Journal of Big Data

RESEARCH Open Access

Big data decision tree for continuous-valued attributes based on unbalanced cut points

Shixiang Ma[1] and Junhai Zhai[1*]

*Correspondence:
mczjh@126.com

[1] Hebei Key Laboratory of Machine Learning and Computational Intelligence, College of Mathematics and Information Science, Hebei University, Baoding, China

Abstract

The decision tree is a widely used decision support model, which can quickly mine effective decision rules based on the dataset. The decision tree induction algorithm for continuous-valued attributes, based on unbalanced cut points, is efficient for mining decision rules; however, extending it to big data remains an unresolved. In this paper, two solutions are proposed to solve this problem: the first one is based on partitioning instance subsets, whereas the second one uses partitioning attribute subsets. The crucial of these two solutions is how to find the global optimal cut point from the set of local optimal cut points. For the first solution, the calculation of the Gini index of the cut points between computing nodes and the selection of the global optimal cut point by communication between these computing nodes is proposed. However, in the second solution, the division of the big data into subsets using attribute subsets in a way that all cut points of an attribute are on the same map node is proposed, the local optimal cut points can be found in this map node, then the global optimal cut point can be obtained by summarizing all local optimal cut points in the reduce node. Finally, the proposed solutions are implemented with two big data platforms, Hadoop and Spark, and compared with three related algorithms on four datasets. Experimental results show that the proposed algorithms can not only effectively solve the scalability problem, but also have lowest running time, the fastest speed and the highest efficiency under the premise of preserving the classification performance.

Keywords: Big data, Decision tree, Cut point, Hadoop, Spark

Introduction

In the field of machine learning, big data refers to data that is too large to be treated using traditional machine learning algorithms. Therefore big data poses huge challenges to traditional machine learning algorithms [1]. It is of great theoretical and application interest to study how to extend machine learning algorithms to handle big data. In previous studies, some researchers have extended some machine learning algorithms to the big data environment. For instance, as early as 2006, Chu et al. [2] deeply and extensively studied the scalability of machine learning algorithms for big data scenarios, and developed a broadly applicable parallel programming method by adapting the MapReduce paradigm; as a result, the proposed method can be easily

applied to several learning algorithms, including locally weighted linear regression, k-means, logistic regression, naive Bayes, support vector machine, independent component analysis, principal component analysis, Gaussian discriminant analysis, expectation maximum algorithm, and back-propagation algorithm. Moreover, He et al. [3] considered the parallel implementation of several classification algorithms based on MapReduce; thus, they implemented the parallel K-Nearest Neighbor (K-NN) algorithm, the parallel naive Bayesian network, and the parallel decision tree, respectively. As for Xu et al. [4], they extended the k-means algorithms to big data environments with MapReduce; therefore, they proposed the k-means++ algorithm, which can not only address the problem of optimal selection of parameter K but can also improve the scalability and efficiency of the k-means algorithm. In addition, Duan et al. [5] proposed a parallel multi-classification algorithm for big data using the Extreme Learning Machine (ELM), and implemented it by the big data platform, Spark. In the literature, there has been some work on the scalability of the decision tree algorithms in the big data context. For instance, three parallel implementations of C4.5 decision tree algorithm by MapReduce were proposed in Wang and Gao [6], Mu et al. [7], and Dai and Ji [8], respectively. Furthermore, Wand et al. [9] conducted a study on optimizing and parallelizing of the C4.5 algorithm on the Spark platform, and proposed a Spark-based parallel C4.5 decision tree algorithm. Yuan et al. [10] also investigated the optimization of the C4.5 decision tree algorithm, but unlike [9], it is not on Spark platform but on MapReduce. Chern et al. [11] proposed a decision tree credit assessment approach to solve the credit assessment problem in a big data environment. Wang et al. [12] investigated the expansion of decision tree algorithm in big data environment from the perspective of data mining, taking RainForest Tree and Bootstrapped Optimistic Algorithm for Tree construction as examples. Regarding big data machine learning, including big data decision tree learning, several researchers have presented in-depth and comprehensive reviews [13–17]. To sum up, we find that although there have been some extensions of decision tree algorithms in big data environments, the problem of the extension of continuous-valued decision tree algorithms, based on unbalanced cut points in big data environments is still not solved. Therefore, in this article, we proposed two solutions to solve this problem. This paper makes three significant contributions:

1. Two solutions, based on the divide-and-conquer strategy, are proposed to extend the continuous-value decision tree algorithm based on unbalanced cut points to big data scenarios. The first one is based on partitioning instance subsets, that is, the big data set is divided into several disjoint instance subsets. The second one is based on partitioning attribute subsets, that is, the big data set is divided into several disjoint attribute subsets. More intuitively, the first solution divides the big dataset into several disjoint subsets along horizontal direction; The second solution divides the big dataset into several disjoint subsets along vertical direction.
2. In the induction process of the continuous-valued big data decision tree based on unbalanced cut points, how to find the global optimal cut point from multiple local optimal cut points is a key problem. The second contribution of this paper is to solve this problem. There is only one global optimal cut point, which is relative to the big

dataset, and there are multiple local optimal cut points, which are relative to the subset of the big dataset. Each local optimal cut point is independently calculated from the local subset on a computing node of a big data platform;

3. Extensive experiments are conducted with the two big data platforms, Hadoop[1] and Spark[2] to verify the feasibility and effectiveness of the proposed method. Including the experiments compared with three closed-related algorithms on four big datasets, and the experiments on two artificial data sets to demonstrate the feasibility of the proposed algorithm.

The rest of this paper is organized as follows. In "Related work" section, we briefly review the related work. In "Two solutions" section, we describe the details of the proposed solutions. In "Experimental results and analysis" section, the experiments are conducted to demonstrate the feasibility and effectiveness of the proposed solution. At last, we conclude our work in "Conclusions" section.

Related work

As one of the top ten classical algorithms in machine learning [18], the decision tree has been widely used in classification and regression problems due to its fast learning speed and high prediction accuracy. However, in the big data environment, the decision tree cannot be completely constructed in memory due to the large amount of data to be processed, which requires a large amount of operation time (in case it can be realized). Accordingly, it is meaningful to study the way to extend the decision tree algorithm to a big data environment. Moreover, the methods of extending the decision tree algorithm and its variants to big data environments are mainly divided into two categories: distributed parallel crisp decision trees and distributed fuzzy decision trees.

