Pattern Recognition 107 (2020) 107521

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

A linear multivariate binary decision tree classifier based on K-means splitting

Fei Wang[a], Quan Wang[a,c], Feiping Nie[b,*], Zhongheng Li[a], Weizhong Yu[a], Fuji Ren[c]

[a] National Engineering Laboratory for Visual Information Processing and Applications and Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University, Xi'an 710049, Shaanxi, PR China
[b] School of Computer Science and Center for OPTical IMagery Analysis and Learning (OPTIMAL), Northwestern Polytechnical University, Xi'an 710072, Shaanxi, PR China
[c] Faculty of Engineering, Tokushima University, Tokushima 770-8506, Japan

ARTICLE INFO

Article history:
Received 19 September 2019
Revised 15 April 2020
Accepted 23 June 2020
Available online 24 June 2020

Keywords:
Hierarchical classifier
Binary tree
Multivariate decision tree
K-means
Supervised classification

ABSTRACT

A novel linear multivariate decision tree classifier, Binary Decision Tree based on K-means Splitting (BDTKS), is presented in this paper. The unsupervised K-means clustering is recursively integrated into the binary tree, building a hierarchical classifier. The introduction of the unsupervised K-means clustering provides the powerful generalization ability for the resulting BDTKS model. Then, the good generalization ability of BDTKS ensures the classification performance. A novel non-split condition with an easy-setting hyperparameter which focuses more on minority classes of the current node is proposed and applied in the BDTKS model, avoiding ignoring the minority classes in the class imbalance cases. Furthermore, the K-means centroid based BDTKS model is converted into the hyperplane based decision tree, speeding up the process of classification. Extensive experiments on the publicly available data sets have demonstrated that the proposed BDTKS matches or outperforms the previous decision trees.

1. Introduction

Decision trees, one class of classical classification methods, have been applied into various domains such as medical diagnosis [1], [2], intrusion detection [3], and signal processing [4]. Decision trees also can be referred as hierarchical classifiers because they need multi-level discrimination to determine which class a specific pattern belongs to. They are flexibly able to tackle multi-class as well as binary classification. A relatively recent survey [5] is available, which has investigated decision trees and the related issues. Generally speaking, based on the different means of node splitting, decision trees can be categorized into univariate decision trees, multivariate decision trees, and omnivariate decision trees.

Univariate decision trees refer to those where only a single attribute (also termed as feature/variable) participates in node splitting. The most representative univariate decision trees ID3 [6], CART [7], and C4.5 [8] utilize information gain, Gini index, and gain ratio as splitting criteria (or say, attribute selection criteria) respectively, which are frequency based heuristic measures. A frequency and segment based heuristic measure is developed in [9] to replace the frequency only based heuristic measures for splitting decision tree nodes. Chandra et al. propose in [10] a new node splitting measure better than Gain Ratio and Gini Index and subsequently provide in [11] another one that includes the concept of the distinct classes. Despite of the good interpretability, the axis-parallel splitting in univariate decision trees may be inappropriate when data sets have numerically correlated variables since stair boundaries occur in this case where an oblique boundary could have separated instances better. Additionally, the selection of cut-points in these univariate decision trees involve in greedy search from the sorted candidates, which is time consuming when all the attributes are continuous.

Multivariate decision trees are those in which multiple attributes participate in node splitting. These attributes constantly construct a linear or nonlinear combination which presents a linear/oblique or nonlinear boundary at each split node in the trees. In [7], a linear multivariate version of CART is also given out in which an iterative backfitting algorithm is employed. The OC1 [12] is an extension to CART, combining the deterministic hill-climbing with randomized procedures to search for a good split. Geometric decision tree is proposed in [13] for learning oblique decision trees by finding the two clustering hyperplanes and choosing one of their angle bisectors as the split rule. In place of the binary splitting,

* Corresponding author.
E-mail addresses: wfx@mail.xjtu.edu.cn (F. Wang), bhxspring@stu.xjtu.edu.cn (Q. Wang), feipingnie@gmail.com (F. Nie), lizhongheng2010@gmail.com (Z. Li), yuwz05@xjtu.edu.cn (W. Yu), ren@is.tokushima-u.ac.jp (F. Ren).

https://doi.org/10.1016/j.patcog.2020.107521
0031-3203/

multi-way splitting is presented in LMDT [14], where an iterative algorithm is used for adjusting the parameters of branches to minimize the number of misclassifications, instead of an impurity measure such as entropy or Gini index. Several discriminant analysis based classification trees have been studied. Some are multi-way such as FACT [15] while some others are binary such as QUEST [16] and LDT [17]. Binary trees are more popular than multi-way ones because trees with low branching factor may be interpretable and better match the underlying divide-and-conquer methodology of trees [17]. In addition, Sok et al. [18] presents multivariate alternating decision trees, equipping decision trees with the boosting capability while ensuring comprehensibility. Nonlinear multivariate decision trees are more susceptible to the risks of overfitting due to the highly complex node splitting model, thus less commonly used than linear ones.

Omnivariate decision trees are decision trees in which the splitting at each node is univariate, linear multivariate, or nonlinear multivariate. The omnivariate decision tree is first proposed in [19], node splitting automatically selected among univariate, linear, or nonlinear depending on some statistical tests. Subsequently, a novel classifiability measure that captures the possible sources of misclassification is proposed for model selection at each split node, eventually forming the classifiability-based omnivariate decision trees [20]. In stead of selecting the most appropriate node splitting model, Altincay [21] propose to use the ensemble model that is composed of linear and nonlinear models for node splitting. Besides, SVMDT [22] is a hybrid SVM based decision tree for binary classification, which has univariate and multivariate (SVM) nodes.

Among the above different types of decision trees, linear multivariate decision trees are pretty promising. Firstly, linear multivariate decision trees are more flexible, not bounded to the limitation that each splitting hyperplane must be orthogonal to a attribute's axis as in univariate decision trees. Secondly, they are less sophisticated than nonlinear multivariate decision trees and omnivariate decision trees, not prone to overfitting. Therefore, this paper focuses on linear multivariate decision trees. Existing linear multivariate decision trees have applied supervised node splitting methods. However, the tree model using supervised node splitting methods such as Fisher's discriminant analysis suffers some problems while tackling multi-class classification. On one hand, if the parent node is split into as many child nodes as there are classes like in FACT [15], the learning sample will probably be depleted so rapidly that the tree is too short to reveal interesting characteristics in data. In other words, under-training is prone to occur. On the other hand, if the parent node is split into two child nodes like in QUEST [16] and LDT [17], all classes at the parent node have to be grouped into two superclasses first, which is an intractable task especially when there are too many classes. Grouping algorithms must be designed before node splitting but the resulting two superclasses are not guaranteed to be linearly separable. This weakens the significance of the supervised node splitting methods since the grouping procedure sets up some mandatory rules which will impose constraints on the following supervised node splitting.

In this paper, we develop a novel linear multivariate binary decision tree named Binary Decision Tree based on K-means Splitting (BDTKS). In this tree model, having circumvented the direct construction of the linear function with the corresponding parameters, we bring the K-means [23] clustering into the binary decision tree model to conduct node splitting in an unsupervised fashion. The concise and effective K-means clustering method that minimizes the within-cluster distances promotes the classification ability of the tree model. In fact, the author of K-means [23] has already referred in the paper to that K-means can be applied to distance-based classification trees. However, simply a brief statement, lack of a concrete algorithm description, is provided in that paper. Gaddam et. al [24] have combined K-means clustering with univariate decision tree, but their approach obviously differentiates from ours. Their approach employs only once K-means (with a predefined $K \geq 2$) and then train K ID3 decision trees to each clusters; our BDTKS repeatedly utilizes K-means at each split node in the binary decision tree. The CLM, one of methods in [25] is a little similar to ours, but it does not leverage unsupervised learning like ours during node splitting. The supervised hyperplane generation at each split node may lead to over-training. Moreover, all the methods in [25] is originally developed for the two-class problem and need a one-versus-all or one-versus-one method to extend to the multi-class problem; our BDTKS is naturally qualified for the multi-class problem.

In [26], the semi-supervised decision tree is proposed to utilize both labels and structural characteristics of data for internal node splitting in order to improve the accuracy. Kim [26] combines supervised and unsupervised information during tree model learning. BDTKS by chance shares the similar idea to some extent. As [26] have claimed, partition based on labels in traditional decision trees ignores the structural or topological homogeneity. The BDTKS model employs both class labels and data structures in the sample space. The class labels are used in the non-split condition to minimize the misclassification error while the data structures are reflected by the K-means clustering ensuring compact sample points at each child node. It is K-means splitting in our tree model that helps to reveal the structure information of data.

Some literature has already introduced clustering methods into classification framework. Qu et al. [27] adopt a visual tree of hierarchical spectral clustering for image classification. Nevertheless, rather than using the graph based spectral clustering, BDTKS introduces the concise K-means clustering into the supervised tree classifier learning, which is both effective and efficient. More recently, Gallego et al. [28] present a clustering-based k-nearest neighbor classification method. They first employ a clustering strategy concretely the K-means algorithm in their paper to reach the nearest cluster and then find the k-nearest neighbors within the cluster. Differently, BDTKS relies on the decision tree classification framework rather than k-nearest neighbor.

BDTKS contains a specially designed non-split condition which gives constraints on not only the majority class but also the minority classes of the current node to accommodate the class imbalance cases. A great deal of class-imbalance learning methods [29] have been developed. There are two most widely used classes of class-imbalance learning methods, namely, sampling including over-sampling and under-sampling, and cost-sensitive learning. The most popular over-sampling technique is the Synthetic Minority Over-sampling Technique (SMOTE) [30]. However, Drummond and Holte [31] points out that over-sampling suffers over-fitting risks and under-sampling is more preferred. Tahir et al. [32] proposes inverse random under sampling method. The idea is to severely under sample the majority class multiple times with each subsets having fewer samples than the minority class. Then multiple classifiers trained from those subsets are combined thought fusion to construct a composite boundary between the majority and the minority class. Also, to avoid large information loss in the traditional under-sampling, quintuplet sampling scheme is presented to train deep convolutional neural network in [33]. Anyway, under-sampling discards examples which may probably carry useful information for classification. Cost-sensitive learning is another technique for class-imbalance problems. Bahnsen et al. [34] proposes an example-dependent cost-sensitive decision tree algorithm and Cao and Wang [35] provides a cost-sensitive stacking ensemble method. Nevertheless, they all deal with binary classification. Actually, it is intractable for cost-sensitive learning to tackle multi-class classification especially when extremely large number of classes are included.

