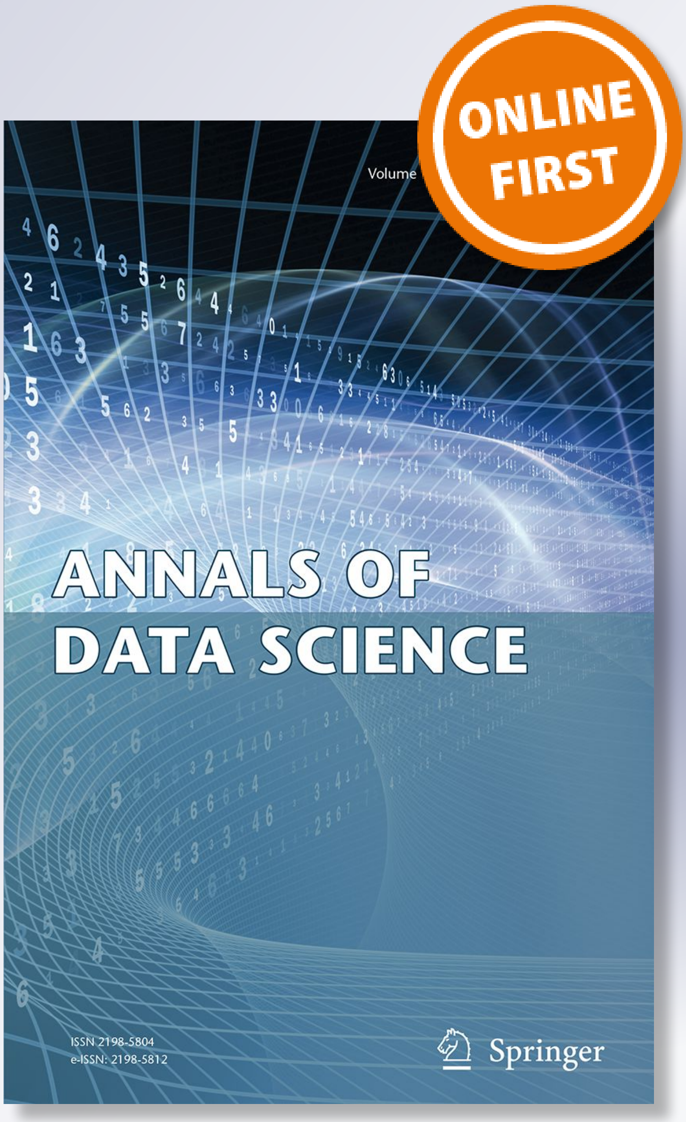
Analytical Split Value Calculation for Numerical Attributes in Hoeffding Trees with Misclassification-Based Impurity

Mehran Mirkhan, Maryam Amir Haeri & Mohammad Reza Meybodi

Annals of Data Science

ISSN 2198-5804

Ann. Data. Sci.
DOI 10.1007/s40745-019-00225-4



Annals of Data Science
https://doi.org/10.1007/s40745-019-00225-4

Analytical Split Value Calculation for Numerical Attributes in Hoeffding Trees with Misclassification-Based Impurity

Mehran Mirkhan[1] · Maryam Amir Haeri[1] · Mohammad Reza Meybodi[1]

Received: 22 August 2018 / Revised: 30 April 2019 / Accepted: 25 May 2019

Abstract
Hoeffding tree is a method to incrementally build decision trees. A common approach to handle numerical attributes in Hoeffding trees is to represent their sufficient statistics as Gaussian distributions. Our contribution in this paper is to prove that by using Gaussian distribution as sufficient statistics and misclassification error as impurity measure, there is an analytical method to exactly calculate the best splitting values. Three different approaches for using this theorem are proposed and all three are tested on both synthetic and real datasets. The experiments suggest that this approach can create smaller trees and learn faster and achieve higher accuracy in most problems.

Keywords Hoeffding tree · Gaussian distribution · Misclassification error · Massive and streaming data

1 Introduction

With the growth of data in recent years and the need for analyzing them, a new field called "big data analysis" has emerged. The aim of this field is to provide fast and efficient methods for analyzing massive datasets without losing the accuracy of the analysis.

These massive datasets usually emerge in the form of data streams. Due to this massive number of data instances, some new challenges also emerge; for example, it

✉ Maryam Amir Haeri
haeri@aut.ac.ir

Mehran Mirkhan
mmirkhan@aut.ac.ir

Mohammad Reza Meybodi
mmeybodi@aut.ac.ir

1 Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran

is assumed that each data instance is observed only once, and the analysis itself should be very fast and memory efficient to be able to deal with such datasets.

One of the most common data analysis tasks is classification. The aim of classification is to construct a function called the classifier, based on the training data that maps the set of data instances into the set of classes. Then this function is used to classify unlabeled data.

There are a wide variety of methods used for data classification. Decision tree (DT) is one of such methods. DT is basically a hierarchy of nodes. Each node has some child nodes and performs a test on a given data and based on the result of the test, sorts the data down into one of its children. Each leaf node is assigned a label; therefore, each data instance can be classified in this manner.

In the last decades, with the emergence of massive datasets, researchers have tried to use DT for this purpose. The main characteristics of DT are that it is very fast and produces an interpretable model, which makes it very suitable for massive datasets [1]. However, the traditional DT algorithms are only applicable to static datasets; in which all instances are available a priori. Therefore such algorithms are useless in case of data streams; in which each data instance is observed only once. Thus, Many methods for incremental training of DTs are proposed so far (e.g. incremental CART [2] and ITI [3]). One of the most famous incremental DT learning algorithms is Hoeffding tree [4]. In this method, the decision tree is constructed incrementally; it uses the Hoeffding bound to find the minimum number of arriving instances needed to reach a distinct level of confidence in splitting the node. After arriving these instances a node is splitted and the tree grows gradually.

There are different modifications and enhancements made on Hoeffding tree in the literature of DT for streaming data. The original authors in the same paper propose some improvements and refer to the new approach as “Very Fast Decision Tree (VFDT)”; which is meant to make Hoeffding tree faster. They proposed another improvement in later years named CVFDT [5] that is able to handle concept drift (the meaning of data instances changes over time).

Recently, many researchers have been focused on improving DT tree for data streams and massive datasets. There are several methods which use ensemble learning to improve the accuracy of the DT over data stream or massive data such as [6–9]. Several other researchers suggested DT methods with different splitting criteria which lead to more accurate or faster DT such as [10–13].