The former is mainly based on the distributed extension of ID3, C4.5, CART, SPRING, and other baseline algorithms, so as to improve the accuracy and efficiency of the algorithm on big data; however, there is still a need to consider the computational complexity of the decision tree algorithm itself, the resident memory, and other characteristics of the optimization before implementing this algorithm. Based on MapReduce, two parallelized C4.5 algorithms were proposed in Wang and Gao [6] and Mu et al. [7]. Moreover, Genuer et al. [19] extended the random forest algorithm for big data classification. Based on the work of Genuer et al. [19], Juez-Gil et al. [20] proposed the rotation forest algorithm for big data classification. Furthermore, Shivaraju et al. [21] proposed a parallel decision tree based on attribute partition. In this case, the whole dataset was divided first into the training set for building a decision tree and the test set for testing the decision tree model according to certain rules. Then, the attributes of both datasets were partitioned, and a decision tree was generated on the training set according to each partition. Moreover, according to the partition of the test set and the training tree, the test tree was generated, and the weighted voting method was used in the generated test tree to deliver the final prediction classification result. To solve the time-consuming problem relative to the calculation of the information gain rate of the C4.5 algorithm, Yuan et al.

[1] https://hadoop.apache.org/.

[2] http://spark.apache.org/.

[22] proposed an improved C4.5 algorithm, which used the McLaughlin formula to calculate the information gain in order to improve computational efficiency. By comparing the improved C4.5 with the traditional C4.5 on the Hadoop platform, the results show that the improved algorithm has higher accuracy and efficiency. Added to that, Desai and Chaudhary studied the scalability of the distributed decision tree and proposed two distributed decision tree algorithms: DDTv1 [23] and DDTv2 [24]. In more detail, DDTv1 is a distributed implementation of the Hadoop-based Distributed Decision Tree (DDT) and the Spark-based Spark Tree (ST). As for DDTv2, it optimizes DDT and ST based on DDTv1 and makes a compromise between partitioning and accuracy. In other words, DDTv2 provides as much parallelism as possible without any loss of accuracy. For instance, Chen et al. [25] proposed a parallel random forest algorithm for big data; the tests were performed on Spark. To improve the generalization performance of the parallel random forest algorithm, it is optimized based on the hybrid approach combining data-parallel and task-parallel optimization. Therefore, Es-sabery et al. [26] proposed an improved ID3 algorithm and used it for the classification of Twitter big data. The main contributions of the proposed work included three points: (1) The information gain feature selector was used to reduce the dimensionality of Twitter's high-dimensional data; (2) Regarding the dimension reduction, the weighted information gain was used to replace the information gain in the ID3 algorithm as a heuristic to induce the decision trees; (3) The improved ID3 algorithm was implemented with the big data programming framework, MapReduce. Moreover, Jurczuk et al. [27] proposed a global induction of the classification trees for large-scale data mining using multi-GPU for accelerating the algorithm. Abuzaid et al. [28] introduced an optimized system for training deep decision trees at scale; it was based on partitioning the feature of the data along with a set of optimized data structures to reduce the CPU occupancy and communication costs of training. Chen et al. [29] designed a distributed decision tree algorithm and implemented it with the big data platform Spark. In addition, En-nattouh et al. [30] applied decision tree algorithm to select the optimal configuration and enhance parameter optimization in clustered platforms for Big Data. Specifically, the number of tasks for each node by analyzing the internal elements of YARN was first calculated, the decision tree is then used to find the optimal configuration. To reduce the time complexity in parsing big data decision tree, Liu et al. [31] designed a novel data structure termed as collision-partition tree, which can lead a more balanced tree structure, thus achieving the goal of reducing the computational time complexity. Jin et al. [32] introduced a sampling scheme with and without the replacement and designed an algorithm to improve the adaptation and generalization ability of classification rules in a big-data environment. Online decision tree algorithms can tackle the problem of big data learning by concurrently training with incoming samples and providing inference results. Based on this point, Lin et al. [33] proposed a high-performance and scalable online decision tree learning algorithm. Weinberg et al. [34] proposed an algorithm to select a representative decision tree from an ensemble of decision-tree models for fast big data classification.

The distributed fuzzy decision tree is an extension of the fuzzy decision tree in a big data environment for handling large-scale data learning. Compared to the distributed crisp decision tree, the research on the distributed fuzzy decision tree is relatively limited. Four representative works are Segatori et al. [15], Mu et al. [35], Wu et al. [36],

and Fernandez-Basso et al. [37]. In Segatori et al. [15], proposed a distributed fuzzy decision tree learning scheme based on the MapReduce programming model where the learning scheme can generate both binary and multiway fuzzy decision trees from this big data. The key idea of this learning scheme is that it uses fuzzy information entropy to discretize each Continuous-Valued attribute. In Mu et al. [35], proposed a parallel fuzzy rule-base based decision tree, the main contribution of this work lay in developing a parallel fusing fuzzy rule based classification system; moreover, the parallelization was implemented via MapReduce, and the ensemble was used to evaluate the obtained fuzzy rule-base. In [36], a Hadoop-based high fuzzy utility pattern mining algorithm was developed to discover high fuzzy utility patterns from big datasets. In Fernandez-Basso et al. [37], gave a Spark-based solution for discovering the fuzzy association rules associated with big data.

As far as we know, the problem of how to extend the continuous-valued decision tree induction algorithm based on unbalanced cut points to a big data environment, has not been solved; therefore the main goal of this paper is to solve this problem.

Two solutions

The goal of this paper is to extend the continuous-valued decision tree induction algorithm based on unbalanced cut points to big data scenarios. This section first briefly reviews the baseline algorithm, and then introduces the two solutions: one based on partitioning instance subsets, and the other based on partitioning attribute subsets.