Our contributions includes the following: First, a new linear multivariate binary decision tree, BDTKS, is presented, enriching the multivariate decision tree methods; Second, BDTKS embeds unsupervised K-means clustering into the binary tree structure, which removes the dilemma of supervised node splitting on how many branches each split node should own for multi-class cases and also improves the generalization performance; Third, a non-split condition with an easy-setting hyperparmeter which focuses more on minority classes of the current node is proposed and applied in the BDTKS model, avoiding ignoring the minority classes in the class imbalance cases; Finally, a conversion of the K-means centroid based BDTKS model into the hyperplane based decision tree speeds up the classification process.

2. Related work review

A recent work [36] about multivariate decision trees is reviewed here due to some intrinsic connection with and certain similarities such as both using binary trees and unsupervised node splitting methods to our proposed methods. Wang et al. [36] have proposed two multivariate decision tree classifiers in the form of a full binary tree, namely the randomly partitioned multivariate decision tree (MDT1) and the Principal Component Analysis (PCA)-partitioned multivariate decision tree (MDT2).

2.1. MDT1 & MDT2

In fact, MDT1 and MDT2 share a common framework. On the whole, it is a procedure to grow a binary tree in a top-down manner. Recursive partition is conducted until there is no node remaining to be partitioned again. The root node is the one first to be partitioned, with all examples of the whole training data in it. We conclude the whole framework as follows:

Step 1: Construct a multivariate hyperplane to split the current node into the left and right child nodes.

Step 2: Determine a split condition. For each child node, if it satisfies the split condition, turn it into the node to be partitioned again; otherwise, turn it into a leaf node with the majority class as the predicted class.

Step 3: Update the current node to the next node to be partitioned again and repeat above procedure from Step 1 until there is no node remaining to be partitioned again.

Obviously, the two key issues in this framework are how to construct the multivariate hyperplane at each split node and what the split condition is like. In the following, we will explain these two issues in detail. For convenience, we discuss a single split in the tree. The split is conducted on a node with a sample matrix $\boldsymbol{X} \in \mathbb{R}^{d\times n}$ and the corresponding class label $\boldsymbol{Y} \in \mathbb{R}^{1\times n}$ known. Wherein, n and d are the number of the examples and the dimensions, respectively. Each element of $\boldsymbol{Y}$ is $y_i \in \{l_1, l_2, \ldots, l_c\}, i = 1, 2, \ldots, n$, where c is the number of classes.

2.1.1. Splitting via hyperplane

In order to complete node splitting, both MDT1 and MDT2 employ a hyperplane in the form of

$$\boldsymbol{w}\cdot\boldsymbol{x} - p = 0, \tag{1}$$

where $\boldsymbol{w}$ is the normal direction of the hyperplane, $\boldsymbol{x}$ is the sample point coming from $\boldsymbol{X}$ and p is the cut-point. Node splitting is accomplished by partitioning the sample points into the two child nodes according to:

$$\begin{cases} \boldsymbol{x} \in \boldsymbol{X}_1, & if \quad \boldsymbol{w}\cdot\boldsymbol{x} > p \\ \boldsymbol{x} \in \boldsymbol{X}_2, & if \quad \boldsymbol{w}\cdot\boldsymbol{x} \le p, \end{cases} \tag{2}$$

where $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ are the sample subsets corresponding to the left and the right child nodes, respectively. Fig. 1 gives an illustration of node splitting via a hyperplane in the tree.

It is clear that, two parameters in Eq. (1), $\boldsymbol{w}$ and p, uniquely determine a hyperplane. MDT1 and MDT2 utilize the same way to determine the cut-point p. Given a certain $\boldsymbol{w}$, all the sample points in the sample matrix $\boldsymbol{X}$ are projected onto the normal direction $\boldsymbol{w}$: $\{\boldsymbol{x}_i, i = 1, 2, \ldots, n\} \to \{P(\boldsymbol{x}_i) = \boldsymbol{w}\cdot\boldsymbol{x}_i, i = 1, 2, \ldots, n\}$, thus generating n projections. Then, the median value of the projections is taken to act as the cut-point:

$$p = median\{P(\boldsymbol{x}_i)|\boldsymbol{x}_i \in \boldsymbol{X}, i = i_1, i_2, \ldots, n\}. \tag{3}$$

The different strategies used for obtaining the normal direction $\boldsymbol{w}$ of the hyperplane make MDT1 and MDT2 different from each other. MDT1 randomly generates a vector $\boldsymbol{v} = [v_1, v_2, \ldots, v_d]^T \in \mathbb{R}^{d\times 1}$, having the same dimension as the sample point $\boldsymbol{x}$. Every element in the vector $\boldsymbol{v}$ is a random number in the range $[-1, 1]$. Then, MDT1 utilizes the unitized $\boldsymbol{v}$ to act as the normal direction $\boldsymbol{w}$:

$$\boldsymbol{w} = \frac{\boldsymbol{v}}{||\boldsymbol{v}||} = \frac{\boldsymbol{v}}{\sqrt{v_1^2 + v_2^2 + \ldots + v_d^2}}. \tag{4}$$

Instead of random generation, MDT2 present a heuristic strategy to determine the normal direction $\boldsymbol{w}$. This strategy is to select the largest principal component of PCA on the sample matrix $\boldsymbol{X}$ to serve as the normal direction. Given $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]$, a centralized sample matrix is obtained by subtracting the mean of all the sample points: $\tilde{\boldsymbol{X}} = [\boldsymbol{x}_1 - \boldsymbol{m}, \boldsymbol{x}_2 - \boldsymbol{m}, \ldots, \boldsymbol{x}_n - \boldsymbol{m}]$, where $\boldsymbol{m} = \frac{1}{n}(\boldsymbol{x}_1 + \boldsymbol{x}_2 + \ldots + \boldsymbol{x}_n)$. Then, the eigenvalue decomposition $\boldsymbol{S}\boldsymbol{\xi} = \lambda\boldsymbol{\xi}$ on the covariance matrix $\boldsymbol{S} = \tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^T$ is performed, receiving d eigenvalues $\lambda_1 > \lambda_2 > \ldots > \lambda_d$ and their corresponding eigenvectors $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_d$. MDT2 chooses the eigenvector corresponding to the largest eigenvalue of $\boldsymbol{S}$ as the normal direction:

$$\boldsymbol{w} = \boldsymbol{\xi}_1. \tag{5}$$

This method of obtaining the normal direction $\boldsymbol{w}$ ensures that the variance of the sample set on the direction $\boldsymbol{w}$ is at its maximum. This means that the partition is performed according to the orientation on which the degree of distinction of the instances in the sample set is at its largest.

2.1.2. Split condition

For the resulting child nodes after each split, a split condition is applied to decide whether they require further split. Specifically, further split will be performed if the child node satisfies the split condition. This split condition is defined as follows:

$$purity(\boldsymbol{R}) = \max\{\frac{n_i}{r}|i = l_1, l_2, \ldots, l_c\} < \delta, \tag{6}$$

where $\boldsymbol{R}$ denotes $\boldsymbol{X}_1$ or$\boldsymbol{X}_2$, r the number of the sample points in sample set $\boldsymbol{R}$, n_i the number of the sample points belonging to the class i, and δ a predefined threshold in the range $(\frac{1}{c}, 1]$. By the way, if a certain child node does not satisfy the split condition, it will become a leaf node assigned the class label $l^* = \arg\max_i\{\frac{n_i}{r}|i = l_1, l_2, \ldots, l_c\}$.

2.1.3. Classification

Once there is no node remaining to be split any more, a multivariate decision tree model (MDT1 or MDT2) has been completely constructed, with the split nodes storing $\boldsymbol{w}$ and p and the leaf nodes storing $l^* \in \{l_1, l_2, \ldots, l_c\}$. An illustration of the multivariate decision tree model (MDT1 or MDT2) is presented in Fig. 2.

Classification can be conducted based on the resulting multivariate decision tree model. For a data point with unknown class, the target is to find which class it should belong to. The data point is first put onto the root node of the tree model, and reaches the next node according to which side the data point lies in of the hyperplane determined by $\boldsymbol{w}$ and p at the split node. It constantly goes though the split nodes and finally reaches a leaf node where l^* indicates its class.

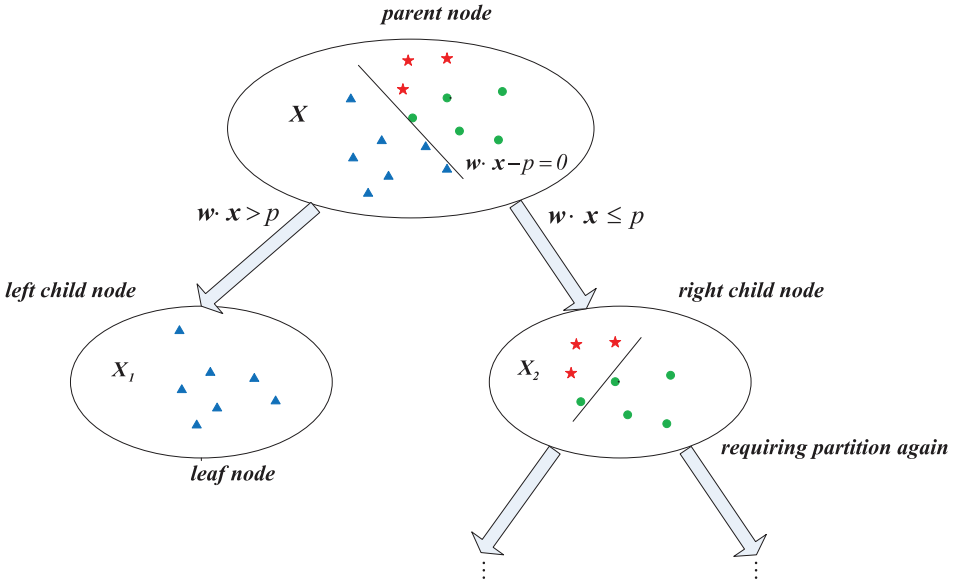

Fig. 1. Node splitting via hyperplane in the tree.

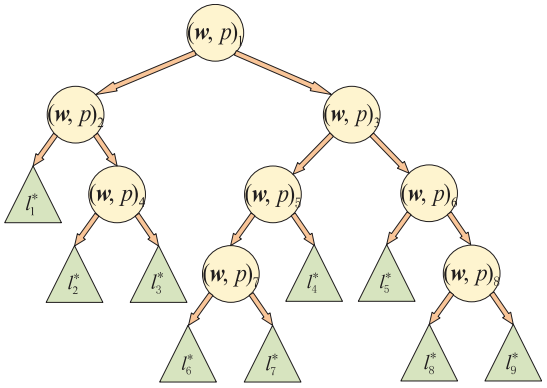

Fig. 2. An illustration of the multivariate decision tree model (MDT1 or MDT2).

