International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:6, No:4, 2012

Binary Classification Tree with Tuned Observation-based Clustering

Maythapolnun Athimethphat and Boontarika Lerteerawong

***Abstract*—**There are several approaches for handling multiclass classification. Aside from one-against-one (OAO) and one-against-all (OAA), hierarchical classification technique is also commonly used. A binary classification tree is a hierarchical classification structure that breaks down a *k*-class problem into binary sub-problems, each solved by a binary classifier. In each node, a set of classes is divided into two subsets. A good class partition should be able to group similar classes together. Many algorithms measure similarity in term of distance between class centroids. Classes are grouped together by a clustering algorithm when distances between their centroids are small. In this paper, we present a binary classification tree with tuned observation-based clustering (BCT-TOB) that finds a class partition by performing clustering on observations instead of class centroids. A merging step is introduced to merge any insignificant class split. The experiment shows that performance of BCT-TOB is comparable to other algorithms.

***Keywords*—**multiclass classification, hierarchical classification, binary classification tree, clustering, observation-based clustering

I. Introduction

MANY classification studies consider only a two-class problem while a real world classification problem sometimes requires classifying examples into more than two categories. Several approaches to handle such multiclass problem have been introduced. Among the simplest but effective ones are one-against-all (OAA) and one-against-one (OAO) decomposition schemes. Decomposition is a technique that handles a multiclass problem by breaking it down into several binary sub-problems, each solved by a binary classifier. A more general approach is P-against-Q (PAQ), where $P \geq 1$, $Q \geq 1$, and $P + Q = m$, where m is a codeword length. OAA could be represented as PAQ with $P = 1$ and $Q = m - P$, where m is equal to the number of classes k.

Decomposition can be implemented in two ways, single-call and multi-call. A single-call classification requires modifying original learning algorithms. Alternatively, a multi-call classification can be chosen in order to avoid changing the existing algorithms. Using a one-against-all approach, one needs k binary classifiers to discriminate each of the k classes from the rest. Similarly, a one-against-one approach requires $k(k - 1)/2$ binary classifiers, one for each pair of classes.

Maythapolnun Athimethphat is with Assumption University, Bangkok 10240 Thailand (phone: +66 2719-1515 Ext. 3681, 3682; Fax: +66 2719-1639; e-mail: maytha@scitech.au.edu).

Boontarika Lerteerawong was graduated from Assumption University, Bangkok, Thailand. (e-mail: l.boontarika@gmail.com).

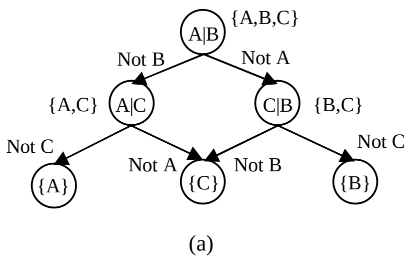
An ensemble of binary classifiers combines classification results from all classifiers by a voting technique such as max-win voting or confidence-based voting [1].

Instead of running in parallel, an ensemble can be constructed in a way that classification in one step is based on classification in previous steps in a hierarchical fashion, forming a tree structure. Thus, hierarchical classification does not need a voting mechanism but determines a class label of an instance by tracing a tree down to a leaf node, which contains one of the class labels. A balanced classification tree usually takes less processing time than one-against-all and one-against-one decompositions, and since it does not need a voting mechanism, it does not experience ambiguity in classification outputs, such as contradicting results by two or more classifiers [1]-[3].

One of the common classification trees is a binary classification tree. At the root of a top-down binary classification tree, an original *k*-class problem is transformed into a binary problem by grouping classes into two groups according to certain criteria such as a class size or class similarity. Many algorithms use distances between class centers, or centroids, to measure class similarity. In this paper, we propose a new approach in partitioning classes by considering similarity between observations instead of class centroids. Class partition is determined by class labels of observations in each group. This helps the algorithm to investigate any possible subpatterns in a class. However, it could increase runtime due to the introduction of redundancy. Thus, we introduce a merging step to merge a minor class split to a larger one. In the experiments, the algorithm is compared with other existing binary classification trees.

