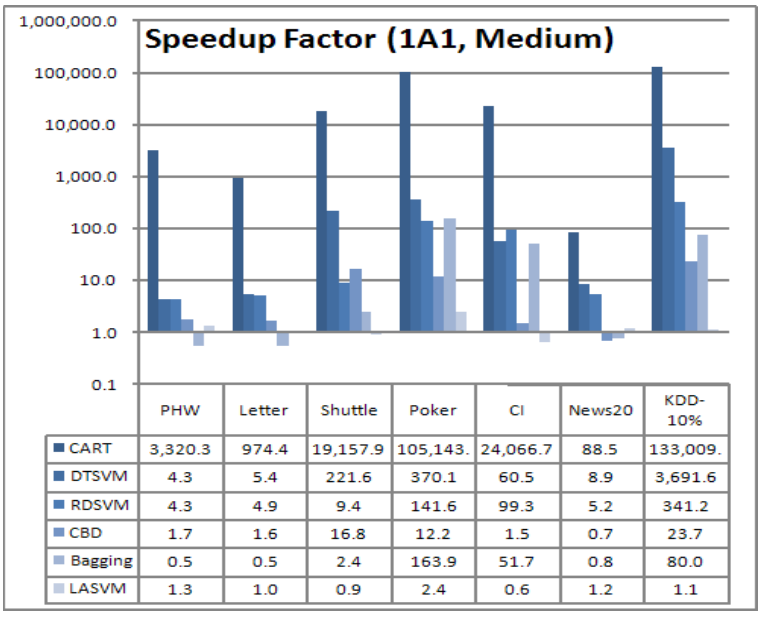
3. RDSVM, being an alternative approach to DTSVM, achieved comparable test accuracy to DTSVM. However, it performed worse in terms of training time on "Shuttle", "Poker" and "KDD-10%". It also performed worse in terms of NESV on "Shuttle", "Poker", "CI" and "KDD-10%".

4. CBD achieved speedup factors above 1 and NESV ratios above 1 on most data sets; however, its scores were overall not as high as those of DTSVM. In addition CBD lagged behind DTSVM in terms of test accuracy on "Letter" and "News20".



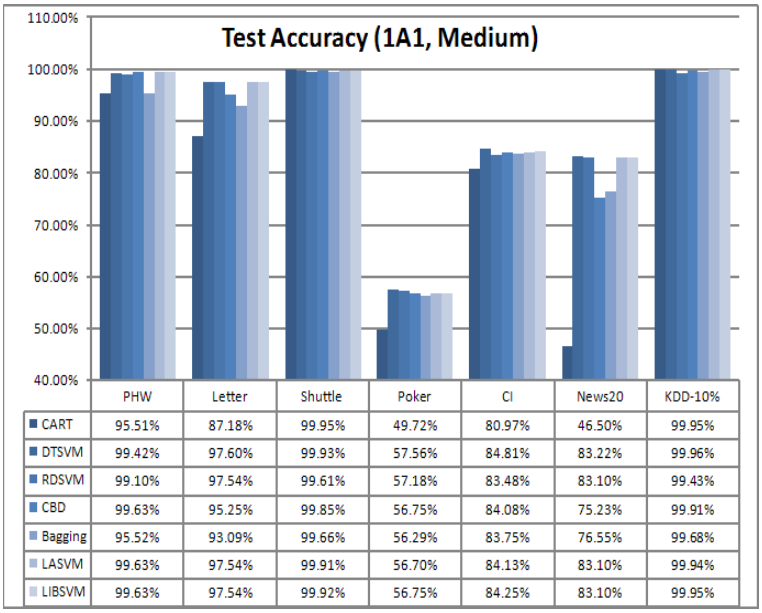
	PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
CART	0.4	4.3	0.3	12.4	13.1	306.0	10.3
DTSVM	275	768	23	3,533	5,209	3,053	371
RDSVM	278	841	542	9,234	3,174	5,223	4,014
CBD	697	2,598	303	106,819	215,219	39,590	57,789
Bagging	2,204	7,575	2,100	7,979	6,100	35,176	17,123
LASVM	924	4,001	5,670	546,417	492,764	23,339	1,261,639
LIBSVM	1,192	4,157	5,096	1,307,667	315,130	27,071	1,369,600

Figure 4: Training times of the seven compared methods, expressed in *seconds*. Training type = 1A1. CART, DTSVM and RDSVM outperformed the other methods.



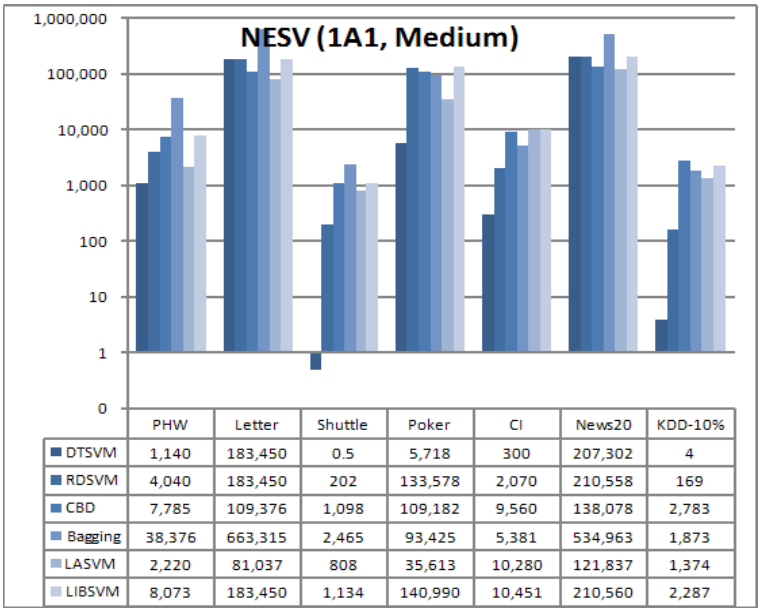
	PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
CART	3,320.3	974.4	19,157.9	105,143.	24,066.7	88.5	133,009.
DTSVM	4.3	5.4	221.6	370.1	60.5	8.9	3,691.6
RDSVM	4.3	4.9	9.4	141.6	99.3	5.2	341.2
CBD	1.7	1.6	16.8	12.2	1.5	0.7	23.7
Bagging	0.5	0.5	2.4	163.9	51.7	0.8	80.0
LASVM	1.3	1.0	0.9	2.4	0.6	1.2	1.1

Figure 5: Speedup factors of all the methods except LIBSVM. Training type = 1A1. CART, DTSVM and RDSVM outperformed the other methods.



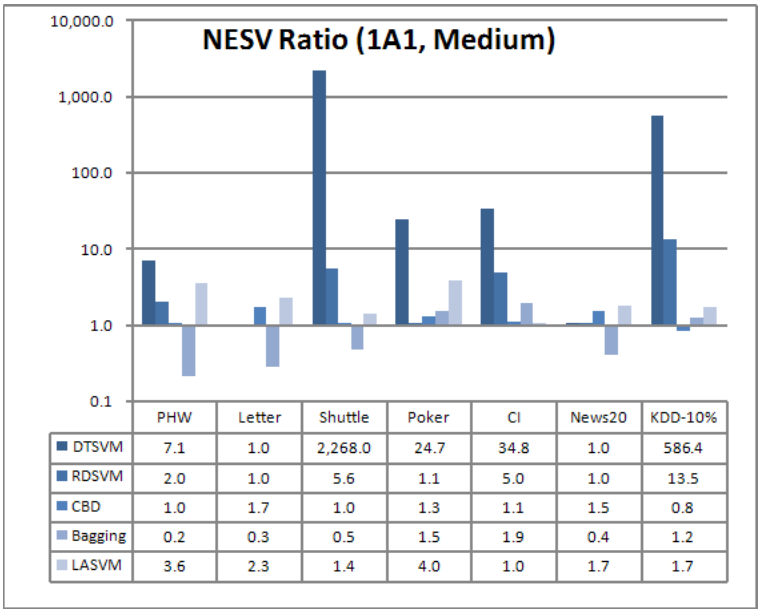
	PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
CART	95.51%	87.18%	99.95%	49.72%	80.97%	46.50%	99.95%
DTSVM	99.42%	97.60%	99.93%	57.56%	84.81%	83.22%	99.96%
RDSVM	99.10%	97.54%	99.61%	57.18%	83.48%	83.10%	99.43%
CBD	99.63%	95.25%	99.85%	56.75%	84.08%	75.23%	99.91%
Bagging	95.52%	93.09%	99.66%	56.29%	83.75%	76.55%	99.68%
LASVM	99.63%	97.54%	99.91%	56.70%	84.13%	83.10%	99.94%
LIBSVM	99.63%	97.54%	99.92%	56.75%	84.25%	83.10%	99.95%

Figure 6: Test accuracy rates of all the methods. Training type = 1A1. CART performed poorly on several data sets; while CBD and Bagging lagged behind DTSVM on some data sets.



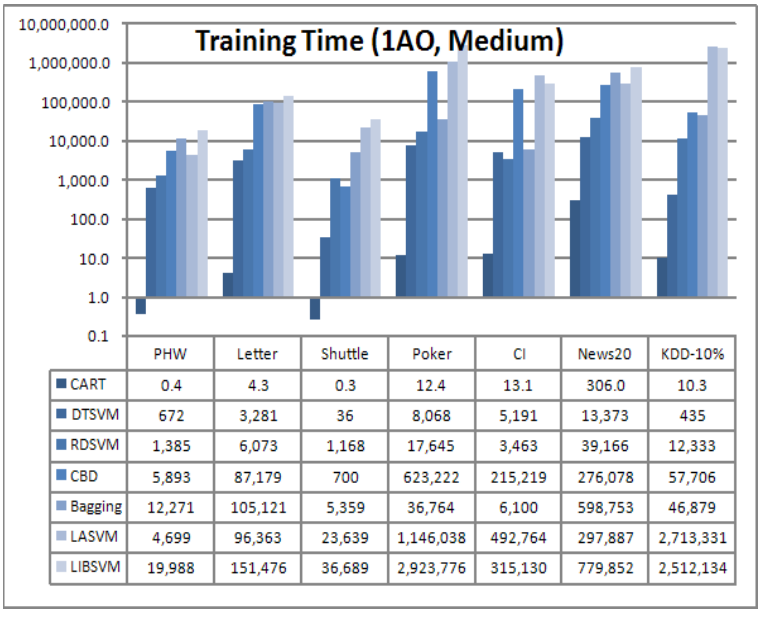
	PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
DTSVM	1,140	183,450	0.5	5,718	300	207,302	4
RDSVM	4,040	183,450	202	133,578	2,070	210,558	169
CBD	7,785	109,376	1,098	109,182	9,560	138,078	2,783
Bagging	38,376	663,315	2,465	93,425	5,381	534,963	1,873
LASVM	2,220	81,037	808	35,613	10,280	121,837	1,374
LIBSVM	8,073	183,450	1,134	140,990	10,451	210,560	2,287

Figure 7: The NESVs of all the methods except CART. Training type = 1A1. DTSVM outperformed all the other methods.