3. The proposed methods

In this section, we present a new linear multivariate binary decision tree. The primary difference from MDT1 and MDT2 is we conduct splitting using K-means. In fact, all the linear multivariate binary decision trees separate the examples at the split node by looking for an appropriate hyperplane. It is NP-hard to find the hyperplane that best partitions the examples. Therefore, taking advantage of K-means provides an alternative scheme. Besides, in order to avoid ignoring minority classes, a novel non-split condition which focuses more on minority classes is proposed in place of the majority class proportion based split condition used in MDT1 and MDT2. All these changes prompt our new model termed as BDTKS. We exhibit the general procedure of building the BDTKS model in Algorithm 1 and an illustration of the resulting BDTKS model for a simple three-class data set in Fig. 3.

It is known clearly from Algorithm 1 and the illustration in Fig. 3 that the resulting BDTKS model is a binary tree with each split node storing the centroids $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ for the corresponding child nodes and each leaf node storing the assigned class label l^*. Given a data point $\boldsymbol{x}$ with unknown class, it is directly put onto the root node of the BDTKS model and is led to the child node whose centroid is $\boldsymbol{c}_{i_*}$ ($i_* = \arg\min_i ||\boldsymbol{x} - \boldsymbol{c}_i||^2, i = 1, 2$). The given data point is repeatedly led to the subsequent child node whose centroid is the centroid to which the data point is closer than the other centroid until it reaches a leaf node assigned to l^*, and it is classified to class l^*.

Algorithm 1 BDTKS.

Input: Training sample matrix $\boldsymbol{D} \in \mathbb{R}^{d\times N}$ with known class label $\boldsymbol{L} \in \mathbb{R}^{1\times N}$.

Output: A binary tree, the split nodes storing the K-means centroids $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ and the leaf nodes storing the assigned class label l^*.

```
Initialize Ω as an empty set.
View the node with the training sample matrix D as the root node and add it to Ω.
while Ω is not empty do
    Select a node from Ω, of which the sample data matrix is denoted by X ∈ R^{d×n} and the corresponding class label by Y ∈ R^{1×n}.
    Partition X into X_1 and X_2 using K-means, the node split into the left and the right child nodes. Store the two centroids c_1 and c_2 of clusters.
    for i = 1, 2 do
        if non-split condition w.r.t X_i and Y_i holds then
            Turn the child node with X_i into a leaf node, and assign it the majority class label l*.
        else
            Add the child node with X_i to Ω.
        end if
    end for
    Remove the node with X from Ω.
end while
```

Algorithm 1 has roughly sketched out the process of forming the BDTKS model from the overall perspective. The following will introduce some detailed issues, such as how to split using K-means and how to define the non-split condition.

3.1. Splitting using K-means

Although MDT1 and MDT2 have already adopted unsupervised learning methods for node splitting, they do not learn enough information and structure that benefits properly partitioning data.

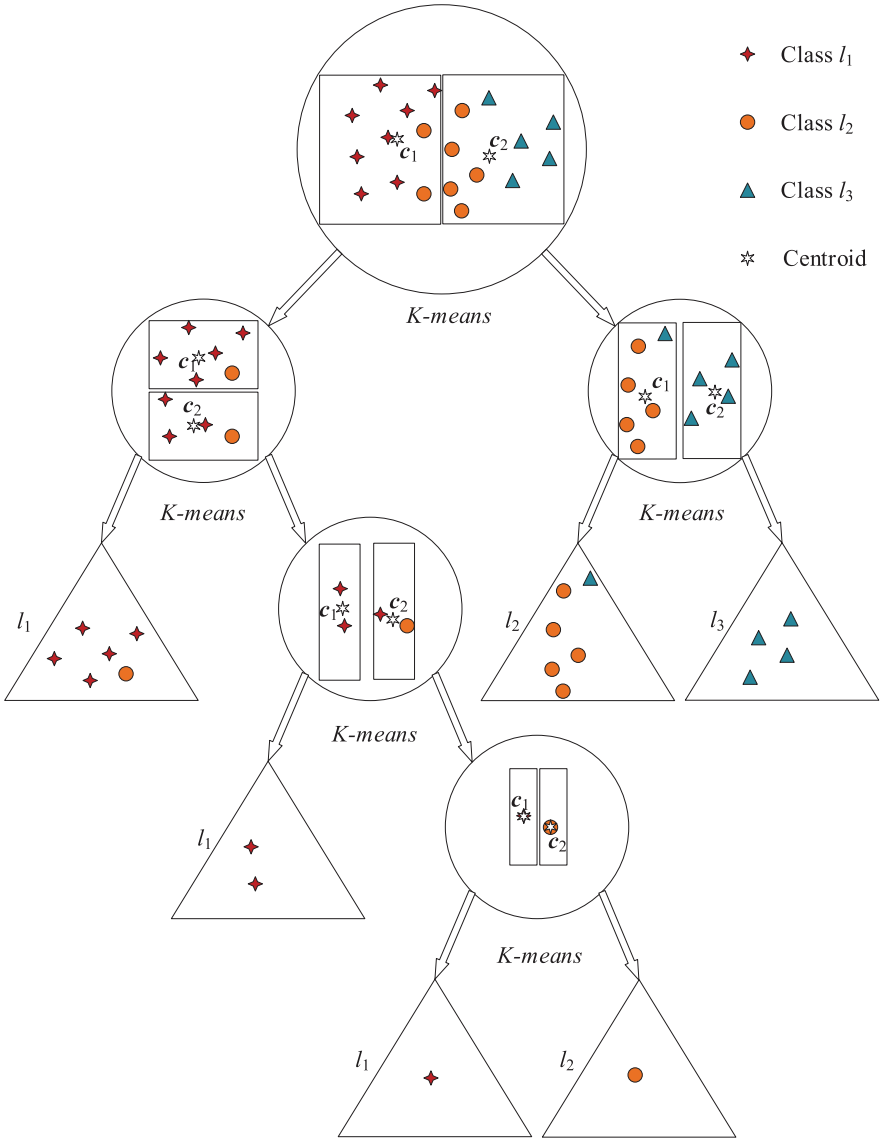

Fig. 3. An illustration of the BDTKS model for a simple three-class data set.

Randomly generating the normal direction of the hyperplane in MDT1 does not have any guidance and heuristic to data partition, lack of inductive ability; and using the PCA largest principal component as the normal direction of the hyperplane in MDT2 only considers the distribution of data in the direction with the largest variance. The imperfection of MDT1 and MDT2 prompts us to seek another strategy to deal with the node splitting problem.

Rather than directly constructing a hyperplane at each split node as in MDT1 and MDT2, BDTKS takes advantage of K-means to complete the splitting task at the split node. BDTKS masterly borrows the unsupervised clustering method K-means with $K = 2$ to partition the sample points into two parts, thus completing the task of node splitting naturally. The K-means method is repeatedly utilized to constantly separate the subsets of the original training sample at the root node into two superclasses from top to down until all the reaching nodes stand for a single class. K-means clustering inherently enables the within-cluster points to be compact, which is consistent with the requirement of decision node splitting that points with similarity are to be partitioned to the same part. What's more, K-means clustering partitions data according to the distribution in the whole data space rather than that at the special largest principal component direction as in MDT2, which covers more comprehensive data information. K-means clustering discovers the intrinsic laws of data without excessive restrictions or interventions. Therefore, compared with node splitting strategies in MDT1 and MDT2, K-means clustering automatically and naturally learns more information and structure that will help the tree model exhibit more powerful generalization ability. Furthermore, the clustering centroids of K-means can naturally guide the new example to the predicted label in the classification stage.

As is well-known, K-means can be formulated as the optimization problem $\min_{C_1,C_2,\ldots,C_K} \sum_{i=1}^{K} \sum_{\boldsymbol{x}\in C_i} ||\boldsymbol{x} - \boldsymbol{c}_i||_2^2$, where $\boldsymbol{c}_i$ is the mean of the vectors in the cluster represented by C_i. The common practice to solve this problem is the EM algorithm which requires the cluster center initialization. Due to two different ways to specify the cluster center initialization, two different versions of BDTKS model are presented, which are termed as BDTKS.v1 and BDTKS.v2.

3.1.1. BDTKS.v1

BDTKS.v1 utilizes the K-means++ algorithm [37] for the cluster center initialization when performing K-means at each split in the

model. Consider a split at a certain node with the sample matrix $\boldsymbol{X}$. Denote the two initial cluster centers by $\boldsymbol{c}_1^{(0)}$ and $\boldsymbol{c}_2^{(0)}$. In BDTKS.v1, we perform K-means with the cluster center initialization method described in Algorithm 2.

Algorithm 2 Initialization method of K-means in BDTKS.v1.

Input: Sample matrix $\boldsymbol{X} \in \mathbb{R}^{d\times n}$.
Output: $\boldsymbol{c}_1^{(0)}$ and $\boldsymbol{c}_2^{(0)}$.

1: Choose an initial center $\boldsymbol{c}_1^{(0)}$ uniformly at random from $\boldsymbol{X}$.
2: Choose the second center $\boldsymbol{c}_2^{(0)} = \boldsymbol{x}' \in \boldsymbol{X}$ at random with the probability $\frac{d(\boldsymbol{x}')^2}{\sum_{\boldsymbol{x}\in\boldsymbol{X}} d(\boldsymbol{x})^2}$ where $d(\boldsymbol{x})$ denotes the distance from the data point $\boldsymbol{x}$ to $\boldsymbol{c}_1^{(0)}$.

Note that this method of initializing the cluster centers introduces the randomness, or say, the uncertainty. This directly leads to different BDTKS models in different runs.

3.1.2. BDTKS.v2

In order to circumvent the uncertainty, BDTKS.v2 employs a deterministic method to initialize the cluster center when performing K-means at each split. The details about this initialization method is presented in Algorithm 3.

Algorithm 3 Initialization method of K-means in BDTKS.v2.

Input: Sample matrix $\boldsymbol{X} \in \mathbb{R}^{d\times n}$.
Output: $\boldsymbol{c}_1^{(0)}$ and $\boldsymbol{c}_2^{(0)}$.

1: Generate a hyperplane in the form of Eq.~(1) based on Eq.~(3) and Eq.~(5).
2: Divide data points in $\boldsymbol{X}$ into two parts $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ according to Eq.~(2).
3: $\boldsymbol{c}_i^{(0)} = \frac{1}{|\boldsymbol{X}_i|}\sum_{\boldsymbol{x}\in\boldsymbol{X}_i} \boldsymbol{x}, i = 1, 2.$

This method is heuristic, meaning that this initialization method stimulates the initial cluster centers closer to the real ones. The reason is that the largest principal component produced by PCA represents the direction with the largest variance in $\boldsymbol{X}$ and the means of the two parts divided according to Eq. (2) with high probability are close to the centers of the two final clusters.