II. Hierarchical Multiclass Classification

An ensemble of classification tree can be constructed in two ways, bottom-up or top-down. The construction of a bottom-up tree begins with leaf nodes, each of which contains one of the k classes. Therefore, the tree will have altogether k leaf nodes. The leaf nodes are paired, and they are merged with their siblings, or one of the nodes in each pair is eliminated. The process repeats until one node is left at the root level [4]. On contrary, top-down tree construction starts at the root. Decision Directed Acyclic Graph (DDAG) [5] performs one-against-one classification at each node. At the root, binary classification is performed based on two classes selected from the initial list of k classes. The label that does not win in the classification will be removed from the list, and the process is repeated with the remaining classes until one class is left as shown in Figure 1. In general, bottom-up and top-down approaches could yield a tree with comparable classification performance. Yet, a top-down tree is more likely to have a balance structure and is easier to implement.



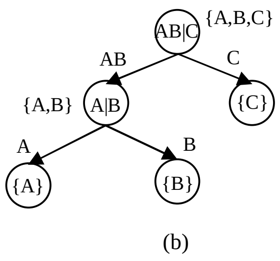


Fig. 1 Examples of (a) DDAG and (b) binary classification tree. A class set in a node and its member are shown in curly brackets

There are many other ways a top-down tree can split. For instance, Unbalanced Decision Tree (UDT) [6] splits by discriminating one class from the rest at each level, creating a very unbalanced binary tree structure. Using a one-against-all concept, UDT could suffer from class imbalance but perform faster than DDAG. On contrary, nested dichotomies [7] algorithm creates a balanced tree structure by dividing classes into two groups, or meta-classes. Members for each meta-class are determined randomly. Class-balanced nested dichotomies (ECBND) algorithm divides classes so that the two meta-classes contain approximately the same number of classes. Another alternative, data-balanced nested dichotomies (EDBND), tries to make the number of observations in the two groups approximately equal. Algorithms like Half-Against-Half (HAH) [8] and SVM Binary Decision Tree (SVM-BDT) [9] focus on obtaining meta-classes in a way that similar classes are in the same meta-class.

III. Top-down Binary Classification Tree

A binary classification tree is an ensemble of classifiers, with a binary tree structure, that breaks down an original k-class classification problem into $(k - 1)$ binary sub-problems when the tree is balanced. As a result, it can effectively deal with a problem where k is large. Its performance is largely influenced by how classes or meta-classes are chosen, or how a tree splits. A tree can be constructed such that partition separability near the top of the tree is higher than at the bottom. This makes it easier to solve classification problems near the root. Classification performance at the root is especially important because misclassification in one step cannot be recovered in the later steps due to the structure of the tree that requires tracing the tree downward. Thus, higher overall accuracy can be achieved when more discriminative problems are solved first. With this concept, many algorithms try to group similar classes in the same subsets so that confusing classes are kept for the later steps. Unsupervised techniques like hierarchical clustering and k-means are used in finding the best partition for splitting [8]-[10].

As illustrated in Table 1, building a top-down tree starts at the root node with an initial training data set D and a set of k classes. The class set S is partitioned into two disjoint subsets, S_1 and S_2, with a class partition function (refer to Table 2). Ideally, we want a partition function that is able to balance the sizes of the two subsets and group similar classes together. A class subset is named collectively with a meta-class label. Observations in the training set D are relabeled according to their meta-classes, for example, C_1 for the first class subset S_1 and C_2 for S_2. As a result, the k-class problem is transformed

TABLE I
Build Top-Down BCT (D, S)

	Steps	Illustration
1.	Given the original multiclass problem, start at the root node.	Dataset: D Class: S = {A, B, C, D, E}
2.	Transform the original problem with PARTITION (D, S).	- D is partitioned into D_1 and D_2. - S is partitioned into S_1 and S_2.
3.	Given the new binary problem (data relabeled with 2 meta-classes), train a binary classifier.	f = a binary classifier
4.	Recursively construct the left and right subtrees with the two sub-problems.	- Repeat the process on D_1 (and the corresponding S_1) for the left subtree. - Repeat the process on D_2 (and S_2) for the right subtree.
5.	Stop when the given class set contains only one class.	