The continuous-valued decision tree algorithm based on unbalanced cut points

The continuous-valued decision tree algorithm based on unbalanced cut points was proposed by Fayyad and Irani in 1992 [38], the core concept of this algorithm is the unbalanced cut point. Given a continuous-valued decision table $D = (U, A \cup Y)$, where U is a set of n instances in R^d, denoted by $U = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$, $\boldsymbol{x}_i \in R^d (1 \le i \le n)$, A is a set of d attributes, denoted by $A = \{a_1, a_2, \cdots, a_d\}$, $a_j \in R (1 \le j \le d)$, Y is a set of labels of instances in U, denoted by $Y = \{1, 2, \cdots, k\}$, k is the number of classes of instances in U. The intuitive representation of the decision table D is shown in Table 1. The values in column j (i.e., the values of attribute a_j) of Table 1 are sorted in ascending order, the sorted result is denoted by $x_{i_1 j}, x_{i_2 j}, \cdots, x_{i_n j}$. The mid-value t_s of $x_{i_s j}$ and $x_{i_{s+1} j}$ is called a cut point of attribute a_j, $1 \le s \le n-1$ and $1 \le j \le d$. Obviously, $\forall a_j \in A$, it has $n-1$ cut points. If the samples x_{i_s} and $x_{i_{s+1}}$ on both sides of the cut point t_s belong to different classes, then t_s is called the unbalanced cut point. Otherwise, it is called the balanced cut point.

Table 1 A decision table containing n instances

$\boldsymbol{x}$	$\boldsymbol{a_1}$	$\cdots$	$\boldsymbol{a_j}$	$\cdots$	$\boldsymbol{a_d}$	$\boldsymbol{y}$
$\boldsymbol{x}_1$	x_{11}	$\cdots$	x_{1j}	$\cdots$	x_{1d}	y_1
$\boldsymbol{x}_2$	x_{21}	$\cdots$	x_{2j}	$\cdots$	x_{2d}	y_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\boldsymbol{x}_n$	x_{n1}	$\cdots$	x_{nj}	$\cdots$	x_{nd}	y_n

Given a continuous-valued decision table $D = (U, A \cup Y)$, the instances in U are classified into k classes, the number of instances in class i is $n_i, 1 \leq i \leq k$. The Gini index of U is defined as:

$$Gini(U) = 1 - \sum_{i=1}^{k} p_i^2 \tag{1}$$

where $p_i = \frac{n_i}{n}$. Similar to information entropy, Gini index also measures the uncertainty of the classes to which the instances belong.

Given a continuous-valued decision table $D = (U, A \cup Y)$. Let t_s be a cut point of attribute $a_j, 1 \leq s \leq n-1, 1 \leq j \leq d$, it partitions U into two subsets U_1 and U_2. The Gini index of t_s is defined by

$$Gini(t_s, a_j, U) = \frac{|U_1|}{|U|} Gini(U_1) + \frac{|U_2|}{|U|} Gini(U_2). \tag{2}$$

It can be seen from (2) that the Gini index of the cut point t_s is the average value of the Gini index of the two subsets U_1 and U_2. In other words, the Gini index of the cut point t_s measures the uncertainty of the classes of the two subsets partitioned by the cut point t_s. Obviously, the smaller the Gini index of the cut point t_s, the more important the cut point t_s is.

For $\forall a_j \in A$, it has $n-1$ cut points $t_s (1 \leq s \leq n-1)$, the optimal cut point of attribute a_j is a cut point that satisfies the following condition:

$$t_j^* = \underset{1 \leq s \leq n-1}{argmin} \{Gini(t_s, a_j, U)\} \tag{3}$$

The global optimal cut point is defined as the optimal cut point with respect to the attribute set A, which is a cut point that satisfies the following condition.

$$t^* = \underset{1 \leq j \leq d}{argmin} \{Gini(t_j^*, a_j, U)\} \tag{4}$$

The attribute corresponding to the optimal cut point t^* is called the optimal extended attribute. Regarding the global optimal cut point, the following theorem holds [38].

Theorem 1 *The global optimal cut point must be a unbalanced cut point.*

This theorem suggests that when searching for the optimal cut point, only the Gini index of the unbalanced cut points need to be calculated, whereas it is unnecessary to calculate the Gini index of the balanced cut points, which can greatly reduce the computational complexity and improve the efficiency of the algorithm. For each attribute $a \in A$, let T_a is the set of all cut points of a. The pseudo-code of continuous-valued decision tree algorithm based on unbalanced cut points is given in Algorithm 1.

Algorithm 1: The continuous-valued decision tree algorithm based on unbalanced cut points

Input: A continuous-valued decision table $D = (U, A \cup Y)$.
Output: A decision tree.

Set $T = \varnothing$;
for (*each attribute* $a \in A$) **do**
  for (*each unbalanced cut point* $t \in T_a$) **do**
    Calculate $Gini(t, a, U)$ by Eq.(2);
  end
end
$t' = \underset{t \in T_a}{argmin} \{Gini(t, a, U)\}$;
$T = T \cup \{t'\}$;
$t^* = \underset{t' \in T}{argmin} \{Gini(t', a, U)\}$;
Select the attribute corresponding to t^* as the extended attribute;
Partition U by t^* into two subsets U_1 and U_2;
// $U_1 = \{\boldsymbol{x} | (\boldsymbol{x} \in U) \wedge (\boldsymbol{x}(a) \leq t^*)\}$, $U_2 = \{\boldsymbol{x} | (\boldsymbol{x} \in U) \wedge (\boldsymbol{x}(a) > t^*)\}$;
for $(i = 1; i \leq 2; i++)$ **do**
  if (*the instances in* U_i *belong to the same class*) **then**
    Generate a leaf node;
  else
    Goto 2;
  end
end
Output a decision tree.

When the continuous-valued decision table D is a big dataset, Algorithm 1 will become infeasible, so how can Algorithm 1 be extended to big data scenarios? The general strategy of big data processing is divide and conquer, that is, the big dataset is divided into several subsets which are distributed to different computing nodes for parallel processing. Big dataset can be divided into subsets in both horizontal and vertical directions. The partition in horizontal direction is to divide the big dataset into instance subsets, while the partition in vertical direction is to divide big dataset into attribute subsets. In this paper, based on horizontal and vertical partitioning, we give two solutions to extend the continuous-valued decision tree algorithm based on unbalanced cut points to big data scenario.

The solution based on partitioning instance subsets

In this solution, the cut points of an attribute $a_j (1 \leq j \leq d)$ fall into two categories:

1. Cut points within a subset, or a computing node, which are the cut points corresponding to the local data subset and can be viewed as local cut points;
2. Cut points between two subsets, or two computing nodes, which are naturally generated by dividing big dataset into several subsets.

