An Efficient Classifier Based on Hierarchical Mixing Linear Support Vector Machines

Di Wang, Xiaoqin Zhang*, Mingyu Fan, Xiuzi Ye
College of Mathematics & Information Science, Wenzhou University
Zhejiang, China
wangdi@amss.ac.cn, zhangxiaoqinnan@gmail.com, {fanmingyu, yexiuzi}@wzu.edu.cn

Abstract

Support vector machines (SVMs) play a very dominant role in data classification due to their good generalization performance. However, they suffer from the high computational complexity in the classification phase when there are a considerable number of support vectors (SVs). Then it is desirable to design efficient algorithms in the classification phase to deal with the datasets of real-time pattern recognition systems. To this end, we propose a novel classifier called HMLSVMs (Hierarchical Mixing Linear Support Vector Machines) in this paper, which has a hierarchical structure with a mixing linear SVMs classifier at each node and predicts the label of a sample using only a few hyperplanes. We also give a generalization error bound for the class of locally linear SVMs (LLSVMs) based on the Rademacher theory, which ensures that overfitting can be effectively avoided. Experimental evaluations shows, while maintaining a comparable classification performance to kernel SVMs (KSVMs), the proposed classifier achieves the high efficiency in the classification stage.

1 Introduction

Support vector machines (SVMs) have enjoyed excellent performance in many areas due to their great flexibility and good generalization performance. However, they still suffer from a high computational complexity in the classification phase if the number of SVs is large. We want the time for classification to be as short as possible, especially in real-time pattern recognition systems, whereas classification must be performed in a few millionseconds to satisfy the needs of online tasks. The methods for reducing the computational complexity of SVM classification can be divided into following two groups.

One group consists of algorithms to prune the SVs of SVMs. Representative papers include [Schoelkopf *et al.*, 1999; Downs *et al.*, 2001; Nguyen and Ho, 2006; Liang, 2010]. However, these methods need to obtain a standard SVM in advance, and this limits their applications to large-scale problems. To address this issue, several methods for selecting a set of basis vectors are proposed. They include sampling from the training set in the Nystrom method [Williams and Seeger, 2001] and variants of the Incomplete Cholesky factorization [Bach and Jordan, 2005], core vector machine (CVM) [Tsang *et al.*, 2005], relevance vector machine (RVM)[Tipping, 2001], and relevance units machine (RUM)[Gao and Zhang, 2009]. Wu *et al.* [Wu *et al.*, 2006] add one constraint on the number of basis vectors to the standard SVM optimization problem, and then solve this modified nonconvex problem to build sparse kernel learning algorithms (SKLA). Joachims and Yu [Joachims and Yu, 2009] explore a new sparse kernel SVMs via cutting plane training, called cutting-plane subspace pursuit (CPSP).Although the above methods prunes the SVs and reduces computational complexity in classification phase, when a new test sample is introduced, they still need to compare it with these pruned SVs via kernel calculations to predict the label of the test sample. In addition, these methods also have to cache substantial kernel values of the important samples in the training process.

The second group of techniques, exploring the divide-and-conquer strategy, are to construct the ensemble models of linear decision functions in the original space, thus avoiding the kernel calculations of the SVs. Classical ensemble methods include bagging [Breiman, 1996], boosting [Pavlov *et al.*, 2000] and random forests [Breiman, 2001]. But these methods tend to use a great amount of linear classifiers to correctly predict the sample label for linearly inseparable datasets, and thus are possibly to increase the computational complexity in classification stage. Bennett and Blue [Bennett and Blue, 1998] propose a primal and dual formulation for SVM approaches to decision trees with three decisions, and claim the method can be applied to trees of any size which needs to be determined beforehand. Fei and Liu [Fei and Liu, 2006] present a binary tree of SVMs for multiclass problems. Fornoni *et al.*[Fornoni *et al.*, 2011] propose a multi class local classifier based on a latent SVM formulation. Ladicky and Torr [Ladicky and Torr, 2013] learn all the local models in a single optimization problem by using the inverse Euclidean distances as a form manifold learning. In the spirit of the mixture-of-expert framework, Fu *et al.*[Fu *et al.*, 2010] propose a mixing linear SVMs (MLSVM) model which par-

*Corresponding author

titions the feature space into several linearly inseparable subregions based on a probabilistic model of the joint data and label distributions, and the testing samples are classified by the weighted average over the mixture of classifiers. Chen *et al.* [Cheng *et al.*, 2010] divide the whole training set into several clusters via MagKmeans algorithm, and then train the LSVM classifier in each cluster. Li *et al.* [Li *et al.*, 2011] propose a piecewise linear classifier named "multiconlitron", which is based on the "convexly separable" concept. Qi *et al.* [Qi *et al.*, 2011] partition the whole training samples into several local regions by applying the locality-sensitive hashing (LSH) method, and construct the local classifier model in each region.

Motivated by the low classification and training cost of linear SVMs, we continue the fruitful line of the second group and aim to provide a model which generates nonlinear classification boundaries via the ensemble of locally linear SVM (LLSVM). To this end, we propose a novel classifier called HMLSVMs (Hierarchical Mixing Linear SVMs). Specifically, it has a hierarchical tree-like structure with a mixing linear SVM classifier at each node. The main advantages of the proposed classifier are as follows. Firstly, HMLSVMs does not require a large memory to store kernel values because HMLSVMs is composed of LSVMs, and there are very efficient algorithms for training LSVMs [Joachims, 2011; Hsieh *et al.*, 2008]. Secondly, the hierarchical structure makes HMLSVMs predict the labels of new arrived samples via a few of LSVMs, and it is much faster than nonlinear SVMs. Thirdly, we quantify the generalization error bound for the class of LLSVMs based on the Rademacher complexity, and the stop criterion based on minimizing this bound ensures that HMLSVMs can effectively avoid overfitting and have a good classification performance. Hence, HMLSVMs has both the advantage of high speed in the classification stage, comparable to LSVMs, and the advantage of good classification performance comparable to KSVMs.