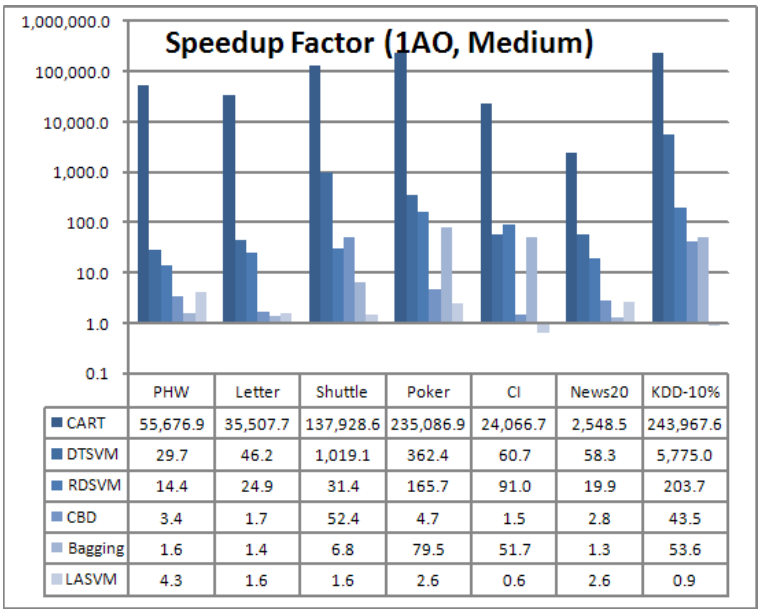
	PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
DTSVM	7.1	1.0	2,268.0	24.7	34.8	1.0	586.4
RDSVM	2.0	1.0	5.6	1.1	5.0	1.0	13.5
CBD	1.0	1.7	1.0	1.3	1.1	1.5	0.8
Bagging	0.2	0.3	0.5	1.5	1.9	0.4	1.2
LASVM	3.6	2.3	1.4	4.0	1.0	1.7	1.7

Figure 8: The NESV ratios of all the methods except CART and LISBSM. Training type = 1A1. DTSVM outperformed all the other methods.



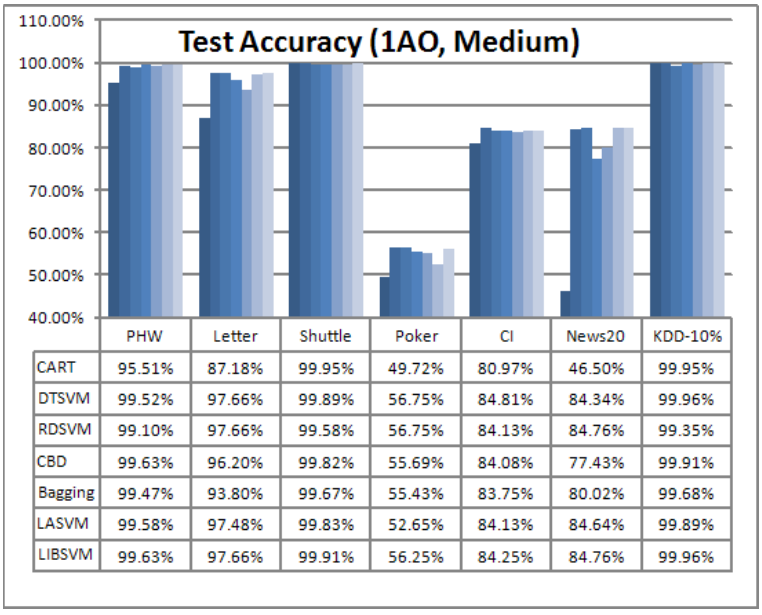
	PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
CART	0.4	4.3	0.3	12.4	13.1	306.0	10.3
DTSVM	672	3,281	36	8,068	5,191	13,373	435
RDSVM	1,385	6,073	1,168	17,645	3,463	39,166	12,333
CBD	5,893	87,179	700	623,222	215,219	276,078	57,706
Bagging	12,271	105,121	5,359	36,764	6,100	598,753	46,879
LASVM	4,699	96,363	23,639	1,146,038	492,764	297,887	2,713,331
LIBSVM	19,988	151,476	36,689	2,923,776	315,130	779,852	2,512,134

Figure 9: Training times of all the methods, expressed in *seconds*. Training type = 1AO. CART, DTSVM and RDSVM outperformed the other methods.



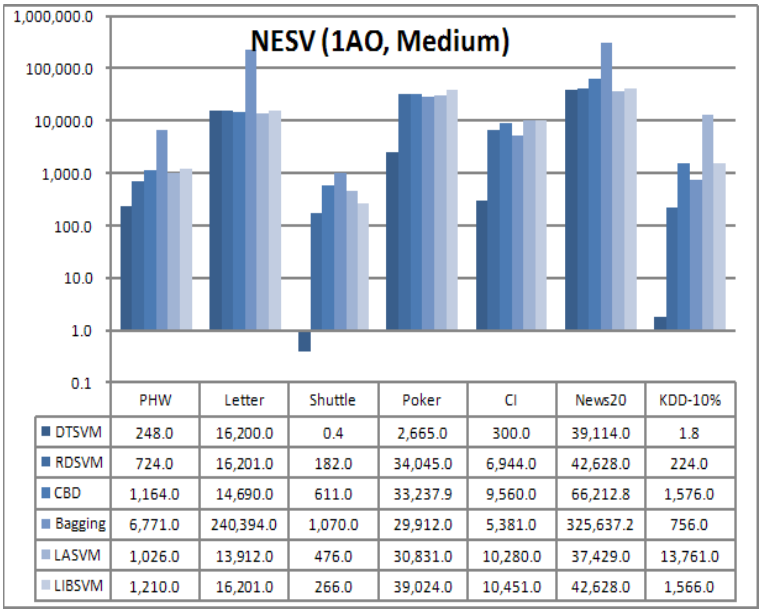
	PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
CART	55,676.9	35,507.7	137,928.6	235,086.9	24,066.7	2,548.5	243,967.6
DTSVM	29.7	46.2	1,019.1	362.4	60.7	58.3	5,775.0
RDSVM	14.4	24.9	31.4	165.7	91.0	19.9	203.7
CBD	3.4	1.7	52.4	4.7	1.5	2.8	43.5
Bagging	1.6	1.4	6.8	79.5	51.7	1.3	53.6
LASVM	4.3	1.6	1.6	2.6	0.6	2.6	0.9

Figure 10: Speedup factors of all the methods except LIBSVM. Training type = 1AO. CART, DTSVM and RDSVM outperformed the other methods.



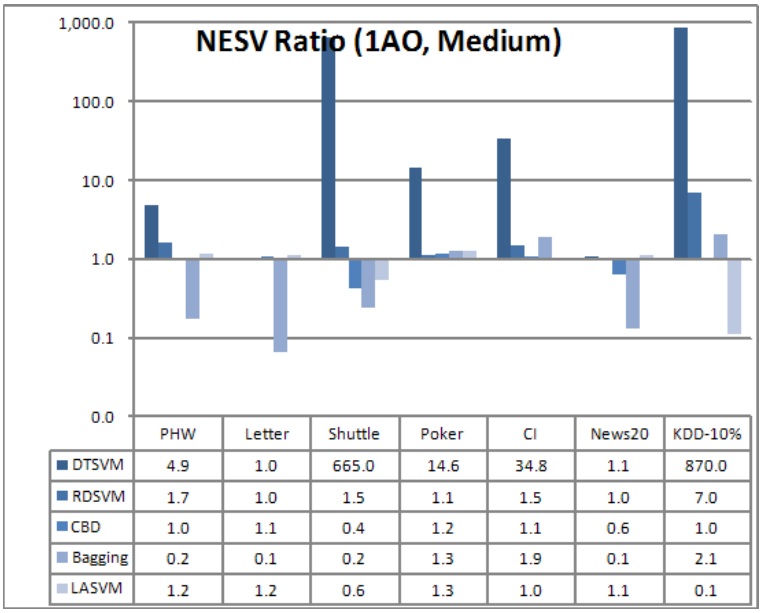
	PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
CART	95.51%	87.18%	99.95%	49.72%	80.97%	46.50%	99.95%
DTSVM	99.52%	97.66%	99.89%	56.75%	84.81%	84.34%	99.96%
RDSVM	99.10%	97.66%	99.58%	56.75%	84.13%	84.76%	99.35%
CBD	99.63%	96.20%	99.82%	55.69%	84.08%	77.43%	99.91%
Bagging	99.47%	93.80%	99.67%	55.43%	83.75%	80.02%	99.68%
LASVM	99.58%	97.48%	99.83%	52.65%	84.13%	84.64%	99.89%
LIBSVM	99.63%	97.66%	99.91%	56.25%	84.25%	84.76%	99.96%

Figure 11: Test accuracy rates of all the methods. Training type = 1AO. CART performed poorly on several data sets; while CBD and Bagging lagged behind DTSVM on some data sets.



	PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
DTSVM	248.0	16,200.0	0.4	2,665.0	300.0	39,114.0	1.8
RDSVM	724.0	16,201.0	182.0	34,045.0	6,944.0	42,628.0	224.0
CBD	1,164.0	14,690.0	611.0	33,237.9	9,560.0	66,212.8	1,576.0
Bagging	6,771.0	240,394.0	1,070.0	29,912.0	5,381.0	325,637.2	756.0
LASVM	1,026.0	13,912.0	476.0	30,831.0	10,280.0	37,429.0	13,761.0
LIBSVM	1,210.0	16,201.0	266.0	39,024.0	10,451.0	42,628.0	1,566.0

Figure 12: The NESVs of all the methods except CART. Training type = 1AO. DTSVM outperformed all the other methods.



	PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
DTSVM	4.9	1.0	665.0	14.6	34.8	1.1	870.0
RDSVM	1.7	1.0	1.5	1.1	1.5	1.0	7.0
CBD	1.0	1.1	0.4	1.2	1.1	0.6	1.0
Bagging	0.2	0.1	0.2	1.3	1.9	0.1	2.1
LASVM	1.2	1.2	0.6	1.3	1.0	1.1	0.1

Figure 13: The NESV ratios of all the methods except CART and LIBSVM. Training type = 1AO. DTSVM outperformed all the other methods.

		PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
1A1	DTSVM	7,694	183,450	181	88,935	8,975	207,290	1,638
	LIBSVM	8,073	183,450	1,134	140,990	10,451	210,560	2,287
1AO	DTSVM	1,775	16,200	195	42,775	8,975	39,122	1,580
	LIBSVM	1,210	16,201	266	39,024	10,451	42,628	1,566

Table 3: The total number of SVs produced by DTSVM and LIBSVM on medium-size data sets. The two methods produced about the same numbers of SVs, even though DTSVM has much smaller NESVs than LIBSVM.

5. Bagging achieved speedup factors below 1 and NESV ratios below 1 on several data sets. It also lagged behind DTSVM in terms of test accuracy on "Letter" and "News20".

6. LASVM, being an alternative numerical method to LIBSVM, achieved speedup factors slightly above 1 on most data sets, but its scores were much lower than those of DTSVM. LASVM also achieved NESV ratios above 1 on most data sets, although they were generally not as high as those of DTSVM. An unexpected result occurred in the 1AO training on "KDD-10%", where LASVM obtained very high NESVs compared to LIBSVM, resulting in a low NESV ratio (0.1). We double checked the process to confirm that the above result was correct.

Finally, we provide some additional information about DTSVM. In Table 3, we show the *total number of support vectors* (TNSV) produced by DTSVM and LIBSVM on all medium-size data sets. TNSV expresses the space-complexity of a training process, whereas NESV expresses the time-complexity of a test process. For DTSVM, the NESV is smaller than the TNSV in most cases, because a test sample usually encounters only some, rather than all, SVs. For LIBSVM, NESV is always the same as TNSV. Note that DTSVM achieved much smaller NESVs on many data sets, but it produced about the same TNSV on all the data sets.

Table 4 shows the DTSVM testing times, as well as the LIBSVM and CART testing times for comparison. As expected from the NESV results, DTSVM's testing time is shorter than that of LIBSVM on all the data sets. We further divide DTSVM's testing time into the time spent on the decision-tree component (DTC) and that spent on local SVMs (lSVMs). In Table 4, the times are separated by a semi-colon. Clearly, the DTC testing time takes an extremely small proportion of DTSVM's testing time. In fact, it is so small that it cannot be measured accurately by the timing mechanism. CART's testing time, on the other hand, is higher than that of the DTC because CART-trees usually grow to deeper levels than DTC-trees.