De Rosa and Cesa-Bianchi [11] introduced a method to determine accurate confidence intervals to estimate the gain of each split in DT. In their approach, the confidence intervals depend on more tree parameters (rather than Hoeffding bound) and thus their Decision trees are more accurate than the Hoeffding trees.

Rutkowski et al. [12] suggested a new approach for constructing DT for stream data. They derived a new splitting criterion based on the misclassification error. By considering this criterion, they proved a theorem showing that with a high probability, the best attribute computed in a considered node based on the available data instances is the same as the attribute computed from the whole data stream. Then they used this theorem with the splitting criterion which is combined by Gini index and demonstrated that this combination leads to more accurate DT classification. They extended this idea in another research work and introduced two other hybrid splitting criteria based on the combination of the misclassification error and Gini index [13].

Another challenge in learning DT is handling numerical attributes. This problem has always been challenging in both static and incremental DTs; since for numerical attributes, the split value can be any real number and to find the best value, one has to check all real values (which is not possible).

In static DT, different approaches have been proposed. One approach is to define some bins and discretized the values based on those bins [14]. Another approach is to sort all the values and find the best split value by checking all points in the middle of observed values [15].

None of the above methods are applicable to incremental algorithms. The problem with discretization is that to define the bins, one has to know the range of values a priori (which is not the case in data streams) [16]. The problem with sorting is that it can only be used when all data instances are available at once (also not the case in data streams). Gama and Pinto [17] used bins to discretize the values and proposed a method to adjust the bins over time when new data is observed. The problem with this approach is that it is not very accurate and also it is prone to the problem of outliers.

Kirkby [18] proposes using Gaussian distribution to store sufficient statistics of the numerical attributes. Moreover, minimum and maximum values arrived so far are also stored. To check for the split, the range between minimum and maximum values is divided into some equal width intervals and each point is analyzed and the best point is chosen. The benefits of Gaussian approximation are that it is easy and efficient to maintain and also it is robust against noise and outliers. Many other researchers tried to find heuristic methods for finding appropriate split values for numerical attributes during the DT classification over massive or streaming data. However, most of these methods still need several preprocessing computations or require to check different points to find the best split values. For example, Salperwyck and Lemaire [19] proposed a DT method for streaming data that uses *Naive Bayes* classifier in the leaves and grow the tree gradually. Their method discretizes a numerical attribute into several intervals if that discretization leads to a better solution (in a Bayesian sense) than the model with no split on that attribute. Thus, it checks different points and uses the Bayes criteria to choose the splitting values among them. Moreover, Zhang and Cheung [20] introduced a discretization method based on windowing and hierarchical clustering for training DT over big data.

The main aim of this paper is to suggest a mathematical (analytical) method to find the best split values for numerical attributes in learning Hoeffding DT for massive or streaming data, by considering Gaussian approximation [18] for numerical attributes and misclassification [12] as the impurity measure. Finding the best split values analytically can increase the accuracy and the efficiency of training DT over data streams. We provide a theorem which shows that the best split point is a point where the Gaussian distributions intersect each other. Thus, by the proposed approach, it is possible to find the best split values for numerical attributes analytically by a closed formula and there is no need to try different split point experimentally. The experimental results show that the proposed method can find smaller and more accurate treas in a more efficient manner.

The rest of the paper is organized as follows. Section 2 is devoted to the preliminary notions which are the prerequisites of the proposed method. In Sect. 3, the proposed method for finding the split point mathematically is explained in detail. This section contains a proof that we provide for the correctness of the splitting method. In Sect. 4 the efficiency of the proposed method is investigated experimentally. Finally, Sect. 5 concludes the paper.

2 Preliminaries

In this section, the preliminary notions of DT for streaming data which are related to our research are explained, formally.

2.1 Decision Tree

A decision tree (DT) is a tree-like graph in which each non-leaf node represents a decision (hence called a decision node) and each leaf node is assigned a label. Each node represents a subset of input space. The purpose of a decision node is to divide this space into some smaller spaces; Therefore, DT's approach toward classification is a divide and conquer approach.

Let us assume that data instances are described by D attributes. Each attribute can be either nominal or numerical. If an attribute i is nominal, then the number of possible values it can have is finite. Furthermore, each data instance is assigned to a class. Let K denote the number of classes.

Even though there are DTs that use multiple attributes in their decision nodes (multivariate DTs), most DTs only use one attribute for this purpose (univariate DTs). This makes DT very fast and easier to interpret. Here we consider univariate DTs. Moreover, different approaches have been proposed so far to divide the space. Here, we assume a binary decision for numerical attributes (in the form $x \leq a$ and $x > a$); and for nominal attributes with v_i possible values, we divide the space into v_i sub-spaces (each sub-space for each attribute value). In this approach, each nominal attribute is used only once in each path from root to a leaf.

The most famous approach to train a DT is the top-down approach. In this approach, one begins with a leaf node (root) and recursively splits the node and extends the tree. Let S denote the set of data instances arrived at a node and let $|S|$ be the number of instances in S and $|S|_k$ be the number of instances in set S that are assigned to class k. In this approach, an impurity measure is defined for a set of data ($g(S)$). This function indicates how impure the set of data is. Any impurity measure g has to satisfy two following conditions [12]:

1. If the set S is dominated by instances of one class, i.e., $\exists_{k_0 \in \{1,\ldots,K\}} |S|_{k_0} = |S|$, then $g(S) = 0$ (the set S is maximally pure).
2. The function $g(S)$ is maximized if each class is represented by the same number of instances, i.e., $\forall_{k \in \{1,\ldots,K\}} |S|_k = \frac{|S|}{K}$ (the set S is maximally mixed).

For each attribute i, set S can be divided into Q disjoint subsets: $\{S_q^i\}_{q=1}^{Q}$. The best split is the one that decreases impurity the most. Therefore the following function is defined (called "purity gain") to represent how good a split is:

$$G^i(S) = g(S) - \sum_{q=1}^{Q} \frac{|S_q^i|}{|S|} g(S_q^i). \tag{1}$$

Once the best split is identified, the node is split into some child nodes and each sub-space (S_q^i) resulted from this split is assigned to the corresponding child. This process is repeated until all data sets assigned to leaf nodes are pure enough.