TABLE II
Partition (D, S) (Class-based Clustering)

	Steps	Illustration
1.	Use clustering to group classes in S into two class subsets.	S_1 = {A, B, C} S_2 = {D, E}
2.	Divide the given dataset D into two subsets.	D_1 = observations with class labels in S_1 D_2 = observations with class labels in S_2
3.	Relabel observations corresponding to their clusters.	- Relabel observations in D_1 as C_1 to represent the original label in S_1. - Relabel observations in D_2 as C_2.

to the two-class problem. Then a binary classifier learns from the relabeled training set.

After training, the dataset D is divided into two subsets D_1 and D_2, according to the two meta-classes. The data set D_1, along with its corresponding class set S_1, is passed on to the left child node, and so D_2 and S_2 to the right child node. The process is repeated recursively on each child node until a class set of the node contains only one class. As a result, it finally produces a binary tree of k leaf nodes, one for each class, and at least $(k - 1)$ internal nodes, each of which contains an independently trained classifier [3], [8], [10].

A popular approach in finding a class partition is to perform unsupervised clustering to group different classes into two clusters. A margin tree algorithm [11] finds a partition by considering margins between classes. Alternatively, other algorithms such as Half-Against-Half (HAH) and SVM binary decision tree (SVM-BDT) measure distance between class centroids. A centroid, or a center, is an average point of a cluster. In Figure 2, a centroid of each class is calculated, and a clustering technique such as k-means and hierarchical clustering is applied on the centroids. Usually Euclidean distance is used as a metric in measuring a distance between centroids.

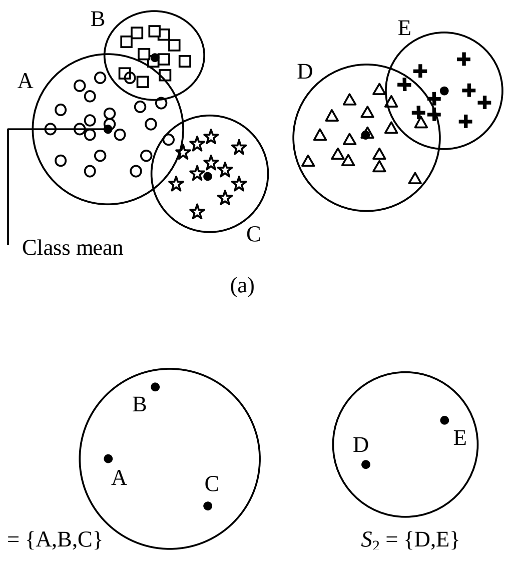


Fig. 2 Illustration of how classes (A, B, C, D, and E) are partitioned according to their centroids. (a) First, class means are calculated to obtain class centroids. (b) Then, class centroids are grouped using unsupervised clustering

After the classification tree is constructed, a class label of an unlabeled observation can be obtained by following the path of the tree, from the root to the leaf, where a class label is determined. Starting at the root, an input of unlabeled data is fed to the tree trained during the tree construction. A classifier in the node assigns each observation into one of the meta-classes, and so to one of the child nodes. When observations move on to the respective child nodes, the process repeats until they reach the leaf level, where observations are assigned with class labels [8]-[10].

IV. Class Partition with Observation-based Clustering

In this paper, we propose an alternative for binary classification tree construction. Our algorithm follows the same process as how a binary tree is built top-down as in Table 1. However, it splits the tree differently. Instead of finding S_1 and S_2 with a clustering algorithm and then dividing D into D_1 and D_2 according to the class subsets, the proposed algorithm splits the other way around. In Figure 3, unsupervised clustering is used to partition the dataset D into D_1 and D_2 first. Then the subsets of classes S_1 and S_2 is derived from the two data subsets. We refer to this class partition process as observation-based clustering.