3.4 Results on Large-Size Data Sets

The results of applying four methods, CART, DTSVM, RDSVM and LIBLINEAR, to the four large-size data sets are shown in Figures 14-16 for 1A1 training, and in Figures 17-19 for 1AO training. Once again, we used the LIBSVM software for all SVM training sessions of DTSVM and RDSVM.

We summarize the results on large-size data sets as follows.

1. Even though DTSVM was not as fast as CART and LIBLINEAR in training, it achieved consistently high test accuracy rates on all the four data sets. It outperformed LIBLINEAR

		PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
CART		0.063	0.125	0.360	0.157	0.204	0.093	2.187
1A1	DTSVM	0.047	3.844	0.000	0.406	0.219	38.141	0.063
		0; 0.047	0; 3.844	0; 0.000	0; 0.406	0; 0.219	0; 38.141	0; 0.063
	LIBSVM	0.265	4.078	0.187	7.406	8.453	39.062	7.344
1AO	DTSVM	0.062	7.344	0.000	1.125	0.219	126	0.063
		0; 0.062	0; 7.344	0; 0.000	0; 1.125	0; 0.219	0; 126	0; 0.063
	LIBSVM	0.328	7.328	0.250	17.313	8.453	138.000	19.875

Table 4: The testing time required by CART, DTSVM and LIBSVM on medium-size data sets. The time required by DTSVM is lower than that required by LIBSVM. The DTC testing time takes a very small proportion of DTSVM's testing time. CART's testing time is higher than that of DTC.

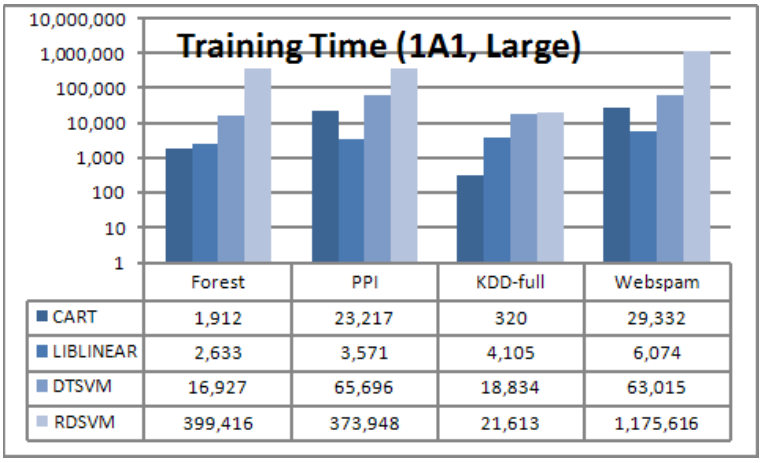
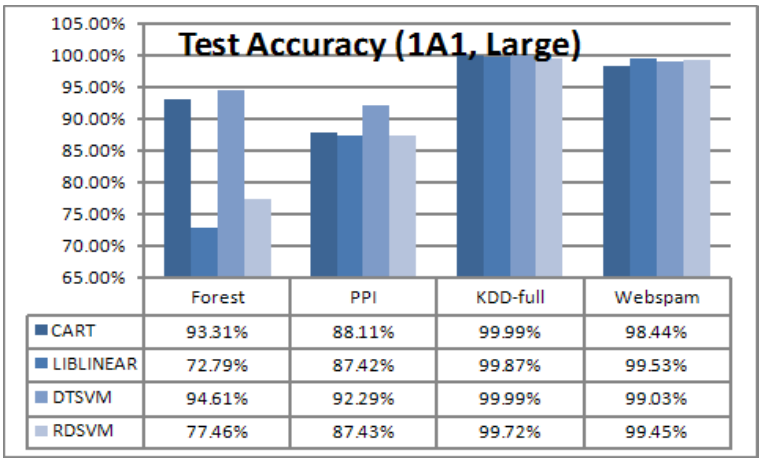


Figure 14: Training times of the four methods, expressed in *seconds*. Training type = 1A1. CART and LIBLINEAR outperformed the other methods.

and RDSVM on "Forest" and "PPI", and surpassed CART on "PPI" by a significant margin. Moreover, DTSVM achieved much lower NESVs than RDSVM on all the data sets.

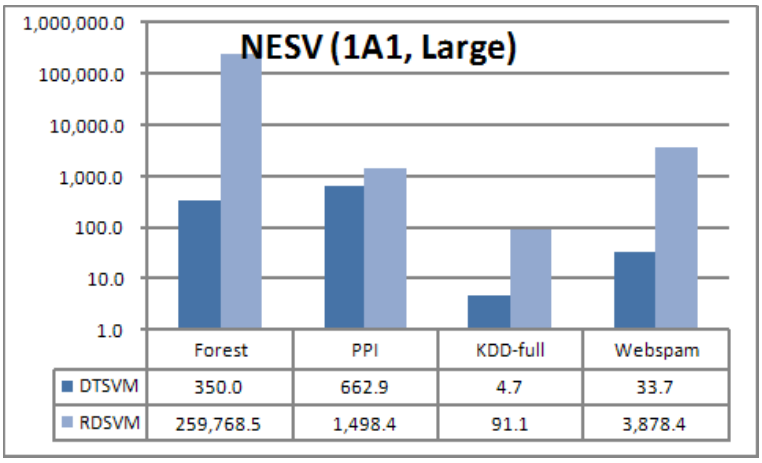
2. Recall that RDSVM achieved equally good test accuracy rates on all the medium-size data sets. However, on the large-size data sets, it achieved much lower accuracy rates on "Forest" and "PPI", and it yielded much higher NESVs on all the data sets. The results show that RDSVM is *not* a good substitute for DTSVM in solving large-scale SVM problems.

3. The results also show that, despite their efficiency in training, CART and LIBLINEAR are *not* good substitutes for DTSVM in solving large-scale problems. LIBLINEAR achieved the best test accuracy rate on "Webspam", presumably because a linear model fits this data set



	Forest	PPI	KDD-full	Webspam
CART	93.31%	88.11%	99.99%	98.44%
LIBLINEAR	72.79%	87.42%	99.87%	99.53%
DTSVM	94.61%	92.29%	99.99%	99.03%
RDSVM	77.46%	87.43%	99.72%	99.45%

Figure 15: Test Accuracy of the four methods. Training type = 1A1. DTSVM outperformed, or performed comparably to, the other methods. CART performed rather well compared to LIBLINEAR.

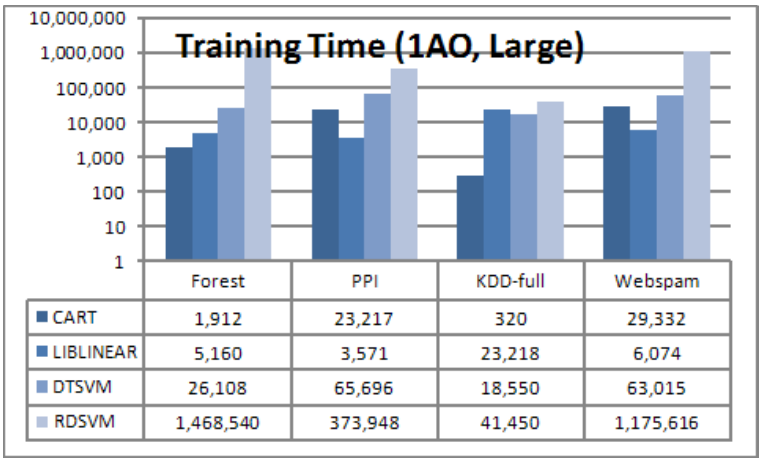


	Forest	PPI	KDD-full	Webspam
DTSVM	350.0	662.9	4.7	33.7
RDSVM	259,768.5	1,498.4	91.1	3,878.4

Figure 16: The NESVs of DTSVM and RDSVM. Training type = 1A1. DTSVM achieved much lower NESVs than RDSVM.

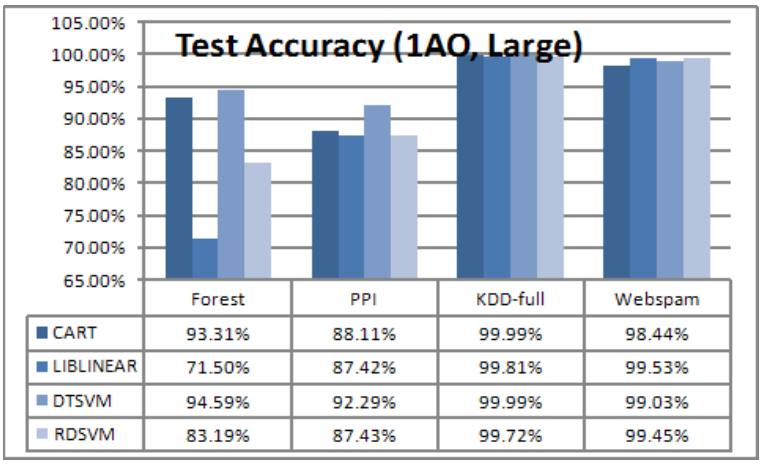
rather well. However, to verify this assumption, we need to compare the test accuracy rates of linear and non-linear models. DTSVM offers us an opportunity to make such a comparison.

We also show the total number of SVs produced by DTSVM in Table 5, while the testing times required by DTSVM and CART are shown in Table 6. DTSVM's testing time is further divided into the amount of time required by the DTC and that required by lSVMs. Once again, it is clear that DTC's testing time only takes a very small proportion of DTSVM's testing time. CART's testing time is higher than that of DTC because CART-trees usually grow to deeper levels than DTC-trees.



	Forest	PPI	KDD-full	Webspam
CART	1,912	23,217	320	29,332
LIBLINEAR	5,160	3,571	23,218	6,074
DTSVM	26,108	65,696	18,550	63,015
RDSVM	1,468,540	373,948	41,450	1,175,616

Figure 17: Training times of the four methods, expressed in *seconds*. Training type = 1AO. CART and LIBLINEAR outperformed the other methods.



	Forest	PPI	KDD-full	Webspam
CART	93.31%	88.11%	99.99%	98.44%
LIBLINEAR	71.50%	87.42%	99.81%	99.53%
DTSVM	94.59%	92.29%	99.99%	99.03%
RDSVM	83.19%	87.43%	99.72%	99.45%

Figure 18: Test Accuracy of the four methods. Training type = 1AO. DTSVM outperformed, or performed as well as, the other methods.

3.5 Further Discussion

To gain insight into why DTSVM is so effective, we show in Table 7 the σ_{opt} derived by DTSVM on medium-size data sets, along with the proportion of training samples that flow to homogeneous leaves. Note that a single table suffices to show all the results because 1A1 training and 1AO training employ the same decision trees and DTSVM yields the same σ_{opt} value for both approaches.

First, we observe that DTSVM required a low ceiling size of 1,500 on all the data sets, except "Letter" and "News20." This explains why DTSVM generally achieved good speedup factors and NESV ratios. Furthermore, the proportion of training samples that flowed to homogeneous leaves under DTSVM was very high in "Shuttle" and "KDD-10%". Since no SVM classifier is involved

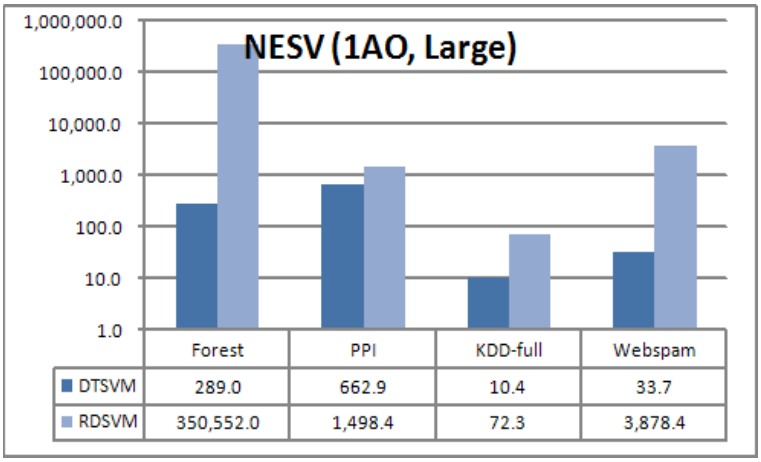


Figure 19: The NESVs of DTSVM and RDSVM. Training type = 1AO. DTSVM achieved much lower NESVs than RDSVM.