3.2. Non-split condition

In classification trees, the split condition is a crucial factor to impact the final tree model and thus classification performance. Although the majority class proportion based split condition used in MDT1 and MDT2 works, there exists a shortcoming that in class imbalance cases the classification performance will degrade sharply if the hyperparameter δ is not well adjusted. That is because it only focuses on the majority class, without explicit constraints on the minority classes. In the class imbalance cases, a relatively small δ does not impose any constraint on minority classes, causing the tree model to label all leaf nodes as the majority class. In order to overcome this shortcoming, a novel non-split condition with an easy-setting hyperparameter λ is proposed. The proportions of classes in the original training sample are utilized to constrain the purity of the sample at leaf nodes. Instead of only concerning the proportion of the majority class as in MDT1 and MDT2, the proposed non-split condition also concerns the proportions of the minority classes at the current node. Actually, in the proposed non-split condition, a stricter constraint is imposed on the proportions of the minority classes than the majority class to accommodate class imbalance cases. In order to expound the proposed non-split condition better, some definitions are given first and then the formula of the non-split condition is presented.

3.2.1. Proposed definitions

Let N denote the total number of the training examples at the root node to learn the tree model. N_i denotes the number of examples belonging to class i in the training sample, herein $i \in \{l_1, l_2, \ldots, l_c\}$ where $l_1, l_2, \ldots, l_c$ are the class labels of c classes. Assuming that a tree model has been obtained, there are m leaf nodes with their individual sample matrices $\boldsymbol{X}^{(l)}, l = 1, 2, \ldots, m$, with m_k leaf nodes assigned to the label $l_k, k = 1, 2, \ldots, c$ such that $\sum_{k=1}^{c} m_k = m$. For a certain child node with the sample set $\boldsymbol{X}^{(t)}$ of $n^{(t)}$ examples, there are $n_i^{(t)}$ examples belonging to class i in it.

Definition 1. *(Model Precision Ratio)* In the tree model, the model precision ratio of the assigned class i is defined as:

$$MP_i^i = \sum_{\boldsymbol{X}^{(t)} is assigned to i} n_i^{(t)} / \sum_{\boldsymbol{X}^{(t)} is assigned to i} n^{(t)} \tag{7}$$

Definition 2. *(Model Recall Ratio)* In the tree model, the model recall ratio of the real class i on the assigned class i is defined as:

$$MR_i^i = \sum_{\boldsymbol{X}^{(t)} is assigned to i} n_i^{(t)} / N_i \tag{8}$$

Definition 3. *(Model Fallout Ratio)* In the tree model, the model fallout ratio of the assigned class i on the real class j, $j \neq i$ is defined as:

$$MF_j^i = \sum_{\boldsymbol{X}^{(t)} is assigned to i} n_j^{(t)} / \sum_{\boldsymbol{X}^{(t)} is assigned to i} n^{(t)} \tag{9}$$

Definition 4. *(Model Omission Ratio)* In the tree model, the model omission ratio of the real class j, $j \neq i$ on the assigned class i is defined as:

$$MO_j^i = \sum_{\boldsymbol{X}^{(t)} is assigned to i} n_j^{(t)} / N_j \tag{10}$$

Definition 5. *(Model Local Precision Ratio)* In the tree model, the model local precision ratio on a certain leaf node with $\boldsymbol{X}^{(t)}$ which is assigned to class i is defined as:

$$MLP_i^{(t)} = n_i^{(t)} / n^{(t)} \tag{11}$$

Definition 6. *(Model Local Recall Ratio)* In the tree model, the model local recall ratio of the real class i on a certain leaf node with $\boldsymbol{X}^{(t)}$ which is assigned to class i is defined as:

$$MLR_i^{(t)} = n_i^{(t)} / N_i \tag{12}$$

Definition 7. *(Model Local Fallout Ratio)* In the tree model, the model local fallout ratio on a certain leaf node with $\boldsymbol{X}^{(t)}$ which is assigned to class i whose real class label is j, $j \neq i$ is defined as:

$$MLF_j^{(t)} = n_j^{(t)} / n^{(t)} \tag{13}$$

Definition 8. *(Model Local Omission Ratio)* In the tree model, the model local omission ratio of the real class j, $j \neq i$ on a certain leaf node with $\boldsymbol{X}^{(t)}$ which is assigned to class i is defined as:

$$MLO_j^{(t)} = n_j^{(t)} / N_j \tag{14}$$

Some theorems based on the above definitions are presented in the supplementary materials to reveal some relationships between them.

3.2.2. Non-split formula

Let $p_i^{(r)} = N_i/N$ denote the proportion of class i in the training sample. $p_i^{(t)} = n_i^{(t)}/n^{(t)}$ denotes the proportion of class i in the sample set $\boldsymbol{X}^{(t)}$ at a certain child node. The proposed non-split condition at each child node with $\boldsymbol{X}^{(t)}(t = 1, 2, \ldots, m)$ is defined as follows:

$$\begin{cases} p_{l_{major}}^{(t)} > p_{l_{major}}^{(r)} \\ p_j^{(t)} < \lambda \cdot p_j^{(r)} \cdot \frac{N}{(n^{(t)})^2} \, for j \neq l_{major}, j = l_1, l_2, \ldots, l_c, \end{cases} \tag{15}$$

where l_{major} is the label of the majority class in $\boldsymbol{X}^{(t)}$ and λ is a hyperparameter which theoretically can take any value in the range of the open interval $(0, +\infty)$. However, if $\lambda \to +\infty$, Eq. (15) degrades into $p_{l_{major}}^{(t)} > p_{l_{major}}^{(r)}$, which goes against the expectation that a stricter constraint should be imposed on the minority classes. What we expect is that $p_j^{(t)}$ is at least lower than $p_j^{(r)}$. However, if $\lambda \geq N$, $\lambda \cdot \frac{N}{(n^{(t)})^2} > 1$. So, we assume $\lambda < N$ although λ should be lower than $\frac{(n^{(t)})^2}{N}$ which is smaller than N to satisfy $\lambda \cdot \frac{N}{(n^{(t)})^2} < 1$.

First, an intuitive understanding of the non-split condition in Eq. (15) is presented here. Once the non-split condition holds, the corresponding child node turns into a leaf node in the tree model. So, this non-split condition means that the proportion of the majority class in the sample data at each leaf node must be larger than the proportion of this class in the training sample data. At the same time, the proportions of those minority classes must be smaller than a certain percentage. Therefore, the purity of each leaf node is always at a relatively high level. What's more, the part of the non-split condition with regard to those minority classes is automatically elastic. If the considered node has a relatively larger number of examples ($n^{(t)}$ is relatively larger), the proportion of those minority classes $p_j^{(t)}$ needs to be lower than a relatively smaller threshold $\lambda \cdot p_j^{(r)} \cdot \frac{N}{(n^{(t)})^2}$ to satisfy the non-split condition; and vice versa. Therefore, the formula can promote to form the tree model quickly when the considered node has a relatively smaller number of examples. Actually, it is reasonable to regard them as a same class since if they are still together after so many times of previous clustering partition they may be really too similar to be partitioned away.

Then, the non-split condition is expounded again from the perspective of precision and recall. It is not difficult to find that, in the non-split condition (Eq. (15)), the proportion $p_{l_{major}}^{(t)}$ of the majority class in the sample set of $\boldsymbol{X}^{(t)}$ is exactly the model local precision ratio $MLP_{l_{major}}^{(t)}$ and $p_j^{(t)}$ is exactly the model local omission ratio $MLF_j^{(t)}$. So, Eq. (15) can be rewritten into:

$$\begin{cases} MLP_{l_{major}}^{(t)} > N_{l_{major}}/N \\ MLF_j^{(t)} < \lambda \cdot N_j/(n^{(t)})^2 \, for j \neq l_{major}, j = l_1, l_2, \ldots, l_c. \end{cases} \tag{16}$$

Consider the leaf nodes with $\boldsymbol{X}^{(t)}(t = a_1, a_2, \ldots, a_{m_k})$ which are all the leaf nodes assigned to $l_{major} = l_k$ in the tree model. Then, according to Theorems 1 and 3 in the supplementary materials, it can be derived that

$$\begin{cases} MP_{l_k}^{(t)} > N_{l_k}/N \\ MF_j^{l_k} < \lambda \cdot N_j \sum_{t=a_1}^{a_{m_k}} \frac{1}{n^{(t)}} / \sum_{t=a_1}^{a_{m_k}} n^{(t)}, j \neq l_k, j = l_1, l_2, \ldots, l_c. \end{cases}$$

For all the $k = 1, 2, \ldots, c$, this conclusion can be reached. This means that the proposed non-split condition makes the model precision ratio higher than a certain value and the model fallout ratio lower than a certain value. In addition, Eq. (15) can be transformed into:

$$\begin{cases} MLR_{l_{major}}^{(t)} > n^{(t)}/N \\ MLO_j^{(t)} < \lambda/n^{(t)} \, for j \neq l_{major}, j = l_1, l_2, \ldots, l_c. \end{cases} \tag{17}$$

According to Theorems 2 and 4 in supplementary materials it can be derived that

$$\begin{cases} MR_{l_k}^{(t)} > \frac{1}{N} \sum_{t=a_1}^{a_{m_k}} n^{(t)} \\ MO_j^{l_k} < \lambda \cdot \sum_{t=a_1}^{a_{m_k}} \frac{1}{n^{(t)}}, j \neq l_k, j = l_1, l_2, \ldots, l_c. \end{cases}$$

This means that the proposed non-split condition makes the model recall ratio higher than a certain value and the model omission ratio lower than a certain value. The unsupervised K-means clustering makes the BDTKS tree model own powerful generalization ability. Therefore, the model precision ratio and the model recall ratio reflect to somewhat extent the precision ratio and the recall ratio of classification. The relatively high values on both precision and recall ratio of classification will correspond to relatively high classification accuracy.

The hyperparameter λ is easy to set. A larger range of effective values offers λ more flexible choice. λ can take values on the interval (0,1] but also those larger than 1. Generally speaking, those relatively high λ values still bring good classification performance.

3.3. Acceleration

Since two distances to the centroids $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ at each split node in the BDTKS model must be calculated, the classification time may be longer than the hyperplane based multivariate decision trees which only compute a projection onto the normal direction. To improve classification speed, the BDTKS model is further modified.

In the training stage, once K-means is completed at a split node, a hyperplane is generated based on the two centroids $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. The normal direction of the hyperplane is obtained as follows:

$$\boldsymbol{w} = \frac{\boldsymbol{c}_2 - \boldsymbol{c}_1}{||\boldsymbol{c}_2 - \boldsymbol{c}_1||}. \tag{18}$$

Besides, the cut-point of the hyperplane is obtained in the following way:

$$p = \boldsymbol{w} \cdot \frac{\boldsymbol{c}_1 + \boldsymbol{c}_2}{2}. \tag{19}$$

In this way, the centroid based BDTKS model is transformed into the hyperplane based model successfully. Since the hyperplane based BDTKS model accelerates the classification speed, it is termed as Accelerated BDTKS (A-BDTKS). A sketch is given in Fig. 4 to show how to generate the hyperplane $\boldsymbol{w} \cdot \boldsymbol{x} - p = 0$ based on the centroids $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ at a split node.