If the number of computing nodes is m (i.e., the number of subsets or partitions of big dataset), then the number of this kind of cut points is $m - 1$ (see Fig. 1). If we artificially control the partition of big dataset, so that the $m - 1$ cut points are all balanced cut points, then it is unnecessary to calculate the Gini index of the $m - 1$ cut points, and the difficulty of processing will be reduced.

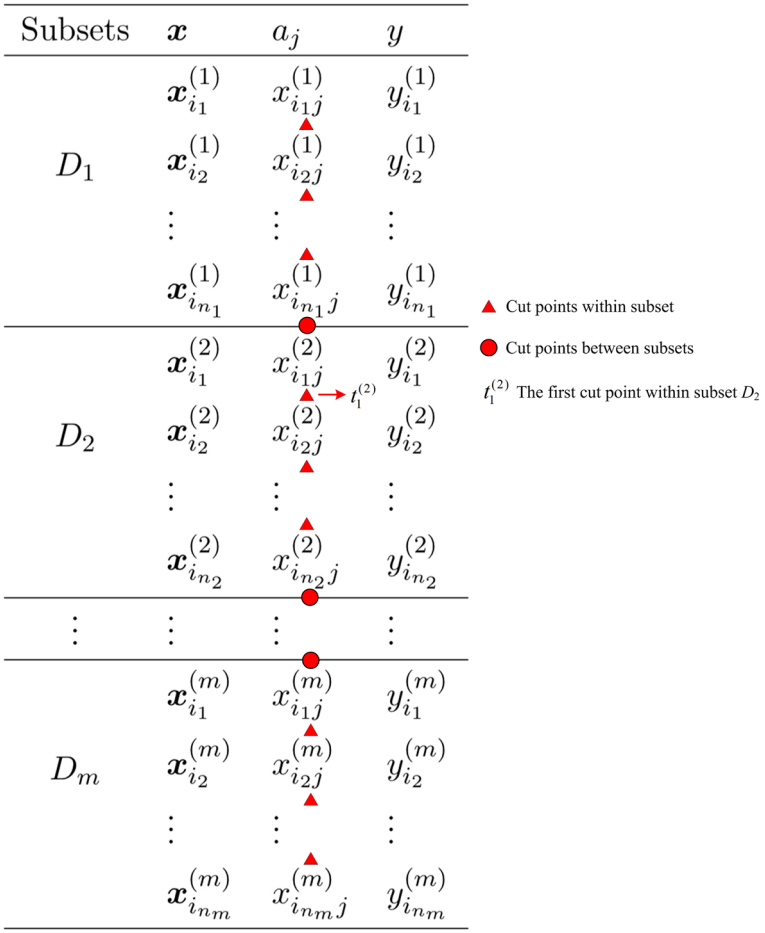

Fig. 1 Schematic diagram of cut points within subset and cut points between subsets

This solution needs to overcome the following two difficulties.

1. When calculating the Gini index of local cut points in a node, it needs to use the information of subsets on other nodes, so how to calculate the Gini index of local cut points across nodes is a difficult problem to be solved;
2. After the d local optimal cut points corresponding to the d attributes are found, then we need to find the global optimal cut point from the d local optimal cut points. Finding the global optimal cut point is another difficulty to be overcome.

The schematic diagram of the solution based on partitioning instance subsets is illustrated in Fig. 2.

The attribute corresponding to the global optimal cut point is the most important attribute, which is used as the extension attribute of the continuous-valued decision tree. In the following, we first take the cut point $t_1^{(2)}$ of the attribute a_j as an example to explain how to calculate the Gini index of the cut points within a subsets.

