Binary Classification Tree for Multiclass Classification with Observation-based Clustering

Maythapolnun Athimethphat[1] and **Boontarika Lerteerawong**[2], Non-members

ABSTRACT

Many classification techniques are originally designed to solve a binary problem, but practically many classification problems involve more than two classes. A multiclass problem can be decomposed into binary sub-problems, each solved by a binary classifier. Aside from using one-against-one (OAO) or one-against-all (OAA) decomposition scheme, an ensemble of binary classifiers can be constructed hierarchically. In this study, we focus in multiclass classification with a binary classification tree and propose a new approach in splitting a top-down tree by grouping observations into two clusters with k-mean clustering. Unlike a traditional class-clustering approach, this observation-based algorithm allows one class to appear in two meta-classes so it can be examined in both sub-trees. A data cleaning process is also performed to avoid insignificant tree splits. The experiment shows how our proposed algorithm (BCT-OB) performed on different data sets, compared with other binary classification tree algorithms. Then advantages and disadvantages of the algorithm are discussed.

Keywords: Multiclass Classification, Support Vector Machine, Hierarchical Classification, Binary Classification Tree, Observation-Based Clustering

1. INTRODUCTION

Many robust classification techniques like Support Vector Machine are originally designed to solve a binary problem. Nevertheless, many classification problems practically involve more than just two classes. They are usually handled by decomposing the original multiclass problem into multiple binary subproblems. Decomposition can be achieved by several approaches, and the most common ones are one-against-all (OAA) and one-against-one (OAO). One-against-all decomposition is the simplest approach, which discriminates one class from the rest, while one-against-one performs pairwise discrimination, considering only two classes at a time.

Manuscript received on July 15, 2012 ; revised on October 30, 2012.

[1,2] The authors are with Faculty of Science and Technology, Assumption University 6th Floor, E Building, Ramkamhaeng Soi 24 Hua Mak, Bangkok 10240, Thailand, E-mail: maytha@scitech.au.edu and l.boontarika@gmail.com

Decomposition can be implemented in two ways, single-call and multi-call. A single-call classifier requires modifying original learning algorithms. In Support Vector Machine, the implementation involves changing its optimization formulation. Alternatively, a multi-call classification can be chosen in order to avoid changing existing algorithms. Using a one-against-all approach, k binary classifiers are needed to discriminate each class from the remaining classes. On the other hand, one-against-one requires $k(k-1)/2$ binary classifiers, one for each pair of the k classes. Unlike a single-call implementation, a multi-call classifier requires an additional decision module to combine classification results from multiple classifiers in an ensemble. Commonly, results are determined by counting votes on each class. The winning class receiving most votes is a class label of an instance. [1]

Hierarchical classification is a newer approach to multiclass classification. Because an ensemble of classifiers has a hierarchical structure, it requires no voting mechanism. A binary tree is a common hierarchical classification ensemble, and it can be constructed bottom-up or top-down. In general, both approaches could yield a tree with comparable performance. Yet, a top-down tree is more likely to have a balanced structure and is easier to implement. A binary classification tree like Half-Against-Half [2], Divide-by-2 [3], and SVM Binary Decision Tree [4] uses different unsupervised clustering techniques on class centroids to group classes into two subsets in order for a binary classifier to work on a problem. In this paper, we propose a new approach in partitioning classes by performing clustering on observations instead of class means.

2. HIERARCHICAL MULTICLASS CLASSIFICATION

Ambiguity can occur in a multi-call classification, using OAA or OAO decomposition. Sometimes more than one class receives the highest vote. More ties can occur in OAO than in OAA because of redundancy in pairwise classifications. Moreover, an OAO ensemble might also give contradicting results by two or more pairwise classifiers. In OAA, it is possible to have no class assigned to an instance. One possible solution to such ambiguity is to use a continuous confidence value instead of a discrete class vote. However, this requires a classifier to estimate a confidence value. Instead, an ensemble of classifiers can be constructed hierarchically so that classification results in one level will be passed on to the next in a hierarchy. The common structure for hierarchical classification is a binary tree, and it can be constructed bottom-up or top-down.