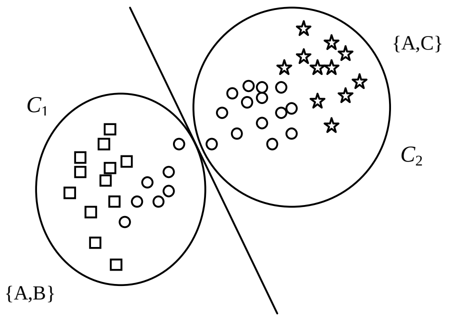


Fig. 3 Illustration of how three classes might be partitioned when clustering is performed on observations. A straight line shows a possible classification function that discriminates the two clusters

From Table 3, given the original dataset D, unsupervised clustering is used to group observations in D into two clusters D_1 and D_2. Assume that the original classification problem initially has five classes, S = {A, B, C, D, E}. Because D_1 contains only observations of class A, B, and C, let S_1 = {A, B, C}. Similarly for D_2, which contains observations of class A, D, and E, let S_2 = {A, D, E}. Because clustering is based on observations, S_1 and S_2 might not always be disjoint as they do in the traditional class-based clustering approach. In this case, they are not disjoint because class A is in both clusters.

The observation-based clustering used in a class partition step allows observations of one class to be examined in different subtrees, so a tree can better detect subpatterns that might exist in a class. However, this approach could increase runtime due to the introduction of redundancy. It takes away the advantage of the traditional binary tree that has a tree size bounded to the number of classes. In addition, it might also lead to an overfitting problem.To prevent an insignificant split of any class into two subclasses (to appears in both S_1 and S_2), a merging threshold ths_m is introduced. If a sample size of any of the two subclasses is too small (having a proportion smaller than ths_m), the smaller subpattern will be merged to the larger

TABLE III
PARTITION (D, S) (TUNED OBSERVATION-BASED CLUSTERING)

	Steps	Illustration
1.	Perform clustering to group observations in D into two clusters D_1 and D_2.	
2.	From the two data subsets, let S_1 and S_2 be a set of distinct class labels in D_1 and D_2 respectively.	S_1 = distinct classes of observations in D_1 = {A, B, C} S_2 = distinct classes of observations in D_2 = {A, D, E}
3.	Let y_i be a class in S, where $i = 1$ to k. Let $p_{1,\,yi}$ be the proportion of observations of class y_i in cluster D_1 and $p_{2,\,yi}$ in cluster D_2. If $p_{1,\,yi}$ or $p_{2,\,yi}$ is smaller than ths_m, move observations of class y_i in the smaller cluster to the larger cluster.	$p_{1,\,\mathrm{A}}$ = the proportion of observations of class A in cluster D_1 = 0.20 $p_{2,\,\mathrm{A}}$ = the proportion of observations of class A in cluster D_2 = 0.80 Given $ths_m = 0.30$, $p_{1,\,\mathrm{A}} < ths_m$. Move data of class A in D_1 to D_2.
4.	Update S_1 and S_2.	S_1 = {B, C} S_2 = {A, D, E}
5.	Relabel observations corresponding to their clusters.	- Relabel observations in D_1 as C_1. - Relabel observations in D_2 as C_2.

subpattern so that the class will appear only in either S_1 or S_2. From the previous example, class label A is in both S_1 and S_2. Assume 20 percent of observations of class A is in D_1 and the remaining 80 percent in D_2. If we set ths_m to 0.30, the proportion of observations of class A in D_1, which is 0.20, is less than ths_m. Thus all observations of class A in D_1 are moved to D_2. Consequently, S_1 now only contains class B and C. Therefore, if the threshold value is zero, the merging is not performed. On the other hand, if it is greater than 0.5, then the merging step is always performed.

V. EXTENSION FOR *K*-CLUSTER CLUSTERING

Unsupervised clustering algorithms like k-means require a user to input the number of clusters. Because clustering results are largely influenced by this parameter, there are many solutions introduced to determine the appropriate number of clusters. A user can determine the number of clusters with prior knowledge, visualization (plot or graph), or some measures (such as Bayesian Information Criterion). Many algorithms are developed to determine the number of clusters automatically. In [12], unsupervised clustering performance was cross-checked with the true class label. The number of clusters and the corresponding cluster centers found by the algorithm were close to the actual classes. This implies that we can let the number of clusters equal to the number of classes if class labels are given.