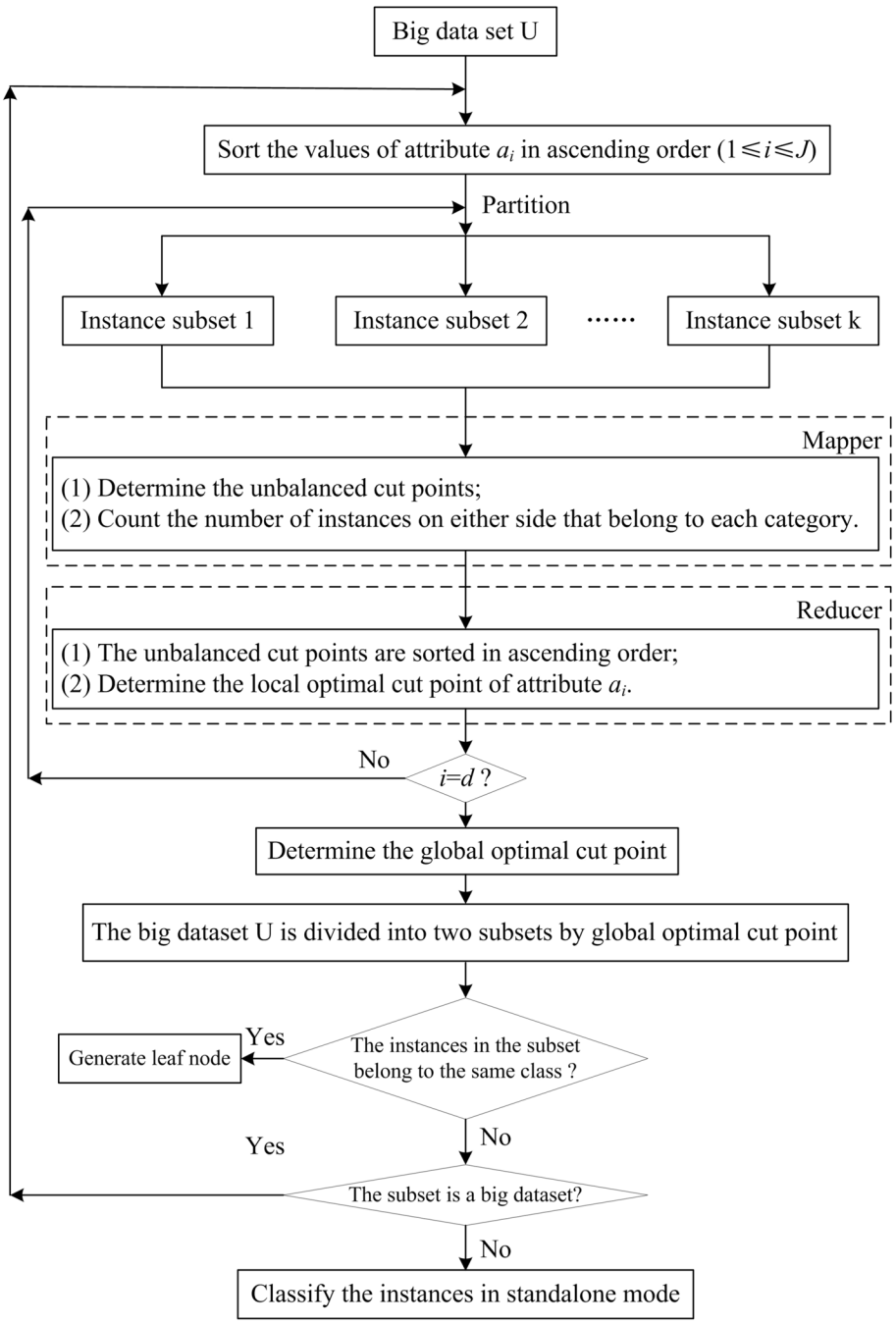

Fig. 2 The schematic diagram of the solution based on partitioning instance subsets

From Fig. 1, we can find that the cup point $t_1^{(2)}$ partitions the subset D_2 into two subsets $\{x_{i_1 j}^{(2)}\}$ and $\{x_{i_2 j}^{(2)}, \cdots, x_{i_{n_2} j}^{(2)}\}$. On the other hand, from the view point of big dataset, the cup point $t_1^{(2)}$ partitions the big dataset U into two subsets U_1 and U_2, where

$U_1 = D_1(a_j) \cup \{x_{i_1 j}^{(2)}\}$ and $U_2 = \{x_{i_2 j}^{(2)}, \cdots, x_{i_{n_2} j}^{(2)}\} \cup D_3(a_j) \cup \cdots \cup D_m(a_j)$. According to Eq. (2), after calculating the Gini index of U_1 and U_2, we can easily calculate the Gini index of the cut point $t_1^{(2)}$. To calculate the Gini index of U_1 and U_2, we only need to count the number of instances in U_1 and U_2 that belong to each category respectively, and it is easy to accomplish.

For the general case, let $t_p^{(q)}$ be the pth cut point of the qth subset D_q, $1 \le p \le n_q$, $1 \le q \le m$. The cut point $t_p^{(q)}$ partitions the subset D_q into two subsets $\{x_{i_1 j}^{(q)}, \cdots, x_{i_p j}^{(q)}\}$ and $\{x_{i_{p+1} j}^{(q)}, \cdots, x_{i_{n_q} j}^{(q)}\}$. At the same time, the cut point $t_p^{(q)}$ partitions the big data U into two subsets U_1 and U_2, where $U_1 = D_1(a_j) \cup \cdots \cup D_{q-1}(a_j) \cup \{x_{i_1 j}^{(q)}, \cdots, x_{i_p j}^{(q)}\}$, and $U_2 = \{x_{i_{p+1} j}^{(q)}, \cdots, x_{i_{n_q} j}^{(q)}\} \cup D_{q+1}(a_j) \cup \cdots \cup D_m(a_j)$. Similarly, in order to calculate the Gini index of the cut point $t_p^{(q)}$, we only need to count the number of instances in U_1 and U_2 that belong to each category respectively. It should be noted that when we count the number of instances belonging to each class in the map phase, the number of instances in the local subset is counted in parallel on each node. After the map phase is completed, the statistical results of all nodes are summarized in the reduce phase to obtain the number of instances belonging to each class in the big dataset. The design of the corresponding map and reduce functions are given in Algorithms 2 and 3, respectively.

Algorithm 2: The map function

Input: Key-value pair $< k_1, v_1 >$.
Output: Key-value pair $< k_2, v_2 >$, where k_2 is the value of t_i^m, v_2 is the tuple pair $<$ SummaryL, SummaryR $>$.

1 Traverse all cut points, select unbalanced cut point t_i^m, and add it to T_j^m;
2 Count the number of instances belonging to different classes between cut points t_i^m and t_{i-1}^m, and store the results in SummaryL;
3 Count the number of instances belonging to different classes between cut points t_i^m and t_{i+1}^m, and store the results in SummaryR;
4 Output $< k_2, v_2 >$.

Algorithm 3: The reduce function

Input: Key-value pair $< k_2, v_2 >$.
Output: The optimal cut point t_j^*.

1 **for** ($each\ unbalanced\ cut\ point\ t_i^m \in T_j$) **do**
2 Summarize the number of instances belonging to different categories in U_L;
3 Summarize the number of instances belonging to different categories in U_R;
4 Calculate the Gini index of each unbalanced cut point;
5 Select the optimal cut point t_j^* for the current attribute;
6 **end**
7 Output t_j^*.

In the framework of MapReduce, The corresponding big data decision tree algorithm based on partitioning instance subsets is denoted by BS-CDT-MR, and its pseudo-code is given in Algorithm 4.

Algorithm 4: BS-CDT-MR Algorithm

Input: Big dataset U.
Output: Continuous-valued decision tree tree based on unbalanced cut points.

```
for (each attribute a_j ∈ A) do
    Sort the values of a_j in ascending order;
    Partition U into k subsets and deploy them to k computing nodes;
    Call Algorithm 2;
    Call Algorithm 3;
end
Select the global optimal cut point t*;
Partition the big dataset U into two subsets U_1 and U_2 by t*;
for (i = 1; i ≤ 2; i++) do
    if (the instances in U_i belong to same class) then
        Generate a leaf node;
    end
    else
        if (U_i is also a big dataset) then
            Repeat the above process;
        end
        else
            Directly construct the decision tree in the mode of standalone;
        end
    end
end
Output decision tree.
```

In the framework of Spark, the pseudo-code of the corresponding algorithm denoted by BS-CDT-SP is given in Algorithm 5.

