Binary Tree of SVM: A New Fast Multiclass Training and Classification Algorithm

Ben Fei and Jinbai Liu

***Abstract*—We present a new architecture named Binary Tree of support vector machine (SVM), or BTS, in order to achieve high classification efficiency for multiclass problems. BTS and its enhanced version, c-BTS, decrease the number of binary classifiers to the greatest extent without increasing the complexity of the original problem. In the training phase, BTS has $N-1$ binary classifiers in the best situation (N is the number of classes), while it has $\log_{4/3}((N+3)/4)$ binary tests on average when making a decision. At the same time the upper bound of convergence complexity is determined. The experiments in this paper indicate that maintaining comparable accuracy, BTS is much faster to be trained than other methods. Especially in classification, due to its Log complexity, it is much faster than directed acyclic graph SVM (DAGSVM) and ECOC in problems that have big class number.**

***Index Terms*—Binary tree of support vector machine (BTS), c-BTS, multiclass classification, probabilistic output, support vector machine (SVM).**

I. Introduction

SUPPORT VECTOR MACHINE (SVM) performs well in predictive data analysis. Book [1] is one of the best references on this subject. Reference [2] is an elementary introduction of SVM, and it introduces the basic concepts of SVM detailedly. SMO [3] implemented a fast training of binary model. Keerthi and Gilbert [21] studied and proved the convergence property of SMO, while the more rigorous proof was given in [22]. Shilton *et al.* [20] introduced an incremental training algorithm, which is suitable for problems of sequentially arriving data and fast constraint parameter variation. For multiclass SVM, there are mainly two types of approaches for training and classification, one of which is to combine several binary classifiers, and another is to consider all the classes in one big optimization problem. Suppose the number of classes is N, we make a brief introduction of these approaches.

1) One-against-all: Each class is trained against the remaining $N-1$ classes that have been collected together. Thus, it has N binary classifiers, but it introduces more nonlinear into the original problem. We always expect the linearly separability between different classes. It has to test N binary decision functions to predict a sample data point.
2) One-against-one: Here $N(N-1)/2$ binary classifiers are trained, and each classifier separates a pair of classes. Outputs $N(N-1)/2$ of times binary tests are required to make a final decision with majority voting.
3) Directed acyclic graph SVM (DAGSVM): Introduced by Platt *et al.* [4]. A total of $N(N-1)/2$ classifiers are trained. DAGSVM depends on a rooted binary directed acyclic graph to make a decision. When a test sample x reaches the leaf node, the final decision will be made. The binary tests times are the number of the nodes in the decision path.
4) Error correcting output codes (ECOC) of kernel machine: Introduced by Dietterich and Bakiri [18]. In this method, the error correcting output codes have been employed to improve the decision accuracy. one-against-one as well as one-against-all are the two wildly used coding schemes. Depending on the encoding scheme, N times binary tests at least or $N(N-1)/2$ times at most are needed to make a decision.

Besides the above four methods, for the method of solving multiclass SVM in one step referenced in [8]–[10], we agree with its theoretical contribution but currently it seems this method is not practical to many applications, for it generates a large optimization problem, which leads to time-consuming training. We will not test this method in this paper. Detailed experiments and comparison of this kind of method can be found in reference [5], which suggested that one-against-one and DAGSVM are more practical for real applications.

However, for all these methods, too many binary tests are required to make a decision. Although DAGSVM and ECOC may reduce the binary test times, they at least need N times tests yet. This shortcoming results in inefficiency of multiclass SVM in applications with many classes.

Tree architecture is always leveraged in decision theory. Support vector machines with Binary Tree Architecture (SVM-BTA) [11] has been introduced to reduce the number of binary classifiers and to achieve a fast decision. SVM-BTA needs to train $N-1$ classifiers and test $\log_2 N$ times for the final decision. But to get a good classifier of one node, it has to evaluate 2^N grouping possibilities with N classes in this node. That is poorly acceptable for a real application. Another problem that should be considered seriously is that although grouping different classes together to get global dichotomies can decrease the number of binary classifiers, it may increase the complexity of the original problem.

Here we present a new architecture named Binary Tree of SVM, or BTS for abbreviation. BTS is also based on binary classifications, but it decreases the number of binary classifiers to the greatest extent without increasing the complexity of the

Manuscript received May 18, 2004; revised November 14, 2005.

B. Fei is with the Department of Mathematics, Tongji University, Shanghai 200092, China (e-mail: feibenalgebra@hotmail.com).

J. Liu is with the Department of Information and Control Engineering, Tongji University, Shanghai 200092, China.

Digital Object Identifier 10.1109/TNN.2006.872343

origin problem. Instead of grouping different classes together to train a global classifier, BTS selects two classes for training in every node, and employs probabilistic outputs to measure the similarity between remaining samples and the two classes used in training. And then, all samples in the node are assigned to the two subnodes derived from the two classes. Repeating the same steps on every node till each node contains only one class samples, and then we will get a binary tree. Besides BTS, we will also introduce a more balanced and more efficient version of BTS, named c-BTS (centered BTS). The average convergent efficiency of BTS is $\log_{4/3}((N+3)/4)$ (Theorem 1). So BTS is fast to make decision. For the readers' convenience, we will introduce SVM briefly in Section II. The algorithm of BTS and c-BTS will be presented in Section III, followed by performance analysis in Section IV. Experiments are given in Section V, which demonstrate the excellent capability of BTS and c-BTS in multiclass problems that have big class number.