2.1 Bottom-up Tree

Constructing a bottom-up tree starts at the leaf level where initially each node represents each class (Fig. 1). Thus, there are altogether k leaf nodes. In a filter tree [5], nodes are paired with their siblings, and binary classification is performed on each. Winning class labels will be considered in the next level. The process repeats until the root node where one class is left. For a bottom-up multi-view forest [6], it is a merging process similar to agglomerative hierarchical clustering. The two nodes with the closest distance (the most similar classes) are merged to form a node of a new meta-class, a group of classes. This method is likely to create an unbalanced tree.

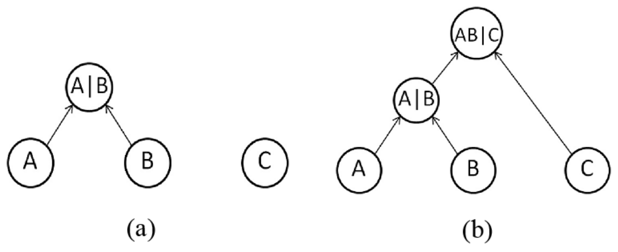


Fig.1: *A bottom-up tree construction, (a) starting at the leave level to (b) the root.*

2.2 Top-down Tree

Top-down tree construction starts with k classes at the root (Fig. 2). Classes are divided into two meta-classes, and binary classification is performed to classify observations accordingly, splitting the root into two child nodes. Observations will be examined recursively in each child node until it cannot be split further. Unbalanced Decision Tree (UDT) [8], for example, discriminates one class from the rest at each level, which creates a very unbalanced tree. Moreover, using an OAA concept, UDT could suffer from class imbalance. On contrary, balanced nested dichotomies [9] divide classes into two equal parts to construct a balanced binary tree. Other algorithms do not divide into two exact halves but focus more on how classes are similar or different. A binary classification tree, such as Half-Against-Half [2] and Divide-By-2 [3], groups similar classes to the same subsets.

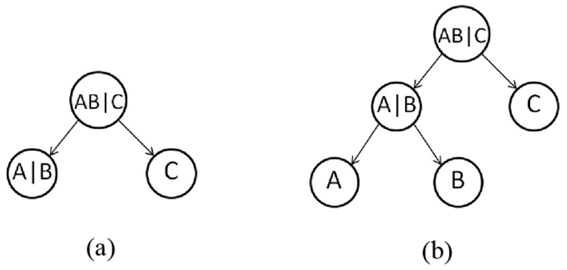


Fig.2: *A top-down tree construction, (a) starting at the root and (b) breaking down nodes to the leave level.*

2.3 Tree Splitting and Merging

Performance of a tree is largely influenced by how classes or meta-classes are chosen, or how a tree splits or merges. To improve runtime, a tree should be balanced [2]. This means a top-down tree should split equally and a bottom-up tree should merge nodes of the same level. In [9], such tree is called a system of balanced nested dichotomies. A tree can be constructed in a way that the number of classes in the left subset is about equal to the right subset, making the tree height bounded to log2 k. This approach creates an ensemble of class-balanced nested dichotomies (ECBND). However, it does not guarantee that a tree will be balanced in term of sample size because the numbers of observations in different classes might not be equal. By partitioning classes into two groups with approximately equal sample size, we get data-balanced nested dichotomies (EDBND). Similarly, in one-against-higher-order (OAHO) [10], classes are sorted by class sizes so the largest class is always compared against the rest of the smaller classes.

On the other hand, a tree can be constructed in a way that partition separability near the top of the tree is higher than the bottom (Fig. 3). This makes it easier to solve classification problems near the root. Classification performance at the root is especially important because when tracing a tree downward, misclassification cannot be recovered in the later steps. Thus, higher overall accuracy can be achieved when more discriminative problems are solved first. [4] [11].