Algorithm 5: BS-CDT-SP Algorithm

Input: Big dataset U.
Output: Continuous-valued decision tree tree based on unbalanced cut points.

```
for (each attribute a_j ∈ A) do
    Sort the values of a_j in ascending order;
    Partition dataset U by self-defined MInputFormat, and obtain
      trainInitRDD = sc.newAPIHadoopFile(U);
    Traverse trainInitRDD, select unbalanced cut points and count the number of instances
      belonging to different classes between two side of each unbalanced cut point tsRDD;
    Calculate the Gini index of each unbalanced cut point by
      giniTRDD = tsRDD.calculateGini();
    Obtain local optimal cut point t*_j = min(giniRDD), and add it to the set of local optimal
      cut points tRDD;
end
Select the global optimal cut point t*;
Partition the big dataset U into two subsets U_1 and U_2 by t*;
for (i = 1; i ≤ 2; i++) do
    if (the instances in U_i belong to same class) then
        Generate a leaf node;
    end
    else
        if (U_i is also a big dataset) then
            Repeat the above process;
        end
        else
            Directly construct the decision tree in the mode of standalone;
        end
    end
end
Output decision tree.
```

The solution based on partitioning attribute subsets

The partition based on attribute subsets is to use attribute subsets to divide the big dataset into several data subsets, intuitively speaking, that is, to divide the big dataset into several data subsets along the vertical direction. Each data subset corresponds to the result of a projection operation in the database system. Each data subset contains all instances, but each instance is represented only by the values of the attributes in the attribute subset, that is, the dimension of the instance vector is the potential of the attribute subset. For example, if a subset of attributes contains three attributes, then each instance is represented by the values of the three attributes, which is a three-dimensional vector. This solution consists of partitioning the dataset with attribute subsets and deploying these subsets in the computing node so that all cut points of the same attribute are on one computing node. Therefore, the counting of the unbalanced cut points and the calculation of the Gini index are similar to those in the standalone environment, and the counting and calculation are relatively easy. The difficulty of this solution is that the user needs to design the partition and the implementation scheme. Hence, a java class, *MTextInputFormat*, is developed and it is extended from the Hadoop class *FileInputFormat*, and two functions *createRecordReader*() and *getSplits*() are overloaded to split the big dataset with attribute subsets. The partition scheme is given in Fig. 3, the schematic diagram of the solution based on partitioning attribute subsets is illustrated in Fig. 4.

If the large dataset U has J attributes, the big data platform has K computing nodes. Obviously, an instance of U consists of J attribute values and a class label. After partitioning the big dataset equally, each subset $A_i(1 \leq i \leq K)$, has $\lceil \frac{J}{K} \rceil$ attribute columns and a class label column. Since Hadoop function *getSplits*() does not support splitting a dataset by attribute subset, it must be overload. Specifically, if the first column is a class label, the ith data of the first row is used as the starting position of the current split, and the ith data of each subsequent row is read into the split to end flag delimiter, indicating that the partitioned big dataset with the attribute $a_i(1 \leq i \leq K)$ is complete. Finally, the Hadoop function *createRecordReader*() is modified to use the column offset as the *key* and the contents of the attributes and class label columns as the *value*. The pseudo-code of the corresponding map and reduce functions are given in Algorithm 6 and 7 respectively.

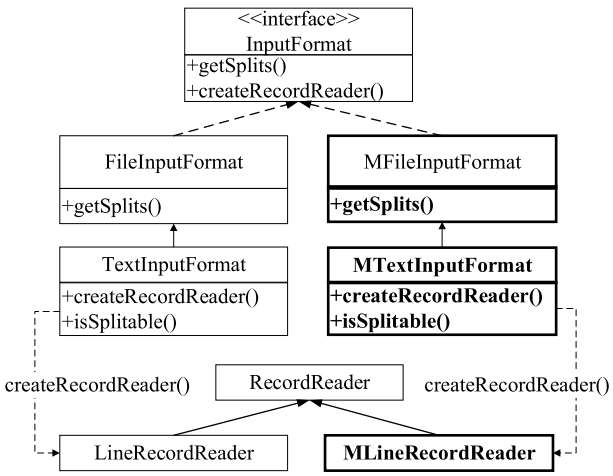

Fig. 3 The design scheme of partitioning attribute subsets

Algorithm 6: The map function

Input: Key-value pair $< k_1, v_1 >$.
Output: Key-value pair $< k_2, v_2 >$, where k_2 is the index of attribute a_j, v_2 is the $Gini(t_j^*)$.

Sort the values of attribute a_j in ascending order, $sortValue(a_j)$;
Calculate the Gini index of unbalanced cut points $Gini(t_i) = calculateGini(t_i)$;
Calculate $t_j^* = minGini(t_i)$;
$context.write(< j, t_j^*.value >, Gini(t_j^*))$;
Output $< k_2, v_2 >$.