II. SVM Preliminary

For binary problem, we have training data points: $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$, where $x_i \in R^d$, $y_i \in \pm 1$. A total m samples.

Solving the optimized separating plane

$$f(x) = (w \cdot x) - b \quad \operatorname{sgn}(f(x)) = \begin{cases} 1 & \text{if } f(x)>0 \\ -1 & f(x)<0 \end{cases} \tag{1}$$

where $(\cdot)$ means inner product of vector.

The optimization problem can be formulated as

$$\min_{w,b} \frac{1}{2} w^T w, \qquad \text{subject to } y_i(w \cdot x_i - b) \geq 1 \tag{2}$$

It is a quadratic optimization problem in a convex set.

We can solve the Wolfe dual instead

$$\max_\alpha L_D \equiv \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j}^{m} \alpha_i \alpha_j y_i y_j x_i \cdot x(j) \tag{3}$$

where $\alpha_i, \alpha_2, \ldots, \alpha_m \geq 0$ are Lagrange multipliers. When α_i are solved, w can be calculated as

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i. \tag{4}$$

For the nonlinear case, we may solve the problem in a certain high dimension space by a kernel map $\Phi(\cdot)$

$$\max_\alpha L_D \equiv \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j}^{m} \alpha_i \alpha_j y_i y_j \Phi(x_i) \cdot \Phi x(j) \tag{5}$$

where $\Phi(x_i) \cdot \Phi(x_j) = k(x_i, x_j)$. That is, the dot product in that high dimensional space is equivalent to a kernel function of the input space.

The radial basis function (RBF) kernel is used in our experiments

$$k(x_i, x_j) = \exp\left(i\gamma \|x_i - x_j\|^2\right), \qquad \gamma > 0. \tag{6}$$

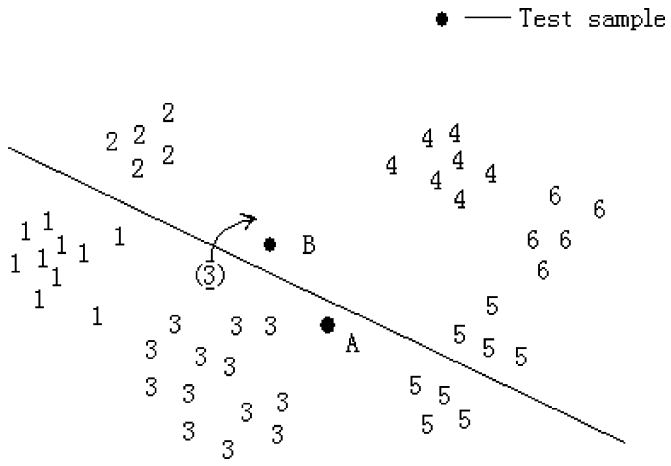

Fig. 1. Multiclass problem with six classes.

RBF is a reasonable first choice. The RBF kernel nonlinearly maps samples into a higher dimensional space, so unlike the linear kernel, it can handle the case when the relation between class labels and attributes is nonlinear.

For imperfect separation, the penalty C is used to penalize the data points that cross the boundaries. The only difference from the perfectly separating case is that α_i is bounded by C

$$\begin{cases} 0 \leq \alpha_i \leq C \\ \sum_i \alpha_i y_i = 0 \end{cases}. \tag{7}$$

III. Algorithm of BTS and c-BTS

A. The BTS

A sample with d features corresponds to a point in d dimension Euclidean space. Samples of different classes are distributed in the same Euclidean space. Now we will introduce the basic motivation of BTS.

Fig. 1 shows a multiclass problem. If the pair of class 1 and 2 has been trained, we will get a separating plane (or named a binary decision function $f(x)$, or a binary classifier). For class 1, $\operatorname{sgn}(f(x)) = -1$. And for class 2, $\operatorname{sgn}(f(x)) = 1$. At the same time the separating plane separates class 3 from class 4 and 6 (although not the optimized one). And class 5 has been scattered by the plane. In the testing phase, when an input testing sample (denoted as x') is at position A $(\operatorname{sgn}(f(x)) = -1)$, then x' does not likely belong to class 4 or 6, in other words, we need not train between class 3–4, or between 3–6 (after all training is a time consuming process). To reduce training times to the greatest degree, we have developed a new architecture named binary tree of SVM (BTS). We introduce BTS at first. The enhanced version, c-BTS, which needs additional processing to build a more balanced tree, will be introduced at the end of Section III.

At the beginning, all the samples are added to a tree's root node, from which, first, we randomly select a pair of classes, train them to get a separating plane, create two empty subnodes, and then designate the two classes' samples to two subnodes, respectively. Second, assign all the other samples in the root node to the two subnodes according to the output of the pair's classifier (the separating plane).