The remainder of the paper is organized as follows. In Section 2, we present the details of training and testing process of HMLSVMs. In Section 3, we quantify the generalization error bound for HMLSVMs based on the Rademacher complexity. Experimental results and related discussions are given in Section 4. Some conclusions are given in Section 5.

2 Description of the HMLSVMs

Before describing the proposed method, we first introduce the symbols used in the remaining parts of the paper.

- I: a row vector which represents a node in HMLSVMs. $I = [0]$ represents the root node; $[I, 1]$ and $[I, -1]$ are respectively the left and right child nodes of I.
- $f^{(t)}$: The weighted LSVM (W-LSVM) classifer trained in the t-th iteration of the training process of HMLSVMs.
- f_I: the classifier associated to the node I.
- S_I: subset of training samples in the node I.
- F_t: HMLSVMs in the t-th iteration.
- $V^{(t)}$: the evaluations of all training samples in the t-th iteration, and its i-th element is $V_i^{(t)} = F_t(\mathbf{x}_i)$.

2.1 Training Stage of HMLSVMs

Calculation of the weights

There are two major steps in the proposed strategy for calculating the training weights.

Step1: Training center selection Let the evaluation of sample $\mathbf{x}_i$ be V_i in current iteration, then the "host" training center is defined as:

$$\mathbf{x}_{\text{hc}} = \arg\max_{\mathbf{x}_i \in \hat{S}} \sum_{\mathbf{x}_j \in N_\varepsilon(\mathbf{x}_i)} \exp\{-\frac{\|\mathbf{x}_j - \mathbf{x}_i\|^2}{\sigma^2}\} E_j \quad (1)$$

where $E_j = \min\{1, \max\{0, 1 - y_j V_j\}\}$, $\hat{S} = \{x_i | y_i V_i < 1\}$, and $N_\varepsilon(\mathbf{x}_i)$ is the set of $\mathbf{x}_i$'s ε- neighborhoods with the same label as $\mathbf{x}_i$. From the definition of $\mathbf{x}_{\text{hc}}$ in Eq. (1), we can concluded that if a sample is selected as the "host" training center, it must meet two conditions: (1) the sample density of its ε-neighborhoods is high, (2) the samples in its ε-neighborhoods have large "training errors". Hence, the samples in local set $N_\varepsilon(\mathbf{x}_{\text{hc}})$ are currently misclassified "most". To correctly classify these misclassified samples, we must retrain a LSVM with them. Accordingly, we define the opposite of the "host" training center, which is called as "guest" training center

$$\mathbf{x}_{\text{gc}} = \underset{\mathbf{x}_i \in \bar{N}_\varepsilon(\mathbf{x}_{\text{hc}})}{\operatorname{argmax}} \sum_{\mathbf{x}_j \in N_\varepsilon(\mathbf{x}_i)} \exp\left\{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2 + \|\mathbf{x}_j - \mathbf{x}_{\text{hc}}\|^2}{\sigma^2}\right\} \quad (2)$$

where $\bar{N}_\varepsilon(\mathbf{x}_{\mathbf{hc}})$ is the set of $\mathbf{x}_{\mathbf{hc}}$'s ε- neighborhoods with different labels from $\mathbf{x}_{\mathbf{hc}}$. Generally speaking, the "guest" training center is $\mathbf{x}_{\text{hc}}$'s "nearest" sample with different labels from $\mathbf{x}_{\text{hc}}$, and the structural information defined in (2) avoids selecting isolated noisy samples.

Step2: Sample weights calculation The training weights of samples in sets $N_\varepsilon(\mathbf{x}_{\text{hc}})$ and $N_\varepsilon(\mathbf{x}_{\text{gc}})$ are calculated according to the sample similarities to the "host" and "guest" training center:

$$v_{\mathbf{x}} = \exp\{-\frac{\|\mathbf{x} - \mathbf{x}_{\text{hc}}\|^2}{\sigma^2}\}, \text{ for } \mathbf{x} \in S_{\text{h}_\varepsilon} \quad (3)$$

$$v_{\mathbf{x}} = \exp\{-\frac{\|\mathbf{x} - \mathbf{x}_{\text{gc}}\|^2}{\sigma^2}\}, \text{ for } \mathbf{x} \in S_{\text{g}_\varepsilon} \quad (4)$$

We assign the largest weights to the samples which are currently misclassified "most". This makes it more likely that the next classifier will correctly classify these samples.

Training Algorithm

The training scheme of HMLSVMs is summarized in Algorithm 1. In Step 7, we find the leaf node I which the "host" training center belongs to. This implies that the samples in set S_I are misclassified "most" by current HMLSVMs. I_p is the parent node of I. I_l is the last element of I, and $I_l = 1$ (I_l=-1) indicates that the samples in S_I are all classified as "positive" samples ("negative samples"). However, there may be some samples are misclassified by the current HMLSVMs. To separate these misclassified samples from S_I, we should assign a classifier to the node I to further split S_I. We define the classifier of node I by $f_I = \min\{f^{(t)}, f_{I_p}\}$ if $I_l = 1$ (separate the negative samples from the "positive" samples),

Algorithm 1: Training of HMLSVMs

Input: The set of training samples $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$.