In the classification stage, the same practice as in MDT1 and MDT2 is completely applicable for A-BDTKS.

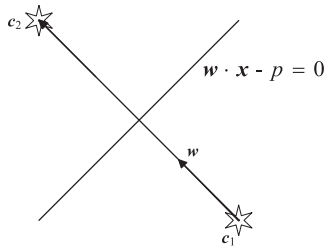

Fig. 4. A sketch of generating the hyperplane from the two centroids at a split node.

Table 1
Characteristics of data sets used in experiments.

Dataset	#training	#testing	#dimensions	#classes	#largest class in training	#smallest class in training
acoustic	78797	19699	50	3	39355	18263
aloi	83961	20990	128	1000	98	28
combined	78791	19698	100	3	39401	18363
covtype	464810	116202	54	7	227033	2165
gisette	5600	1400	5000	2	2805	2795
ijcnn1	113353	28338	22	2	102443	10910
letter	14934	3734	16	26	620	419
madelon	1050	262	500	2	534	516
mushrooms	2919	730	111	2	2512	407
pendigits	8794	2198	16	10	930	819
phishing	809	202	67	2	449	360
protein	19510	4877	357	3	9046	4185
satimage	5148	1287	36	6	1235	491
segment	894	224	19	7	134	121
seismic	78795	19699	50	3	39487	18306
shuttle	46400	11600	9	7	36433	7
skin-nonskin	41146	10287	3	2	29437	11709
usps	7438	1860	256	10	1248	576
w8a	25528	6382	300	2	24762	766

4. Experiments

In this section, extensive experiments are conducted to demonstrate the superiority of our proposed BDTKS model. All experiments are run in *MATLAB*9.2.0(*R*2017*a*) on the computer with an $Intel(R)Core(TM)i7-8700CPU@3.20GHz$ processor with 64.0*GB* memory. All the 19 data sets used for these experiments are publicly available from the web[1]. The original data sets from the web is processed by first eliminating the repeated examples and then randomly choosing 80% for training and the rest 20% for testing/classification. The characteristics of the finally used data sets are depicted in Table 1.

4.1. Experimental design

We have implemented BDTKS.v1, BDTKS.v2, and A-BDTKS.v2. Because of the uncertainty of BDTKS.v1, we do not implement A-BDTKS.v1. The comparative methods contain CART [7], C4.5 [8], MDT1 [36], MDT2 [36], λ-MDT1, λ-MDT2, δ-BDTKS.v1, and δ-BDTKS.v2. λ-MDT1 and λ-MDT2 are variants of MDT1 and MDT2 modified by replacing the majority class proportion based split condition with the proposed non-split condition. δ-BDTKS.v1 and δ-BDTKS.v2 are varied from BDTKS.v1 and BDTKS.v2 by using the majority class proportion based split condition instead of the non-split condition.

We directly use the MATLAB built-in function *fitctree*() to implement CART without pruning and let all other settings be default. For C4.5, we do not split nodes whose sample size is lower than 5% of the training sample size. These two methods finish their training procedure without any validation set. For all the methods containing the parameter δ or λ, we have adjusted their parameter (δ or λ) to take 10 different values. For δ, these values are obtained by taking 10 values at equal intervals on the interval $(\frac{1}{c}, 1]$. For λ, these values are fixed to take values from $\{10^{-6}, 10^{-5}, \ldots, 10^{3}\}$ for all different data sets. To select the best parameter value, we randomly select 20% examples from the training sample to act as the validation set and the rest as the learning set. The parameter value enabling the accuracy to be the largest on the learning set is taken as the best parameter value and is again used to learn the final model on the training sample. For those methods with uncertainty, the process of learning on the learning set as well as the training sample is repeatedly run 5 times. We record some criteria such as the classification accuracy, the macro-F_1 score, the macro-average precision and recall, the training and the classification time, and the number of splitting in the tree model. For those methods with uncertainty, the mean and the standard deviation of the corresponding criteria on 5 runs are recorded. For A-BDTKS.v2, just the classification time is recorded because that is the only notable difference from BDTKS.v2.

[1] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/ .

4.2. Classification performance

First of all, we compare our proposed BDTKS.v1 and BDTKS.v2 with C4.5, CART, MDT1, and MDT2 on classification accuracy. Table 2 shows the comparison of these methods on classification accuracy. It can be observed that for all the data sets except for "phishing" our methods remain quite high accuracy which is much higher than or close to MDT2 (the better one among MDT1 and MDT2). In fact, for the data set "phishing", since the training sample size is relatively small (809 as shown in Table 1), the hold-out validation method is unfavourable to the parameter selection, meaning that a parameter value with a poor performance is probably taken as the best parameter value. From Table 2, it seems that on the whole our BDTKS (BDTKS.v1 or BDTKS.v2) and CART have advantages over each other on a approximately peer-to-peer number of data sets. However, there are 5 data sets for which our BDTKS is two more percent higher than CART but only 1 data sets ("phishing") for which CART is two more percent higher than BDTKS. Clearly, BDTKS.v1 or BDTKS.v2 has overwhelming advantages over or matches C4.5 on all the data sets. Actually, most of the data sets for which C4.5 and CART can perform better than our BDTKS have the characteristic that all or part of the attributes are discrete. Specially, for the data sets "mushrooms" and "w8a", all the values of any attribute are either 0 or 1, and there are only 3 distinct attribute values among the whole "phishing" data sets. Therefore, the univariate decision trees C4.5 and CART still have some advantages in dealing with data sets owning discrete attributes. This may owe to No Free Lunch Theorem. That is to say, there is no method which can beat all others in all cases. In addition, it can be discovered that C4.5 is very clumsy when dealing with multi-class data sets such as "aloi", "covtype", "pendigits", and "usps". But our BDTKS can tackle multi-class problems well.

In order to manifest the success of applying K-means to the binary tree, we also specially make comparisons among the original methods (MDT2 and BDTKS.v2) and the modified ones. The macro-F_1 score of all these methods are presented in Table 3. It can be seen that if the condition to judge whether split or not is the same,

Table 2
Classification accuracy of BDTKS.v1 and BDTKS.v2 compared with that of C4.5, CART, MDT1, and MDT2. ("↑" represents that the classification accuracy of BDTKS.v1 or BDTKS.v2 is higher than both MDT1 and MDT2. "*" represents that classification accuracy of this corresponding univariate or multivariate decision tree is two more percent higher than all listed multivariate (MDT1, MDT2, BDTKS.v1, and BDTKS.v2) or univariate (C4.5 and CART) decision trees.).

ACC(%)	C4.5	CART	MDT1	MDT2	BDTKS.v1	BDTKS.v2
acoustic	69.85	68.61	65.78 ± 0.54	70.33	**71.39** ± 0.22↑	**71.39**↑
aloi	3.59	76.01	47.99 ± 0.34	78.29 *	**85.86** ± 0.22↑*	85.64↑*
combined	77.02	76.58	73.27 ± 0.42	76.38	80.22 ± 0.13↑*	**80.30**↑*
covtype	40.96	**93.37**	85.82 ± 0.13	90.07	90.87 ± 0.04↑	91.59↑
gisette	92.14	**93.57**	61.76 ± 2.59	87.79	91.27 ± 0.64↑	92.93↑
ijcnn1	94.75	**97.96**	93.98 ± 0.18	96.43	97.74 ± 0.04↑	97.65↑
letter	22.82	83.74	58.50 ± 0.78	79.24	**86.36** ± 0.60↑*	86.31↑*
madelon	58.02	72.52	53.44 ± 4.03	67.94	63.28 ± 3.15	**76.72**↑
mushrooms	**100.00**	**100.00**	94.66 ± 0.79	99.59	**100.00** ± 0.00↑	99.59
pendigits	78.03	95.77	88.50 ± 0.88	96.82	**98.10** ± 0.20↑	97.73↑
phishing	90.10	**91.09** *	68.71 ± 3.69	83.17	86.73 ± 2.02↑	83.66↑
protein	52.08	51.36	46.00 ± 0.55	48.94	52.29 ± 0.35↑	**52.59**↑
satimage	82.13	86.79	80.03 ± 0.53	88.66	89.42 ± 0.62↑*	**89.90**↑*
segment	91.96	92.41	81.88 ± 3.58	87.95	**92.95** ± 0.73↑	91.52↑
seismic	**72.72**	66.92	63.13 ± 3.72	69.21	71.39 ± 0.12↑	71.55↑
shuttle	99.68	**99.98**	99.84 ± 0.02	99.91	99.92 ± 0.03↑	99.91
skin-nonskin	92.62	99.69	99.55 ± 0.06	**99.78**	99.73 ± 0.04	99.77
usps	82.42	87.47	66.33 ± 1.65	88.71	92.42 ± 0.59↑*	**92.74**↑*
w8a	98.50	**98.81**	96.90 ± 0.00	96.90	96.73 ± 0.15	96.79

Table 3
Macro-F_1 score of MDT2 and BDTKS.v2 and modified ones. ("↑" or "↓" represents the macro-F_1 score of this method is higher or lower than the immediate preceding method which is the corresponding method using majority class proportion based split condition. "*" represents methods applying K-means to the binary tree have higher macro-F_1 score: δ-BDTKS.v2 versus MDT2, BDTKS.v2 versus λ-MDT2.).

Dataset	with under-sampling				without under-sampling			
	MDT2	λ-MDT2	δ-BDTKS.v2	BDTKS.v2	MDT2	λ-MDT2	δ-BDTKS.v2	BDTKS.v2
acoustic	0.6476	0.6476	0.6550 *	0.6550 *	0.6897	0.6981↑	0.6962 *	0.7086↑*
aloi	0.6785	0.6785	0.7658 *	0.7658 *	0.7855	0.7855	0.8580 *	0.8580 *
combined	0.7437	0.7437	0.7526 *	0.7526 *	0.7493	0.7893↑	0.7677 *	0.7957↑*
covtype	0.6087	0.6087	0.6364 *	0.6364 *	0.8368	0.8368	0.8637 *	0.8637 *
gisette	0.8921	0.8921	0.9351 *	0.9351 *	0.8779	0.8799↑	0.9308 *	0.9294↓*
ijcnn1	0.8091	0.8091	0.8755 *	0.8755 *	0.8913	0.8913	0.9284 *	0.9284 *
letter	0.7657	0.7657	0.8420 *	0.8420 *	0.7939	0.7939	0.8640 *	0.8640 *
madelon	0.7188	0.7188	0.7042	0.7042	0.6779	0.6698↓	0.7614 *	0.7663↑*
mushrooms	0.9898	0.9898	0.9730	0.9974↑*	0.9921	0.9948↑	0.9921	0.9921
pendigits	0.9673	0.9673	0.9762 *	0.9762 *	0.9683	0.9683	0.9776 *	0.9776 *
phishing	0.8100	0.8149↑	0.8999 *	0.8999 *	0.8296	0.8146↓	0.8497 *	0.8352↓*
protein	0.3966	0.3966	0.4383 *	0.4383 *	0.4240	0.4411↑	0.4724 *	0.4849↑*
satimage	0.8335	0.8312↓	0.8782 *	0.8867↑*	0.8700	0.8421↓	0.8846 *	0.8846 *
segment	0.8843	0.8843	0.9156 *	0.9156 *	0.8782	0.8943↑	0.9149 *	0.9149 *
seismic	0.6118	0.6118	0.6262 *	0.6192↓*	0.6481	0.6806↑	0.6481	0.6864↑*
shuttle	0.2088	0.2088	0.2384 *	0.2717↑*	0.6789	0.6789	0.7817 *	0.7817 *
skin-nonskin	0.9943	0.9943	0.9955 *	0.9955 *	0.9973	0.9973	0.9971	0.9971
usps	0.8644	0.8644	0.9145 *	0.9145 *	0.8749	0.8749	0.9197 *	0.9197 *
w8a	0.6416	0.6407↓	0.6581 *	0.6565↓*	0.4921	0.6533↑	0.4921	0.6742↑*

those methods applying K-means show higher macro-F_1 score. Using the same majority class proportion based split condition, δ-BDTKS.v2 exhibit better performance than MDT2. Likewise, using the same non-split condition, BDTKS.v2 are better than λ-MDT2. Therefore, applying K-means to the binary tree to obtain linear multivariate decision trees has successfully improved the classification performance.