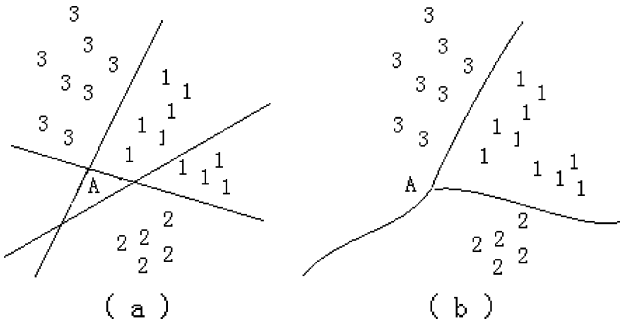

Fig. 2. (a) Separating planes generated by one-against-one, testing samples in position A will not be well classified. (b) Probabilistic output method makes samples in region A classifiable.

After one step separating, two subnodes, denoted as 0 and 1, have been created. Each contains a portion of the samples from its parent node. In our example, class 1 and 3 are assigned to subnode 0. Class 2, 4, and 6 are assigned to subnode 1, while the samples in class 5 are scattered to the two different subnodes. To call this separating procedure recursively upon subnode 0 and 1, respectively, until every new created node contains only one class, then we build up the whole BTS.

Obviously the above process can reduce the number of binary classifiers, but is it reasonable enough? For example, if a testing sample is at position B, it will be classified to subnode 1 by the separating plane of class 1 and 2. So this sample will not have a chance to be classified to class 3, but in fact it most likely belongs to class 3. This problem may cause severe decrease of classification accuracy. The experiments in Section V clearly demonstrate this decrease.

So the probabilistic output is leveraged to solve this problem. Firstly, we introduce the probabilistic output in the previous works and explain what kind of problems it can solve. Then we will use probabilistic output to improve BTS's classification accuracy.

We begin with an example, presented in Fig. 2(a), which is a paradigm of a three-class problem. To train it by one-against-one, we will get three separating planes. If a testing sample is at position A, the majority voting method cannot decide which class the sample belongs to.

One method for fitting probabilistic is to fit Gaussian to the class condition densities $P(f|y=-1)$ and $P(f|y=1)$ which is firstly proposed by Hastie, Tibshirani in [7]. As shown in Fig. 2(b), samples in region A become classifiable according to the posterior probability. In [6], the posterior probability is estimated by (8)

$$P(y=1|f)=\frac{1}{1+\exp(-f)}$$
$$P(y=1|f)=50\% \text{ when } f=0. \tag{8}$$

Or one can use another Sigmoid

$$P(y=1|f)=\frac{1}{1+\exp(af^2+bf+c)}. \tag{9}$$

But the function in (9) is nonmonotonic. Nonmonotonic means that (9) cannot be used as a global estimation of posterior probability, its output will dramatically departure from the real value near the inflexion. So (9) is not a suitable model.

In [12], Platt explained it detailedly and he suggested a parametric form of a sigmoid as

$$P(y=1|f)=\frac{1}{1+\exp(Af+B)}. \tag{10}$$

It works well, but fitting the two parameters (A and B) needs more computational time. In the process of building a BTS, we always expect a faster procedure, and we only need a simple method to measure whether assigning a sample to a subnode is reasonable; so we adopt (8) to calculate the reasonability.

Return to Fig. 1, class 3 has been totally assigned to subnode 0, but some of its samples are too close to the plane (i.e., their probabilistic outputs are less than a certain threshold), so we reassign these samples to subnode 1, then class 3 will have chance to be trained with classes in subnode 1. Class 4 and 6 are far away from the plane, so no reassignment occurs.

Still in Fig. 1, when a testing sample is at position B, from root node, it will be classified to subnode 1. With the reassignment, it may be classified to class 3. Otherwise this data point will lose the possibility to be classified to class 3 at the root node. Experiments in Section V report high improvement of accuracy after implementing the reassignment. Now we list the algorithm in details.

Here is the description of symbols and data structures.

1) Sample_vector: The vector contains total training data points.
2) Trained_list: The list contains the trained models. And each model contains the parameters of a binary classifier.
3) δ: The threshold indicates the distance between a data point and the separating plane of node.
4) f_K: The binary decision function of node K.
5) $\mathrm{sgn}(f_K)$: If $f_K(x)>0$, then $\mathrm{sgn}(f_K)=1$, else if $f_K(x)<0$, then $\mathrm{sgn}(f_K)=-1$.
6) $\Delta P_K(x_i)$: The reasonability of a sample x_i in node K belonging to subnode 0 or 1. If it is smaller than δ, x_i may be reassigned to the other subnode, seeing step2.f for detail. The $\Delta P_K(x_i)$ is defined as

$$\Delta P_K(x_i)=P\left(y|f_K(x_i)\right)-50\%=\frac{1}{1+\exp\left(-f_K(x_i)\right)}-50\% \tag{11}$$