1: Train the standard LSVM classifier $f^{(1)}$ with sample set S, and assign $f^{(1)}$ to the root node, i.e., $f_{[0]}=f^{(1)}$. Initialize $t = 1$.
2: Let $F_1 = \{f_{[0]}\}$, and $\tilde{S}_1 = \{S_{[0,1]}, S_{[0,-1]}\}$, where $S_{[0,1]}=\{\mathbf{x}|f_{[0]}(\mathbf{x}) \geq 0, (\mathbf{x}, y) \in S\}$ and $S_{[0,-1]} = \{\mathbf{x}|f_{[0]}(\mathbf{x}) < 0, (\mathbf{x}, y) \in S\}$.
3: **loop**
4: Let t = t+1.
5: Find the "host" and "guest" training center via Eq. (1) and (2), and calculate the training weights of samples in the local sets $N_\varepsilon(\mathbf{x}_{\text{hc}})$ and $N_\varepsilon(\mathbf{x}_{\text{gc}})$ via Eq. (3) and (4).
6: Train the W-LSVM $f^{(t)}$ with the weighted samples in the sets $N_\varepsilon(\mathbf{x}_{\text{hc}})$ and $N_\varepsilon(\mathbf{x}_{\text{gc}})$.
7: Find the leaf node which the "host" training center $\mathbf{x}_{\text{hc}}$ belongs to, and denote by I. Let $I_l = I(|I|)$ and $I_p = I(1 : |I| - 1)$, where $|I|$ is the number of elements of I.
8: **if** $I_l == -1$ **then**
9: Let $f_I = \max\{f^{(t)}, f_{I_p}\}$.
10: **else**
11: Let $f_I = \min\{f^{(t)}, f_{I_p}\}$.
12: **end if**
13: Let $F_t = F_{t-1} \cup \{f_I\}$, $S_{[I,1]} = \{\mathbf{x}|f_I(\mathbf{x}) \geq 0, \mathbf{x} \in S_I\}$, $S_{[I,-1]} = \{\mathbf{x}|f_I(\mathbf{x}) < 0, \mathbf{x} \in S_I\}$, $\tilde{S}_t = (\tilde{S}_{t-1} \cup \{S_{[I,1]}, S_{[I,-1]}\}) \setminus \{S_I\}$.
14: Update the evaluations of training samples $V^{(t)}$ by using current HMLSVMs.
15: **end loop**

Output: F_t, and $\tilde{S}_t$.

and by $f_I = \max\{f^{(t)}, f_{I_p}\}$ if $I_l = -1$ (separate the positive samples from the "negative" samples), where $f^{(t)}$ is the W-LSVM classifier trained in the current iteration and f_{I_p} is the classifier associated to the parent node of I. This setting method ensures that the classifier associated to node I considers not only the current trained W-LSVM $f^{(t)}$, but also the classifier associated to its parent node. Therefore, the classifier at each node is a mixture of maximums and minimums of LSVMs associated to its ancestor nodes. In Step 13, f_I splits S_I into subsets $S_{[I,1]}$ and $S_{[I,-1]}$ to enhance the impurity of current leaf nodes.

The outputs of the training process are F_t and $\tilde{S}_t$. F_t is the set of classifiers associated to the nodes of HMLSVMs. $\tilde{S}_t$ includes all subsets in the leaf nodes, and it gives the classification results of training samples. For any $S_I \in \tilde{S}_t$, if $I_l = 1$, the samples in set S_I are classified as positive samples. Otherwise, they are classified as negative samples.

Remark 1. *The total number of classifiers at nodes of HMLSVMs is a key parameter which determines the generalization performance for HMLSVMs. We will give a proper stop criterion in Section 3.*

2.2 Classification Stage of HMLSVMs

In classification stage, we make the new arrived sample go through from the root node to a terminal node of H-MLSVMs. When the sample has arrived at a terminal node, if its H-MLSVMs classifier value is positive, then it is classified as positive sample, otherwise, it is classified as negative sample. Since only the terminal nodes have no classifier, then $f_{[I,1]} \in F_t$ (or $f_{[I,-1]} \in F_t$) implies that the testing sample does not arrive at a terminal node of H-MLSVMs. We should continue classifying the sample using the classifier $f_{[I,1]}$ (or $f_{[I,-1]}$). Otherwise, if $f_{[I,1]} \notin F_t$ ($f_{[I,-1]} \notin F_t$), this implies that $\mathbf{x}$ has arrived at a terminal node, then we output the result $f_I(\mathbf{x})$.

2.3 Computational Complexity

Before giving the computational complexity, we first introduce the symbols used below. n is the number of samples, d is the dimension of samples, and $\bar{p}$ and $\bar{q}$ are the average number of elements of $\hat{S}$ and $\varepsilon-$neighborhoods $N_\varepsilon(\cdot)$ in (1)respectively. We assume the total number of classifiers at nodes of HMLSVMs is L, and the average complexity of computing a value of an exponential function $\exp(\cdot)$ is a constant. We use the dual coordinate descent (DCD) method [Hsieh *et al.*, 2008] to train LSVMs because it is simple and reaches an $\epsilon-$accurate solution with the computational complexity $O(\log(1/\epsilon)n\bar{d})$ for n training samples, where $\bar{d}$ is the average number of nonzero elements per sample.

Complexity of training: In Step 1, computing the LSVM of the root node takes $O(\log(1/\epsilon)n\bar{d})$ operations. In Step 2, the complexity of the evaluations of training samples is $O(nd)$. In Step 5, selecting the ε-neighborhoods of the samples in set $\hat{S}$ takes $O(\bar{p}nd)$ operations, and computing the "host" and "guest" center take $O(\bar{p}\bar{q}d)$ and $O(\bar{q}^2d)$ operations respectively. Then the complexity of Step 5 is $O((\bar{p}n + \bar{q}^2)d)$ since $\bar{q} < n$. Training the classifier with the samples of the local regions in Step 6 takes $O(\log(1/\epsilon)\bar{q}\bar{d})$ operations. From Step 13-14, the complexity is at most $O(nd)$ since we only need to update the evaluations of the training samples in node I. Hence, the complexity of Algorithm 1 is $O(\log(1/\epsilon)n\bar{d} + L(\bar{p}n + \bar{q}^2)d)$. Generally speaking, $\bar{p}$ increases with n. Thus the complexity becomes expensive when n is very large. In our experiments, we use the probabilistic speedup method. The idea is to randomly sample a sufficiently large subset $\hat{S}'$ from $\hat{S}$, and select the sample training center $\mathbf{x}_{\text{hc}}$ in set $\hat{S}'$ instead of $\hat{S}$. This makes the algorithm much faster as $|\hat{S}'| \ll |\hat{S}|$.

Complexity of classification: Since we classify a sample by making it go through from the root node to a leaf node of H-MLSVMs, and all classifiers associated to the nodes are composed of LSVMs, then only a few hyperplanes need to be evaluated in the classification step, no kernel computation is required. If the tree depth of HMLSVMs is $\bar{L}$, then the complexity for classifying a sample is at most $O(\bar{L}d)$. This guarantees the high speed in the classification phase.