As for the advantage of the proposed non-split condition, let us first observe Table 3. There are many cases where the macro-F_1 score varies when methods change the majority class proportion based split condition (using δ) into the proposed non-split condition (using λ). There are many ups and some downs (on "gisette", "madelon", "phishing", and "satimage") when using δ is changed into using λ. The four data sets with downs have the common characteristics that they all have small training sample size and relatively balanced classes. It is reasonable to infer that the small training sample size has caused the relatively inaccurate results during the parameter selection. Besides, the non-split condition relies more on large training sample size due to using the constraint of the proportion of classes. The larger training sample size may reflect the more real state of the classes, thus more correct classification. In addition, the non-split condition may slightly degrade the performance for some relatively balanced data sets. Note that, there still exist five imbalanced data sets ("acoustic", "combined", "protein", "seismic", and "w8a") keeping stable ups when using δ is changed into using λ. To sum up, compared with using the majority class proportion based split condition, methods using the proposed non-split condition present better or same performance for all data sets except for some balanced ones with small sample size. Actually, the non-split condition makes BDTKS model accommodate the imbalanced cases more easily. The under-sampling technique is a commonly used method for handling class-imbalance problem. So, we also use the random under-sampling technique to preprocess the training sample to become the totally balanced sub-

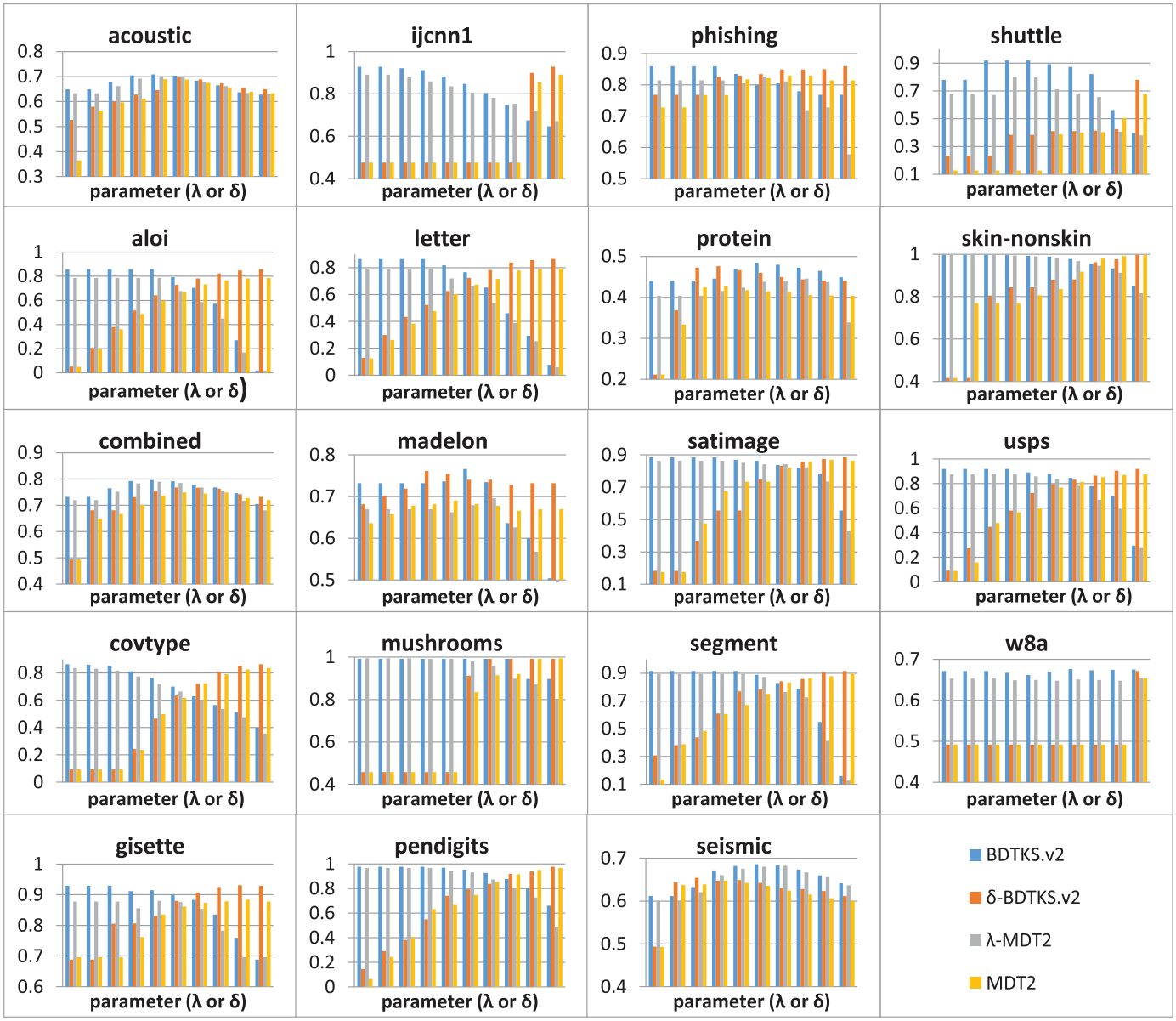

Fig. 5. Macro-F_1 score varying with the corresponding parameter.

set and then learn MDT2 and BDTKS.v2 and their modified ones. The macro-F_1 score values are also reported in Table 3. From those results, it can be found that the listed methods outperform the corresponding ones with under-sampling in most cases and just for some relatively balanced data sets with small sample size those methods with under-sampling beat those without under-sampling. Therefore, it can be inferred that under-sampling may be effective for nearly balanced data sets while BDTKS may play to its strengths in more imbalanced cases.

To explore the parameter sensitivity, we have conducted experiments on BDTKS.v2 and MDT2 and the modified ones as the parameter (λ or δ) varies. λ varies from 10^{-6} to 10^{3} to take 10 different values for all data sets. δ takes 10 values at equal intervals on the interval $(\frac{1}{c}, 1]$. Fig. 5 has shown the macro-F_1 score values of the four methods with those different parameter values. Table 4 has shown a more clear comparison between BDTKS.v2 and δ-BDTKS.v2 on the distributions of λ and δ and the corresponding macro-average precision and recall. From Fig. 5 and Table 4, some conclusions can be drown. First, the non-split condition has lower parameter sensitivity than the majority class proportion based split condition in most cases. Faced with class imbalanced problem, λ varying in a wide range still makes trees classify well while δ only in a very limited range is meaningful. For some imbalanced data sets ("ijcnn1", "mushrooms", "shuttle", and "w8a"), on the macro-F_1 score, methods using δ have cliff changes while those using λ gradually vary. Second, the best classification accuracy of our BDTKS can usually be obtained when λ takes a certain value on the interval (0,1] and values around 10^{-3} enables BDTKS to behave well enough for most data sets. Generally speaking, smaller λ is more proper for data sets with large number of classes or extremely imbalanced classes. Third, the proposed non-split condition makes BDTKS present quite high level in classification precision and recall simultaneously in a quite wide range of the parameter λ.

4.3. Training and classification time

We show the training time of C4.5, MDT1, MDT2, BDTKS.v1 and BDTKS.v2 in Table 5. We do not show the training time of CART,

Table 4
Distributions of λ (in BDTKS.v2) and δ (in δ-BDTKS.v2) and corresponding precision and recall.