The algorithm mainly contains three procedures.

```
Procedure 1: Initialize
Add all the training data points into
the Root-Node. Set Trained list to be
blank.
Procedure 2: Build subtree of Binary
Tree of SVM
step1. Input node K, and check if it
contains different classes.
step2. Evaluation: If it contains dif-
ferent classes, then we randomly se-
lect two denoted as i and j, and create
```

two child nodes named 0 and 1. Begin the evaluation process:
a. Search in Trained list to check if the ith and jth class have been trained as a binary problem.
b. If not, take out all the data points that belong to the ith and jth class from Sample_vector, generate a binary problem, train it by SMO to get a model, and add the model into the Trained_list.
c. If the two classes as a binary problem has been trained, then the model is already in the Trained_list.
d. Assign data points in the ith class of node K to the child node 0, and the data points in the jth class of node K to the child node 1.
e. Test remaining sample points of the other classes in node K, if the output of $\mathrm{sgn}(f_K(x))$ is same with class i, then assign x to the child node 0, if $\mathrm{sgn}(f_K(x))$ is same with class j, then assign x to the child node 1. At the same time, the probabilities of these data points are also calculated.
f. If data points of a certain class in node K (except the two classes already trained) have been assigned to child node 0 (child node 1) wholly, but some of their probabilistic outputs $|\Delta P_K(x_i)| < \delta$, then reassign these data points to child node 1 (child node 0). The purpose of this step is to enable this class to be trained with classes in another child node. Obviously, if the probabilities of a class's data points are all greater than δ, the reassignment will not happen.
g. If all of the remaining classes in node K have been scattered into two child nodes, then clear the two child nodes:
g.1 Select another pair in node K again.
g.2 If we can find out a pair of two classes that has not been evaluated in node K, then go to step2.a.
g.3 If all pairs in node K have been evaluated, then the node encounters the worst situation. So delete the two child nodes, save models of these pairs in node K and return. (The worst situation will be studied in section IV).
h. Else the pair of i and j is named the "*accepted pair*."

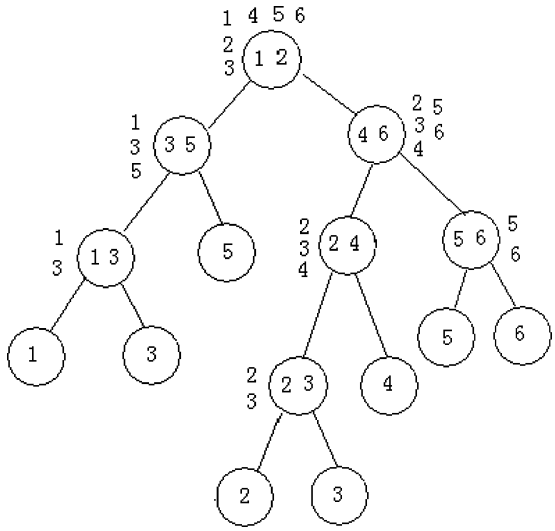

Fig. 3. Illustration of BTS with the training samples shown in Fig. 1. A circle is a node of BTS and the pair inside a circle is the accepted pair, which has been trained to get a binary decision function. Notice that class 3 and 5 is scattered into two leaf nodes. Class 3 is scattered by reassignment and class 5 is scattered by the decision function of class 1 and 2.

i. The procedure 2 is called recursively for the two child nodes.
step3. If only one class belongs to node K, return.

Remark: In step 2.b, we take out all the samples of class i and j from the original dataset.

After finishing building a BTS, every leaf node contains data points which belong to one or more than 2 (except 2) classes, while one class may be scattered to different leaf nodes. Fig. 3 is a possible BTS of a 6-class problem in Fig. 1. For that the two classes in a node is selected randomly, we will introduce c-BTS to exclude the randomicity.

Procedure 3: Classification in BTS
Starting from the root node, a testing sample is classified into subnode 0 or 1 by the classifier of the accepted pair in the current node, until it reaches the leaf node. If a leaf node contains more than two models (the worst situation occur in this leaf node, as explained in step 2.g.3 of Procedure 2), the final decision will be made by majority voting just as one-against-one.

B. The Accepted Pair and c-BTS

In procedure 2, step2, the class i and j are selected randomly. If a different pair has been selected, the final structure of BTS may be different. Moreover, we always expect that the BTS to be a balanced tree, which will reach the best decision efficiency. The following method will be helpful to get a more balanced tree.

First, the centers of all the classes in a node are calculated. Then the average value of all the centers is calculated as the node's center. The classes in the node will be sorted according to the Euclidean distances from their centers to the node's center. The nearest two are firstly evaluated till we find the *accepted pair*. After doing so, we can get a unique binary tree. It will be more balanced. We name the new tree as c-BTS, where "c" means "centered."

C. Probabilistic Model and δ

At present, the probabilistic model (8) is adopted to measure the reasonability of reassignment. This model may not fit the real probabilistic distribution of the problem exactly, but here we do not need an accurate model. The advantage of doing so is that (8) can be intuitively seen as a different representation of Euclidean distance of instance space. When a sample is close to the classification plane, the output of (8) is close to 50%. Without fitting parameters process, the calculation is thereby fast.

The δ is selected globally. Considering the efficiency of the BTS architecture, it is difficult to select different δ for distinct nodes. We currently tune the global δ to improve the accuracy. In general, a bigger δ leads to a higher accuracy, while the number of classifiers and the training time will increase at the same time.

IV. Performance Analysis

Now we analyze the performance of BTS. The best and the worst performance will be analyzed, respectively. The general convergence analysis is in Section IV-C.