To find a split, an algorithm can use one of the two extremes, to randomly select a tree structure [12] or to search for the optimal tree structure from all possible combinations [11]. While the former one is simple but less likely to achieve the optimal performance, the later one is too costly, especially when the number of classes is large. Many algorithms, however, 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 (e.g., [2 - 4], [6], [13]).

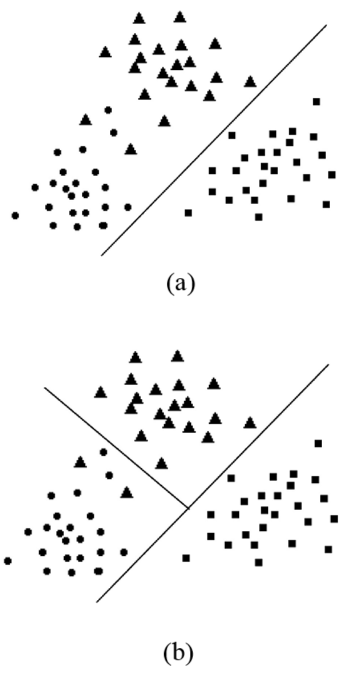


Fig.3: *A tree tries to solve (a) a more discriminative problem first and (b) a more difficult one later.*

3. 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 [14]. In this section, we will focus on constructing a top-down binary classification using a class-based splitting technique. Then, in the next section, our observation-based algorithm will be discussed.

3.1 Tree Construction

Building a top-down tree starts at the root node with an initial training data set D and a set of k classes, S. The class set S is partitioned into two disjoint subsets, S_1 and S_2, with a class partition function. Ideally, we want a partition function that is able to balance the sizes of the two subsets and group similar classes together into meta-class. Observations in the training set D are relabeled according to their meta-classes, resulting in a two-class problem. Then a binary classifier learns from the relabeled training set. After training, the data set 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 applied 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. Below summarizes a general algorithm for constructing a binary classification tree. [2] [11] [14].

```
**Algorithm** BUILD TOP-DOWN BCT $(D, S)$
INPUT:
$D = x_1, \ldots, x_n$, a data s et of n observations
$S = y_1, \ldots, y_k$, a set of $k$ class labels

IF $S$ contains only one class
    Terminate the process
    RETURN the class label in S
ENDIF
Partition $S$ into two subsets $S_1$ and $S_2$
Let $C_1$ be a meta-class label for $S_1$ and $C_2$ for $S_2$
FOR each $x$ in $D$
    Label $x$ with a meta-class label according to its
    label $y$
ENDFOR
Train a binary classifier $f$ with a data set $D$ and the
meta-class labels $\{C_1, C_2\}$
Divide $D$ into two subsets, $D_1$ and $D_2$, according to
meta-classes
Call BUILD TOP-DOWN BCT $(D_1, S_1)$
Call BUILD TOP-DOWN BCT $(D_2, S_2)$
```

3.2 Class Partitioning

A popular approach in finding a class partition is to perform unsupervised clustering to group different classes into two clusters. A margin tree [13] algorithm finds a partition by considering margins between classes in 3 ways, greedy, single linkage, and complete linkage. A greedy approach searches for a partition that yields the maximum margin between the two clusters from all possible partitions. Single linkage and complete linkage find a partition with the minimum within-cluster margins. While single linkage uses the shortest distance between classes, complete linkage considers the largest distance.

Alternatively, other algorithms such as Half-Against-Half (HAH) [2], SVM binary decision tree (SVM-BDT) [4], and multi-view forest [6] measure distance between class centroids. A centroid, or a center, is an average point in a cluster. Thus, when finding a class centroid, a mean value of each dimension in a particular class is calculated. An alternative to centroids is medoids, where medians are used, as applied in Partitioning Around Medoids (PAM). [15] After finding class centroids, a clustering technique like k-means and hierarchical clustering is applied on the centroids. Usually Euclidean distance is used as a metric in measuring a distance between two clusters.

Figure 4 shows how to find a class partition with centroid clustering. First, a mean value of observations in each class is calculated. We obtain k class centroids, and these data points are an input of a clustering algorithm. Class centroids are then grouped into 2 clusters. Any clustering techniques can be used. If k-means clustering is used, distances between centroids within a group are minimized. Hence, similar classes are grouped together to represent a meta-class.