3 Generalization Error Bound for HMLSVMs

For classification problems, we are interested in bounding the difference between empirical and expected risk over some

function class. Because HMLSVMs can be consider as an ensemble method of LLSVMs, we will quantify the generalization error bound for the class of LLSVMs based on the Rademacher complexity in this section.

The empirical Rademacher complexity of a class of function $\mathfrak{F}$ defined on domain $\mathcal{Z}$ is

$$\hat{R}_n(\mathfrak{F}) = \mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{f\in\mathfrak{F}}\left(\frac{1}{n}\sum_{i=1}^{n}\sigma_i f(\mathbf{z}_i)\right)\middle| T\right] \tag{5}$$

where the elements of $T = \{\mathbf{z}_1, \cdots, \mathbf{z}_n\}$ are generated by a distribution D on $\mathcal{Z}$, and $\sigma_1, \cdots, \sigma_n$ are independent Rademacher variables such that $P(\sigma_i = 1) = P(\sigma_i = -1) = 0.5$ for $i = 1, \cdots, n$.

The following theorem states that the difference between the empirical and the true expected value of a function is upper bounded by the empirical Rademacher complexity of the function class.

Theorem 1. *(Rademacher theorem) [Bousquet* et al.*, 2004] Let $\mathfrak{F}$ be a class of functions with domain $\mathcal{Z}$, which map from $\mathcal{Z}$ to $[0, 1]$, and $\{\mathbf{z}_1, \cdots, \mathbf{z}_n\}$ be generated independently according to a probability distribution D. Then for any fixed real value $\delta \in (0, 1)$, the following equation is satisfactory with probability at least $1 - \delta$ for every $f \in \mathfrak{F}$*

$$\mathbb{E}_D[f(\mathbf{z})] \le \frac{1}{n}\sum_{i=1}^{n}[f(\mathbf{z}_i)] + 2\hat{R}_n(\mathfrak{F}) + 3\sqrt{\frac{\ln(2/\delta)}{2n}} \tag{6}$$

Assume a LLSVM f defined on the domain $\mathcal{Z}$ is composed of κ LSVMs, denoted by

$$f = \{\langle \mathbf{w}_t, \mathbf{x}\rangle + b_t, \mathbf{x} \in \mathcal{Z}_t\}_{t=1}^{\kappa}. \tag{7}$$

$\mathcal{Z}_t$ is the subregion to which the LSVM $f_t(\mathbf{x}) = \langle \mathbf{w}_t, \mathbf{x}\rangle + b_t$ should be applied, such that $\mathcal{Z} = \cup_{t=1}^{\kappa}\mathcal{Z}_t$ and $\mathcal{Z}_i \cap \mathcal{Z}_j = \emptyset$ for $i \ne j$. Let $\mathfrak{F}_{B,\kappa,\mathcal{Z}_t}$ be the function class of LLSVM defined by

$$\mathfrak{F}_{B,\kappa,\mathcal{Z}_t} = \left\{ f \middle| f = \{\langle \mathbf{w}_t, \mathbf{x}\rangle + b_t, \mathbf{x} \in \mathcal{Z}_t\}_{t=1}^{\kappa}, \sup_t \|\tilde{\mathbf{w}}_t\| \le B \right\} \tag{8}$$

where $\tilde{\mathbf{w}}_t = [\mathbf{w}_t^\top b_t]^\top$. If we use the soft loss function

$$\ell_s(y, f(\mathbf{x})) = \min(1, \max(0, 1 - yf(\mathbf{x})/r)) \tag{9}$$

then the following theorem gives an generalization error bound for LLSVMs.

Theorem 2. *Let $\mathfrak{F}_{B,\kappa,\mathcal{Z}_t}$ be a function class defined by (8), and suppose the training samples $\mathbf{x}_{t1}, \cdots, \mathbf{x}_{tn_t}$ are in the subregion $\mathcal{Z}_t$ for $t = 1, \cdots, \kappa$. Then for any fixed real value $\delta \in (0, 1)$, the expected risk of LLSVM $f \in \mathfrak{F}_{B,\kappa,\mathcal{Z}_t}$ with probability at least $1 - \delta$ can be bounded by*

$$L(f) \le L_{emp}(f) + \frac{2B}{nr}\sum_{t=1}^{\kappa}\left(\sum_{i=1}^{n_t}\tilde{\mathbf{x}}_{ti}^\top\tilde{\mathbf{x}}_{ti}\right)^{\frac{1}{2}} + 3\sqrt{\frac{\ln(2/\delta)}{2n}} \tag{10}$$

where $L(f)$ is the expected risk based on the $0/1$ loss, $L_{emp}(f)$ is the empirical risk based on the soft loss (9), and $\tilde{\mathbf{x}}_{ti} = [\mathbf{x}_{ti}^\top\ 1]^\top$ for $i = 1, \cdots, n_t$ and $t = 1, \cdots, \kappa$.

Proof. Define the function class $\mathfrak{L}_{B,\kappa,\mathcal{Z}_t}$ as

$$\{\ell_s \,|\ell_s = \min(1, \max(0, 1 - yf(\mathbf{x})/r)),\ f \in \mathfrak{F}_{B,\kappa,\mathcal{Z}_t}\}\ .$$

Since $|\ell_s(f_1(\mathbf{x}), y) - \ell_s(f_2(\mathbf{x}), y)| \le \frac{1}{r}|f_1(\mathbf{x}) - f_2(\mathbf{x})|$ for $\forall\ f_1, f_2 \in \mathfrak{F}_{B,\kappa,\mathcal{Z}_t}$, i.e., ℓ_s satisfies Lipschitz condition with constant $1/r$, then from Lemma 7 of [Meir and Zhang, 2003], we have

$$\hat{R}_n(\mathfrak{L}_{B,\kappa,\mathcal{Z}_t}) \le \hat{R}_n(\mathfrak{F}_{B,\kappa,\mathcal{Z}_t})/r \tag{11}$$