		Forest	PPI	KDD-full	Webspam
DTSVM	1A1	140,008	594,687	2,752	8,116
	1AO	114,958	594,687	2,781	8,116

Table 5: The total number of SVs produced by DTSVM on large-size data sets.

		Forest	PPI	KDD-full	Webspam
CART		0.109	0.563	0.281	151.860
DTSVM	1A1	2.485	38.453	1.094	127.000
		0.047; 2.438	0.235; 38.218	0.25; 0.844	75; 52.000
	1AO	3.859	38.453	1.485	127.000
		0.047; 3.812	0.235; 38.218	0.25; 1.235	75; 52.000

Table 6: The testing time required by CART and DTSVM on large-size data sets. DTC's testing time only takes a very small proportion of DTSVM's testing time. CART's testing time is higher than that of DTC.

	PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
σ_{opt}	1,500	24,000	1,500	1,500	1,500	24,000	1,500
Proportion	0%	0%	98.42%	0%	15.76%	0%	97.35%

Table 7: The σ_{opt} obtained by DTSVM on the medium-size data sets and the proportion of training samples that flow to homogeneous leaves.

		PHW	Letter	Shuttle	Poker	CI	News20	KDD-10%
1A1	DTSVM	1,500	24,000	1,500	1,500	1,500	24,000	1,500
	RDSVM	1,500	24,000	1,500	24,000	6,000	24,000	1,500
1AO	DTSVM	1,500	24,000	1,500	1,500	1,500	24,000	1,500
	RDSVM	1,500	24,000	1,500	24,000	24,000	24,000	1,500

Table 8: The σ_{opt} values derived by DTSVM and RDSVM on the medium-size data sets.

Data Set	Training Mode	1,500	6,000	24,000
Letter	1A1	633	45	90
	1AO	2,730	178	373
News20	1A1	1,631	665	757
	1AO	7,056	2,952	3,365

Table 9: The DTSVM training times required for different ceiling sizes.

Data Set	Training Mode	1,500	6,000	24,000
Letter	1A1	95.35%	96.61%	97.60%
	1AO	95.71%	96.91%	97.66%
News20	1A1	67.02%	76.92%	83.22%
	1AO	70.82%	79.06%	84.34%

Table 10: The DTSVM test accuracy rates that correspond to different ceiling sizes.

in any homogeneous leaves, DTSVM achieved very high speedup factors and NESV ratios on these two data sets. The same fact also explains why DTSVM achieved such low NESVs, which even fell below 1 on "Shuttle". Note that this phenomenon occurs because decision trees group neighboring samples into the same leaf. RDSVM, on the other hand, does not produce the same effect because the probability that all samples will carry the same label in the same randomly decomposed region is extremely small.

The lack of homogeneous leaves is not the only reason for RDSVM's poor performance in training. Table 8 shows the σ_{opt} values derived by DTSVM and RDSVM. The results explain why RDSVM achieved much smaller speedup factors and NESV ratios on "CI" and "Poker".

Next, we examine the DTSVM results for the "Letter" and "News20" data sets in which the optimal sizes The σ_{opt} exceeded the size of the training component. Thus, the output DTSVM classifier was trained on the full training component. Even so, DTSVM still achieved speedup factors above 1 because it only trained lSVMs for all the parameter values on leaves with a ceiling size of 1,500, which took much less time than training them on the full training component. The amount of time spent on higher ceiling sizes did not increase at a faster rate, because DTSVM only trained a small number of lSVMs. Moreover, the lSVMs were trained with top-ranked parameters, which tended to require less time than those trained with bottom-ranked parameters.

	Forest	PPI	KDD-full	Webspam
σ_{opt}	1,500	1,500	1,500	1,500
Proportion	5.55%	1.64%	42.05%	80.63%

Table 11: The σ_{opt} values obtained by DTSVM on the large-size data sets and the proportion of training samples that flowed to homogeneous leaves.

		Forest	PPI	KDD-full	Webspam
1A1	DTSVM	1,500	1,500	1,500	1,500
	RDSVM	96,000	1,500	1,500	96,000
1AO	DTSVM	1,500	1,500	1,500	1,500
	RDSVM	1,536,000	1,500	1,500	96,000

Table 12: The σ_{opt} values obtained by DTSVM and RDSVM on the large-size data sets.

Table 9 shows the training times required for different ceiling sizes. We observe that DTSVM spent most of its time on the leaves with the lowest ceiling size. In addition, Table 10 shows the test accuracy rates corresponding to different ceiling sizes, assuming that the training was terminated at those sizes. The results demonstrate the benefit of searching for the σ_{opt} because, if we terminated the training at ceiling size 1,500 or 6,000, we would obtain significantly lower test accuracy rates.

Thus far, we have only discussed the DTSVM results for medium-size data sets. For completeness, Table 11 shows the σ_{opt} values obtained by DTSVM on the large-size data sets, along with the proportion of training samples that flowed to homogeneous leaves. In Table 12, we show the σ_{opt} values obtained by DTSVM and RDSVM. The results explain why RDSVM required much longer training times and more NESVs for "Forest", "PPI" and "Webspam".

Since DTSVM's $\sigma_{opt} = 1,500$ for all four data sets, we do not have any tables for the large-size data sets correspond to Tables 9 and 10 for the medium-size data sets.

4. Generalization Error Bounds for the DTSVM Classifier

To provide a generalization error bound of DTSVM, we start with the following framework. Let $\mathbf{R}^d$ be the d-dimensional Euclidean space. We assume that a set of training examples $X_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ is given, where $(\mathbf{x}_k, y_k) \in \mathbf{R}^d \times \{-1, 1\}$ for $k = 1, \ldots, n$. The DTSVM method produces a classifier $h(\mathbf{x}, \pi, \mathbf{f})$, in which π is a binary tree comprised of L leaves, $\mathbf{f} = (f_1, \ldots, f_L)$, and f_i is related to the lSVM trained on leaf i of π for $i = 1, \ldots, L$. The binary tree π produces a partition function that maps an input in $\mathbf{R}^d$ to $\{1, \ldots, L\}$, and let $\pi(\mathbf{x})$ be the leaf that $\mathbf{x}$ flows to. On the other hand, the lSVM trained on leave i, $i = 1, \ldots, L$ is expressed as $f_i \circ \Phi$, where Φ maps an input in $\mathbf{R}^d$ to a Hilbert space $\mathbf{H}$, and f_i is a *linear* function from $\mathbf{H}$ to $\mathbf{R}$. Note that for a linear function $g : \mathbf{H} \to \mathbf{R}$, there exists some $w \in \mathbf{H}$ such that

$$g(z) = \langle w, z \rangle$$

for all $z \in \mathbf{H}$, and we define $\|g\| = \langle w, w \rangle^{1/2}$.

Note that if $\pi(\mathbf{x}) = i$, then $h(\mathbf{x}, \pi, \mathbf{f}) = sign(f_i(\Phi(\mathbf{x}))$. Let us define the function $\mathbf{f}^\pi : \mathbf{R}^d \to \{-1, 1\}$ by

$$f^\pi(\mathbf{x}) = f_{\pi(\mathbf{x})}(\Phi(\mathbf{x})).$$

It follows that $h(\mathbf{x}, \pi, \mathbf{f}) = sign(\mathbf{f}^\pi(\mathbf{x}))$.

Sometimes, Φ is only defined implicitly. That is, instead of specifying the functional form of Φ, only the inner product of $\Phi(\mathbf{u})$ and $\Phi(\mathbf{v})$ is specified as

$$\langle \Phi(\mathbf{u}), \Phi(\mathbf{v}) \rangle = k(\mathbf{u}, \mathbf{v}),$$

where $\mathbf{u}, \mathbf{v} \in \mathbf{R}^d$, and $k(\cdot, \cdot)$ is a kernel function. In the remainder of this section, we assume that the function Φ is given and fixed.

Next, we define several notations used in this section. $\mathbf{N}$ is the set of natural numbers; $\mathbf{R}^+$ is the set of positive real numbers; $\mathcal{P}_L(\mathbf{R}^d)$ is the class of all functions from $\mathbf{R}^d$ to $\{1, \ldots, L\}$; $\mathcal{R}(\mathbf{H})$ is the class of all functions from $\mathbf{H}$ to $\mathbf{R}$; and $\mathcal{L}(\mathbf{H})$ is the class of all *linear* functions from $\mathbf{H}$ to $\mathbf{R}$. Moreover, if T is a set, we define $T^L = \{(t_1, \ldots, t_L) : t_i \in T \text{ for } i = 1, \ldots, L\}$; that is, T^L comprises all the L-tuples of T's elements.

To provide a bound for the generalization error of $h(\mathbf{x}, \pi, \mathbf{f})$, we need the standard notions of shatter coefficients, margins and covering numbers, which are defined below. Further details can be found in Vapnik (1995) and Cristianini and Shawe-Taylor (2000). First, we define the notion of the shatter coefficient, which we use to measure the complexity of the partition functions corresponding to binary decision trees. Informally, the n^{th} shatter coefficient of a class $\mathcal{G} \subseteq \mathcal{P}_L(\mathbf{R}^d)$ is the maximum number of ways in which n points can be partitioned into L parts by functions in $\mathcal{G}$. Formally, we have the following definition.

Definition 1 *Let $\mathcal{G} \subseteq \mathcal{P}_L(\mathbf{R}^d)$. For any $n \in \mathbf{N}$, the n^{th} shatter coefficient of $\mathcal{G}$ is*

$$V(\mathcal{G}, n) = \max_{S \subseteq R^d, |S| = n} |\{\pi_S : \pi \in \mathcal{G}\}|,$$

where π_S is the function obtained by restricting π to the domain S.

Next, we extend the standard notion of the margin to a collection of margins, one for each part of a partition, in the following way.

Definition 2 *Let $\mathbf{f} = (f_1, \ldots, f_L) \in (\mathcal{R}(\mathbf{H}))^L$, $\pi \in \mathcal{P}_L(\mathbf{R}^d)$, $X_n \subseteq \mathbf{R}^d \times \{-1, 1\}$, and $\gamma = (\gamma_1, \ldots, \gamma_L) \in (\mathbf{R}^+)^L$. We say that $\mathbf{f}^\pi$ has margin γ on X_n, or $mg(\mathbf{f}^\pi, X_n) \geq \gamma$, if*

$$y \cdot \mathbf{f}^\pi(\mathbf{x}) \equiv y \cdot f_i(\Phi(\mathbf{x})) \geq \gamma_i$$

for any $i \in \{1, \ldots, L\}$ and any $(\mathbf{x}, y) \in X_n$ with $\pi(\mathbf{x}) = i$.

In addition, we adopt the following notion of a covering number proposed by Alon et al. (1997). We use it to measure the complexity of the classifiers corresponding to lSVMs. Informally, a covering of a class $\mathcal{F} \subseteq \mathcal{R}(\mathbf{H})$ of functions with respect to a set D of n inputs is a collection of functions in $\mathcal{R}(\mathbf{H})$ such that any function $f \in \mathcal{F}$ is covered by some function g in the collection, in the sense that their values are within some distance η on all the inputs in D. The goal is to find the smallest such collection for any set D of n inputs in some domain E. Formally, we define the covering number as follows.