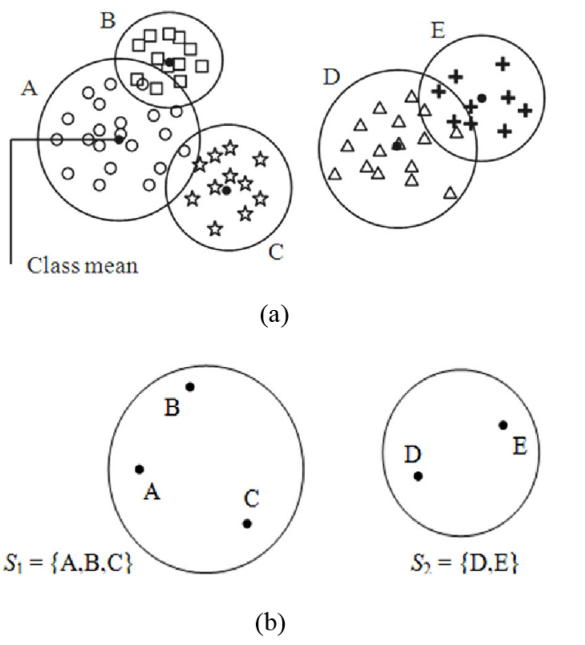


Fig.4: *Clustering class centroids by (a) calculating class means and then (c) grouping centroids together.*

3.3 Classification

A class label of a sample can be obtained by following the path of the tree, from the root to the leaf, where a class label is determined (Fig. 5). Starting at the root, an input of unlabeled data is fed to the tree. 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. In Fig. 5, examples of Class A will move to the left child (AB) of the root and then to the leaf node of Class A. [11]

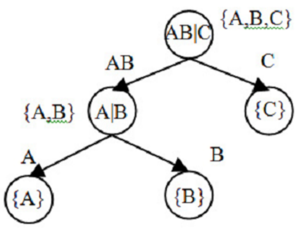
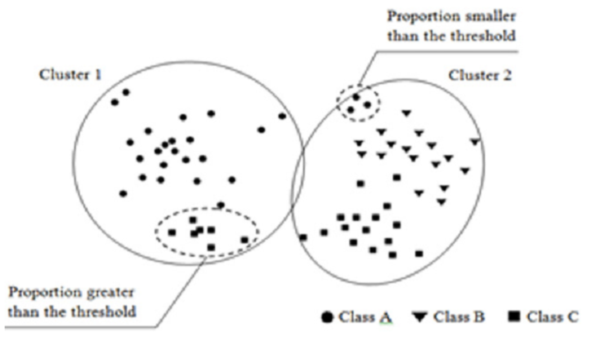


Fig.5: *An example of binary classification tree.*

4. BINARY CLASSIFICATION TREE WITH OBSERVATION-BASED CLUSTERING

In this paper, we would like to propose a different approach in splitting a binary tree. Instead of clustering class centroids, our algorithm partitions data by clustering observations, regardless of their class labels. Because the number of observations is usually much larger than the number of classes, we will use k-means as a clustering technique in our algorithm. Because we cluster observations and determine meta-classes from partitioned data points, one class is allowed to appear in both clusters as illustrated in Fig. 6. Consequently, one class will be examined in both sub-trees. This helps a tree to detect any existing sub-patterns in a class.