A. The Best Situation: Train $N-1$ Binary Problems

In the best situation, we only need to train $N-1$ binary classifiers, because in this situation, no class is scattered into different subnodes. If the numbers of the classes in one node and its subnodes are N_0, N_1, N_2, respectively, we have $N_0 = N_1 + N_0$ because the two subnodes have no common class. Finally, the BTS must have N leaf nodes and totally $2N-1$ nodes in the tree. For every node, except the leaf nodes, we need to solve a binary classification problem; so the BTS will have $(2N-1)-N = N-1$ binary classifier.

B. The Worst Situation: Train $N(N-1)/2$ Binary Problems

In the worst situation, we have to train 2 binary $N(N-1)/2$ classifiers. This situation happens when the classifiers generated by any pair in every node scatter the other classes in the same node. Suppose that the node contains N_0 classes, each subnode has $N_0 - 1$ classes. So we record the trained class pair and its classifier into a list (Trained_list in Section III), if a pair has been trained in a node, it will never be trained again in another node. Procedure 2, step 2, b and c are the implementation. In such a situation each pair will be trained as one-against-one or DAGSVM. The only difference is that samples of the same class will be scattered to different leaf nodes. Noticing that here the BTS is a balanced binary tree, which has $2^k - 1$ nodes at kth layer, every node in this layer has $N - k + 1$ classes. Because every leaf node has only one class, so the number of layers of BTS is N, the number of leaf nodes is 2^{N-1}. The number of nodes in BTS is $2^N - 1 = 1 + 2 + 2^2 + 2^3 + \ldots + 2^{N-1}$. It is an exponential increase of the number of nodes. That is why this situation is named the "worst" one. The classification can be made after testing $N-1$ binary decision functions.

TABLE I
Problem List

Problem	#Class	#Training data	#Testing data	#Feature
letter	26	15000	5000	16
optdigit	10	3823	1797	64
pendigit	10	7494	3498	12
texture	100	1500	1000	45

We have to adopt some techniques to deal with the worst situation. When a node encounters the worst situation, we record the trained pair and its classifier to the Trained_list, delete the two subnodes, and select another pair of the node to train. If the worst situation still occurs, then we repeat the above steps until i) we find a pair that does not lead to the worst situation, or ii) all pairs in the node have been evaluated.

In the second condition, no subnode will be created. In classification phase, when a testing sample reaches such a leaf node, we will make a final classification by majority voting just as one-against-one.

Procedure 2- >step 2- >g.1, g.2, g.3 is the implementation of the above process. As a result, we can guarantee the nodes in BTS will not increase exponentially.

C. General Analysis of the Convergence

Suppose that no worst situation occurs in any node of BTS, we have the following theorem on the general convergence property.

Theorem 1: The complexity order of the convergence of BTS is $O(\log N)$. More precisely, the average convergence performance is $\log_{4/3}((N+3)/4)$.

Proof: In ith layer of BTS, a node has N_i classes. If one class is scattered and reassigned into two subnodes, the total number of classes in the subnodes of node i is $N_i + 1$. If the worst situation occurs, then no subnode will be generated. Suppose that no worst situation occurs during training in any node, the number of the classes in subnodes could be N_i (the best situation), N_i+1, N_i+2, ..., or N_i+N_i-3. All of the N_i-2 cases have equal chance to occur. So the average number of classes in one subnode is

$$N_{i,s} = \frac{N_i + (N_i+1) + \ldots + (N_i+N_i-3)}{2(N_i-2)} = \frac{3(N_i-1)}{4} \tag{12}$$

Suppose that total class number is N. The root node containing N classes is at 0th layer of BTS, we will prove that at the ith layer of BTS, one node averagely has

$$N_i = \left(\frac{3}{4}\right)^i (N+3) - 3 \tag{13}$$

classes.

TABLE II
LETTER

Method	C, γ	#Binary Classifiers	#Binary Tests			Accuracy (%)
			Max	Min	Ave	
One-against-one	16, 4	325	325	325	325	95.2
DAGSVM	16, 4	325	26	26	26	95.2
ECOC-L-One-All	16, 2	26	26	26	26	95.06
ECOC-L-One-One	16, 2	325	325	325	325	95.2
BTS, $\delta = 0.0\%$	16, 2	261	22	6	13.05	85
BTS, $\delta = 1.5\%$	16, 2	277	22	6	13.8694	93.4
BTS, $\delta = 5\%$	16, 2	310	23	7	15.312	93.9
BTS, $\delta = 8\%$	16, 2	323	23	7	15.3706	95.2
c-BTS, $\delta = 0.0\%$	16, 2	268	24	6	13.9422	93.16
c-BTS, $\delta = 1.5\%$	16, 2	286	24	6	14.6878	93.94
c-BTS, $\delta = 5\%$	16, 2	305	25	7	14.7566	95.12