Different impurity measures have been proposed so far. Three famous impurity measures are as follows [12]:

- Information Entropy:

$$g(S) = -\sum_{k=1}^{K} p_k(S) log_2[p_k(S)], \tag{2}$$

- Gini Index:

$$g(S) = 1 - \sum_{k=1}^{K} (p_k(S))^2. \tag{3}$$

- Misclassification Error:

$$g(S) = 1 - \max_{k \in \{1,\ldots,K\}} \{p_k(S)\}, \tag{4}$$

where

$$p_k(S) = \frac{|S|_k}{|S|}. \tag{5}$$

2.2 Hoeffding Tree

In static datasets, purity gain is calculated by observing the whole dataset; however, in the case of data streams, each data instance is observed only once. To overcome this challenge, one has to build the DT incrementally. Hoeffding tree is such a technique [4]. In Hoeffding tree, instead of storing all data in each node, some statistics about the data are stored in each leaf node (called "sufficient statistics"). For each new incoming data, it is sorted down through the tree until it reaches a leaf; then, the sufficient statistics of the leaf are updated. Periodically, the leaf is checked to see whether a split is required or not. For this purpose, two attributes that achieve highest gains are identified. Split is performed only if the difference between these two gains is larger than a split threshold. Authors proposed the Hoeffding bound (hence the name Hoeffding tree) as split threshold, which is defined as:

$$\text{Hoeffding bound} = R\sqrt{\frac{ln\left(\frac{1}{\delta}\right)}{2|S|}}, \tag{6}$$

where R is a range of random variable and δ specifies the degree of confidence in splits.

This method has been modified to handle concept drift phenomenon too [8,21]. Using "Naive Bayes" for classification in leaf nodes is also a very common practice in Hoeffding trees [22].

2.3 Misclassification Error

Rutkowsky et al. [12] proposed some improvements on Hoeffding trees. They first suggested that Hoeffding bound is not mathematically justified [23]. Then they proposed the use of misclassification error as impurity measure:

$$g(S) = 1 - \max_{k\in\{1,\ldots,K\}}\{p_k(S)\}, \tag{7}$$

and showed that it performs better than other methods [12]. They proposed the following split threshold for misclassification-based impurity measure:

$$\text{Split Threshold} = z_{(1-\delta)}\sqrt{\frac{1}{2|S|}}, \tag{8}$$

where $z_{(1-\delta)}$ is the $1-\delta$ quantile of the standard normal distribution ($\mathcal{N}(0,1)$).

In this paper, we use this measure as the impurity measure in learning DT.

3 Proposed Method

Our main contribution in this paper is to provide a theorem to show that by using Gaussian approximation as sufficient statistics for numerical attributes and misclassification error as impurity measure, there exists a closed formula for calculating the best splitting value. This theorem enables us to calculate split values very quickly and very accurately. This not only makes the training process much faster in most cases (which is highly desirable) but also creates much smaller trees (which is also highly desirable).

Thus, in this section, first a detailed description is provided to show how Gaussian approximation can be used as sufficient statistics. Next, a theorem is provided to show that the best split point is a point where Gaussian distributions intersect each other. Finally, three different approaches to practically use this theorem are proposed.

3.1 Gaussian Distribution as Sufficient Statistics

To store sufficient statistics in a node for nominal attributes, one stores a $K \times \nu_i$ table for each attribute i, where K is the number of classes, and ν_i is the number of possible values that attribute i can take. With each incoming new data instance, the corresponding elements of these tables are incremented. To check whether a split is required, each column of the table is considered as a sub-space and its impurity is measured. Then according to Eq. (1), the gain obtained by splitting on this attribute is calculated.

A similar approach can be adopted for numerical attributes. For each attribute, K Gaussian distributions are defined. With each incoming new data instance, the distributions can be updated as Algorithm 1 [18]:

Algorithm 1 Updating Gaussian distribution with each incoming data instance

```
1: weightSum ← 0
2: valueSum ← 0
3: valueSqSum ← 0
4: for all incoming data instances (value, weight) do
5:     weightSum ← weightSum + weight
6:     valueSum ← valueSum + value × weight
7:     valueSqSum ← valueSqSum + value × value × weight
8: end for
9: anytime output:
10: return mean ← $\frac{\text{valueSum}}{\text{weightSum}}$
11: return variance ← $\frac{\text{valueSqSum}-\text{mean}\times\text{valueSum}}{\text{weightSum}-1}$
```

To calculate split gain for a given split point x, one should calculate how many data instances would go into each branch (as mentioned earlier, for numerical attributes only binary splits are considered here). Let $S_l^i(x)$ and $S_r^i(x)$ respectively denote set of data that would go into left and right branches respectively by splitting on attribute i at value x. Then, CDF of Gaussian distribution can be used for this purpose:

$$|S_l^i(x)|_k = |S|_k F_k^i(x), \tag{9}$$

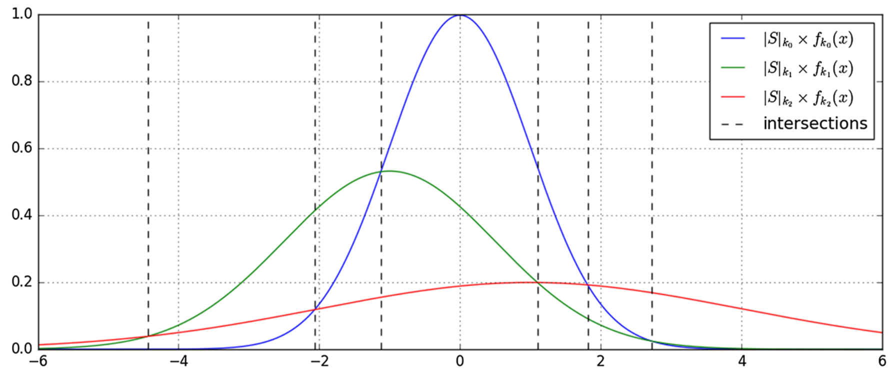
$$|S_r^i(x)|_k = |S|_k (1 - F_k^i(x)), \tag{10}$$

where $F_k^i(x)$ is cumulative distribution function of kth Gaussian distribution of attribute i at point x. Since there are only two branches, we have