Let σ_{ti} be the Rademacher variable corresponding to the sample $\mathbf{x}_{ti}$, and let $\boldsymbol{\sigma}_t = [\sigma_{t1}, \ldots, \sigma_{tn_t}]$, then the empirical Rademacher complexity of $\mathfrak{F}_{B,\kappa,\mathcal{Z}_t}$ satisfies

$$\begin{aligned}
&\hat{R}_n(\mathfrak{F}_{B,\kappa,\mathcal{Z}_t}) \\
=\ & \mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{f\in\mathfrak{F}_{B,\kappa,\mathcal{Z}_t}}\frac{1}{n}\sum_{t=1}^{\kappa}\sum_{i=1}^{n_t}\sigma_{ti}(\mathbf{w}_t^\top\mathbf{x}_{ti} + b_t)\right] \\
\le\ & \frac{1}{n}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{f\in\mathfrak{F}_{B,\kappa,\mathcal{Z}_t}}\sum_{t=1}^{\kappa}\left|\sum_{i=1}^{n_t}\langle\tilde{\mathbf{w}}_t, \sigma_{ti}\tilde{\mathbf{x}}_{ti}\rangle\right|\right] \\
\le\ & \frac{B}{n}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sum_{t=1}^{\kappa}\left\|\sum_{i=1}^{n_t}\sigma_{ti}\tilde{\mathbf{x}}_{ti}\right\|\right] \\
\le\ & \frac{B}{n}\sum_{t=1}^{\kappa}\left[\mathbb{E}_{\boldsymbol{\sigma}_t}\left(\sum_{i,j=1}^{n_t}\sigma_{ti}\sigma_{tj}\tilde{\mathbf{x}}_{ti}^\top\tilde{\mathbf{x}}_{tj}\right)\right]^{\frac{1}{2}} \\
=\ & \frac{B}{n}\sum_{t=1}^{\kappa}\left(\sum_{i=1}^{n_t}\tilde{\mathbf{x}}_{ti}^\top\tilde{\mathbf{x}}_{ti}\right)^{\frac{1}{2}}
\end{aligned} \tag{12}$$

The disappearance of the mixed terms $\sigma_{ti}\sigma_{tj}\tilde{\mathbf{x}}_{ti}^\top\tilde{\mathbf{x}}_{tj}$ for $i \ne j$ in Eq. (12) follows from the fact that the four possible combinations of -1 and 1 have equal probability with two of the four having the opposite signs and hence cancelling out. On the other hand,

$$L(f) = \mathbb{E}_D[\ell_{0/1}(f(\mathbf{x}), y)] \le \mathbb{E}_D[\ell_s(f(\mathbf{x}), y)]$$

According to Eq. (11), (12) and (6), the proof is complete. □

Theorem 2 gives us an upper bound on the generalization error in terms of the empirical risk and the complexity of the function class $\mathfrak{F}_{B,\kappa,\mathcal{Z}_t}$, where we measure the complexity of $\mathfrak{F}_{B,\kappa,\mathcal{Z}_t}$ by the minimal "margin" of all LSVMs ($2/\|\tilde{\mathbf{w}}_t\|$ is the margin of the t-th LSVM in the extended feature space $\tilde{\mathcal{X}} = \{\tilde{\mathbf{x}}|\tilde{\mathbf{x}} = [\mathbf{x}^\top 1]^\top\}$) and the partition of the training space. Note that if a LLSVMs partitions the training sample set into a small number of parts (i.e.,κ is small), then the second term in the right hand of Eq. (10) (the complexity of $\mathfrak{F}_{B,\kappa,\mathcal{Z}_t}$) is likely small, however, the empirical risk are likely large especially for datasets which are far from being linearly separable. Hence, there is a need to trade off the empirical error with the class complexity. To this end, we use the minimization of

$$\begin{aligned}
&\sum_{t=1}^{\kappa}\sum_{i=1}^{n_t}\min(1, \max(0, 1 - y_{ti}f(\mathbf{x}_{ti})/r)) \\
&+\lambda\left(\sup_t\|\tilde{\mathbf{w}}_t\|\right)\cdot\sum_{t=1}^{\kappa}\left[\mathrm{tr}\left(Z_t^\top\tilde{X}^\top\tilde{X}Z_t\right)\right]^{\frac{1}{2}}
\end{aligned} \tag{13}$$

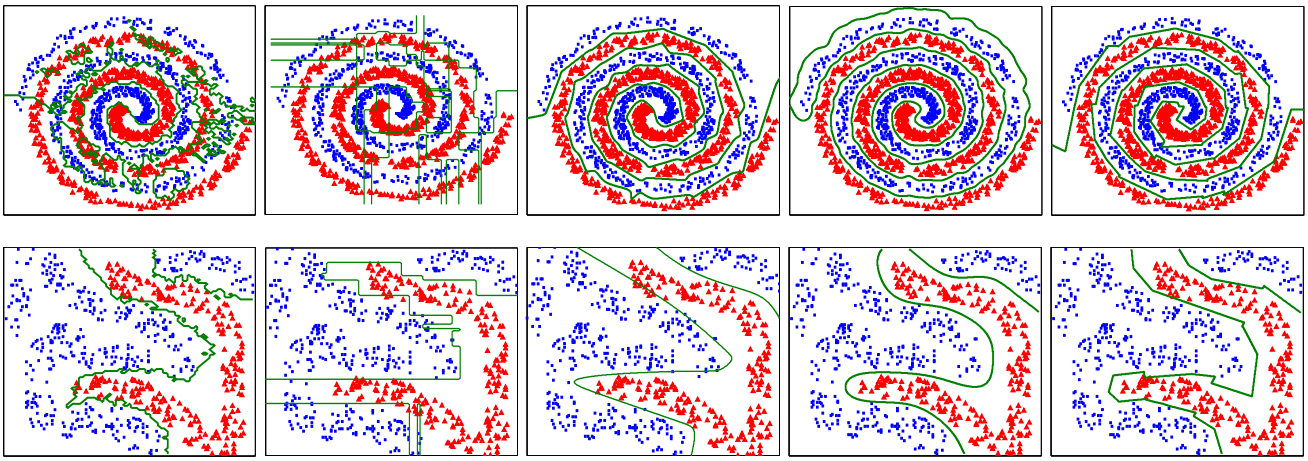

Figure 1: Decision boundaries of Ada-LR, CART, MLSVM, LIBSVM and HMLSVMs on the two synthetic datasets.