TABLE IV
PARTITION (D, S) (K-CLUSTER EXTENSION)

	Steps	Illustration
1.	Perform clustering to group observations in D into k clusters $D_1, \ldots, D_k$.	D contains observations of 5 classes, A, B, C, D, and E.
2.	Let $S_1, \ldots, S_k$ be a set of distinct class labels in $D_1, \ldots, D_k$ respectively.	S_1 = {A, B} S_2 = {B, C} S_3 = {C} S_4 = {D} S_5 = {D, E}

3.	Let $P_1, \ldots, P_k$ be cluster centers of cluster $D_1, \ldots, D_k$ obtained from a clustering algorithm in Step 1.
	Perform clustering to group centroids $P_1, \ldots, P_k$ into 2 clusters.
		Cluster 1 = {P_1, P_2, P_3}
		Cluster 2 = {P_4, P_5}

4.	If P_1 is in the first cluster, observations in D_1 are assigned to Cluster 1 (relabel as C_1). Otherwise, assign them to Cluster 2 (relabel as C_2). Repeat for all k cluster centers.	Examples in Step 2-3 imply that… Cluster 1 contains observations from {A, B, C}. Cluster 2 contains observations from {D, E}
5.	Continue Step 3-5 in the two-cluster version (Table III)	

In Table 4, the process starts with clustering observations in the dataset D, given that the cluster number is equal to the number of classes k. Alternatively, we can provide a clustering function with the class centroids as the initial cluster centers. After obtaining k clusters from the clustering algorithm, each observation is relabeled according to its cluster. Because a problem will be solved by a binary classifier, a partition function should provide a classifier with 2 meta-classes. Therefore, we have to reduce k clusters to 2 clusters by performing clustering on the cluster centers on the second layer. Observations are relabeled corresponding to the clustering results in the first and the second layers. An observation will be labeled as C_1 if its cluster on the first layer has a centroid belong to Cluster 1 in the second layer. Otherwise, it will be labeled as C_2.

VI. EXPERIMENT

We have conducted an experiment to study performance of the binary classification tree with tuned observation-based clustering (BCT-TOB) on eight UCI datasets [13]. The unsupervised clustering technique used in this study is k-means because it is more suitable for applying to a large number of data points than hierarchical clustering. Support vector machine (SVM) was used as a base classifier. The best merging threshold for each dataset was determined empirically.For each dataset, a training set and a test set were randomly chosen for classification and measuring performance. For datasets with sample size below 200, fifty percent of the data was used as a training set, and the remaining as a test set. Otherwise, a training set and a test set were sampled in a ratio of 70:30. Only Pendigits was divided into 7,494 samples for training and 3,498 for testing, as

TABLE V
PERFORMANCE OF BCT-TOB ON UCI DATASETS

Dataset	Sample size	No. of classes	No. of features	ths_m	Avg. accuracy (%)	
					2 clusters	*k* clusters
Iris	150	3	4	0.00	97.20 ± 1.93	97.20 ± 1.83
Wine	178	3	13	0.20	91.01 ± 1.68	83.52 ± 12.56
Balance-scale	625	3	4	0.40	***92.66 ± 2.20**	81.81 ± 2.65
Glass	214	6	9	0.00	66.92 ± 2.21	63.45 ± 6.68
Ecoli	336	8	7	0.20	84.95 ± 3.84	85.25 ± 3.11
Yeast	1,484	10	8	0.06	52.38 ± 2.26	49.82 ± 5.79
Pendigits	10,992	10	16	0.10	97.71 ± 0.79	***98.87 ± 0.62**
Vowel	990	11	10	0.10	88.62 ± 2.23	***91.75 ± 2.45**