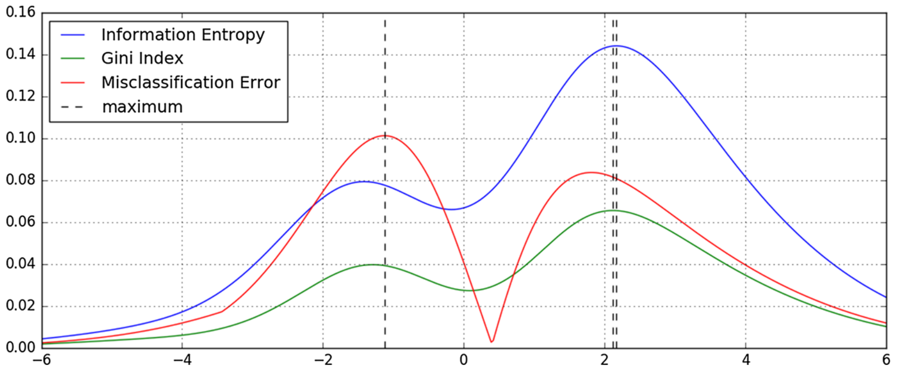
$$|S_l^i(x)| + |S_r^i(x)| = |S^i(x)|. \tag{11}$$

3.2 Analytical Split Value Calculation

Figure 1 shows the gains obtained by three different impurity measures at each point [Eqs. (2), (3) and (7)]. In this figure, one can observe that by using misclassification error as impurity measure, highest gain occurs exactly at a point where two Gaussian



(a) Gaussian distributions of a three class problem (multiplied by the number of instances that belong to each class). Vertical lines indicate intersection points.



(b) Gains calculated for every split point for the above distributions based on three different impurity measures.Vertical lines indicate maximum gain obtained by each impuritymeasure.

Fig. 1 The behavior of three different impurity measures are illustrated. As can be observed, maximum gain obtained by misclassification error occurs at an intersection point

distributions intersect with each other. We express this observation as a theorem and will provide a proof.

Theorem 1 *In Hoeffding trees where sufficient statistics of numerical attributes are represented by Gaussian distributions, and purity gain is defined as Eq.* (1) *and misclassification error specified in Eq.* (7) *is used as impurity measure, the highest gain always occurs where two Gaussian distributions intersect each other.*

Proof For simplicity, attribute index is removed from equations.

Simplifying Eq. (1):

$$G(S, x) = g(S) - \frac{1}{|S|}\big[|S_l(x)|g(S_l(x)) + |S_r(x)|g(S_r(x))\big]. \tag{12}$$

Substituting Eq. (7):

$$G(S,x) = g(S) - \frac{1}{|S|}\left[|S_l(x)|\left(1 - \max_{k\in\{1,\dots,K\}}\{p_k(S_l(x))\}\right)\right.$$
$$\left. + |S_r(x)|\left(1 - \max_{k\in\{1,\dots,K\}}\{p_k(S_r(x))\}\right)\right]. \tag{13}$$

There are two max operations in this equation. Let assume $x \in (x_1, x_2)$ in which, class k_1 wins the first max operation and class k_2 wins the second one and outside of this range, the winners change. Then

$$G(S,x) = g(S) - \frac{1}{|S|}\left[|S_l(x)|\left(1 - \frac{|S_l(x)|_{k_1}}{|S_l(x)|}\right) + |S_r(x)|\left(1 - \frac{|S_r(x)|_{k_2}}{|S_r(x)|}\right)\right]. \tag{14}$$

Using Eq. (11):

$$G(S,x) = g(S) - 1 + \frac{|S_l(x)|_{k_1} + |S_r(x)|_{k_2}}{|S|}. \tag{15}$$

Substituting Eqs. (9) and (10):

$$G(S,x) = g(S) - 1 + \frac{|S|_{k_1}F_{k_1}(x) + |S|_{k_2}(1 - F_{k_2}(x))}{|S|}. \tag{16}$$

To find the maximum gain, derivative of G with respect to x should be set to zero:

$$\frac{\partial G(S,x)}{\partial x} = \frac{\partial}{\partial x}\left[|S|_{k_1}F_{k_1}(x) + |S|_{k_2}(1 - F_{k_2}(x))\right] \tag{17}$$
$$= |S|_{k_1} f_{k_1}(x) - |S|_{k_2} f_{k_2}(x)$$
$$= 0, \tag{18}$$

where $f_k(x)$ represents the PDF of Gaussian distribution of kth class at point x. This concludes to

$$|S|_{k_1} f_{k_1}(x) = |S|_{k_2} f_{k_2}(x), \tag{19}$$

which indicates the intersection of two Gaussian distributions (multiplied by the number of instances belong to each one).

It was assumed that $x \in (x_1, x_2)$; therefore, the boundaries of this interval should also be tested. For a point x_0 to maximize G, it is required that

$$\begin{cases} \left.\frac{\partial G(S,x)}{\partial x}\right|_{x=x_0^-} > 0 \\ \left.\frac{\partial G(S,x)}{\partial x}\right|_{x=x_0^+} < 0 \end{cases}. \tag{20}$$

where

$$\begin{cases} x_0^+ = x_0 + \epsilon \\ x_0^- = x_0 - \epsilon \end{cases}, \epsilon > 0 \text{ and approaches zero.} \tag{21}$$

Since x_0 is a boundary point, the winner of one of the max operations changes in its neighborhood:

- Let k_1 and k_2 be the winners of first and second max operations at x_0^+ and k_3 and k_2 be the winners of first and second max operations at x_0^-. Using Eq. (18) in Eq. (20):

$$\begin{cases} |S|_{k_3} f_{k_3}(x_0) - |S|_{k_2} f_{k_2}(x_0) > 0 \\ |S|_{k_1} f_{k_1}(x_0) - |S|_{k_2} f_{k_2}(x_0) < 0 \end{cases}. \tag{22}$$

  Therefore

$$|S|_{k_1} f_{k_1}(x_0) < |S|_{k_3} f_{k_3}(x_0). \tag{23}$$

  Since k_1 wins the first max operation at x_0^+ and k_3 wins at x_0^-, therefore

$$\begin{cases} p_{k_1}(S_l(x_0^-)) < p_{k_3}(S_l(x_0^-)) \\ p_{k_1}(S_l(x_0^+)) > p_{k_3}(S_l(x_0^+)) \end{cases} \tag{24}$$

$$|S_l(x_0^+)|_{k_1} - |S_l(x_0^-)|_{k_1} > |S_l(x_0^+)|_{k_3} - |S_l(x_0^-)|_{k_3} \tag{25}$$

$$\frac{\partial(|S|_{k_1} F_{k_1}(x))}{\partial x}\bigg|_{x=x_0} > \frac{\partial(|S|_{k_3} F_{k_3}(x))}{\partial x}\bigg|_{x=x_0} \tag{26}$$

$$|S|_{k_1} f_{k_1}(x_0) > |S|_{k_3} f_{k_3}(x_0), \tag{27}$$

  which clearly contradicts Eq. (23).
- Let k_1 and k_2 be the winners of first and second max operations at x_0^+ and k_1 and k_3 be the winners of first and second max operations at x_0^-. Using Eq. (18) in Eq. (20):

$$\begin{cases} |S|_{k_1} f_{k_1}(x_0) - |S|_{k_3} f_{k_3}(x_0) > 0 \\ |S|_{k_1} f_{k_1}(x_0) - |S|_{k_2} f_{k_2}(x_0) < 0 \end{cases}. \tag{28}$$