Definition 3 *Let $\eta \in \mathbf{R}^+$ and $\mathcal{F} \subseteq \mathcal{R}(\mathbf{H})$. For a subset $D \subseteq \mathbf{H}$, let $\mathcal{C}(\mathcal{F}, D, \eta)$ be the smallest collection of functions from D to $\mathbf{R}$ such that, for each $f \in \mathcal{F}$, we have $g \in \mathcal{C}(\mathcal{F}, D, \eta)$ with $|f(z) - g(z)| \leq \eta$ for each $z \in D$. For $E \subseteq \mathbf{H}$ and $n \in \mathbf{N}$, we define the covering number of $\mathcal{F}$ with respect to E, n and η as*

$$N(\mathcal{F}, E, n, \eta) = \max_{D \subseteq E, |D|=n} |\mathcal{C}(\mathcal{F}, D, \eta)|.$$

4.1 Hard Margin Bounds

We have a set X_n of n training examples drawn independently and at random according to the distribution $\mathcal{D}$. Moreover, we have a learned classifier $sign(\mathbf{f}^\pi)$, with $\pi \in \mathcal{P}_L(\mathbf{R}^d)$ and $\mathbf{f} \in (\mathcal{R}(\mathbf{H}))^L$, which classifies all the training examples correctly with a large margin. Our objective is to bound the *generalization error* of the classifier $sign(\mathbf{f}^\pi)$, which is defined as the probability that $sign(\mathbf{f}^\pi(\mathbf{x})) \neq y$, with $(\mathbf{x}, y)$ sampled according to $\mathcal{D}$. The following lemma gives such a bound, which generalizes a known result for SVMs (cf. Cristianini and Shawe-Taylor, 2000).

Lemma 4 *Let $\mathcal{G} \subseteq \mathcal{P}_L(\mathbf{R}^d)$, $\gamma = (\gamma_1, \ldots, \gamma_L) \in (\mathbf{R}^+)^L$, and $\mathcal{F} = \mathcal{F}_1 \times \cdots \times \mathcal{F}_L$, where $\mathcal{F}_i \subseteq \mathcal{R}(\mathbf{H})$ for $1 \leq i \leq L$. In addition, let $\mathcal{D}$ be a probability distribution on $\mathbf{R}^d \times \{-1, 1\}$, and let n be a large enough integer. Suppose a set of n samples X_n are drawn independently and at random according to $\mathcal{D}$, and consider any classifier $sign(\mathbf{f}^\pi)$, with $\pi \in \mathcal{G}$ and $\mathbf{f} \in \mathcal{F}$, such that $mg(\mathbf{f}^\pi, X_n) \geq \gamma$. Then, with probability $1 - \delta$, the generalization error of $mg(\mathbf{f}^\pi)$ will be at most*

$$\frac{2}{n}\left[\sum_{i=1}^{L} \log N(\mathcal{F}_i, E, 2n, \gamma_i/2) + \log V(\mathcal{G}, 2n) + \log(2/\delta)\right],$$

where $E = \{\Phi(\mathbf{x}) : (\mathbf{x}, y) \in supp(\mathcal{D})\}$ and $supp(\mathcal{D})$ is the support of $\mathcal{D}$.

We provide the proof in Appendix A, as it is rather lengthy and closely follows the standard approach and that of Cristianini and Shawe-Taylor (2000). The idea is to show that the functions $\mathbf{f}^\pi$, with $\pi \in \mathcal{G}$ and $\mathbf{f} \in \mathcal{F}$, can be "well covered" by a small number of functions, so that a union bound can be applied to provide an upper bound on the probability that our classifier has a large generalization error.

Before proceeding further, we explain the meaning of Lemma 4. Suppose for some $\mathcal{G}$ and $\mathcal{F} = \mathcal{F}_1 \times \cdots \times \mathcal{F}_L$, we can build a classifier $sign(\mathbf{f}^\pi)$ with $\pi \in \mathcal{G}$ and $\mathbf{f} \in \mathcal{F}$ that has a large margin and zero training error. Then, Lemma 4 gives us an upper bound on the generalization error of $sign(\mathbf{f}^\pi)$ in terms of the complexity of $\mathcal{G}$ and $\mathcal{F}_i$, where we measure the complexity of $\mathcal{G}$ by its shatter coefficient $V(\mathcal{G}, 2n)$ and the complexity of each $\mathcal{F}_i$ by its covering number $N(\mathcal{F}_i, E, 2n, \gamma_i/2)$. To obtain a small generalization error, we need to have a $\mathcal{G}$ with a small $V(\mathcal{G}, 2n)$ and an $\mathcal{F}_i$ with a small $N(\mathcal{F}_i, E, 2n, \gamma_i/2)$. However, this does not suggest that we can simply choose any $\mathcal{G}$ and $\mathcal{F}_i$ with small $V(\mathcal{G}, 2n)$ and $N(\mathcal{F}_i, E, 2n, \gamma_i/2)$ for any classification task. This is because we may not be able to build a classifier from $\mathcal{G}$ and $\mathcal{F}_i$ to classify every training example correctly with a large margin. For example, if we do not choose $\mathcal{G}$ properly (say, by using a random partition), the training examples in some parts of the partition may not be separated by SVMs with a large margin. Our main contribution is the discovery that decision trees are good partition functions when combined with SVMs, since they allow a large margin and only require a short training time for lSVMs.

Lemma 4 states a general result for any $\mathcal{G}$ and any $\mathcal{F}_i$ for $1 \leq i \leq L$. We now consider our choice of $\mathcal{G}$ and $\mathcal{F}_i$, which correspond to decision trees and SVMs respectively. For $\beta \in \mathbf{R}^+$, we define

$\mathcal{L}(\mathbf{H},\beta)$ as the class of all linear functions $f \in \mathcal{L}(\mathbf{H})$ with $\|f\| \leq \beta$. A bound can be obtained for the covering number of $\mathcal{L}(\mathbf{H},\beta)$ with respect to E, n and η, provided that E is a bounded subset of $\mathbf{H}$ (see, for example, Bartlett and Shawe-Taylor, 1998). This shows that, although there is an infinite number of functions in $\mathcal{L}(\mathbf{H},\beta)$, they can be covered by a small number of functions in $\mathcal{R}(\mathbf{H})$ with respect to any set of n points in E. As a result, the complexity of the class $\mathcal{L}(\mathbf{H},\beta)$ is low when measured by its covering number.

Lemma 5 *Let $\alpha,\beta,\eta \in \mathbf{R}^+$ and let $n \in \mathbf{N}$. Consider any $E \subseteq \mathbf{H}$ with $\|z\| \leq \rho$ for every $z \in E$. Then, there is a constant c such that*

$$\log N(\mathcal{L}(\mathbf{H},\beta),E,n,\eta) \leq c\frac{\rho^2\beta^2}{\eta^2}\log^2 n.$$

We also define $\mathcal{B}_L(\mathbf{R}^d)$ as the class of partition functions associated with binary trees containing L leaves that partition the space $\mathbf{R}^d$ into L axis-aligned parts, as described in Section 2. Clearly, $\mathcal{B}_L(\mathbf{R}^d) \subseteq \mathcal{P}_L(\mathbf{R}^d)$. The following lemma provides a bound on the n^{th} shatter coefficient of $\mathcal{B}_L(\mathbf{R}^d)$.

Lemma 6 *Let $d,n,L \in \mathbf{N}$. Then*

$$V(\mathcal{B}_L(\mathbf{R}^d),n) \leq L\log(dnL^2).$$

Proof Consider any n-element subset $S \subseteq \mathbf{R}^d$. We need to cut S into L parts. Initially, there is only one part in S. We perform the cut operation iteratively in the following way. Each time, we choose one part from at most $L-1$ existing parts of S and cut it into two parts. To do this, we pick one of the d dimensions and at most one of the $n-1$ cutting hyperplanes on that dimension. Thus, there are at most $(L-1)d(n-1) \leq dnL$ ways to perform one cut operation. To obtain L parts, we repeat the cut operation L-1 times; hence, the number of possible partitions is at most $(dnL)^{L-1} \leq (dnL)^L$. Finally, there are $L!$ ways to order the L parts of each partition, yielding the following result:

$$V(\mathcal{B}_L(\mathbf{R}^d),n) \leq (dnL)^L \cdot (L!) \leq (dnL^2)^L.$$

■

From the above three lemmas, we immediately derive the following theorem.

Theorem 7 *Let $\rho \in \mathbf{R}^+$, $\beta_i \in \mathbf{R}^+$ for $1 \leq i \leq L$, $\gamma = (\gamma_1,\ldots,\gamma_L) \in (\mathbf{R}^+)^L$, and $\mathcal{F} = \mathcal{L}(\mathbf{H},\beta_1) \times \cdots \times \mathcal{L}(\mathbf{H},\beta_L)$. In addition, let $\mathcal{D}$ be a probability distribution on $\mathbf{R}^d \times \{-1,1\}$ such that $\|\Phi(\mathbf{x})\| \leq \rho$ for every $(\mathbf{x},y) \in supp(\mathcal{D})$, and let n be a large enough integer. Suppose a set X_n of n samples are drawn independently and at random according to $\mathcal{D}$, and consider any classifier $sign(\mathbf{f}^\pi)$, with $\pi \in \mathcal{B}_L(\mathbf{R}^d)$ and $\mathbf{f} \in \mathcal{F}$, such that $mg(\mathbf{f}^\pi,X_n) \geq \gamma$. Then, with probability $1-\delta$, the generalization error of $sign(\mathbf{f}^\pi)$ will be at most*

$$\frac{c}{n}\left(\sum_{i=1}^{L}\frac{\rho^2\beta_i^2}{\gamma_i^2}\log^2 n + L\log(dnL^2) + \log(1/\delta)\right),$$

for some constant c.

From Theorem 7, we conclude that if a classifier partitions the training data set into a small number of parts (i.e., L is sufficiently small), and it classifies each training sample with a large margin (i.e., γ_i is large for each i), then the classifier is likely to have a small generalization error. However, this theorem does not indicate how to find a good value for L because, in general, it is hard to know how L affects the margins and the generalization error bound. Instead of being led by any analytical result, the DTSVM learning algorithm takes a data driven approach to find a good L.

Note that the bound in Theorem 7 has a similar form to the generalization error bound of the perceptron decision trees proposed by Bennett et al. (2000). The main difference is that in their bound, γ_i is the margin at the i^{th} internal node and the sum is over all the internal nodes. We remark that it is hard to tell which one of these two bounds is better because, in general, we do not know their actual values for different learning tasks.

4.2 Soft Margin Bounds

Note that Theorem 7 works in the case where the training data X_n can be separated with a margin vector γ. If the data is non-separable or noisy, we need to consider the notion of a soft margin (Cristianini and Shawe-Taylor, 2000). Suppose we have $\pi \in \mathcal{P}_L(\mathbf{R}^d)$, which partitions X_n into L parts: $X_n^1, \ldots, X_n^{(L)}$, where $X_n^{(i)} \equiv \{(\mathbf{x}, y) \in X_n : \pi(\mathbf{x}) = i\}$, and let us denote $X_n^{(i)} = \{(\mathbf{x}_{i,1}, y_{i,1}), \ldots, (\mathbf{x}_{i,n_i}, y_{i,n_i})\}$ for some $n_i \in \mathbf{N}$, for $1 \le i \le L$. In addition, suppose we have $\mathbf{f} = (f_1, \ldots, f_L) \in \mathcal{L}(\mathbf{H}, \beta_1) \times \cdots \times \mathcal{L}(\mathbf{H}, \beta_L)$, where $\beta_i \in \mathbf{R}^+$ for $1 \le i \le L$. Then, we can define the margin slack vector of f_i as follows.