Dataset	ACC_{best}		$\geq 0.95 \times ACC_{best}$					
	λ	δ	λ	P(%)	R(%)	δ	P(%)	R(%)
acoustic	10^{-2}	0.73	$[10^{-3}, 1]$	[67.46,69.94]	[69.25,71.81]	[0.73,0.87]	[67.62,69.74]	[66.87,69.50]
aloi	$[10^{-6}, 10^{-2}]$	1	$[10^{-6}, 10^{-2}]$	85.99	85.62	[0.80,1]	[82.56,85.99]	[82.10,85.62]
combined	10^{-2}	0.73	$[10^{-4}, 1]$	[74.95,78.41]	[77.94,80.77]	[0.60,1]	[73.07,76.87]	[73.20,76.89]
covtype	10^{-6}	1	$[10^{-6}, 10^{-4}]$	[83.64,86.38]	[86.13,86.43]	[0.83,1]	[84.23,86.38]	[77.93,86.36]
gisette	$[10^{-6}, 10^{-4}]$	0.90	$[10^{-6}, 1]$	[88.29,92.94]	[88.29,92.93]	[0.85,1]	[90.71,93.08]	[90.66,93.08]
ijcnn1	$[10^{-6}, 10^{-5}]$	1	$[10^{-6}, 1]$	[81.76,94.85]	[79.11,91.08]	[0.95,1]	[94.50,94.85]	[85.75,90.90]
letter	$[10^{-6}, 10^{-3}]$	1	$[10^{-6}, 10^{-3}]$	86.42	86.38	[0.81,1]	[83.94,86.42]	[83.89,86.38]
madelon	10^{-1}	0.70	$[10^{-6}, 1]$	[73.17,76.59]	[73.24,76.67]	[0.70,1]	[72.84,76.26]	[72.89,76.02]
mushrooms	$[10^{-6}, 1]$	[0.85,1]	$[10^{-6}, 10]$	99.76	98.67	[0.80,1]	[97.53,99.76]	[85.84,98.67]
pendigits	$[10^{-6}, 10^{-3}]$	1	$[10^{-6}, 10^{-1}]$	[95.46,97.76]	[95.39,97.76]	[0.91,1]	[94.10,97.76]	[94.08,97.76]
phishing	$[10^{-6}, 10^{-3}]$	1	$[10^{-6}, 10^{-2}]$	[83.49,86.06]	[83.55,85.90]	[0.70,1]	[82.58,86.06]	[82.35,85.90]
protein	10^{-1}	0.53	$[10^{-2}, 10^{2}]$	[47.48,49.67]	[44.21,47.35]	[0.47,0.67]	[33.47,49.45]	[41.02,46.63]
satimage	$[10^{-6}, 10^{-3}]$	1	$[10^{-6}, 1]$	[85.02,88.85]	[82.61,88.07]	[0.75,1]	[86.65,88.85]	[80.30,88.07]
segment	$[10^{-6}, 10^{-3}]$	1	$[10^{-6}, 10^{-1}]$	[89.32,91.33]	[88.44,91.66]	[0.91,1]	[90.37,91.33]	[90.89,91.66]
seismic	10^{-1}	0.73	$[10^{-3}, 10^{2}]$	[65.48,68.01]	[66.61,69.27]	[0.47,0.93]	[62.45,64.77]	[62.22,66.19]
shuttle	$[10^{-6}, 10^{-5}]$	1	$[10^{-6}, 10^{2}]$	[55.73,99.28]	[56.74,86.44]	[0.66,1]	[40.62, 85.26]	[41.43,72.16]
skin-nonskin	$[10^{-6}, 10^{-5}]$	1	$[10^{-6}, 10]$	[93.97,99.67]	[96.84,99.79]	[0.90,1]	[95.71,99.67]	[96.64,99.76]
usps	$[10^{-6}, 10^{-3}]$	1	$[10^{-6}, 10^{-1}]$	[87.92,92.16]	[87.38,91.78]	[0.91,1]	[91.02,92.16]	[90.26,91.78]
w8a	10^{3}	[0.55,0.95]	$[10^{-6}, 10^{3}]$	[68.29,72.54]	[63.15,64.18]	[0.55,1]	[48.45,71.40]	[50.00,63.35]

Table 5
Comparison of training time T_{tr}.

T_{tr}(s)	C4.5	MDT1	MDT2	BDTKS.v1	BDTKS.v2
acoustic	1.39E+04	1.14E+03	**2.44E+02**	1.81E+03	4.02E+02
aloi	6.49E+04	5.18E+03	**4.98E+02**	2.20E+03	6.62E+02
combined	4.05E+04	7.67E+02	**1.66E+02**	1.42E+03	3.46E+02
covtype	4.14E+03	1.92E+04	2.81E+03	1.11E+04	**2.76E+03**
gisette	1.11E+04	**2.33E+02**	4.35E+02	7.12E+02	6.09E+02
ijcnn1	6.86E+03	1.15E+02	**3.76E+01**	5.61E+02	1.53E+02
letter	**1.17E+00**	9.03E+01	2.17E+01	1.89E+02	4.93E+01
madelon	7.70E+02	8.72E+00	**5.86E+00**	3.11E+01	9.56E+00
mushrooms	**1.25E-01**	3.75E+00	9.06E-01	4.48E+00	1.39E+00
pendigits	5.32E+00	1.41E+01	**3.02E+00**	3.72E+01	1.06E+01
phishing	**1.57E-01**	3.53E+00	9.71E-01	1.12E+01	2.68E+00
protein	2.22E+03	2.09E+02	**1.11E+02**	7.76E+02	2.14E+02
satimage	6.30E+00	1.04E+01	**3.25E+00**	5.28E+01	1.40E+01
segment	2.19E+00	3.25E+00	**9.24E-01**	9.21E+00	2.44E+00
seismic	1.71E+04	8.42E+02	**2.31E+02**	1.78E+03	3.97E+02
shuttle	4.45E+00	8.94E+00	**2.44E+00**	4.39E+01	1.22E+01
skin-nonskin	**2.45E+00**	9.68E+00	2.49E+00	3.12E+01	8.04E+00
usps	2.08E+03	3.50E+01	**1.32E+01**	1.10E+02	3.80E+01
w8a	5.88E+01	**8.29E+00**	9.24E+00	4.30E+02	1.07E+02

because CART is very fast due to the MATLAB built-in function *fitctree*() employing the MEX file which greatly shortens the training time. It can be seen from Table 5 that either C4.5 or MDT2 wins in the training time for nearly all data sets and our BDTKS.v2 is always quite close to the winner in training time. As a matter of fact, this also reflects that univariate decision trees and multivariate decision trees have their respective merits in different cases.

In addition, comparison of the classification time of MDT1, MDT2, BDTKS.v1, BDTKS.v2, and A-BDTKS.v2, all of which are multivariate decision trees, is shown in Table 6. We have reported the total classification time T_{te} on all the testing examples. From Table 6, it can be discovered that A-BDTKS.v2 has indeed shortened the classification time in comparison with BDTKS.v2. Herein, we point out that A-BDTKS.v2 should have been approximately twice as fast as BDTKS.v2. Actually, Table 6 seems not to prove that point for the reason that we have used "parfor" statement in the classification stage of all the experiments. Anyway, Table 6 indeed exhibits that A-BDTKS.v2 is the fastest in the classification time among the listed multivariate decision trees for most data sets. Moreover, our proposed methods especially A-BDTKS.v2 takes comparable or shorter classification time compared with MDT1 and MDT2.

4.4. Number of splitting

Generally speaking, the number of splitting in decision tree models can be one of the criteria that indicate the complexity of models. Under the premise of ensuring the classification performance, the smaller number of splitting in the model is the better. Table 7 has shown the number of splitting in the models of C4.5, CART, MDT1, λ-MDT1, MDT2, λ-MDT2, δ-BDTKS.v1, BDTKS.v1, δ-BDTKS.v2, and BDTKS.v2. For most of the data sets, the number of splitting in the models using our proposed non-split condition is smaller than or equals to that in the models using the majority class proportion based split condition. Note that, for the data set "w8a", although MDT1, MDT2, δ-BDTKS.v1, and δ-BDTKS.v2 own just one splitting, they classify all data points into one class, which does not make any sense. Also, for most of the data sets, the number of splitting in the K-means embedded models is smaller than that in the corresponding models without K-means embedded, namely δ-BDTKS.v1 smaller than MDT1, BDTKS.v1 smaller than λ-MDT1, δ-BDTKS.v2 smaller than MDT2, and BDTKS.v2 smaller than λ-MDT2. Additionally, for some data sets such as "acoustic", "combined", "protein", and "seismic", BDTKS.v1 and BDTKS.v2 own smaller number of splitting compared to CART on the condition that they present higher classification accuracy as has been pre-

Table 6
Comparison of classification time T_{te}.

T_{te}(s)	MDT1	MDT2	BDTKS.v1	BDTKS.v2	A-BDTKS.v2
acoustic	7.03E-01	**5.78E-01**	9.91E-01	9.21E-01	7.65E-01
aloi	7.10E+00	3.95E+00	3.48E+00	3.67E+00	**2.96E+00**
combined	9.35E-01	1.03E+00	1.01E+00	9.68E-01	**7.34E-01**
covtype	1.08E+01	7.85E+00	1.09E+01	8.50E+00	**7.43E+00**
gisette	**4.82E-01**	7.68E-01	1.21E+00	1.19E+00	7.34E-01
ijcnn1	**8.48E-01**	1.19E+00	1.51E+00	1.34E+00	1.22E+00
letter	2.72E-01	2.35E-01	2.31E-01	2.19E-01	**2.03E-01**
madelon	5.30E-02	6.30E-02	6.26E-02	**3.10E-02**	**3.10E-02**
mushrooms	5.00E-02	4.70E-02	3.76E-02	**3.10E-02**	**3.10E-02**
pendigits	1.00E-01	9.40E-02	9.70E-02	**9.30E-02**	**9.30E-02**
phishing	3.14E-02	**3.10E-02**	3.42E-02	**3.10E-02**	**3.10E-02**
protein	**1.16E-01**	2.03E-01	4.79E-01	3.44E-01	2.34E-01
satimage	7.82E-02	7.80E-02	9.06E-02	7.80E-02	**6.20E-02**
segment	3.74E-02	**3.10E-02**	3.74E-02	**3.10E-02**	**3.10E-02**
seismic	**3.40E-01**	3.92E-01	8.32E-01	8.14E-01	6.59E-01
shuttle	2.53E-01	2.66E-01	3.38E-01	2.98E-01	**2.50E-01**
skin-nonskin	1.94E-01	1.89E-01	2.03E-01	2.03E-01	**1.87E-01**
usps	1.81E-01	**1.26E-01**	1.59E-01	1.57E-01	**1.26E-01**
w8a	**5.62E-02**	6.30E-02	4.63E-01	2.20E-01	1.81E-01

Table 7
Number of splitting in models.

Name	C4.5	CART	MDT1	λ-MDT1	MDT2	λ-MDT2	δ-BDTKS.v1	BDTKS.v1	δ-BDTKS.v2	BDTKS.v2
acoustic	114	13191	13359.4	3534.6	8221	3273	14601.4	3932	14022	3293
aloi	76	16565	61783.4	64121.6	38860	38860	24791.2	24832	26355	26355
combined	164	9817	17601.4	2894.8	21781	2505	10391.8	3145.4	5279	2533
covtype	85	31149	161319.6	160925	130842	130842	102614.6	95002.6	104188	104188
gisette	213	221	577.6	252.2	1250	397	1317.2	1271.2	833	1211
ijcnn1	37	2111	10531.4	19432	15281	15281	10273.8	10132.2	10831	10831
letter	39	2137	9693.4	9793.6	6536	6536	4421.2	4435.2	4622	4622
madelon	244	161	327.4	750	515	605	97.2	412.8	175	112
mushrooms	4	9	483	579	42	58	10	14	16	16
pendigits	65	415	2158.8	2730.4	1237	1237	730.4	711.4	832	832
phishing	26	55	69.8	94.4	156	266	222	247.2	155	153
protein	1053	4583	381.8	201.6	1905	357	2392.2	773.2	1493	836
satimage	74	499	1668.8	2087.8	1100	259	1324.4	1320	1334	1334
segment	19	55	343.8	373.6	203	238	173.6	168.8	184	184
seismic	120	13509	4726	1077	5661	1051	5799.6	1277.8	5354	1060
shuttle	15	57	900.4	885	526	526	336.2	332.6	349	349
skin-nonskin	24	295	1120.8	1147.4	693	693	480.4	443.4	516	516
usps	147	525	4129.8	4188.2	1880	1880	1588	1630.6	1701	1701
w8a	107	353	1	880.8	1	2546	1	2055.8	1	803

sented in Table 2. Although generally owning the very small number of splitting for most of the data sets, C4.5 gains relatively lower classification accuracy, which can be observed from Table 2.