  Therefore

$$|S|_{k_3} f_{k_3}(x_0) < |S|_{k_2} f_{k_2}(x_0). \tag{29}$$

  Since k_2 wins the second max operation at x_0^+ and k_3 wins at x_0^-, therefore

$$\begin{cases} p_{k_2}(S_r(x_0^-)) < p_{k_3}(S_r(x_0^-)) \\ p_{k_2}(S_r(x_0^+)) > p_{k_3}(S_r(x_0^+)) \end{cases} \tag{30}$$

$$|S_r(x_0^+)|_{k_2} - |S_r(x_0^-)|_{k_2} > |S_r(x_0^+)|_{k_3} - |S_r(x_0^-)|_{k_3} \tag{31}$$

$$\frac{\partial(|S|_{k_2}(1 - F_{k_2}(x)))}{\partial x}\bigg|_{x=x_0} > \frac{\partial(|S|_{k_3}(1 - F_{k_3}(x)))}{\partial x}\bigg|_{x=x_0} \tag{32}$$

$$|S|_{k_2} f_{k_2}(x_0) < |S|_{k_3} f_{k_3}(x_0), \tag{33}$$

which contradicts Eq. (29).

Therefore, maximum gain occurs exactly at an intersection of Gaussian distributions. □

Theorem 1 implies that for a numerical attribute with Gaussian approximation, only intersection points are needed to be checked for split. This enables one to accurately and efficiently calculate the best attribute values for splitting.

Moreover, this theorem implies that intersection points in which the two distributions are not both winners of the max operations, can be rejected. This decreases the number of candidate points (although the checking of this phenomenon itself can be costly).

The intersection points of two Gaussian distributions can be calculated as follows:

$$n_1 \frac{1}{\sqrt{2\pi}\sigma_1} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} = n_2 \frac{1}{\sqrt{2\pi}\sigma_2} e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}}, \tag{34}$$

where n_1 and n_2 represent the number of instances belong to the first and second distributions respectively, and μ and σ represent mean and standard deviation. The solutions of this equation are the solutions of the following quadratic equation:

$$\left(\frac{1}{\sigma_1^2} - \frac{1}{\sigma_2^2}\right)x^2 - 2\left(\frac{\mu_1}{\sigma_1^2} - \frac{\mu_2}{\sigma_2^2}\right)x + \left(\frac{\mu_1^2}{\sigma_1^2} - \frac{\mu_2^2}{\sigma_2^2} - 2ln\frac{n_1\sigma_2}{n_2\sigma_1}\right) = 0. \tag{35}$$

3.3 Practical Usage of the Proposed Method

In Hoeffding tree, each node first finds the best split for each attribute and then checks whether a split is required by comparing two best attributes. In VFDT (which is an improved version of Hoeffding tree), this process of checking is performed every n steps. This technique greatly increases the speed.

In [18], as explained earlier, to check whether a split is required, the range between minimum and maximum values arrived is divided into some bins; then, the gain for each bin is calculated. Authors suggested that best performance is achieved by using 10 bins. Therefore, in this paper, 10 bins are used similarly. We refer to this approach as “mDT-bin” (misclassification-based decision tree with bins).

Theorem 1 can be used in this scheme with three different approaches:

- *mDT-intersect* In this approach, to check for split, only intersections of Gaussian distributions are calculated and evaluated. If no intersection occurs, no check would be performed.

- *mDT-delayed* The problem with mDT-intersect is that computing intersection points can be costly. For a K class problem, each attribute has K distributions and therefore at most $\binom{K}{2}$ intersections can occur. This means the computational complexity is $O(K^2)$.

  A workaround is to find intersections only when a split is going to happen. In this approach, first the bin method is used to find the best value and calculate gain. Then the difference between the best two gains is compared with split threshold [Eq. (8)]. If the conditions are satisfied (a split is going to happen), intersections are calculated and gains are evaluated for these new points. Finally split is performed on the best point.
- *mDT-combined* In this approach, both bin and intersection points are considered all the time. The calculation cost will be much higher, but perhaps the performance would increase.

4 Experimental Results

In this section, some experiments are performed on the proposed methods. First, the methods are tested on synthetic datasets. The use of synthetic datasets is that the number of attributes and classes can be changed and this enables one to perform better comparisons. Next, some real datasets are used. Real datasets better show the performance of models on real world applications.

4.1 Experiment on Synthetic Data

4.1.1 Gaussian Dataset

To generate this dataset, the algorithm "make_classification" in scipy library was used [24]. This algorithm initially creates clusters of points normally distributed (standard deviation = 1) about vertices of an n-dimensional hypercube; then assigns an equal number of clusters to each class.

The experiments are performed on different number of attributes and classes. For each specific number of attributes and classes, 10 datasets with different random seeds where generated, and a tenfold cross validation is performed on each dataset; finally, the 100 results are averaged. Results are presented in Table 1.

4.1.2 Decision Tree Dataset

Decision trees can also be used to generate data [4]. For this purpose, first a random DT is constructed. To construct a random DT, one begins with a leaf (root). If the depth of the tree is less than $d_{\min}$, the leaf is turned into a binary decision node and a random decision value and a numerical attribute are assigned to it. Otherwise, the node remains as a leaf with probability ω. If the node is remained a leaf, a random class is assigned to it. All the nodes that are at depth $d_{\max}$ remain as leaf. When the