Table 1: Overall description of the datasets and parameters used in experiments.

Datasets	TR#	C_l	C_{LIB}	γ_{LIB}	C_{CVM}	γ_{CVM}	C_{NYS}	γ_{NYS}	γ_{MLSVM}	σ^2	λ
Spiral	2000	10	10	1	10	1	10	0.1	0.5	0.5	0.001
Fourclass	862	10	1	10	10	0.1	1000	1	1	0.02	0.01
Monks1	556	0.1	10	1	10	1	10	1	0.05	0.01	0.1
Pokerhand	6028	1	1	0.1	10	0.025	100	0.01	0.005	10	0.001
Letter	1536	10	10	1	10	2.5	10	10	0.1	0.6	0.001
EMG	80044	0.1	10	10	10	0.1	1000	10	0.1	0.3	0.001
Statlog	1410	10	10	0.1	10	0.5	100	1	10	1	0.001
Segment	660	10	10	1	10	5	10	10	0.1	0.1	0.001
Iris	150	10	10	0.05	0.1	1	0.01	1	0.1	4	0.001
Pendigit	2198	10	10	1	10	1	10	10	0.1	0.6	0.05
Chess	6179	10	10	0.1	10	0.1	100	0.1	1	4	0.001
SVMguide	7089	10	10	0.0001	10	0.0005	10	0.0001	0.001	1000	0.001
Building	1800	0.01	10	0.001	10	0.0003	1000	0.001	0.0001	1000	2
Face	8977	0.1	10	0.05	10	0.025	100	0.1	0.01	15	0.01

as the stop criterion, where λ is a tunable parameter and the second term in Eq. (13) can be seen as a regularization for the complexity of $\mathfrak{F}_{B,\kappa,\mathcal{Z}_t}$. In the algorithm of HMLSVMs, we train a series of LSVMs with the total number of classifiers at nodes of HMLSVMs in the increasing order, and the repetition is not stopped until the minimization of the value in (13) is achieved. The regularization term ensures that overfitting can be effectively avoided.

4 Experiments

In this section, HMLSVMs is compared with the ensemble methods of combining linear classifiers, (1) adaboost with decision stump as weak learners (Ada-Stump), (2) adaboost with linear regression as weak learners (Ada-Percep), (3) adaBoost with linear regression as weak learners (Ada-LR), (4) mixing linear SVMs (MLSVM) [Fu *et al.*, 2010], as well as the kernel methods (with RBF kernel $k(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma\|\mathbf{x}_i-\mathbf{x}_j\|^2}$), (5) LIBSVM [Fan *et al.*, 2005], (6) CVM [Tsang *et al.*, 2005], (7) Nystrom method [Williams and Seeger, 2001], and decision tree method (8) CART [Breiman *et al.*, 2004]. The parameters are obtained by using 5-fold cross validation. For Nystrom method, the basis vectors size is set to 200 (except for CBCL Face and EMG database, which is set to 500). For adaboost methods, we fix the number of boosting iterations to 1000 and 200 for synthetic and real-world datasets (except for EMG database, which is set to 1000) respectively, and select the classifier number which has the highest classification accuracy for testing samples. A list of the datasets and the optimal parameters is given in Table 1. We decompose each dataset into two parts: training samples and testing samples, which are independent from each other. 20 independent sets of training and testing samples are generated by running 20 random divisions on data samples in each set with a fixed proportion $8:2$.

Two synthetic datasets, Spiral and Fourclass datasets, are used to test the above algorithms. These two datasets are often considered to be highly nonlinear datasets. In Fig. 1, from left to right, we show the decision boundaries produced by Ada-LR, CART, MLSVM, LIBSVM, and HMLSVMs respectively. The decision boundaries of HMLSVMs are smoother than that of the ensemble methods and nearly approach that of the kernel methods. The mean number of support vectors used by LIBSVM, CVM and Nystrom, and the

Table 2: Testing accuracy of the ten methods. The mean rate together with standard deviation is reported.

Datasets	LSVM	LIBSVM	CVM	Nystrom	Ada-Stu	Ada-Per	Ada-LR	MLSVM	CART	HMLSVMs
Spiral	55.1±2.1	100±0	99.9±0.1	99.2±0.6	70.4±1.7	67.3±2.0	70.8±1.8	99.4±0.2	73.5±1.9	99.7±0.3
Fourclass	76.9±2.9	100±0	100±0	99.5±0.3	82.6±3.2	82.6±3.8	98.7±0.4	97.5±0.9	96.2±1.6	99.5±0.6
Monks1	66.3±3.7	99.5±0.8	99.6±0.8	98.4±1.8	75.1±3.4	76.7±5.8	77.8±3.5	97.5±1.9	74.1±4.6	99.8±0.5
Pokerhand	66.5±1.4	99.3±0.2	99.6±0.2	98.5±0.3	66.5±1.4	81.3±2.5	75.6±3.3	97.3±0.4	77.5±2.7	96.4±1.1
Letter	97.7±0.6	99.9±0.1	99.9±0.1	99.4±0.7	99.4±0.3	98.3±0.7	98.3±0.5	99.2±0.7	87.9±2.0	99.7±0.2
EMG	59.6±1.9	97.3±0.4	97.1±0.3	95.5±0.7	73.6±1.8	71.2±2.4	69.7±1.9	93.4±2.7	89.3±2.5	94.2±1.3
Statlog	95.8±1.2	98.3±0.7	98.6±0.7	98.4±0.4	93.6±1.9	96.0±1.3	95.8±1.4	96.5±2.6	89.0±2.5	97.2±1.2
Segment	89.7±2.3	94.7±1.6	95.3±1.6	94.3±2.8	93.5±2.1	93.1±1.4	93.0±1.8	93.2±2.7	85.2±1.3	94.0±3.6
Iris	70.2±9.1	96.9±1.9	96.7±2.8	96.0±3.4	90.0±6.4	92.4±3.9	92.1±5.2	92.7±2.8	89.3±5.8	96.7±2.5
Pendigit	97.1±1.0	99.8±0.2	99.8±0.2	99.5±0.4	97.5±0.9	98.5±0.8	97.7±0.8	99.6±0.1	91.8±3.2	99.4±0.3
Chess	75.6±1.1	96.6±0.3	96.9±0.3	92.5±0.6	84.2±1.8	77.7±1.7	77.0±1.1	92.3±3.9	71.3±2.0	92.8±0.9
SVMguide	95.3±0.5	96.8±0.5	96.9±0.4	96.7±0.4	96.2±0.2	94.7±0.8	93.8±0.6	95.9±0.5	85.1±0.7	96.5±0.3
Building	94.7±0.8	99.6±0.6	99.0±0.6	98.9±0.6	97.3±0.9	95.2±0.9	95.1±0.7	98.0±0.9	84.5±2.2	98.3±0.4
Face	98.0±0.3	99.6±0.1	99.7±0.1	99.3±0.2	98.4±0.3	97.9±0.3	97.5±0.4	99.0±0.3	85.2±1.4	99.2±0.1