Proportion smaller than the threshold
Cluster 1
Cluster 2
Proportion greater than the threshold
● Class A
▼ Class B
■ Class C

Fig.6: *Clustering by observations.*

We applied a similar concept of tree construction by the class-based approach but modified the way a tree splits. We named our algorithm a binary classification tree with observation-based clustering (BCT-OB), and its algorithm is provided as follow:

```
Algorithm BUILD BCT-OB ($D, S, ths$)
INPUT:
$D = \{x_1, \ldots, x_n\}$, a data set of n observations
$S = \{y_1, \ldots, y_k\}$, a set of $k$ class labels
$ths$ = a threshold of value between 0 and 1

IF S contains only one class
    Terminate the process and return the class label in $S$
ENDIF
Partition $D$ into two clusters $D_1$ and $D_2$ with k-means algorithm

FOR each $D_i$, where $i = \{1, 2\}$
    Let $S_i$ be a set of distinct class labels $y$ corresponding to each $x$ in $D_i$
ENDFOR
Let $C_1$ be a meta-class label for $S_1$ and $C_2$ for $S_2$
Label each $x$ in $D_1$ as $C_1$ and $D_2$ as $C_2$

FOR each $y$ in $S_i$, where $i = \{1, 2\}$
    Let $p_{yi}$ = (no. of samples of class $y$ in $S_i$) / (no.
    of samples of $y$)
    IF $p_{yi} < ths$
        Delete all samples in $D_i$ labeled with class $y$
        Delete the corresponding $y_4$ from $S_i$
    ENDIF
ENDFOR
Train a binary classifier with data set $(D_1 \cup D_2)$ to
classify samples into two meta-class $\{C_1, C_2\}$

Call BUILD BCT-OB $(D_1, S_1, ths)$
Call BUILD BCT-OB $(D_2, S-2, ths)$
```

Starting at the root, k-means is used in grouping observations from the original data set D into two clusters D_1 and D_2. Unique class labels of all samples in a cluster are included in a class subset (S_1 or S_2) of the corresponding cluster. From Fig. 6, the first subset S_1 contains Class A and C because observations in the first cluster D1 are from Class A and C, and so the second subset S_2 = {A, B, C}. Next, samples in each cluster are relabeled with their respective meta-class label. All observations in D1 are relabeled as Meta-class C_1, where C_1 represents Class A and C, and observations in D2 as C_2, which represents Class A, B, and C. Then, a binary classifier is trained to classify data into two meta-classes. The process continues recursively on the node's left child and right child with the corresponding data subsets D_i and class subsets S_i. It terminates at the leaf level when only one class is left in a class subset. This label determines a class of an observation reaching this node in the classification phrase.

During splitting, a data cleaning process also takes place. After a data set is partitioned, a proportion of a class appearing in each cluster is calculated. Observations of a class in a cluster with a proportion less than a given threshold (ths) will be removed from the cluster. For example, in Fig. 6, Class A appears in both clusters, 98 percent of Class A in D_1 and the remaining two percent in D_2. If the threshold is set to 0.025, observations of Class A in D_2 will be deleted because its proportion (0.020) is smaller than ths (0.025). On the other hand, observations of Class C are allowed to be in both clusters because their proportions in D_1 and D_2 are above the threshold. This cleaning process prevents the algorithm to further learn from an insignificantly small subgroup which deviates from the majority to improve computational time (avoid nodes to split when not necessary) and avoid overfitting.

Figure 7 illustrates how a BCT-OB can be constructed. Classification started at the root with a 3-class problem. Observations were clustered into two groups, and a binary classifier was trained to differentiate between the two. The first cluster, which was examined on the left child node, contained observations from only one class, so the node did not split further.