Definition 8 *Let $\gamma = (\gamma_1, \ldots, \gamma_L) \in (\mathbf{R}^+)^L$. For $1 \le i \le L$ and $1 \le j \le n_i$, let*

$$\xi_{i,j} = \max(0, \gamma_i - y_{i,j} \cdot f_i(\Phi(\mathbf{x}_{i,j}))).$$

The definition reflects how far the elements of $X_n^{(i)}$ are from having a margin γ_i. Therefore, for $1 \le i \le L$, we call the vector $\xi_i = (\xi_{i,1}, \ldots, \xi_{i,n_i})$ the margin slack vector of f_i with respect to π and γ_i over X_n.

To find a bound for the generalization error of $sign(\mathbf{f}^\pi)$ in the case of such a soft margin, we follow the approach of Shawe-Taylor and Cristianini (1999, 2002), which works for the case of $L = 1$. The idea is to map points of $\mathbf{H}$ to a higher dimensional space $\hat{\mathbf{H}}$ so that, for each i, the image of $X_n^{(i)}$ can be separated by some function $\hat{f}_i$ with the desired margin. Theorem 7 can then be applied. To find each $\hat{f}_i$, we present the following lemma, which is a simple extension of Shawe-Taylor & Cristianini's approach. For completeness, we provide the proof in Appendix B.

Lemma 9 *Suppose that $\|\Phi(\mathbf{x})\| \le \rho$ for any $(\mathbf{x}, y) \in supp(\mathcal{D})$. Then, there exists a space $\hat{\mathbf{H}}$, a mapping $\tau_\rho : \mathbf{H} \to \hat{\mathbf{H}}$, and a sequence $\hat{\mathbf{f}} = (\hat{f}_1, \ldots, \hat{f}_L)$ of L functions such that the following four facts hold.*

1. *For any $(\mathbf{x}, y) \in supp(\mathcal{D})$, $\|\tau_\rho(\Phi(x))\| \le \sqrt{2}\rho$.*
2. *For $1 \le i \le L$, $\hat{f}_i \in \mathcal{L}(\hat{\mathbf{H}}, \hat{\beta}_i)$ with $\hat{\beta}_i \le \sqrt{\beta_i^2 + \|\xi_i\|^2/\rho^2}$.*
3. *For $1 \le i \le L$, any $(\mathbf{x}_{i,j}, y_{i,j}) \in X_n^{(i)}$ will be classified correctly with a margin*

$$y_{i,j} \cdot \hat{f}_i(\tau_\rho(\Phi(\mathbf{x}_{i,j}))) \ge \gamma_i.$$

4. For $1 \le i \le L$ and for any $(\mathbf{x}, y) \notin X_n$, $\hat{f}_i(\tau_\rho(\Phi(\mathbf{x}))) = f_i(\Phi(\mathbf{x}))$.

According to this lemma, we define

$$\Psi = \tau_\rho \circ \Phi, \text{ and}$$

$$\hat{\mathbf{f}}^\pi(\mathbf{x}) = \hat{f}_{\pi(\mathbf{x})}(\Psi(\mathbf{x})).$$

Then, we know that $\hat{\mathbf{f}}^\pi$ has a margin $(\gamma_1, \ldots, \gamma_L)$ on X_n. Moreover, for $(\mathbf{x}, y) \in \mathit{supp}(\mathcal{D})$, we have

$$\|\Psi(\mathbf{x})\| \le \|\tau_\rho(\Phi(\mathbf{x}))\| \le \sqrt{2}\rho.$$

Now we can apply Theorem 7, with the space $\mathbf{H}$ replaced by $\hat{\mathbf{H}}$ and the mapping Φ replaced by Ψ, to obtain a bound on the generalization error of $\mathit{sign}(\hat{\mathbf{f}}^\pi)$, but with the quantity $\rho^2\beta^2/\gamma^2$ replaced by

$$\frac{2\rho^2(\beta_i^2 + \|\xi_i\|^2/\rho^2)}{\gamma_i^2} = \frac{2(\rho^2\beta_i^2 + \|\xi_i\|^2)}{\gamma_i^2}.$$

Finally, to bound the generalization error of $\mathit{sign}(\mathbf{f}^\pi)$, by Lemma 9, we know that for any $(\mathbf{x}, y) \notin X_n$, $\mathit{sign}(\mathbf{f}^\pi(\mathbf{x})) = \mathit{sign}(\hat{\mathbf{f}}^\pi(\mathbf{x}))$; therefore, $\mathit{sign}(\mathbf{f}^\pi)$ and $\mathit{sign}(\hat{\mathbf{f}}^\pi)$ have the same generalization error for inputs that do not fall within X_n. However, it is possible that the elements of X_n misclassified by $\mathit{sign}(\mathbf{f}^\pi)$ take a nontrivial measure in D, which results in $\mathit{sign}(\mathbf{f}^\pi)$ having a larger generalization error over $\mathcal{D}$ than $\mathit{sign}(\hat{\mathbf{f}}^\pi)$. As suggested by Shawe-Taylor and Cristianini (2002), this can be handled by modifying $\mathit{sign}(\mathbf{f}^\pi)$ on the misclassified elements in X_n. We call this new function the X_n-filtered version of $\mathit{sign}(\mathbf{f}^\pi)$. Then, we have the following theorem.

Theorem 10 *Let $\rho \in \mathbf{R}^+$, $\beta_i \in \mathbf{R}^+$ for $1 \le i \le L$, $\gamma = (\gamma_1, \ldots, \gamma_L) \in (\mathbf{R}^+)^L$, and $\mathcal{F} = \mathcal{L}(\mathbf{H}, \beta_1) \times \cdots \times \mathcal{L}(\mathbf{H}, \beta_L)$. In addition, let $\mathcal{D}$ be a probability distribution on $\mathbf{R}^d \times \{-1, 1\}$ such that $\|\Phi(\mathbf{x})\| \le \rho$ for every $(\mathbf{x}, y) \in \mathit{supp}(\mathcal{D})$, and let n be a large enough integer. Suppose a set of n samples, X_n, are drawn independently and at random according to $\mathcal{D}$; and consider any $\pi \in \mathcal{B}(\mathbf{R}^d)$ and $\mathbf{f} = (f_1, \ldots, f_L) \in \mathcal{F}$, such that for $i \le i \le L$, f_i has a margin slack vector ξ_i with respect to π and γ_i over X_n. Then, with probability $1 - \delta$, the generalization error of the X_n-filtered version of $\mathit{sign}(\mathbf{f}^\pi)$ will be at most*

$$\frac{c}{n}\left(\sum_{i=1}^{L}\left(\frac{\rho^2\beta_i^2 + \|\xi_i\|^2}{\gamma_i^2}\right)\log^2 n + L\log(dnL^2) + \log(1/\delta)\right),$$

for some constant c.

Theorem 10 shows that if we can build a classifier with a small L and a small $\|\xi_i\|$ for every i, then the first two terms inside the parentheses of the bound will be small, and the classifier is likely to have a small generalization error. As with Theorem 7, this does not suggest a way to choose the number L. Instead, the learning algorithm in Section 2 finds a good L in a data-driven manner. To facilitate a better understanding of the bound, we provide numerical values for the two terms derived by DTSVM classifiers. We also compare the numerical values with related quantities derived by global SVM (gSVM) classifiers, that is, SVM classifiers built on the full training data set.

Data Set	1A1		1AO	
	T1	T2	T1	T2
PHW	1,699	55	5,769	55
Shuttle	1,152,495	110	2,756,183	110
CI	202,354	481	202,354	481
Poker	12,253	139	48,893	139
KDD-10%	1,297,678	287	64,238	287

Table 13: The values of two leading terms T_1 and T_2 that appear in the generalization error bound for DTSVM classifiers. Training types = 1A1 and 1AO.

4.3 A Numerical Investigation

Note that in Theorem 10, β_i and γ_i are interdependent quantities for $1 \le i \le L$, so we can fix one of them and try to optimize the other. In the formulation of the SVM optimization problem, the objective is to minimize β_i under the constraint that $\gamma_i = 1$ for $1 \le i \le L$. Under this convention, the bound in Theorem 10 becomes

$$\frac{c}{n}\left(\sum_{i=1}^{L}\left(\rho^2\beta_i^2 + \|\xi_i\|^2\right)\log^2 n + L\log(dnL^2) + \log(1/\delta)\right). \tag{1}$$

Recall that a DTSVM classifier is associated with a tree with L leaves and each leaf is associated with an lSVM. If the tree has only one leaf (i.e., $L = 1$), then the DTSVM classifier will be reduced to a gSVM classifier, which has the following generalization error bound

$$\frac{c}{n}\left(\left(\rho^2\beta^2 + \|\xi\|^2\right)\log^2 n + \log(1/\delta)\right) \tag{2}$$

(cf. Cristianini and Shawe-Taylor, 2000).

Let us compare the terms that appear in parentheses in (1) and (2). The second term $L\log(dnL^2)$ in (1) is the shatter coefficient of the partition function π associated with a binary tree. We claim that the value of (1) achieved by DTSVM is dominated by the first term, which is the sum of L quantities associated with L leaves of a binary tree, as opposed to the single quantity in (2). Moreover, the first term in (1) achieved by DTSVM is comparable to the corresponding quantity in (2) achieved by gSVM. The following results confirm the above two claims.

To validate the first claim, we compare the two leading terms in parentheses in (1). The first term is $T_1 = \sum_{i=1}^{L}(\rho^2\beta_i^2 + \|\xi_i\|^2)\log^2 n$ and the second term is $T_2 = L\log(dnL^2)$. The results, shown in Table 13, confirm the claim that T_1 far exceeds T_2 and (1) is dominated by T_1. Note that we compute T_1 under the following assumptions. (i) When the data set contains more than two labels, T_1 is taken as the average of the quantities over all classifiers. (ii) The value of ρ is always 1 when RBF kernels are involved. (iii) The value of $\|\xi_i\|^2$ is obtained from the solution to the quadratic programming optimization problem. Further details can be found in Cristianini and Shawe-Taylor (2000), Section 6.1.2.

To validate the second claim, we compare $R = \sum_{i=1}^{L}(\rho^2\beta_i^2 + \|\xi_i\|^2)$ and $S = \rho^2\beta^2 + \|\xi\|^2$, which are derived, respectively, from the first terms in the generalization error bounds for DTSVM and

Data Set	1A1			1AO		
	S	R	R/L	S	R	R/L
PHW	74	133	17	326	450	56
Shuttle	114,786	75,630	5,402	593,967	180,990	12,928
CI	16,422	13,599	257	16,422	13,599	257
Poker	370	874	49	2,328	3,486	194
KDD-10%	100,227	70,796	2,212	5,089	3,505	110

Table 14: The values of S and R, which appear in the generalization error bound for gSVM and DTSVM classifiers respectively, and the values of R/L. Training types = 1A1 and 1AO.

gSVM classifiers (i.e., in (1) and (2)) with the common factor $\log^2 n$ removed from them. To ensure a meaningful comparison between R and S, both classifiers have to take the same (C,γ) values, which we specify as the optimal values for gSVM. As a result, we had to train new DTSVM classifiers for some data sets, using the same decomposition schemes (i.e., the same binary trees and same ceiling sizes) as the old classifiers, but different (C,γ) values.

Table 14 shows the values of S, R and R/L, derived from five data sets. The "Letter" and "News20" data sets are not included in the table because the DTSVM classifier using the designated values of (C,γ) would be the same as the gSVM on those data sets. It is clear that the values of R are as small as (less than 150), or of the same order of magnitude as, those of the corresponding S. In fact, S can be viewed as the slack-to-margin ratio and R as the sum of such ratios. The results show that each lSVM generates smaller slack-to-margin ratios than the corresponding gSVM, while the sum of lSVM ratios is comparable to the corresponding gSVM ratio. This explains why the test accuracy rates of DTSVM classifiers are comparable to those of gSVM classifiers.