TABLE III
OPTDIGIT

Method	C, γ	#Binary Classifiers	#Binary Tests			Accuracy (%)
			Max	Min	Ave	
One-against-one	32, 0.0078125	45	45	45	45	97.4958
DAGSVM	32, 0.0078125	45	10	10	10	97.5
ECOC-L-One-All	32, 0.0078125	10	10	10	10	97.2732
ECOC-L-One-One	32, 0.0078125	45	45	45	45	97.4958
BTS, $\delta = 0.0\%$	32, 0.0078125	37	23	4	14.47468	97.5
BTS, $\delta = 1.5\%$	32, 0.0078125	37	23	4	14.49527	97.5
BTS, $\delta = 5\%$	32, 0.0078125	37	23	4	14.49527	97.5
c-BTS, $\delta = 0.0\%$	32, 0.0078125	36	11	4	7.40568	97.1063
c-BTS, $\delta = 1.5\%$	32, 0.0078125	38	11	5	7.18364	97.1619
c-BTS, $\delta = 5\%$	32, 0.0078125	42	11	6	7.93489	97.2732

At the 1th layer, one node averagely has $N_1 = (3/4)(N + 3) - 3$ classes, (13) holds; Suppose that at ith layer, the (13) holds, then when in $(\mathrm{i}+1)$th layer, according to (12)

$$\begin{aligned} N_{i+1} &= \frac{3(N_i - 1)}{2^2} \\ &= \left(\frac{3}{4}\right)\left[\left(\frac{3}{4}\right)^i (N+3) - 3 - 1\right] \\ &= \left(\frac{3}{4}\right)^{i+1} (N+3) - 3. \end{aligned}$$

So (13) still holds in $(\mathrm{i}+1)$th layer of BTS. Thus (13) holds.

When $N_i = 1$, the BTS will converge, hence

$$\left(\frac{3}{4}\right)^i (N+3) - 3 = 1.$$

Hence

$$i = \log_{\frac{4}{3}}\left(\frac{N+3}{4}\right). \tag{14}$$

So the average convergence performance is $\log_{4/3}((N+3)/4)$. The order of the complexity is $O(\log(N))$. ■

It is obvious that the BTS can classify a testing sample by averagely testing $\log_{4/3}((N+3)/4)$ binary classifiers. From 4 the proof of theorem 1, a subnode i in the ith layer will has $N_{i-1} - 1$ classes at most. Therefore, we have Theorem 2.

Theorem 2: The upper bound of the convergence of BTS is N.

In order to verify the efficiency of the BTS, we carried out the following multiclass classification experiments.

V. EXPERIMENTS

In this section, we provide the results of our multiclass problems' experiments. The first problem is **letter**, a 26-class problem from Statlog collection [14]. It was tested in [5]. **Letter** has 16 features, 15 000 training samples, and 5000 testing samples. The second and the third problems are two 10-class problems from the UCI Repository of machine learning databases [15]: **optdigit** and **pendigit**. The last problem is texture classification, which has 100-class from the Brodatz album [16]. The wavelet decomposition technique is applied to extract texture features [17].

In addition, LibSVM [13] with RBF kernel is used to solve the binary classification problem. Parameters C and γ in (6) and (7) are selected by the tool of LIBSVM, which is based on cross-validation. All the data have been scaled into $[-1, 1]$ as the suggestion of LibSVM. We list these problems in Table I.

Here, we compare BTS and c-BTS with the following methods:

1) one-against-one;
2) DAGSVM [4];
3) ECOC-L-One-All: ECOC with Likelihood decoding function and one-against-all coding scheme [19];
4) ECOC-L-One-One: ECOC with likelihood decoding function and one-against-one coding scheme [19].

We compare BTS and c-BTS with the above four methods in following three aspects.

TABLE IV
PENDIGIT

Method	C, γ	#Binary Classifiers	#Binary Tests			Accuracy (%)
			Max	Min	Ave	
One-against-one	32, 0.0078125	45	45	45	45	97.084
DAGSVM	32, 0.0078125	45	10	10	10	97.084
ECOC-L-One-All	32, 0.0078125	10	10	10	10	97.0555
ECOC-L-One-One	32, 0.0078125	45	45	45	45	97.084
BTS, $\delta = 0.0\%$	32, 0.0078125	33	8	4	6.27759	96.9697
BTS, $\delta = 1.5\%$	32, 0.0078125	36	10	4	6.39565	97.0555
BTS, $\delta = 5\%$	32, 0.0078125	36	10	4	6.39623	97.0555
c-BTS, $\delta = 0.0\%$	32, 0.0078125	30	10	4	6.43482	97.0555
c-BTS, $\delta = 1.5\%$	32, 0.0078125	31	10	4	6.46026	97.084

TABLE V
TEXTURE

Method	C, γ	#Binary Classifiers	#Binary Tests			Accuracy (%)
			Max	Min	Ave	
One-against-one	16, 0.0078125	4950	4950	4950	4950	77.1
DAGSVM	16, 0.0078125	4950	100	100	100	76.5
ECOC-L-One-All	16, 0.0078125	100	100	100	100	65.8
ECOC-L-One-One	16, 0.0078125	4950	4950	4950	4950	77.2
BTS, $\delta = 2.5\%$	16, 0.0078125	433	23	5	11.279	68.5
BTS, $\delta = 5\%$	16, 0.0078125	498	22	5	11.441	70.2
BTS, $\delta = 10\%$	16, 0.0078125	1498	26	7	14.151	75.7
BTS, $\delta = 12\%$	16, 0.0078125	1906	35	8	20.656	76.9
c-BTS, $\delta = 2.5\%$	16, 0.0078125	462	15	6	10.28	70.8
c-BTS, $\delta = 5\%$	16, 0.0078125	663	19	6	11.662	72.8
c-BTS, $\delta = 10\%$	16, 0.0078125	1280	28	8	15.012	76.8
c-BTS, $\delta = 12\%$	16, 0.0078125	1844	31	9	17.17	77.0

1) #Binary classifiers: The total number of binary classifiers that the method needs.
2) #Binary tests: Means how many binary classifiers have been tested when we classify a testing sample; In BTS and DAGSVM, it is the number of nodes in the decision path.
3) Accuracy: The ratio between the numbers of the correctly classified testing samples and the total testing samples.