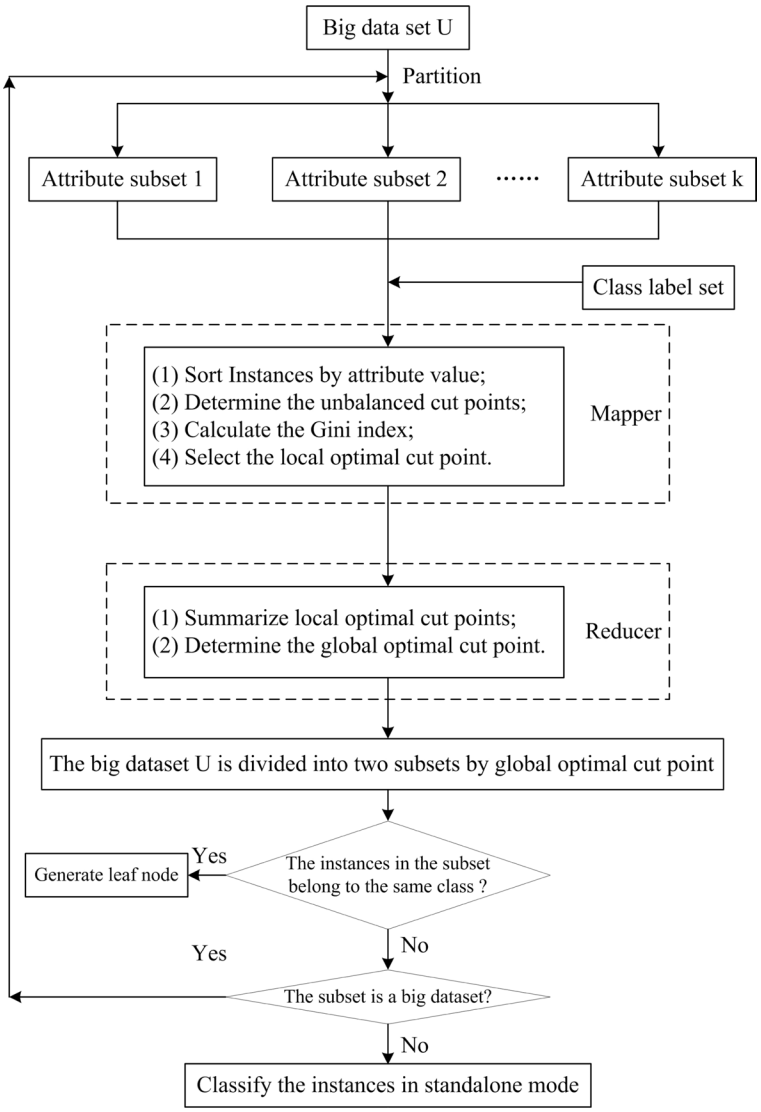

Fig. 4 The schematic diagram of the solution based on partitioning attribute subsets

Algorithm 7: The reduce function

Input: Key-value pair $< k_2, v_2 >$.
Output: The optimal cut point t^*.
Sort $Gini(t_j^*)$ in ascending order;
Calculate the global optimal cut point $t^* = minGini(t_j^*)$;
$context.write(t^*, NullWritable)$;
Output t^*.

The pseudo-code of the corresponding MapReduce-based algorithm denoted by BA-CDT-MR is given in Algorithm 8.

Algorithm 8: BA-CDT-MR Algorithm

Input: Big dataset U.
Output: Continuous-valued decision tree based on unbalanced cut points.
Call Algorithm 6;
Call Algorithm 7;
Select the global optimal cut point t^*;
Partition the big dataset U into two subsets U_1 and U_2 by t^*;
for $(i = 1; i \leq 2; i++)$ **do**
  if ($the\ instances\ in\ U_i\ belong\ to\ same\ class$) **then**
    Generate a leaf node;
  end
  else
    if (U_i *is also a big dataset*) **then**
      Repeat the above process;
    end
    else
      Directly construct the decision tree in the mode of standalone;
    end
  end
end
Output decision tree.

In the framework of Spark, the pseudo-code of the corresponding algorithm denoted by BA-CDT-SP is given in Algorithm 9.

Algorithm 9: BA-CDT-SP Algorithm

Input: Big dataset U.
Output: Continuous-valued decision tree based on unbalanced cut points.
$trainInitRDD = sc.newAPIHadoopFile(U)$;
Transform the data to key-value pairs $kvRDD[(value, label)]$;
Sort the values of attribute a_j in ascending order by $sortByKey()$;
Select unbalanced cut points;
Calculate the Gini index of unbalanced cut points by $giniTRDD = tRDD.calculateGini(t_i)$;
Select the cut point with the minimum Gini index as the local optimal cut point t^*;
Partition the big dataset U into two subsets U_1 and U_2 by t^*;
for $(i = 1; i \leq 2; i++)$ **do**
  if ($the\ instances\ in\ U_i\ belong\ to\ same\ class$) **then**
    Generate a leaf node;
  end
  else
    if (U_i *is also a big dataset*) **then**
      Repeat the above process;
    end
    else
      Directly construct the decision tree in the mode of standalone;
    end
  end
end
Output decision tree.

Table 2 The configuration of the big data platform

Items	Configuration
CPU	Intel(R) Xeon(R) Platinum 8255C CPU 2.50GHz
Memory	16GB
Network Card	Broadcom 5720 QP 1Gb
Hard Disk	2TB
Operating System	CentOS 7.0
Hadoop	Hadoop 3.1.4
Sprk	Spark 3.0.0
JDK	JDK 1.8.0
Scala	Scala SDK-2.12.11

Table 3 The configuration of the nodes in the big data platform

Nodes	IP	Type of nodes
Master	192.168.1.110	NameNode, ResourceManager
Worker 1	192.168.1.111	DataNode, NodeManager
Worker 2	192.168.1.112	DataNode, NodeManager
Worker 3	192.168.1.113	DataNode, NodeManager
Worker 4	192.168.1.114	DataNode, NodeManager
Worker 5	192.168.1.115	DataNode, NodeManager

Experimental results and analysis

To demonstrate the effectiveness of the proposed solutions, we conducted experiments on a big data platform with 6 computing nodes using two open-source framewoeks: MapReduce and Spark. The configuration of the big data platform is given in Table 2, and the configuration of computing nodes in the big data platform is given in Table 3. It should be noted that in the big data platform, the configuration of the master node and the slave node are same.

We compared the proposed algorithms with five methods on five datasets, including two artificial and three UCI datasets. The first artificial dataset, denoted by Gaussian1, is two-dimensions. The instances in this dataset are divided into two classes, where both follow the Gaussian distribution. The corresponding parameters are given in Table 4. The second artificial dataset, denoted by Gaussian2, is four-dimensions. The instances in this dataset fall into four classes, all follow the Gaussian distribution. The corresponding parameters are given in Table 5. The basic information of the five datasets is given in Table 6. The five comparison methods are Parallel C4.5 [7],

Table 4 The parameters of the first artificial set Gaussian1

i	μ_i	Σ_i
1	$(1.0, 1.0)^T$	$\begin{bmatrix} 0.6 & -0.2 \\ -0.2 & 0.6 \end{bmatrix}$
2	$(2.5, 2.5)^T$	$\begin{bmatrix} 0.2 & -0.1 \\ -0.1 & 0.2 \end{bmatrix}$

Table 5 The parameters of the second artificial set Gaussian2

i	μ_i	Σ_i
1	$(0.0, 0.0, 0.0)^T$	$\begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$
2	$(0.0, 1.0, 0.0)^T$	$\begin{bmatrix} 1.0 & 0.0 & 1.0 \\ 0.0 & 2.0 & 2.0 \\ 1.0 & 2.0 & 5.0 \end{bmatrix}$
3	$(-1.0, 0.0, 1.0)^T$	$\begin{bmatrix} 2.0 & 0.0 & 0.0 \\ 0.0 & 6.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$
4	$(0.0, 0.5, 1.0)^T$	$\begin{bmatrix} 2.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 3.0 \end{bmatrix}$

Table 6 The basic information of the five datasets

Datasets	Number of instances	Number of attributes	Number of classes
Gaussian1	5,000,000	2	2
Gaussian2	2,500,000	3	4
Covertype	581,012	10	7
SUSY	5,000,000	18	2
HEPMASS	10,500,000	28	2

FRBDT (Parallel Fuzzy Rule-base Based Decision Tree) [35], IS-C4.5 (Improvement Strategy for C4.5) [6], Weka J48-MR in Hadoop machine learning library, and MLlib DT-SP in Spark machine learning library. The parallel C4.5, based on MapReduce and Spark, are denoted by Parallel C4.5-MR and Parallel C4.5-SP, respectively. The indexes for the experimental comparison are the test accuracy and the running time.