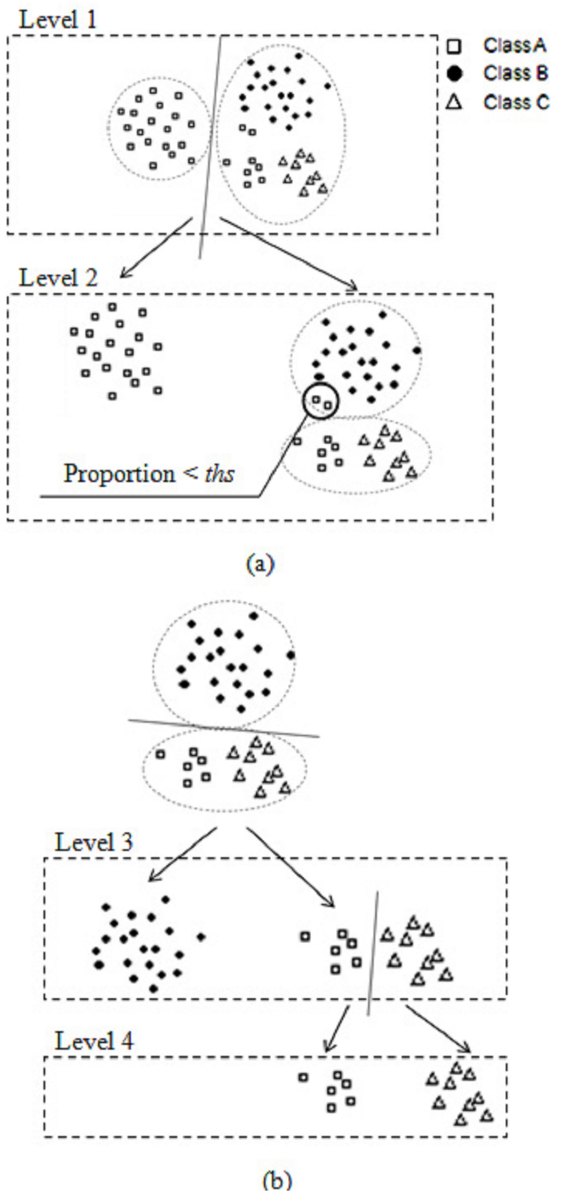


Fig. 7: *An example of BCT-OB and cleaning process. (a) An insignificant split of a class is deleted. (b) Clustering is performed on the clean data; the process continues.*

On the other hand, the right child node, which had three classes, required the algorithm to continue partitioning observations. In Fig. 7 (a), the algorithm detected a class partition of Class A having a proportion less than the threshold. Therefore, it was deleted before the algorithm performed clustering on the remaining observations again as in Fig. 7 (b). The tree construction is performed recursively until no more nodes can be split.

Regarding the complexity, our technique does not split the same way a traditional balanced binary tree does. Because our algorithm allows redundancy (one class can be in two child nodes), time complexity is expected to be higher than a balanced tree. In the worst case, the height of the tree could be very skew, O(k), where k = no. of classes. We have a threshold to avoid insignificant split, which can help decreasing an experimental runtime, but the height of the tree is still bound to the number of classes. The actual runtime also depends on clustering algorithm, unlike most traditional methods that divide classes into halves. Our goal is not to reduce the runtime, but to introduce 'redundancy' to the algorithm so we can handle data set with fuzziness.

5. EXPERIMENTS

5.1 Experimental Setup

Eight data sets from UCI Machine Learning Repository [16] were used in the experiment. Data description and thresholds used in different data set are summarized in Table 1.

In this study, BCT-OB was compared with class-based algorithms, HAH and SVM-BDT, and two balanced nested dichotomies systems, ECBND and EDBND. All techniques were implemented in R statistical package, version 2.13.1. The package 'e1071' provides an interface to SVM implementation 'libsvm' [7], which was used in constructing a base classifier. SVM's parameters were obtained by parameter tuning (the 'tune' function in 'e1071') with 10-fold cross validation. A threshold value required by the proposed algorithm is determined empirically. Selection of classes in the two meta-classes in ECBND and EDBND were determined randomly as described in the original paper [9].

Table 1: *A summary of data description.*

Data set	Sample size	No. of classes	Class distribution	*ths*
Iris	150	3	50; 50; 50	0.025
Wine	178	3	59; 71; 48	0.025
Balance-scale	625	3	288; 49; 288	0.010
Glass	214	6	70; 17; 76; 13; 9; 29	0.010
Ecoli	336	8	143; 77; 52; 35; 20; 5; 2; 2	0.025
Yeast	1,484	10	463; 429; 244; 163; 51; 44; 37; 30; 20; 5	0.010
Pendigits	10,992	10	1165; 1156; 1158; 1032; 1125; 1036; 1069; 1131; 1055; 1065	0.010
Vowel	990	11	90 each	0.025

Random sub-sampling validation was applied to evaluate classification performance on each data set by each classification tree, except for Pendigits dataset which was used as given by the source. Pendigits is composed of 7,494 samples for training and 3,498 for testing. For other data sets, each data set was divided into two parts, training and test sets. For Iris and Wine, sample sizes of training and test sets are equal. The rest was divided into 70 percent for training and 30 for testing. Ten different training and test sets were used for evaluating performance on each data set. Observations in each training and test set are determined with a simple random sampling process.