Table 1 The result of experiments performed on Gaussian datasets

D	K	Method	Train accuracy	Test accuracy	Number of nodes	Depth of tree	Training time (s)
4	2	mDT-bin	0.9940	0.9920	66.24	14.27	2.99
		mDT-intersect	0.9943	0.9937	5.58	3.17	1.61
		mDT-delayed	0.9953	0.9927	51.26	11.57	2.97
		mDT-combined	0.9956	0.9918	39.14	9.27	2.98
6	2	mDT-bin	0.9923	0.9897	70.46	17.95	4.16
		mDT-intersect	0.9870	0.9848	13.82	5.27	2.31
		mDT-delayed	0.9949	0.9911	52.72	12.88	4.11
		mDT-combined	0.9956	0.9904	37.54	9.51	4.11
8	2	mDT-bin	0.9936	0.9902	71.94	18.05	5.36
		mDT-intersect	0.9884	0.9857	12.92	4.90	2.89
		mDT-delayed	0.9940	0.9877	55.04	13.67	5.29
		mDT-combined	0.9942	0.9877	41.48	10.56	5.29
10	2	mDT-bin	0.9909	0.9854	77.14	17.58	5.81
		mDT-intersect	0.9907	0.9883	12.88	4.86	3.51
		mDT-delayed	0.9922	0.9877	60.56	13.50	6.53
		mDT-combined	0.9936	0.9870	47.68	11.19	6.54
6	4	mDT-bin	0.9888	0.9842	92.70	15.52	4.49
		mDT-intersect	0.9930	0.9848	40.28	8.29	2.61
		mDT-delayed	0.9920	0.9856	73.92	12.58	3.84
		mDT-combined	0.9935	0.9847	51.74	9.37	3.98
8	4	mDT-bin	0.9853	0.9762	89.12	13.95	5.92
		mDT-intersect	0.9860	0.9743	50.96	9.29	3.70
		mDT-delayed	0.9871	0.9768	78.00	12.47	5.84
		mDT-combined	0.9891	0.9765	64.04	10.84	6.19

Table 1 continued

D	*K*	Method	Train accuracy	Test accuracy	Number of nodes	Depth of tree	Training time (s)
10	4	mDT-bin	0.9774	0.9592	93.18	13.79	7.51
		mDT-intersect	0.9778	0.9587	60.90	10.29	4.54
		mDT-delayed	0.9794	0.9612	84.46	12.97	7.40
		mDT-combined	0.9804	0.9609	72.04	11.56	8.04
8	6	mDT-bin	0.9789	0.9602	96.04	11.93	6.41
		mDT-intersect	0.9826	0.9607	70.26	9.49	4.53
		mDT-delayed	0.9824	0.9640	88.12	11.03	6.40
		mDT-combined	0.9833	0.9618	76.64	10.21	7.74
10	6	mDT-bin	0.9707	0.9486	107.50	12.49	8.22
		mDT-intersect	0.9766	0.9481	84.68	10.50	5.95
		mDT-delayed	0.9759	0.9505	100.34	11.57	8.22
		mDT-combined	0.9772	0.9489	89.76	10.94	10.35
8	8	mDT-bin	0.9795	0.9651	105.58	12.32	6.56
		mDT-intersect	0.9878	0.9691	69.52	9.69	5.23
		mDT-delayed	0.9863	0.9716	95.24	11.38	5.69
		mDT-combined	0.9882	0.9692	75.38	10.02	8.43
10	8	mDT-bin	0.9607	0.9318	115.16	11.51	8.93
		mDT-intersect	0.9690	0.9362	102.18	10.76	8.85
		mDT-delayed	0.9681	0.9381	112.62	11.21	9.29
		mDT-combined	0.9685	0.9356	106.14	10.93	14.06
10	10	mDT-bin	0.9601	0.9214	118.90	10.46	9.71
		mDT-intersect	0.9676	0.9270	111.66	10.21	12.83
		mDT-delayed	0.9643	0.9258	118.78	10.59	10.21
		mDT-combined	0.9679	0.9261	114.08	10.32	18.51

D and *K* represent number of attributes and number of classes respectively

DT is constructed, one can generate random instances and by sorting them down the tree, the label of the instance would be obtained.

Similar to Gaussian datasets, the experiments are performed on different number of attributes and classes. For each specific number of attributes and classes, 3 random datasets for each value of $\omega = \{0.1, 0.15, 0.2, 0.25\}$ is generated (12 in total); and a tenfold cross validation is performed on each dataset. Finally all the 120 results are averaged. Results are presented in Table 2.

4.1.3 Discussion About the Results

By observing the results achieved on synthetic datasets, the following conclusions can be made:

- mDT-intersect
  - The first observation is that mDT-intersect always produces smaller tree. The reason is that the split values are very precise and therefore fewer splits are required.
  - In mDT-intersect, at most $\binom{K}{2}$ candidate points are checked every time. Therefore in problems with $K < 5$, mDT-intersect always examines fewer points than mDT-bin with 10 bins. However in reality, some Gaussian distributions do not intersect each other. Therefore this method can be faster even if $K > 5$. In these two experiments, mDT-intersect is faster on datasets with $K < 10$ and $K < 8$ respectively.
  - This method outperforms mDT-bin on most cases where $K > 2$. The reason that mDT-intersect achieves lower accuracy on two class problems is that mostly, no intersections occur. Therefore, the resulting tree is very small. Although, the performance achieved with such small trees seems remarkable. This problem can also occur in cases where $K > 2$, since in these cases, by splitting the nodes, the number of classes that a child sees is fewer; therefore, in deeper nodes, two class problems would emerge. This is the reason that mDT-intersect sometimes performs worse than other methods.
- mDT-delayed
  - From size and time perspective, mDT-delayed behaves very similar to mDT-bin. However the size is usually smaller (due to the finer splits achieved by intersections).
  - In most cases, this method achieves higher accuracy than mDT-bin. In cases where it performs worse than mDT-bin, the training accuracy is higher; which suggests that mDT-delayed is overfitted in these cases.
  - This method's accuracy is higher than that of mDT-intersect in some cases and lower in others. In Hoeffding trees, split threshold is the criterion to determine whether split is required or not; however, in mDT-intersect, there is another criterion: whether intersection happens or not. In other words, it is possible that splitting on some bin points might result in a gain higher than threshold, yet no intersection happens between distributions. In such cases, mDT-intersect does not perform a split and waits for more data, however mDT-delayed does