TABLE VI
BCT-TOB PERFORMANCE COMPARED WITH OTHER TECHNIQUES

	BCT-TOB	SVM-BDT	HAH	ECBND	EDBND
Iris	***97.20 ± 1.83**	***97.87 ± 1.80**	***96.00 ± 3.08**	86.93 ± 12.82	86.93 ± 12.82
Wine	91.01 ± 1.68	***93.37 ± 3.02**	***93.71 ± 2.07**	***91.91 ± 3.38**	***91.91 ± 3.38**
Balance-scale	***92.66 ± 2.20**	88.62 ± 0.67	***91.70 ± 2.50**	***91.06 ± 3.05**	***91.06 ± 3.05**
Glass	***66.92 ± 2.21**	62.62 ± 2.81	63.38 ± 5.93	***67.54 ± 6.09**	***66.62 ± 7.14**
Ecoli	85.25 ± 3.11	83.96 ± 2.91	84.75 ± 4.15	86.34 ± 3.26	84.46 ± 6.23
Yeast	52.38 ± 2.26	***59.78 ± 1.96**	***60.33 ± 2.02**	58.49 ± 1.24	58.31 ± 2.03
Pendigits	***98.87 ± 0.62**	97.02 ± 0.32	96.63 ± 0.18	97.00 ± 0.23	96.62 ± 0.31
Vowel	91.75 ± 2.45	97.61 ± 1.30	***97.71 ± 0.71**	***97.64 ± 1.27**	***97.34 ± 1.34**

provided by the source. The process was repeated 10 times to obtain an average accuracy and a standard deviation for each dataset. Table 1 summarizes description of each dataset and its corresponding best merging threshold and performance.

From Table 5, we can see that a threshold value is roughly between 0 and 0.20 for the eight datasets. Only Balance-scale requires a merging threshold as high as 0.40 because data points are highly scattering and setting the threshold too low could cause overfitting. On the other hand, Iris and Glass require no merging. In Iris, changing the threshold value made no significant affect on the performance since the patterns are clearly discriminative.

Comparing performance by 2-cluster and *k*-cluster versions of BCT-TOB, in most datasets, the two versions yielded comparable classification results, implying that clustering observations into 2 clusters is sufficient for obtaining good performance. However, BCT-TOB performed significantly better (at 95% confidence) in Pendigits and Vowel when clustering into *k* clusters. In Balance-scale, performance significantly dropped when using a *k*-cluster extension.

Next, in Table 6, BCT-TOB's best performance was compared with other four classification tree techniques, namely SVM-BDT, HAH, both of which use class-based clustering, and the two nested dichotomies techniques, class-balanced (ECBND) and data-balanced (EDBND), which randomly partition classes in a way that balances sizes of class and data, respectively. All techniques were implemented in R statistical package, and SVM was used as a base classifier for every classification tree algorithm. Table 6 shows how each algorithm performs in each dataset. A significantly (95% confidence) highest average accuracy (in percent) of each dataset is emphasized in bold and marked with an asterisk.

From the table, we can see that BCT-TOB performed well in Iris, Balance-scale, Glass, and Pendigits. Its performance was as good as SVM-BDT and HAH in Iris dataset and as good as ECBND and EDBND in Glass dataset. In Balance-scale and Pendigits, BCT-TOB outperformed other techniques. Nevertheless, BCT-TOB was inferior to other algorithms in Wine, Yeast, and Vowel. In Wine, its accuracy was significantly smaller than those of SVM-BDT and HAH but approximately as large as ECBND and EDBND. Finally, in Ecoli, there is no significant difference in performance across the five methods. In sum, BCT-TOB's performance is satisfying in 5 out of 8 datasets.

VII. CONCLUSION

A binary classification tree is an ensemble of classifiers that breaks down a *k*-class problem into multiple binary sub-problems, each solved by a binary classifier. In each non-leaf node, a given class set is divided into two disjoint subsets. A common approach is to perform unsupervised clustering to group similar classes together. Algorithms like Half-Against-Half (HAH) and SVM binary decision tree (SVM-BDT) measure similarity in term of distance between class centroids. Classes with small distance are grouped together in the same cluster.