5.2 Experimental Results

Table 2 summarizes classification average accuracy (in percent) and standard deviation by BCT-OB, SVM-BDT, HAH, ECBND, and EDBND. A t-test was used to test if performance by one technique is significantly different from the others. An asterisk marked in front of the highest value for each data set indicates that it is significantly different with 95% confidence.

Table 2: *Classification performance of BCT-OB and other binary classification tree algorithms.*

Data set	BCT-OB	SVM-BDT	HAH	ECBND	EDBND
Iris	***96.93 ± 2.09**	***97.87 ± 1.80**	***96.00 ± 3.08**	86.93 ± 12.82	86.93 ± 12.82
Wine	84.79 ± 4.75	***93.37 ± 3.02**	***93.71 ± 2.07**	***91.91 ± 3.38**	***91.91 ± 3.38**
Balance-scale	81.50 ± 2.19	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	**83.56 ± 3.94**	**83.96 ± 2.91**	**84.75 ± 4.15**	**86.34 ± 3.26**	**84.46 ± 6.23**
Yeast	52.04 ± 2.22	***59.78 ± 1.96**	***60.33 ± 2.02**	58.49 ± 1.24	58.31 ± 2.03
Pendigits	***97.12 ± 0.41**	***97.02 ± 0.32**	96.63 ± 0.18	***97.00 ± 0.23**	96.62 ± 0.31
Vowel	85.32 ± 2.11	97.61 ± 1.30	***97.71 ± 0.71**	***97.64 ± 1.27**	***97.34 ± 1.34**

From the Table 2, there was no difference in performance on Iris and Ecoli by BCT-OB, SVM-BDT, and HAH, but performance by the three method was significantly higher than balanced nested dichotomies. BCT-OB outperformed the two class-based techniques in Glass and Pendigits and performed as accurately as ECBND. The average accuracy of BCT-OB was higher than SVM-BDT and ECBND in Pendigits but not statistically significant. In addition, most of the time, classification performance by ECBND and EDBND was comparable. Though, there were few cases that ECBND performs better than EDBND.

Note that variances in performance by the two balanced nested dichotomies were high in Iris and Glass data sets. For Iris, this might due to a small training sample size. For Glass, where there are 6 classes and class imbalance is present, class partitioning might have a strong influence in classification performance. Together with the randomizing process of class selection, accuracy in each run could vary greatly.

Since BCT-OB relies on how well observations are partitioned into two clusters, its performance is influenced by unsupervised clustering algorithm, which is k-means in this case, and performance of k-means also depends on the number of clusters and the initial clusters. Inappropriate choices could lead to less optimal performance. The threshold in BCT-OB also influences tree construction and classification performance. Setting the threshold too low would result in a larger tree, especially when data cannot be partitioned well, and overfitting might also occur. However, setting the threshold too high would possibly degrade classification accuracy due to information loss. The ensemble would not gain the advantage of redundancy occurs when a class can appear in more than one cluster. The experiments suggest a threshold below 0.025.

6. CONCLUSION

A binary classification tree is an ensemble of classifiers that performs hierarchical classification by breaking down an original k-class problem into two-class problems. In each internal node, classes are partitioned into two meta-classes, and there are several approaches to class partitioning. Some algorithms like balanced nested dichotomies concern the algorithm's runtime while some seek a tree structure that optimizes classification accuracy. Algorithms like HAH and SVM-BDT group similar classes together based on distances between class centroids so that difficult classification problems would be solved later in the tree.