Table 2 The result of experiments performed on decision tree datasets

D	*K*	Method	Train accuracy	Test accuracy	Number of nodes	Depth of tree	Training time (s)
4	2	mDT-bin	0.92435	0.89125	106.90	12.33	2.31
		mDT-intersect	0.85820	0.84575	22.03	5.08	0.95
		mDT-delayed	0.92651	0.88970	107.68	12.76	2.33
		mDT-combined	0.92883	0.89108	103.57	12.49	2.36
6	2	mDT-bin	0.90146	0.85667	116.37	12.52	3.34
		mDT-intersect	0.85664	0.84425	17.25	4.45	1.09
		mDT-delayed	0.90222	0.85900	115.30	12.93	3.37
		mDT-combined	0.90394	0.85750	110.52	12.72	3.40
8	2	mDT-bin	0.88999	0.83592	122.55	12.78	4.27
		mDT-intersect	0.83213	0.81367	22.75	4.92	1.71
		mDT-delayed	0.89155	0.83608	122.23	13.03	4.36
		mDT-combined	0.89411	0.83667	117.57	12.78	4.40
10	2	mDT-bin	0.89165	0.83000	119.53	12.61	4.14
		mDT-intersect	0.84126	0.81883	22.68	5.23	1.59
		mDT-delayed	0.89447	0.83708	119.60	13.16	5.23
		mDT-combined	0.89618	0.83933	115.17	12.93	5.27
6	4	mDT-bin	0.87536	0.82200	120.82	12.05	4.25
		mDT-intersect	0.88764	0.83783	97.32	11.70	2.07
		mDT-delayed	0.88122	0.82842	120.35	12.14	4.36
		mDT-combined	0.88689	0.83258	114.47	12.15	5.05
8	4	mDT-bin	0.83687	0.76533	127.10	12.55	5.71
		mDT-intersect	0.84949	0.77975	104.47	13.02	2.82
		mDT-delayed	0.8455	0.77208	125.93	13.18	5.91
		mDT-combined	0.85063	0.77650	122.20	13.53	7.09

Table 2 continued

D	K	Method	Train accuracy	Test accuracy	Number of nodes	Depth of tree	Training time (s)
10	4	mDT-bin	0.85337	0.79117	121.77	11.79	6.89
		mDT-intersect	0.86243	0.79592	103.48	11.98	2.86
		mDT-delayed	0.85543	0.78942	123.50	12.11	7.15
		mDT-combined	0.86170	0.79242	118.23	12.23	8.50
8	6	mDT-bin	0.83154	0.76550	124.03	12.46	6.77
		mDT-intersect	0.84060	0.77308	110.70	12.96	5.63
		mDT-delayed	0.83522	0.77150	124.43	12.91	7.14
		mDT-combined	0.84199	0.77192	120.27	13.27	11.09
10	6	mDT-bin	0.82249	0.75742	124.15	12.49	8.44
		mDT-intersect	0.83666	0.76425	111.20	13.30	6.75
		mDT-delayed	0.82643	0.75658	124.22	13.00	9.00
		mDT-combined	0.83673	0.76733	119.37	13.03	13.50
8	8	mDT-bin	0.81821	0.75333	126.55	12.36	7.63
		mDT-intersect	0.84009	0.77383	115.37	13.54	9.47
		mDT-delayed	0.83169	0.76542	126.07	12.97	8.16
		mDT-combined	0.83865	0.77192	120.45	13.39	15.49
10	8	mDT-bin	0.83317	0.77050	122.65	11.56	9.07
		mDT-intersect	0.84864	0.78500	113.17	12.31	11.38
		mDT-delayed	0.84167	0.78108	123.93	11.90	6.87
		mDT-combined	0.84405	0.78050	119.70	12.41	18.93
10	10	mDT-bin	0.82606	0.76242	122.30	12.29	9.74
		mDT-intersect	0.84784	0.78383	110.70	13.58	15.54
		mDT-delayed	0.83984	0.78117	122.95	12.92	10.59
		mDT-combined	0.84667	0.78033	116.18	13.92	23.59

D and K represent number of attributes and number of classes respectively perform a split. Having more data before splitting can sometimes be beneficial since a better judgment can be made; however, sometimes it is better to make a not very good split now and perform good splits later on.

- mDT-combined
  - This method performs very slowly (as expected).
  - In most cases, mDT-combine achieves higher train accuracy than other methods; yet in these cases achieves lower test accuracy. This means that mDT-combine is more prone to overfitting. Generally, increasing the number of candidate points might not be desirable since it results in more overfitting (in [18] authors compare VFDTs with 10, 100 and 1000 bins and conclude that the one with 10 bins performs better).

4.2 Experiment on Real Data

In this section, proposed methods are tested on some real datasets (datasets are available at [25]). Both small and large datasets are chosen to also examine overfitting. The results are presented in Table 3.

Conclusions that can be made about these results are similar to those of synthetic datasets. mDT-intersect produces smaller trees and is usually faster and in most cases outperforms mDT-bin. mDT-delayed has a similar performance to mDT-intersect except in cases with higher number of classes, is faster. mDT-combined is more prone to overfitting since in most cases it has lower test accuracy despite having higher training accuracy.

5 Conclusions

In this paper, a new approach to analytically calculate the best split values in DT for streaming data is proposed. It was showed with a theorem that by using Gaussian distribution as sufficient statistics and misclassification error as impurity measure, the best split value is where two Gaussian distributions intersect each other. Then, three different approaches were proposed to use this theorem.

Experiments were performed on these methods on both synthetic and real datasets. It was observed that the proposed methods can create smaller trees and learn faster. Also these methods outperformed previous approach in most cases.

This research can be continued in two directions: first, it is interesting to know whether similar theorems exist for other impurity measures with other distributions. Second, it would be better to represent sufficient statistics with more complex models. A Gaussian distribution is not a good representation in most cases (this can cause inconsistencies in the results). Gaussian mixture model is a much better representation. It would be interesting to see how such methods can be used in incremental decision trees and whether similar theorems can be found for these methods.

Table 3 The result of experiments performed on some real datasets

Dataset	*D*	*K*	Method	Train accuracy	Test accuracy	Number of nodes	Depth of tree	Training time (s)
Cover type	10	7	mDT-bin	0.75998	0.61641	2363.0	27.2	155.98
			mDT-intersect	0.75913	0.62333	2277.4	31.1	129.31
			mDT-delayed	0.75806	0.62448	2400.2	26.4	162.97
			mDT-combined	0.76309	0.62702	2408.2	28.6	161.48
MAGIC Gamma Telescope	10	2	mDT-bin	0.86233	0.81141	1146.0	41.6	34.16
			mDT-intersect	0.85112	0.80894	285.6	27.4	16.99
			mDT-delayed	0.85554	0.80142	1147.8	42.2	34.53
			mDT-combined	0.85790	0.80794	1129.6	46.1	34.51
HTRU_2	8	2	mDT-bin	0.98291	0.97647	1327.2	45.5	63.45
			mDT-intersect	0.98069	0.97786	92.6	14.2	24.52
			mDT-delayed	0.98366	0.97462	1165.2	37.9	74.01
			mDT-combined	0.98447	0.97451	974.2	38.8	76.26
Blood Transfusion Service Center	4	2	mDT-bin	0.81780	0.76351	123.4	15.3	6.30
			mDT-intersect	0.78368	0.76891	20.0	5.7	2.11
			mDT-delayed	0.81869	0.75675	122.6	15.6	6.06
			mDT-combined	0.81944	0.75675	122.8	15.5	6.14
Wireless indoor localization	7	4	mDT-bin	0.98239	0.96900	196.2	27.3	28.40
			mDT-intersect	0.97889	0.97000	50.0	8.9	5.89
			mDT-delayed	0.98233	0.96850	148.4	22.8	27.21
			mDT-combined	0.98283	0.96850	126.6	21.6	28.09