In this paper, we propose a different way of splitting a tree. Instead of performing clustering on class centroids, the proposed binary classification tree, BCT-TOB, performs clustering on observations to allow one class to appear in more than one cluster in aim to increase redundancy. A merging step is introduced to merge any insignificant class split to avoid overfitting and reduce runtime. Data subsets of a class in two clusters are merged when one of the subsets has a proportion less than a given threshold. The experiment shows that a good threshold is usually between 0 and 0.20. In general, k-means of 2 clusters is able to provide a good split. In most datasets, BCT-TOB performs better than or at least as good as a class centroid method or a random method. Further study could investigate more into different clustering technique because it greatly affects classification performance. Any technique that can make use of class labels when perform class partitioning could also be considered.

Acknowledgment

This work was fully supported by Faculty of Science and Technology, Assumption University, Thailand.

References

[1] G. Ou, Y. L. Murphey and L. Feldkamp, "Multiclass Pattern Classification Using Neural Networks," in *17th International Conference on Pattern Recognition (ICPR'04)*, vol. 4, J. Kittler, M. Petrou, and M. Nixon, Eds. Cambridge: IEEE Computer Society Press, 2004, pp. 1051-4651.

[2] D. Tax and R. Duin, "Using Two-class Classifiers for Multiclass Classification," in *Proceedings of the 16th International Conference on Pattern Recognition*, vol. 2, 2002, pp. 124-127.

[3] S. Kumar, J. Ghosh and M. M. Crawford, "Hierarchical Fusion of Multiple Classifiers for Hyperspectral Data Analysis," in *Pattern Analysis & Applications*, vol. 5, no. 2. London: Springer London, 2002, pp. 210-220.

[4] A. Beygelzimer, J. Langford and P. Ravikumar, "Multiclass Classification with Filter Trees," June 2007. [Online]. Available: http://hunch.net/~jl/projects/reductions/mc_to_b/invertedTree.pdf.

[5] J. C. Platt, N. Cristianini and J. Shawe-Taylor, "Large Margin DAGs for Multiclass Classification," in *Advances in Neural Information Processing Systems*, vol. 12, S.A. Solla, T.K. Leen, K.R. Muller, Eds. Cambridge: MIT Press, 2000, pp. 547-553.

[6] A. Ramanan, S. Suppharangsan and M. Niranjan, "Unbalanced Decision Trees for Multi-class Classification," in *Second International Conference on Industrial and Information Systems (ICIIS 2007)*, Penadeniya, Sri Lanka, 2007.

[7] L. Dong, E. Frank, and S. Kramer, “Ensembles of Balanced Nested Dichotomies for Multi-Class Problems,” in *Knowledge Discovery in Databases: PKDD 2005*, vol. 3721, A. M. Jorge, et la., Eds. New York: Springer-Verlag, pp.84-95.

[8] H. Lei and V. Govindaraju, "Half-Against-Half Multi-class Support Vector Machines," in *Multiple Classifier Systems*, vol. 3541, N. C. Oza, et la., Eds. New York: Springer-Verlag, 2005, pp. 156-164.

[9] G. Madzarov, D. Gjorgjevikj and I. Chorbev, "A Multi-class SVM Classifier Utilizing Binary Decision Tree," in *Informatica*, vol. 33, no. 2, S. Lian, D. Kanellopoulos, and G.J. Ruffo, Eds. Ljubljana, Slovenia: Slovenian Society Informatika, 2008, pp. 233-241.

[10] J. Lee and I. Oh, "Binary Classification Trees for Multi-class Classification Problems," in *The Seventh International Conference on Document Analysis and Recognition*. A. Antonacopoulos, Ed. Edinburgh, Scotland: IEEE Computer Society Press, 2003, pp. 770-774.

[11] R. Tibshirani, T. Hastie, “Margin Trees for High-dimensional Classification,” in *Journal of Machine Learning Research*, vol. 8, S. Džeroski, P. Geurts, and J. Rousu, Eds. Brookline, MA: Microtome Publishing, 2007, pp. 637-652.

[12] R. Kothari and D. Pitts, "On finding the number of clusters," in *Pattern Recognition Letters*, vol. 20, no. 4. New York: Elsevier Science, 1999, pp. 405-416.

[13] A. Frank and A. Asuncion, *UCI Machine Learning Repository*, 2010. [Online] http://archive.ics.uci.edu/ml.