In the Tables II–IV, all the six methods can reach almost the same accuracy. Among all the methods, c-BTS is the fastest one in decision. Another observation is that c-BTS is more efficient than normal BTS, which is apparent in Table III. However, if the number of the classes is small, the advantage of BTS is not prominent. So we conducted a 100-class texture classification problem.

Texture classification experiment was conducted on a set of 100 texture images from the Brodatz album [16] and the wavelet decomposition technique is applied to extract texture features [17]. At first, each image was divided into 25 segments and every segment has 128×128 pixels. Fifteen of the segments were used for training and the others are used for testing.

Obviously, as shown in Table V, a higher δ leads to a higher accuracy and more binary classifiers. However, when the accuracy of c-BTS is as high as that of one-against-one, increasing δ will not improve the accuracy apparently.

In training phase, when $\delta = 12\%$, c-BTS reaches the accuracy of 77.0% with only 1884 binary classifiers, which is less than half of 4950. A test sample, starting from the root node of BTS, reaches a leaf node after at most 31 binary tests. Due to the $\log_{4/3}((N+3)/4)$ computational complexity in 4 BTS, the average binary tests (17.17) is almost six times faster than that of DAGSVM and ECOC-L-One-All, and is 300 times faster than one-against-one and ECOC-L-One-One in classification phase.

TABLE VI
LETTER: FOUR-FOLD CROSS-VALIDATIONS OF C-BTS, $\delta = 1.5\%$

	C, γ	# Binary Classifiers	# Binary Tests			Accuracy (%)
			Max	Min	Ave	
	16, 2	290	22	6	14.4344	93.42
	16, 2	286	24	6	14.6878	93.94
	16, 2	294	22	6	13.7876	92.78
	16, 2	286	21	5	13.633	94.6
Expected Value		289	22.25	5.75	14.1357	93.685
Variance		11	1.1875	0.1875	0.19198	0.004479

Moreover, c-BTS gets the highest scores in the computational time (average Binary tests). Although ECOC-L-One-All has the least binary classifiers, which means the least storage, it costs more decision making time than c-BTS and BTS, and its accuracy is the lowest in the six methods.

We also examined if c-BTS has stable behavior under different data sets by cross-validation. We mix the training data and the testing data in Table I, and split them into several sets randomly. For letter and texture, fourfold cross-validation is conducted; For optdigit and pendigit, we do threefold cross-validation experiments instead. See Tables VI–IX for the results.

Verified by cross validation, the c-BTS has small variation on the precise and the averaged decision times. So it is stable under different data sets. Especially in Table IX, although the variance of the number of binary classifiers is relatively big (2216.25), the decision time complexity can still be well controlled.

TABLE VII
OPTDIGIT: THREE-FOLD CROSS-VALIDATIONS OF C-BTS, $\delta = 5\%$

	C, γ	# Binary Classifiers	# Binary Tests			Accuracy (%)
			Max	Min	Ave	
	32, 0.0078125	42	11	6	7.93489	97.27320
	32, 0.0078125	36	11	3	7.63908	98.53333
	32, 0.0078125	36	10	4	7.26385	98.64160
Expected Value		38	10.6667	4.3333	7.6126	98.1494
Variance		8	0.2222	1.5556	0.0754	0.00385

TABLE VIII
PENDIGIT: 3-FOLD CROSS-VALIDATIONS OF C-BTS, $\delta = 1.5\%$

	C, γ	# Binary Classifiers	# Binary Tests			Accuracy (%)
			Max	Min	Ave	
	32, 0.0078125	31	10	4	6.46026	97.0840
	32, 0.0078125	31	18	4	7.37603	99.0392
	32, 0.0078125	31	18	4	7.37640	99.0390
Expected Value		31	15.3333	4	7.0709	98.3874
Variance		0	14.2222	0	0.1864	0.008494

TABLE IX
TEXTURE: 4-FOLD CROSS-VALIDATIONS OF C-BTS, $\delta = 10\%$

	C, γ	# Binary Classifiers	# Binary Tests			Accuracy (%)
			Max	Min	Ave	
	16, 0.0078125	1280	28	8	15.012	76.8
	16, 0.0078125	1297	25	7	15.46046	77.0
	16, 0.0078125	1316	28	7	15.518	74.7
	16, 0.0078125	1193	27	7	15.522	71.5
Expected Value		1271.5	27	7.25	15.3781	75
Variance		2216.25	1.5	0.1875	0.04527	0.04895

VI. CONCLUSION AND FUTURE WORK

From these experiments, we can draw the following conclusions.