5. Conclusion

This paper develops a novel linear multivariate decision tree named BDTKS. Introducing K-means clustering to serve as node splitting method and proposing the non-split condition are the key components of the proposed approach. The former provides BDTKS with good generalization ability, enhancing the classification performance. The latter enables the tree model to accommodate the class imbalance cases more flexibly than the majority class proportion based split condition in MDT1 and MDT2. Hence, BDTKS can adapt itself to either class balance or class imbalance cases. Experiments on 19 publicly available data sets have demonstrated that BDTKS matches or outperforms the famous univariate decision trees (C4.5 and CART) and the recent multivariate decision trees (MDT1 and MDT2) in classification performance. In terms of efficiency, BDTKS and its accelerated version A-BDTKS take comparable training time as the previous decision trees and shorter classification time. Therefore, good classification performance and high efficiency coexist in the proposed linear multivariate decision tree. However, confronted with high dimension cases, the K-means node splitting method in BDTKS is probably somewhat time consuming. Additionally, though the proposed non-split condition in BDTKS has made the parameter less sensitive than the majority class proportion based split condition, there still is room to improve.

In future work, feature selection or dimensionality reduction technique can be considered into node splitting at each non-leaf node. The non-split condition can be advanced to further reduce the parameter sensitivity. The proposed BDTKS decision tree can be ensembled to construct decision forests thus improving performance again.

Declaration of Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research is supported by National Major Science and Technology Projects of China [grant number 2019ZX01008103]; Xi'an Science and Technology Innovation Program [grant number 201809162CX3JC4]; National Natural Science Foundation of China

[grant numbers 61772427, 61751202]; and the Fundamental Research Funds for the Central Universities.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.patcog.2020.107521.

References

[1] A. Pinto, S. Pereira, D.M.L.D. Rasteiro, C. Silva, Hierarchical brain tumour segmentation using extremely randomized trees, Pattern Recognit. 82 (2018) 105–117.
[2] M.M. Ghiasi, S. Zendehboudi, A.A. Mohsenipour, Decision tree-based diagnosis of coronary artery disease: cart model, Comput. Methods Programs Biomed. 192 (2020) 105400.
[3] S. Benferhat, A. Boudjelida, K. Tabia, H. Drias, An intrusion detection and alert correlation approach based on revising probabilistic classifiers using expert knowledge, Appl. Intell. 38 (4) (2013) 520–540.
[4] N.D. Vanli, M.O. Sayin, N.M. Mohaghegh, H. Ozkan, S.S. Kozat, Nonlinear regression via incremental decision trees, Pattern Recognit. 86 (2019) 1–13.
[5] S.B. Kotsiantis, Decision trees: a recent overview, Artif. Intell. Rev. 39 (4) (2013) 261–283.
[6] J.R. Quinlan, Induction of decision trees, Mach. Learn. 1 (1) (1986) 81–106.
[7] L. Breiman, J.H. Friedman, C.J. Stone, R.A. Olshen, CART: Classification and Regression Trees, Chapman & Hall/CRC, 1984.
[8] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc., 1993.
[9] R. Wang, S. Kwong, X.-Z. Wang, Q. Jiang, Segment based decision tree induction with continuous valued attributes, IEEE Trans. Syst. Man Cybern. 45 (7) (2015) 1262–1275.
[10] B. Chandra, P. Paul Varghese, Moving towards efficient decision tree construction, Inf. Sci. 179 (8) (2009) 1059–1069.
[11] B. Chandra, R. Kothari, P. Paul, A new node splitting measure for decision tree construction, Pattern Recognit. 43 (8) (2010) 2725–2731.
[12] S.K. Murthy, S. Kasif, S. Salzberg, A system for induction of oblique decision trees, J. Artif. Intell. Res. 2 (1994) 1–32.
[13] N. Manwani, P.S. Sastry, Geometric decision tree, IEEE Trans. Syst. Man Cybern. 42 (1) (2012) 181–192.
[14] P.E. Utgoff, C.E. Brodley, Linear Machine Decision Trees, Technical Report, University of Massachusetts, Amherst MA, 1991.
[15] W. Loh, N. Vanichsetakul, Tree-structured classification via generalized discriminant analysis, J. Am. Stat. Assoc. 83 (403) (1988) 715–725.
[16] W.Y. Loh, Y.S. Shih, Split selection methods for classification trees, Stat. Sin. 7 (4) (1997) 815–840.
[17] O.T. Yildiz, E. Alpaydin, Linear discriminant trees, Int. J. Pattern Recognit. Artif. Intell. 19 (3) (2005) 323–353.
[18] H.K. Sok, M.P. Ooi, Y.C. Kuang, S. Demidenko, Multivariate alternating decision trees, Pattern Recognit. 50 (2016) 195–209.
[19] C. Yildiz, E. Alpaydin, Omnivariate decision trees, IEEE Trans. Neural Netw. 12 (6) (2001) 1539–1546.
[20] Y. Li, M. Dong, R. Kothari, Classifiability-based omnivariate decision trees, IEEE Trans. Neural Netw. 16 (6) (2005) 1547–1560.
[21] H. Altincay, Decision trees using model ensemble-based nodes, Pattern Recognit. 40 (12) (2007) 3540–3551.
[22] M.A. Kumar, M. Gopal, A hybrid SVM based decision tree, Pattern Recognit. 43 (12) (2010) 3977–3987.
[23] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics & Probability, 1967, pp. 281–297.
[24] S.R. Gaddam, V.V. Phoha, K.S. Balagani, K-means+id3: a novel method for supervised anomaly detection by cascading k-means clustering and id3 decision tree learning methods, IEEE Trans. Knowl. Data Eng. 19 (3) (2007) 345–354.
[25] M.F. Amasyali, O. Ersoy, Cline: a new decision-tree family, IEEE Trans. Neural Netw. 19 (2) (2008) 356–363.
[26] K. Kim, A hybrid classification algorithm by subspace partitioning through semi-supervised decision tree, Pattern Recognit. 60 (2016) 157–163.
[27] Y. Qu, L. Lin, F. Shen, C. Lu, Y. Wu, Y. Xie, D. Tao, Joint hierarchical category structure learning and large-scale image classification, IEEE Trans. Image Process. 26 (9) (2017) 4331–4346.
[28] A. Gallego, J. Calvozaragoza, J.J. Valeromas, J.R. Ricojuan, Clustering-based k-nearest neighbor classification for large-scale data with neural codes representation, Pattern Recognit. 74 (2018) 531–543.
[29] S. Das, S. Datta, B.B. Chaudhuri, Handling data irregularities in classification: foundations, trends, and future challenges, Pattern Recognit. 81 (2018) 674–693.
[30] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, Smote: synthetic minority over-sampling technique, J. Artif. Intell. Res. 16 (1) (2002) 321–357.
[31] C. Drummond, R. Holte, C4.5, class imbalance, and cost sensitivity: Why under-sampling beats oversampling, in: Proceedings of the ICML'03 Workshop on Learning from Imbalanced Datasets, 2003.
[32] M.A. Tahir, J. Kittler, F. Yan, Inverse random under sampling for class imbalance problem and its application to multi-label classification, Pattern Recognit. 45 (10) (2012) 3738–3750.
[33] C. Huang, Y. Li, C.C. Loy, X. Tang, Learning deep representation for imbalanced classification, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 5375–5384.
[34] A.C. Bahnsen, D. Aouada, B. Ottersten, Example-dependent cost-sensitive decision trees, Expert Syst. Appl. 42 (19) (2015) 6609–6619.
[35] C. Cao, Z. Wang, Imcstacking: cost-sensitive stacking learning with feature inverse mapping for imbalanced problems, Knowl. Based Syst. 150 (2018) 27–37.
[36] F. Wang, Q. Wang, F. Nie, W. Yu, R. Wang, Efficient tree classifiers for large scale datasets, Neurocomputing 284 (2018) 70–79.
[37] D. Arthur, S. Vassilvitskii, K-means++: the advantages of careful seeding, in: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 1027–1035.

Fei Wang received the B.E. degree in electrical and information engineering from Northwest University, the M.E. degree in communication and information system from Xi'an Institute of Optics and Precision Mechanics, Chinese Academy of Sciences, Ph.D. degree in Control science and Engineering from Xi'an Jiaotong University (XJTU), Xi'an, China, in 1998, 2002, and 2009 respectively. Currently, he is a Full Professor in Xi'an Jiaotong University. His research interests include image processing, computer vision and intelligent system.

Quan Wang is pursuing a double Ph.D. Degree with Xi'an Jiaotong University and Tokushima University. She received the M.E. degree from Xi'an Jiaotong University, China, in 2015, the B.E. degree from Xi'an University of Posts and Telecommunications, China, in 2012. Her main research interests include machine learning, data mining, pattern recognition, image processing, and computer vision.

Feiping Nie received the Ph.D. degree in computer science from Tsinghua University, China, in 2009. He is currently a Full Professor with Northwestern Polytechnical University, China. His research interests are machine learning and its applications, such as pattern recognition, data mining, computer vision, image processing, and information retrieval. He has authored over 100 papers in the following top journals and conferences: TPAMI, IJCV, TIP, TNNLS/TNN, TKDE, Bioinformatics, ICML, NIPS, KDD, IJCAI, AAAI, ICCV, CVPR, and ACM MM. His papers have been cited over 7000 times. He is serving as an associate editor or a PC member for several prestigious journals and conferences in the related fields.

Zhongheng Li is a Ph.D. student specialized in pattern recognition and intelligent system at Xi'an Jiaotong University, Xi'an, China. He received the B.E. degree in automation from Xi'an Jiaotong University, China, in 2014. His main research interests include machine learning, pattern recognition, image processing, and computer vision.

Weizhong Yu received the Ph.D. degree from the Xi'an Research Institute of Hi-Tech, China, in 2012. He was also with the Institute of Microelectronics, Tsinghua University, Beijing, China, for the Ph.D. degree. He is currently a Post-Doctoral Research Associate with the Institute of Artificial Intelligence and Robotics of Xi'an Jiaotong University. His main research interests include machine learning and computer vision.

Fuji Ren received the Ph.D. degree from the Faculty of Engineering, Hokkaido University, Japan, in 1991. From 1991 to 1994, he was a Chief Researcher with CSK. In 1994, he joined the Faculty of Information Sciences, Hiroshima City University, as an Associate Professor. Since 2001, he has been a Professor with the Faculty of Engineering, Tokushima University. His current research interests include natural language processing, artificial intelligence, affective computing, and emotional robot. He is the Academician of The Engineering Academy of Japan and EU Academy of Sciences. He is an Editor-in-Chief of International Journal of Advanced Intelligence and the Vice President of CAAI. He is a Fellow of The Japan Federation of Engineering Societies, IECE, and CAAI. He is the President of the International Advanced Information Institute, Japan.