Table 3: Testing complexity in the classification stage. The mean number together with standard deviation is reported.

Datasets	LIBSVM	CVM	Nystrom	Ada-Stu	Ada-Per	Ada-LR	MLSVM	CART	HMLSVMs
Spiral	616.0±10.3	1600±0	195.7±3.5	143.1±29.7	161.2±21.4	169.7±36.5	50±0	6.6±0.2	8.3±0.3
Fourclass	468±2.0	99.8±3.5	40.3±1.8	292.3±252.3	627.4±321.3	743.6±154.6	20±0	6.3±0.2	4.5±0.3
Monks1	290.2±5.3	367.3±3.8	186.1±1.7	40.8±20.0	157.3±39.6	160.2±33.2	20±0	4.9±1.3	4.1±0.4
Pokerhand	2068.5±14.5	775.1±25.2	200±0	22.1±26.0	188.8±13.3	166.7±35.1	40±0	8.8±0.1	10.4±2.2
Letter	98.5±5.6	332.5±6.1	198.4±1.2	79.5±47.1	114.4±65.1	133.0±42.9	20±0	9.5±0.2	4.3±1.5
EMG	4962±35	11922±731	499±0.7	723±95	889±24	785± 54	100±0	13.3±0.1	5.6±0.7
Statlog	108.1±3.9	168.0±5.9	200±0	52.3±48.9	112.9±62.3	98.7±74.1	40±0	7.7±0.4	4.9±1.2
Segment	146.7±4.2	270.5±7.6	98.6±1.3	106±48.8	97.3±66.0	137.3±53.1	20±0	8.4±0.2	4.5±1.4
Iris	26.2±1.6	120.0±0	35.1±0.3	66.6±63.8	60.9±38.0	123.7±53.4	5±0	5.6±0.4	3.8±0.8
Pendigit	89.3±4.1	510.3±9.2	200±0	69.9±60.9	171.6±27.5	154.5±28.9	20±0	10.4±0.7	3.8±0.4
Chess	879.9±11.0	1398.0±16.1	200±0	174.7±25.5	149.5±55.3	115.5±65.5	50±0	6.9±0.2	8.0±0.6
SVMguide	510.5±13.8	1152.2±38.3	87.8±11.5	34.1±41.5	21.2±18.6	116.3±73.1	20±0	6.1±0.7	2.2±0.9
Building	647.6±11.4	340.8±14.2	200±0	131.6±44.2	160.4±43.8	144.9±46.6	40±0	10.5±0.1	4.1±0.9
Face	397.1±7.7	661.1±12.9	499.6±0.7	168.5±17.7	128.1±35.4	101.7±41.3	50±0	9.0±0.0	2.9±0.7

mean number of linear classifiers employed by Ada-Stump, Ada-Perceptron, Ada-LR and MLSVM, and the mean number of linear classifiers for CART and HMLSVMs encountered per testing sample are shown in Table 3[1]. Although the number of linear classifiers employed by HMLSVMs for the two synthetic datasets in our experiments are respectively 66 and 25, the mean number of linear classifiers encountered per testing sample are respectively 8.3 and 4.5. This is due to the hierarchical structure of HMLSVMs, which predicts the label of a testing sample by making the sample go through from the root node to a leaf node of HMLSVMs. From the average numbers of SVs or linear classifiers in Table 3, it can be seen that HMLSVMs is orders of magnitude faster than the kernel and ensemble methods. CART achieves the classification cost comparable to that of HMLSVMs, but its testing accuracy is very poor, even lower than that of LSVM on some datasets.

To further demonstrate the power of HMLSVMs, we also conducted experiments on 12 real-world datasets obtained from sources [Frank and Asuncion, 2010; Chang and Lin, 2001]. From Table 2, it can be seen clearly that CVM and LIBSVM outperform all other methods, however, they are at the cost of generating the largest number of SVs (as shown in Table 3), and this makes them have the highest computational complexity in classification stage. The methods of Nystrom and MLSVM report considerable reductions of the number of SVs or LSVMs, but the numbers are still much higher than that of HMLSVMs. HMLSVMs achieves good classification performance comparable to KSVMs (CVM, LIBSVM, Nystrom). More importantly, it classifies a testing sample only via several (at most 10.4) LSVMs, so the time cost of HMLSVMs in the classification stage is significantly reduced. This is the main focus of HMLSVMs, which is not having the best testing accuracy but provide a method capable of classifying a pattern in a few milliseconds while obtaining a competitive performance. Thanks to the regularization term in (13), this ensures that the HMLSVMs do not fall into overfitting and have the good generalization ability comparable to KSVMs. The hierarchical structure of HMLSVMs makes it

[1]The codes of LIBSVM, CVM and Nystrom are implemented in C++ and the codes of Ada-Stump, Ada-Perceptron, Ada-LR, MLSVM, CART and HMLSVMs are implemented in MATLAB, hence we prefer the number of SVs or linear classifiers to the classification time in fairness. Note that the complexity of MLSVM scales with twice the number of linear classifiers, including gate function and classifier evaluations

achieve the high efficiency in the classification stage.