1) The accuracy is guaranteed by tuning the threshold δ. Generally, increasing δ can improve the accuracy of BTS. However, it makes BTS generate more classifiers. Moreover, all the methods can achieve almost the same accuracy.
2) The average $\log_{4/3}((N+3)/4)$ decision complexity of BTS is the fastest one in the current methods. With bigger number of classes, the advantages of BTS will become more prominent.
3) In order to keep the original complexity and obtain the decision robustness, BTS still adopts one-to-one training scheme. At the same time, the probabilistic model is used to reduce the total number of classifiers. By doing so, the number of classifiers is great less than other methods except ECOC-L-One-All.
4) c-BTS is a unique tree. It can be easily seen that c-BTS is more balanced than a normal BTS, and is faster in decision. c-BTS may not be the best solution to control the selection of the accepted pair in a node, but it is a feasible way to exclude the randomicity and improve a normal BTS.

At present, it is difficult to select different thresholds for every node. We can only control the tree by a global threshold δ. The future task is to find an efficient way to get a globally optimized δ.

REFERENCES

[1] V. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.
[2] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining Knowl. Discov.*, vol. 2, no. 2, pp. 1–47, 1998.
[3] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods—Support Vector Learning*, B. Scholkopf, C. J. C. Burges, and A. J. Smola, Eds. Cambridge, MA: MIT Press, 1999, pp. 185–208.
[4] J. Platt, N. Cristianini, and J. Shawe-Taylor, "Large margin DAGSVM's for multiclass classification," *Advances in Neural Information Processing System*, vol. 12, pp. 547–553, 2000.
[5] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multi-class support vector machines," *IEEE Trans. Neural Netw.*, vol. 13, pp. 415–425, 2002.
[6] G. Wahba, "Support vector machines, reproducing kernel Hilbert spaces and the randomized GACV," in *Advances in Kernel Methods—Support Vector Learning*, B. Scholkopf, C. J. C. Burges, and A. J. Smola, Eds. Cambridge, MA: MIT Press, 1999, pp. 69–88.
[7] T. Hastie and R. Tibshirani, "Classification by pairwise coupling," *Ann. Statist.*, vol. 26, no. 2, pp. 451–471, 1998.
[8] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *J. Mach. Learn. Res.*, vol. 2, pp. 265–292, 2001.
[9] ——, "On the learnability and design of output codes for multiclass problems," in *Comput. Learn. Theory*, 2000, pp. 35–46.
[10] J. Weston and C. Watkins, "Multi-class support vector machines," in *Proc. ESANN99*, M. Verleysen, Ed.. Brussels, Belgium, 1999.
[11] S. Cheong, S. H. Oh, and S.-Y. Lee, "Support vector machines with binary tree architecture for multi-class classification," *Neural Info. Process.—Lett. Rev.*, vol. 2, no. 3, Mar. 2004.
[12] J. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *Advances in Large Margin Classifiers*, A. J. Smola, P. Bartlett, B. Scholkopf, and D. Schuurmans, Eds. Cambridge, MA: MIT Press, 1999.
[13] LibSVM: A Library for Support Vector Machine, C.-C. Chang and C. J. Lin. (2001). *http://www.csie.ntu.edu.tw/cjlin/libsvm* [Online]
[14] Statlog Data Set*ftp://ftp.ncc.up.pt/pub/statlog/* [Online]
[15] UCI Repository of Machine Learning Databases, C. Blake, E. Keogh, and C. Merz. (1998). *http://www.ics.uci.edu/mlearn/MLRepository.html* [Online]
[16] P. Brodatz, *Textures: A Photographic Album for Artists and Designers*. New York: Dover, 1966.
[17] A. Busch and W. W. Boles, "Texture classification using wavelet scale relationships," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 4, pp. 3584–3587, 2002.
[18] T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *J. Artific. Intell. Res.*, vol. 2, pp. 263–286, 1995.
[19] A. Passerini, M. Pontil, and P. Frasconi, "New results on error correcting output codes of kernel machines," *IEEE Trans. Neural Netw.*, vol. 15, no. 1, pp. 45–55, Jan. 2004.
[20] A. Shilton, M. Palaniswami, D. Ralph, and A. C. Tsoi, "Incremental training of support vector machines," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 114–131, Jan. 2005.
[21] S. S. Keerthi and E. G. Gilbert, "Convergence of a generalized SMO algorithm for SVM classifier design," *Mach. Learn.*, vol. 46, pp. 351–360, 2002.
[22] N. Takahashi and T. Nishi, "Rigorous proof of termination of SMO algorithm for support vector machines," *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 774–776, May 2005.

Ben Fei received the B.S. degree in physics and the Ph.D. degree in fundamental mathematics from Tongji University, Shanghai, China, in 1999 and 2005, respectively.

He is currently a researcher with the IBM China Research Lab. His research interests involve algebraic number theory, function analysis, machine learning, and information retrieval. His interests include both pure mathematics and cutting edge computer science research topics.

Jinbai Liu received the B.S. and M.S. degrees in automatic control from Tongji University, Shanghai, China, in 2000 and 2005, respectively.

He is currently a software engineer with Microsoft MSN Technology Center, Shanghai, China. His research interests cover machine learning, image processing, and control theory.