5. Conclusion

We have proposed a method that uses a binary tree to decompose a given data space and trains an lSVM on each of the decomposed regions. The resultant DTSVM classifier can be constructed in a much shorter time than the gSVM classifier, and still achieve comparable accuracy rates to the latter. We also provide a generalization error bound for the DTSVM classifier. Using some data sets to compute the theoretical bounds for gSVM and DTSVM classifiers, we find that DTSVM classifiers generate comparable error bounds to those generated by gSVM classifiers. This finding explains why DTSVM classifiers can achieve more or less the same accuracy rates as gSVM classifiers.

Appendix A. Proof of Lemma 4

Let $\mathcal{G} \subseteq \mathcal{P}_L(\mathbf{R}^d)$, $\gamma = (\gamma_1,\ldots,\gamma_L) \in (\mathbf{R}^+)^L$, and $\mathcal{F}_i \subseteq \mathcal{R}(\mathbf{H})$ for $i = 1,\ldots,L$. The samples in X_n are drawn independently and at random according to the distribution $\mathcal{D}$.

Our goal is to find an upper bound for the probability of the following event.

- A_1: there exist $\pi \in \mathcal{G}$ and $\mathbf{f} = (f_1, \ldots, f_L) \in \mathcal{F}_1 \times \cdots \times \mathcal{F}_L$ such that $mg(\mathbf{f}^\pi, X_n) \geq \gamma$ and $err(\mathbf{f}^\pi, \mathcal{D}) > \varepsilon$, where $err(\mathbf{f}^\pi, \mathcal{D})$ is the probability that $sign(\mathbf{f}^\pi(\mathbf{x})) \neq y$ with $(\mathbf{x}, y)$ being sampled according to $\mathcal{D}$.

Note that $err(\mathbf{f}^\pi, \mathcal{D})$ is the generalization error of $sign(\mathbf{f}^\pi(\mathbf{x}))$. We relate event A_1 to another event A_2 in which an additional set of n independent samples, $\hat{X}_n$, are drawn at random according to $\mathcal{D}$ and the empirical error of $sign(\mathbf{f}^\pi(\mathbf{x}))$ over $\hat{X}_n$ is considered.

- A_2: there exist $\pi \in \mathcal{G}$ and $\mathbf{f} = (f_1, \ldots, f_L) \in \mathcal{F}_1 \times \cdots \times \mathcal{F}_L$ such that $mg(\mathbf{f}^\pi, X_n) \geq \gamma$ and $err(\mathbf{f}^\pi, \hat{X}_n) > \varepsilon/2$, where $err(\mathbf{f}^\pi, \hat{X}_n) = |\{(x, y) \in \hat{X}_n : sign(\mathbf{f}^\pi(\mathbf{x})) \neq y\}|/n$.

Following a standard argument (Vapnik, 1995), one can relate the probability of A_1 with that of A_2. More precisely, we have

$$\Pr_{X_n, \hat{X}_n}[A_2] \geq \Pr_{X_n, \hat{X}_n}[A_2 \wedge A_1] = \Pr_{X_n}[A_1] \cdot \Pr_{X_n, \hat{X}_n}[A_2 | A_1],$$

and by Chebyshev's inequality, one can show that

$$\Pr_{X_n, \hat{X}_n}[A_2 | A_1] = 1 - \Pr_{X_n, \hat{X}_n}[\neg A_2 | A_1] \geq 1 - 1/(n\varepsilon^2) \geq 1/2,$$

for a large enough n. Consequently, we have

$$\Pr_{X_n}[A_1] \leq \left(1 / \Pr_{X_n, \hat{X}_n}[A_2 | A_1]\right) \cdot \Pr_{X_n, \hat{X}_n}[A_2] \leq 2 \cdot \Pr_{X_n, \hat{X}_n}[A_2].$$

To find a bound for $\Pr_{X_n, \hat{X}_n}[A_2]$, let us consider the following event, A_3, where a set of $2n$ samples, X_{2n}, are drawn independently and at random according to $\mathcal{D}$, and X_{2n} is further divided randomly into two disjoint parts of equal size: W_1 and W_2.

- A_3: there exist $\pi \in \mathcal{G}$ and $\mathbf{f} = (f_1, \ldots, f_L) \in \mathcal{F}_1 \times \cdots \times \mathcal{F}_L$ such that $mg(\mathbf{f}^\pi, W_1) \geq \gamma$ and $err(\mathbf{f}^\pi, W_2) > \varepsilon/2$.

We observe that the distribution of $(X_n, \hat{X}_n)$ is identical to that of (W_1, W_2) over random X_{2n}; consequently we have

$$\Pr_{X_{2n}, W_1, W_2}[A_3] = \Pr_{X_n, \hat{X}_n}[A_2].$$

The next step is to find a bound for $\Pr_{X_{2n}, W_1, W_2}[A_3]$.

Let $\mathcal{G}(X_{2n})$ be the family of functions of $\mathcal{G}$ restricted to the domain $\{\mathbf{x} : (\mathbf{x}, y) \in X_{2n}\}$; and for $\pi \in \mathcal{G}(X_{2n})$, let

$$B^\pi_{\gamma/2}(X_{2n}) = \mathcal{C}\left(\mathcal{F}_1, \Phi(X^{(1)}_{2n}), \gamma_1/2\right) \times \cdots \times \mathcal{C}\left(\mathcal{F}_L, \Phi(X^{(L)}_{2n}), \gamma_L/2\right),$$

where $\Phi(X^{(i)}_{2n}) = \{\Phi(\mathbf{x}) : (\mathbf{x}, y) \in X_{2n}, \pi(\mathbf{x}) = i\}$. For $\mathbf{g} = (g_1, \ldots, g_L) \in B^\pi_{\gamma/2}(X_{2n})$, let $g^\pi(\mathbf{x}) = g_i(\mathbf{x})$ for $\mathbf{x} \in \Phi(X^{(i)}_{2n})$ for $1 \leq i \leq L$. Then, for $\pi \in \mathcal{G}(X_{2n})$ and $\mathbf{f} = (f_1, \ldots, f_n) \in \mathcal{F}_1 \times \cdots \times \mathcal{F}_L$, there exists $\mathbf{g} = (g_1, \ldots, g_L) \in B^\pi_{\gamma/2}(X_{2n})$ such that for any $(\mathbf{x}, y) \in X_n$, if $\pi(\mathbf{x}) = i$, then

$$|f_i(\Phi(\mathbf{x})) - g_i(\Phi(\mathbf{x}))| \leq \gamma_i/2.$$

For such $\mathbf{f}^\pi$ and $\mathbf{g}^\pi$, $mg(\mathbf{f}^\pi, W_1) \geq \gamma$ implies that $mg(\mathbf{g}^\pi, W_1) \geq \gamma/2$; and $err(\mathbf{f}^\pi, W_2) \geq \varepsilon/2$ implies that $err_{\gamma/2}(\mathbf{g}^\pi, W_2) \geq \varepsilon/2$, where $err_{\gamma/2}(\mathbf{g}^\pi, W_2)$ is the proportion of $(\mathbf{x}, y)$ in W_2 for which $g_i(\mathbf{x}) < \gamma_i/2$ if $\pi(\mathbf{x}) = i$. Therefore, the probability of the event A_3 cannot exceed that of the following event A_4.

- A_4: there exist $\pi \in \mathcal{G}(X_{2n})$ and $\mathbf{g} \in B^\pi_{\gamma/2}(X_{2n})$ such that $mg(\mathbf{g}^\pi, W_1) \geq \gamma/2$ and $err_{\gamma/2}(\mathbf{g}^\pi, W_2) \geq \varepsilon/2$.

To bound the probability of A_4, let us first consider any fixed X_{2n}, π, and $\mathbf{g}$, and consider an event, denoted as $A_4(X_{2n}, \pi, \mathbf{g})$, over the random division of X_{2n} into W_1 and W_2, such that $mg(\mathbf{g}^\pi, W_1) \geq \gamma/2$ and $err_{\gamma/2}(\mathbf{g}^\pi, W_2) \geq \varepsilon/2$. Note that for the event $A_4(X_{2n}, \pi, \mathbf{g})$ to occur, there are at most $2n - (\varepsilon/2)n$ elements of X_{2n} which can be separated by $\mathbf{g}^\pi$ with margin $\gamma/2$, and these elements must contain all the n elements of W_1. This implies that

$$\Pr_{W_1, W_2}[A_4(X_{2n}, \pi, \mathbf{g})] \leq \frac{\binom{2n-(\varepsilon/2)n}{n}}{\binom{2n}{n}} \leq \left(\frac{n}{2n}\right)^{(\varepsilon/2)n} = 2^{-\varepsilon n/2}.$$

Next, let us fix any X_{2n} and consider an event, denoted as $A_4(X_{2n})$, over the random division of X_{2n} into W_1 and W_2, such that the event $A_4(X_{2n}, \pi, \mathbf{g})$ occurs for some $\pi \in \mathcal{G}(X_{2n})$ and $\mathbf{g} \in B^\pi_{\gamma/2}(X_{2n})$. By a simple union bound, we have

$$\Pr_{W_1, W_2}[A_4(X_{2n})] \leq \sum_{\pi \in \mathcal{G}(X_{2n})} \sum_{g \in B^\pi_{\gamma/2}(X_{2n})} 2^{-\varepsilon n/2} = |\mathcal{G}(X_{2n})| \cdot |B^\pi_{\gamma/2}(X_{2n})| \cdot 2^{-\varepsilon n/2}.$$

Since $|X_{2n}| = 2n$, we have $|\mathcal{G}(X_{2n})| \leq V(\mathcal{G}, 2n)$. Moreover,

$$\begin{aligned}
\left|B^\pi_{\gamma/2}(X_{2n})\right| &= \prod_{1 \leq i \leq L} \left|C\left(\mathcal{F}_i, X^{(i)}_{2n}, \gamma_i/2\right)\right| \\
&\leq \prod_{1 \leq i \leq L} N\left(\mathcal{F}_i, E, |X^{(i)}_{2n}|, \gamma_i/2\right) \\
&\leq \prod_{1 \leq i \leq L} N(\mathcal{F}_i, E, 2n, \gamma_i/2),
\end{aligned}$$

where $E = \{\Phi(\mathbf{x}) : (\mathbf{x}, y) \in supp(\mathcal{D})\}$. Therefore, we have

$$\Pr_{W_1, W_2}[A_4(X_{2n})] \leq V(\mathcal{G}, 2n) \cdot \left(\prod_{1 \leq i \leq L} N(\mathcal{F}_i, E, 2n, \gamma_i/2)\right) \cdot 2^{-\varepsilon n/2}.$$

Note that by randomizing the selection of X_{2n}, the expected value of $\Pr_{W_1, W_2}[A_4(X_{2n})]$ is just the probability of A_4, over the random selection of X_{2n} and its random division into W_1 and W_2, from a simple probability fact.[1] As a result, we have

$$\begin{aligned}
\Pr_{X_{2n}, W_1, W_2}[A_4] &= \mathop{\mathrm{E}}_{X_{2n}}\left[\Pr_{W_1, W_2}[A_4(X_{2n})]\right] \\
&\leq V(\mathcal{G}, 2n) \cdot \left(\prod_{1 \leq i \leq L} N(\mathcal{F}_i, E, 2n, \gamma_i/2)\right) \cdot 2^{-\varepsilon n/2}.
\end{aligned}$$

1. Suppose A is an event over a joint distribution $(\mathbf{X}, \mathbf{W})$. Let $A(X)$ denote the event A conditioned on $\mathbf{X} = X$. Then, $\Pr_{(X,W) \in (\mathbf{X}, \mathbf{W})}[A] = \sum_Z \Pr_{X \in \mathbf{X}}[X = Z] \cdot \Pr_{W \in \mathbf{W}}[A(Z)] = \mathrm{E}_{X \in \mathbf{X}}[\Pr_{W \in \mathbf{W}}[A(X)]]$.