In this study, we have proposed a new approach to tree splitting. Unlike other class-based approach, we split a tree by performing clustering on observations instead of class centroids. Thus, samples of the same class can be split, allowing sub-classes to be considered in different sub-trees. The experiment shows that our proposed BCT-OB performs comparably to other binary classification trees. There is still room for improvement, such as employing different clustering algorithm, handling overlapping clusters, and dealing with class imbalance. We would like to leave this for the future study.

References

[1] G. Ou, Y.L. Murphey, and L. Feldkamp, "Multiclass Pattern Classification Using Neural Networks," in *ICPR'04*, vol. 4, Cambridge, UK, 2004, pp. 585–588.

[2] H. Lei and V. Govindaraju, "Half-Against-Half Multi-class Support Vector Machines," in *Multiple Classifier Systems*, vol. 3541, 2005, pp. 156–164.

[3] V. Vural and J.G. Dy, "A Hierarchical Method for Multi-Class Support Vector Machines," *Proceedings of the Twenty-First International Conference on Machine Learning*, pp. 105, 2004.

[4] G. Madzarov, D. Gjorgjevikj, and I. Chorbev, "A Multi-class SVM Classifier Utilizing Binary Decision Tree," *Informatica*, vol. 33, no. 2, pp. 233–241, 2008.

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

[6] M.F.A. Hady, F. Schwenker, and G. Palm, "Multi-View Forest: A New Ensemble Method based on Dempster-Shafer Evidence Theory," *IJAMAS*, vol. 22, no. S11, pp. 2–19, 2011.

[7] C.-C. Chang and C.-J. Lin. (2001) LIBSVM: a library for support vector machines. [Online]. http://www.csie.ntu.edu.tw/ cjlin/libsvm

[8] A. Ramanan, S. Suppharangsan, and M. Niranjan, "Unbalanced Decision Trees for Multi-class Classification," in *ICIIS 2007*, Penadeniya, Sri Lanka, 2007, pp. 291–294.

[9] L. Dong, E. Frank, and S. Kramer, "Ensembles of Balanced Nested Dichotomies for Multi-Class Problems," in *9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, vol. 3721, Porto, Portugal, 2005, pp. 84–95.

[10] Y.L. Murphey, H. Wang, G. Ou, and L.A. Feldkamp, "OAHO: an Effective Algorithm for Multi-Class Learning from Imbalanced Data," in *International Joint Conference on Neural Networks*, 2007, Orlando, FL, 2007, pp. 406–411.

[11] J.-S. Lee and I.-S. Oh, "Binary Classification Trees for Multi-class Classification Problems," in *The Seventh International Conference on Document Analysis and Recognition*, vol. 2, 2003, pp. 770–774.

[12] E. Frank and S. Kramer, "Ensembles of Nested Dichotomies for Multi-Class Problems," in *ICML2004*, vol. 382, Montreal, Quebec, 2004, pp. 305–312.

[13] R. Tibshirani and T. Hastie, "Margin Trees for High-dimensional Classification," Journal of Machine Learning Research 8, pp. 637-652, 2007.

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

[15] L. Kaufman and P. Rousseeuw, "Clustering by Means of Medoids," in *Statistical Data Analysis Based on the L1-Norm and Related Methods*, Yadolah Dodge, Ed. New York, NY: Birkhauser, 1987, pp. 405-416.

[16] A. Frank and A. Asuncion. (2010) UCI Machine Learning Repository. [Online]. http://archive.ics. uci.edu/ml

Maythapolnun Athimethphat received his BSc degree in Computer Science from Assumption University in Thailand and the MSc degree in Operation Research from The National Institute of Development Administration in Thailand. Currently he has worked as a lecturer and researcher at Faculty of science and technology, Assumption University, Thailand for over 14 years. His research area included image processing, data mining, imbalance data, and multiclass classification.

Boontarika Lerteerawong achieved Bachelor degree (BS) with distinction in Applied Statistics at Assumption University in 2008. Her past researches focus on data mining, classification, and rare-class classification. Currently, she is working in the multinational corporation in the telecommunication industry.