Table 3 continued

Dataset	*D*	*K*	Method	Train accuracy	Test accuracy	Number of nodes	Depth of tree	Training time (s)
Iris	4	3	mDT-bin	0.97704	0.96000	18.4	6.5	2.59
			mDT-intersect	0.98222	0.96667	11.2	5.6	1.34
			mDT-delayed	0.98000	0.96000	17.2	6.2	2.53
			mDT-combined	0.98296	0.96000	16.0	5.9	2.59
Seeds	7	3	mDT-bin	0.95185	0.92857	32.4	7.9	4.03
			mDT-intersect	0.96455	0.94286	23.4	6.2	1.86
			mDT-delayed	0.95820	0.94762	29.0	7.8	3.85
			mDT-combined	0.96349	0.94286	27.4	7.4	3.81
Ecoli	7	8	mDT-bin	0.85116	0.81212	36.2	10.3	6.91
			mDT-intersect	0.86271	0.80000	35.8	10.4	8.63
			mDT-delayed	0.86040	0.80909	36.8	10.8	7.62
			mDT-combined	0.86403	0.80606	36.8	10.8	13.77

D and *K* represent number of numerical attributes and number of classes respectively

References

1. Tidke B, Mehta R (2018) A comprehensive review and open challenges of stream big data. In: Pant M, Ray K, Sharma TK, Rawat S, Bandyopadhyay A (eds) Soft computing: theories and applications. Springer, Berlin, pp 89–99
2. Crawford SL (1989) Extensions to the cart algorithm. Int J Man-Mach Stud 31(2):197–217
3. Utgoff PE, Berkman NC, Clouse JA (1997) Decision tree induction based on efficient tree restructuring. Mach Learn 29(1):5–44
4. Domingos P, Hulten G (2000) Mining high-speed data streams. In: Proceedings of the 6th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 71–80
5. Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: Proceedings of the 7th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 97–106
6. Zhang P, Zhou C, Wang P, Gao BJ, Zhu X, Guo L (2015) E-tree: an efficient indexing structure for ensemble models on data streams. IEEE Trans Knowl Data Eng 27(2):461–474
7. Gomes HM, Bifet A, Read J, Barddal JP, Enembreck F, Pfharinger B, Holmes G, Abdessalem T (2017) Adaptive random forests for evolving data stream classification. Mach Learn 106(9–10):1469–1495
8. Bifet A, Zhang J, Fan W, He C, Zhang J, Qian J, Holmes G, Pfahringer B (2017) Extremely fast decision tree mining for evolving data streams. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 1733–1742
9. Segatori A, Marcelloni F, Pedrycz W (2018) On distributed fuzzy decision trees for big data. IEEE Trans Fuzzy Syst 26(1):174–192
10. Lo W-T, Chang Y-S, Sheu R-K, Chiu C-C, Yuan S-M (2014) Cudt: a cuda based decision tree algorithm. Sci World J 2014 Article ID 745640
11. De Rosa R, Cesa-Bianchi N (2015) Splitting with confidence in decision trees with application to stream mining. In: 2015 international joint conference on neural networks (IJCNN). IEEE, pp 1–8
12. Rutkowski L, Jaworski M, Pietruczuk L, Duda P (2015) A new method for data stream mining based on the misclassification error. IEEE Trans Neural Netw Learn Syst 26(5):1048–1059
13. Jaworski M, Duda P, Rutkowski L (2018) New splitting criteria for decision trees in stationary data streams. IEEE Trans Neural Netw Learn Syst 29(6):2516–2529
14. Dougherty J, Kohavi R, Sahami M (1995) Supervised and unsupervised discretization of continuous features. In: Machine learning proceedings. Elsevier, pp 194–202
15. Quinlan JR (1993) C4.5: Programming for machine learning. Morgan Kauffmann 38:48
16. Ramírez-Gallego S, García S, Mouriño-Talín H, Martínez-Rego D, Bolón-Canedo V, Alonso-Betanzos A, Benítez JM, Herrera F (2016) Data discretization: taxonomy and big data challenge. Wiley Interdiscip Rev Data Min Knowl Discov 6(1):5–21
17. Gama J, Pinto C (2006) Discretization from data streams: applications to histograms and data mining. In: Proceedings of the 2006 ACM symposium on applied computing. ACM, pp 662–667
18. Kirkby RB (2007) Improving hoeffding trees. Ph.D. dissertation, The University of Waikato
19. Salperwyck C, Lemaire V (2013) Incremental decision tree based on order statistics. In: The 2013 international joint conference on neural networks (IJCNN). IEEE, pp 1–8
20. Zhang Y, Cheung Y-M (2014) Discretizing numerical attributes in decision tree for big data analysis. In: 2014 IEEE international conference on data mining workshop. IEEE, pp 1150–1157
21. Jadhav SA, Kosbatwar S (2016) Concept-adapting very fast decision tree with misclassification error. Int J Adv Res Comput Eng Technol (IJARCET) 5(6):1763–1767
22. Krawczyk B, Wozniak M (2015) Weighted naive bayes classifier with forgetting for drifting data streams. In: 2015 IEEE international conference on systems, man, and cybernetics. IEEE, pp 2147–2152
23. Rutkowski L, Pietruczuk L, Duda P, Jaworski M (2013) Decision trees for mining data streams based on the Mcdiarmid's bound. IEEE Trans Knowl Data Eng 25(6):1272–1279
24. Jones E, Oliphant T, Peterson P et al (2001) SciPy: open source scientific tools for Python. http://www.scipy.org/. Accessed 27 May 2019
25. Dheeru D, Karra Taniskidou E (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml. Accessed 20 Aug 2018