Finally, by combining all the bounds derived so far, we obtain

$$\Pr_{X_n}[A_1] \leq 2 \cdot \Pr_{X_{2n},W_1,W_2}[A_4] \leq 2 \cdot V(\mathcal{G},2n) \cdot \left(\prod_{1 \leq i \leq L} N(\mathcal{F}_i, E, 2n, \gamma_i/2) \right) \cdot 2^{-\varepsilon n/2},$$

which is at most d if

$$\varepsilon = \frac{2}{n} \left(\log V(\mathcal{G},2n) + \sum_{1 \leq i \leq L} \log N(\mathcal{F}_i, E, 2n, \gamma_i/2) + \log(2/\delta) \right).$$

This proves the lemma.

Appendix B. Proof of Lemma 9

We begin by describing the space $\hat{\mathbf{H}}$ and the mapping $\tau_\rho : \mathbf{H} \to \hat{\mathbf{H}}$. Consider the inner product space

$$I(\mathbf{H}) = \{ f \in \mathcal{R}(\mathbf{H}) : f(\mathbf{x}) \neq 0 \text{ for a finite number of } \mathbf{x} \in \mathbf{H} \},$$

where the inner product of f and g in $I(\mathbf{H})$ is defined as $\langle f, g \rangle = \sum_z f(z)g(z)$. Let

$$\hat{\mathbf{H}} = \mathbf{H} \times I(\mathbf{H}).$$

Define the function $\tau_\rho : \mathbf{H} \to \hat{\mathbf{H}}$ by

$$\tau_\rho(z) = (z, \rho \cdot \delta_z),$$

where δ_z is the function defined by

$$\delta_z(z') = \begin{cases} 1 & \text{if } z = z', \\ 0 & \text{otherwise.} \end{cases}$$

Then, the first item of the lemma holds, since for $(\mathbf{x}, y) \in supp(\mathcal{D})$, $\|\Phi(\mathbf{x})\| \leq \rho$; thus,

$$\|\tau_\rho(\Phi(x))\|^2 = \|\Phi(x)\|^2 + \|\rho \delta_{\Phi(x)}\|^2 \leq 2\rho^2.$$

Next, we prove the existence of $\hat{\mathbf{f}} = (\hat{f}_1, \ldots, \hat{f}_L)$. Since $I(\mathbf{H})$ is an inner product space, each of its elements defines a linear function on $I(\mathbf{H})$. Hence, for $(f,g) \in \mathcal{L}(\mathbf{H}) \times I(\mathbf{H})$, the function $(f,g) : \hat{\mathbf{H}} \to \mathbf{R}$, defined by

$$(f,g)(z,h) = f(z) + \langle g, h \rangle,$$

for $(z,h) \in \hat{\mathbf{H}}$, is a linear function. Now, for $1 \leq i \leq L$, we can define $\hat{f}_i : \hat{\mathbf{H}} \to \mathbf{R}$ by

$$\hat{f}_i = (f_i, g_i/\rho),$$

where $g_i \in I(\mathbf{H})$ is defined by

$$g_i = \sum_{j=1}^{n_i} \xi_{i,j} \cdot y_{i,j} \cdot \delta_{\Phi(\mathbf{x}_{i,j})}.$$

Since $f_i \in \mathcal{L}(\mathbf{H}, \beta_i)$, there exists $w_i \in \mathbf{H}$ with $\|w_i\| \leq \beta_i$ such that $f_i(z) = \langle w_i, z \rangle$. It follows that for $\hat{z} \in \hat{\mathbf{H}}$,

$$\hat{f}_i(\hat{z}) = \langle \hat{w}_i, \hat{z} \rangle,$$

where $\hat{w}_i = (w_i, g_i/\rho)$, and

$$\|\hat{w}_i\|^2 = \|w_i\|^2 + \sum_{j=1}^{n_i} |\xi_{i,j}|^2/\rho^2 \leq \beta_i^2 + \|\xi_i\|^2/\rho^2.$$

Therefore, $\hat{f}_i \in \mathcal{L}(\hat{\mathbf{H}}, \hat{\beta}_i)$, where $\hat{\beta}_i = \sqrt{\beta_i^2 + \|\xi_i\|^2/\rho^2}$, and the second item of the lemma holds.

Furthermore, for any $(\mathbf{x}_{i,j}, y_{i,j}) \in X_n^{(i)}$, we have

$$\begin{aligned} y_{i,j} \cdot \hat{f}_i(\tau_\rho(\Phi(\mathbf{x}_{i,j}))) &= y_{i,j} \cdot f_i(\Phi(\mathbf{x}_{i,j})) + y_{i,j} \cdot (\xi_{i,j} \cdot y_{i,j}) \\ &= y_{i,j} \cdot f_i(\Phi(\mathbf{x}_{i,j})) + \xi_{i,j} \\ &\geq \gamma_i, \end{aligned}$$

by the definition of $\xi_{i,j}$. Thus, the third item of the lemma also holds.

Finally, for $1 \leq i \leq L$ and for any $(\mathbf{x}, y) \notin X_n$, we have

$$\hat{f}_i(\tau_\rho(\Phi(\mathbf{x}))) = f_i(\Phi(\mathbf{x})) + \sum_{j=1}^{n_i} \xi_{i,j} \cdot y_{i,j} \cdot \delta_{\Phi(\mathbf{x}_{i,j})}(\Phi(\mathbf{x})) = f_i(\Phi(\mathbf{x})).$$

Thus, the fourth item of the lemma holds as well. This concludes the proof of Lemma 9.

References

N. Alon, S. Ben-David, N. Cesa-Bianchi, and D. Haussler. Scale-sensitive dimensions, uniform convergence, and learnability. *Journal of ACM*, 44(4):615–631, 1997.

P. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

K. P. Bennett and J. A. Blue. A support vector machine approach to decision trees. In *Proceedings IEEE International Joint Conference on Neural Networks*, 1998.

K. P. Bennett, N. Cristianini, J. Shawe-Taylor, and D. Wu. Enlarging the margins in perceptron decision trees. *Machine Learning*, 41:295–313, 2000.

A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, 2005.

A. Bordes, L. Bottou, P. Gallinari, and J. Weston. Solving multiclass support vector machines with LaRank. In *Proceedings of the 24th International Machine Learning Conference*, pages 89–96, 2007.

A. Bordes, L. Bottou, and P. Gallinari. SGD-QN: careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10:1737–1754, 2009.

L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, L. Jackel, Y. LeCun, U. Müller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of classifier methods: A case study in handwriting digit recognition. In *Proc. Int. Conf. Pattern Recognition*, pages 77–87, 1994.

L. Breiman. Bagging predictors. *Machine Learning*, 262:123–140, 1996.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman and Hall, 1984.

C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Journal of Data Mining and Knowledge Discovery*, 2(2):1–47, 1998.

R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computation*, 14(5):1105–1114, 2002.

C. Cortes and V. Vapnik. Support vector machines. *Machine Learning*, 20:1–25, 1995.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

G. R. Dattatreya and L. N. Kanal. Decision trees in pattern recognition. In L. N. Kanal and A. Rossenfeld, editors, *Progress in Pattern Recognition*, volume 2. Elsevier, 1985.

T. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40:139–157, 2000.

R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

V. Franc and S. Sonnenburg. Optimized cutting plane algorithm for support vector machines. In *Proceedings of the 25th International Machine Learning Conference*, pages 320–327, 2008.

T. Glasmachers and C. Igle. Maximum-gain working set selection for SVMs. *Journal of Machine Learning Research*, 7:1437–1466, 2006.

H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: the cascade SVM. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Neural Information Processing Systems*. MIT Press, 2004.

T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Kernel Methods – Support Vector Learning*. MIT Press, 1998.

T. Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226, 2006.

J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. *IEEE Transactions on Signal Processing*, 52(8), 2004.

S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited: A stepwise procedure for building and training a neural network. In J. Fogelman, editor, *Neurocomputing: Algorithms, Architectures and Applications*. Springer-Verlag, 1990.

N. D. Lawrence, M. Seeger, and R. Herbrich. Fast sparse gaussian process methods: the informative vector machine. In S. Becker, S. Thrun, and K. Obermayer, editors, *Neural Information Processing Systems*. MIT Press, 2003.

A. K. Menon. Large-scale support vector machines: algorithms and theory. Research Exam, University of California, San Diego, 2009.

S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Machine Learning Research*, 2:1–32, 1994.

D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

E. Osuna, R. Freud, and F. Girosi. An improved training algorithm for support vector machines. In *Neural Networks for Signal Processing*, volume VII, pages 276–285, 1997.

N. Panda, E. Y. Chang, and G. Wu. Concept boundary detection for speeding up SVMs. In *Proceedings of International Conference on Machine learning*, pages 681–688, 2006.

D. Pavlov, D. Chudova, and P. Smyth. Towards scalable support vector machines using squashing. In *ACM SIGKDD*, pages 295–299, 2000.

J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. J. Smola, editors, *Kernel Methods: Support Vector Learning*. MIT Press, 1998.

J. C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin dags for multiclass classification. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems*. MIT Press, 2000.

J. R. Quinlan. Induction of decision tree. *Machine Learning*, 1:81–106, 1986.

S. Ramaswamy. Multiclass text classification a decision tree based SVM approach. CS294 Practical Machine Learning Project, 2006.

A. Rida, A. Labbi, and C. Pellegrini. Local experts combination through density decomposition. In *Proceedings of International Workshop on AI and Statistics*, 1999.

H. Sahbi and D. Geman. A hierarchy of support vector machines for pattern detection. *Journal of Machine Learning Research*, 7:2087–2123, 2006.

R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

R. E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. *Journal of Machine Learning Research*, 39(2/3):135–168, 2000.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *Proceedings of International Conference on Machine learning*, pages 807–814, 2007.

J. Shawe-Taylor and N. Cristianini. Margin distribution bounds on generalization. In *Proceedings of the European Conference on Computational Learning Theory*, pages 263–273, 1999.

J. Shawe-Taylor and N. Cristianini. On the generalization of soft margin algorithms. *IEEE Transactions on Information Theory*, 48(10):2721–2735, 2002.

A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proceedings of the International Conference on Machine Learning*, pages 911–918, 2000.

A. J. Smola, S. V. N. Vishwanathan, and Q. V. Le. Bundle methods for machine learning. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. MIT Press, 2007.

J.-Q. Sun, G.-G. Wang, Q. Hu, and S.-Y. Li. A novel SVM detection tree and its application to face detection. In *Proceedings of the 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, pages 385–389, 2007.

R. Tibshirani and T. Hastie. Margin trees for high-dimensional classification. *Journal of Machine Learning Research*, 8:637–652, 2007.

V. Tresp. A bayesian committee machines. *Neural Computation*, 12:2719–2741, 2000.

V. Tresp. Scaling kernel-based systems to large data sets. *Data Mining and Knowledge Discovery*, 5:197–211, 2001.

I. W. Tsang, J. T. Kwok, and P.-M. Cheung. Core vector machines: fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

T.-S. Tseng, C.-Y. Guo, Wen-Lian Hsu, and F. Chang. Protein-protein interface prediction based on a novel SVM speedup. Technical Report, Number TR-IIS-10-001, Institute of Information Science, Academia Sinica, 2010.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, 1995.

D. Wu, K. P. Bennett, N. Cristianini, and J. Shawe-Taylor. Large margin decision trees for induction and transduction. In *Proceedings of International Conference on Machine Learning*, pages 474–483, 1999.

H. Yu, J. Han J. Yang, and X. Li. Making SVMs scalable to large data sets using hierarchical cluster indexing. *Data Mining and Knowledge Discovery*, 11:295–321, 2003.