5 Conclusions

In this paper, we proposed a novel classifier called HMLSVMs by using a greedy strategy, which has a hierarchical structure with a mixture of LSVMs at each node. The hierarchical structure of HMLSVMs and the regularization term in (13) make it have the high speed in the classification stage and the good classification performance comparable to KSVMs. Experiments on both synthetic and real datasets are presented to show the effectiveness of our method. In the future, we will focus on the online learning of HMLSVMs.

Acknowledgement

This work is partly supported by NSFC under Grants 61472285, 6141101224, 61473212, 61100147, 61203241 and 61305035, and partly by the NSF of Zhejiang Province under Grants LY12F03016, LY15F030011 and LQ13F030009, and partly by Project of Science and Technology Plans of Zhejiang Province under Grants 2014C31062.

References

[Bach and Jordan, 2005] F. Bach and M. Jordan. Predictive low-rank decomposition for kernel methods. In *Proceedings of the 22th International Conference on Machine Learning*, Bonn, Germany, 2005.

[Bennett and Blue, 1998] K. P. Bennett and J. A. Blue. A support vector machine approach to decision trees. In *IEEE International Joint Conference on Neural Networks*, pages 2396–2401, 1998.

[Bousquet *et al.*, 2004] O. Bousquet, S. Boucheron, and G. Lugosi. *Introduction to statistical learning Theory*. Springer, Berlin, 2004.

[Breiman *et al.*, 2004] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Wadsworth, 2004.

[Breiman, 1996] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[Breiman, 2001] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[Chang and Lin, 2001] C. C. Chang and C. J. Lin. Libsvm: a library for support vector machine, 2001.

[Cheng *et al.*, 2010] H. B. Cheng, P. N. Tan, and R. Jin. Efficient algorithm for localized support vector machine. *IEEE Transactions on Knowledge and Data Engineering*, 22(4):537–549, 2010.

[Downs *et al.*, 2001] T. Downs, K. Gates, and A. Masters1. Exact simplification of support vector solutions. *Journal of Machine Learning Research*, 2:293–297, 2001.

[Fan *et al.*, 2005] R. E. Fan, P. H. Chen, and C. J. Lin. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

[Fei and Liu, 2006] B. Fei and J. Liu. Binary tree of svm: A new fast multiclass training and classification algorithm. *IEEE Transactions on Neural Networks*, 17(3):696–704, 2006.

[Fornoni *et al.*, 2011] M. Fornoni, B. Caputo, and F. Orabona. Locally linear support vector machines. In *Proceedings of International Conference of Machine Learning*, pages 985–992, 2011.

[Frank and Asuncion, 2010] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[Fu *et al.*, 2010] Z. Fu, A. Robles-Kelly, and J. Zhou. Mixing linear svms for nonlinear classification. *IEEE Transactions on Neural Networks*, 21(12):1963–1975, 2010.

[Gao and Zhang, 2009] J. B. Gao and J. Zhang. Sparse kernel learning and the relevance units machine. 5476:612–619, 2009.

[Hsieh *et al.*, 2008] C. J. Hsieh, K. W. Chang, C. J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008.

[Joachims and Yu, 2009] T. Joachims and C. N. Yu. Sparse kernel svms via cutting-plane training. *Machine Learning*, 76(2–3):179–193, 2009.

[Joachims, 2011] T. Joachims. Training linear svms in linear time. In *PProceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining*, pages 217–226, Philadelphia, PA, 2011.

[Ladicky and Torr, 2013] L. Ladicky and P. H. S. Torr. Multiclass latent locally linear support vector machines. In *JMLR: Workshop and Conference Proceedings*, volume 29, pages 229–244, 2013.

[Li *et al.*, 2011] Y. J. Li, B. Liu, X. W. Yang, Y. Z. Fu, and H. J. Li. Multiconlitron: A general piecewise linear classifier. *IEEE Transactions on Neural Networks*, 22(2):276–289, 2011.

[Liang, 2010] X. Liang. An effective method of pruning support vector machine classifiers. *IEEE Transactions on Neural Networks*, 21(1):26–38, 2010.

[Meir and Zhang, 2003] R. Meir and T. Zhang. Generalization error bounds for bayesian mixture algorithms. *Journal of Machine Learning Research*, 4:839–860, 2003.

[Nguyen and Ho, 2006] D. D. Nguyen and T. B. Ho. A bottom-up method for simplifying support vector solutions. *IEEE Transactions on Neural Networks*, 17(3):792–796, 2006.

[Pavlov *et al.*, 2000] D. Pavlov, J. Mao, and B. Dom. Scaling-up support vector machines using boosting algorithm. In *Proceedings of 15th International Conference on Pattern Recognition*, volume 2, pages 219–222, 2000.

[Qi *et al.*, 2011] G. J. Qi, T. Qi, and T. Huang. Locality-sensitive support vector machine by exploring local correlation and global regularization. In *Proceedings of 24th IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, USA, 2011.

[Schoelkopf *et al.*, 1999] B. Schoelkopf, S. Mika, C. J. C. Burges, P. Knirsch, K. Muller, G. Ratsch, and A. J. mola. Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, 10(5):1000–1017, 1999.

[Tipping, 2001] M. E. Tipping. Sparse bayesian learning and the relevance vector machine. 1:211–244, 2001.

[Tsang *et al.*, 2005] I. W. Tsang, J. T. Kwok, and P. M. Cheung. Core vector machines: fast svm training on very large datasets. *Journal of Machine Learning Research*, 6:363–392, 2005.

[Williams and Seeger, 2001] C. Williams and M. Seeger. Using the nystrom method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, volume 13, pages 682–688. MIT Press, 2001.

[Wu *et al.*, 2006] M. R. Wu, B. Schoelkopf, and G. Bakir. A direct method for building sparse kernel learning algorithms. *Journal of Machine Learning Research*, 7:603–624, 2006.