Since the test accuracy of the same algorithm implemented by different platforms will not change significantly, the experimental comparison results of the test accuracy

Table 7 The experimental comparison results of test accuracy

Methods	Datasets				
	Gaussian1	Gaussian2	SUSY	HEPMASS	Covertype
BA-CDT-MR	0.985	0.523	**0.723**	0.782	0.825
BA-CDT-SP	0.982	**0.529**	0.717	0.781	0.824
BS-CDT-MR	0.985	0.512	0.696	0.789	0.821
BS-CDT-SP	0.983	0.511	0.688	0.792	0.823
Parallel C4.5-MR	**0.987**	0.522	0.662	**0.805**	0.819
Parallel C4.5-SP	0.984	0.516	0.668	0.801	0.819
Weka-J48-MR	0.977	0.503	0.665	0.787	0.792
MLlib-DT-SP	0.981	0.512	0.679	0.772	0.798
FRBDT	0.983	0.527	0.705	0.769	**0.836**
IS-C4.5	0.986	0.522	0.662	0.804	0.819

The data in bold indicate the highest test accuracy of different algorithms on different data sets

Table 8 The experimental comparison results of running time with MapReduce

Methods	Datasets				
	Gaussian1	Gaussian2	SUSY	HEPMASS	Covertype
BA-CDT-MR	**189.7**	**229.7**	**1244.1**	**3121.2**	**3669.3**
BS-CDT-MR	221.0	259.5	1531.5	3710.0	3891.8
Parallel C4.5-MR	234.4	277.9	1557.4	3801.2	4656.4
Weka-J48-MR	274.5	286.4	1648.2	3956.2	4853.9
FRBDT	244.5	272.1	1789.0	3815.2	4386.3
IS-C4.5	255.5	289.4	1448.8	3798.6	3952.0

The data in bold indicate the shortest run times for different algorithms implemented by MapReduce and Spark on different datasets, respectively

Table 9 The experimental comparison results of running time with Spark

Methods	Datasets				
	Gaussian1	Gaussian2	SUSY	HEPMASS	Covertype
BA-CDT-SP	**15.4**	**31.2**	**228.2**	**506.5**	**568.5**
BS-CDT-SP	19.8	35.3	312.5	572.2	626.6
Parallel C4.5-SP	31.5	45.1	325.1	687.6	597.1
MLlib-DT-SP	55.2	59.8	414.0	709.5	583.4
FRBDT	34.1	40.5	219.8	556.7	572.1
IS-C4.5	35.5	43.1	252.0	619.5	595.9

The data in bold indicate the shortest run times for different algorithms implemented by MapReduce and Spark on different datasets, respectively

of the algorithms implemented by MapReduce and Spark platforms are uniformly presented in Table 7. Whereas the algorithms are implemented by MapReduce and Spark, resulting in significant different running time. The experimental comparison results of the running time of the algorithms implemented by MapReduce and Spark are given in Tables 8 and 9, respectively.

From the experimental results, displayed in Table 7, one can find the following:

1. There is no significant difference in test accuracy between the two algorithms based on instance subset partition (i.e., BS-CDT-MR and BS-CDT-SP) and the two algorithms based on attribute subset partition (i.e., BA-CDT-MR and BA-CDT-SP) on the five datasets. The test accuracy of the first two algorithms based on attribute subset partition on datasets Gaussian2 and SUSY is slightly higher than that of the second two algorithms based on instance subset partition. This is mainly due to the fact that all four algorithms use the Gini index as a heuristic, but the partition mode and implementation framework of the big datasets are different;
2. Compared to the other six algorithms (Parallel C4.5-MR, Parallel C4.5-SP, FRBDT, IS-C4.5, Weka J48-MR, and MLlib DT-SP), there is no significant difference in the test accuracy on the five datasets.

Referring to the experimental results shown in Table 8, the two algorithms proposed in this paper have the least running time, the fastest speed, and the highest efficiency compared to the four algorithms based on MapReduce. The reason is that

the proposed algorithms are based on the unbalanced cut point. Therefore, when calculating the heuristic, only the Gini index of the unbalanced cut points needs to be calculated, which is not the case for the balanced cut points. From a statistical point of view, the computing time complexity can be reduced by half. Moreover, from the experimental results in Table 9, one can get a conclusion similar to that of Table 8. That is, the two algorithms proposed in this paper have the lowest running time, the fastest speed and the highest efficiency. We believe that the reason is that the solution based on partitioning attribute subsets makes all cut points of each attribute located on the same computing node, and the calculation of the Gini index and the local optimal cut point does not require communication between the computing nodes, while the solution based on the partitioning instance subsets must communicate between computing nodes to calculate the Gini index and the local optimal cut point for each attribute.

In order to more intuitively show the difference in running efficiency when two different solutions are implemented with two different big data frameworks, MapReduce and Spark, we visualized the running time on five datasets. Figure 5 presents the comparison of the running time of the solution based on partitioning attribute subsets under two frameworks, MapReduce and Spark. It can be clearly seen from Fig. 5 that for the same partitioning method (based on partitioning attribute subsets), the implementation of two different frameworks, MapReduce and Spark, the difference of running time is very significant. The result is similar for the case based on partitioning instance subsets, see Fig. 6. We think there are two main reasons. First, Spark constructs DAG directed acyclic graphs when processing big data. Compared with MapReduce, the times used for shuffle operation can be reduced in most cases, thus reducing a large amount of sorting time. Second, MapReduce needs to write the intermediate results of calculations to disks, while Spark can store the intermediate results in memory in the form of RDD, which greatly reduces disk I/O operations and reduces the running time of algorithms. Compared with Spark, MapReduce has

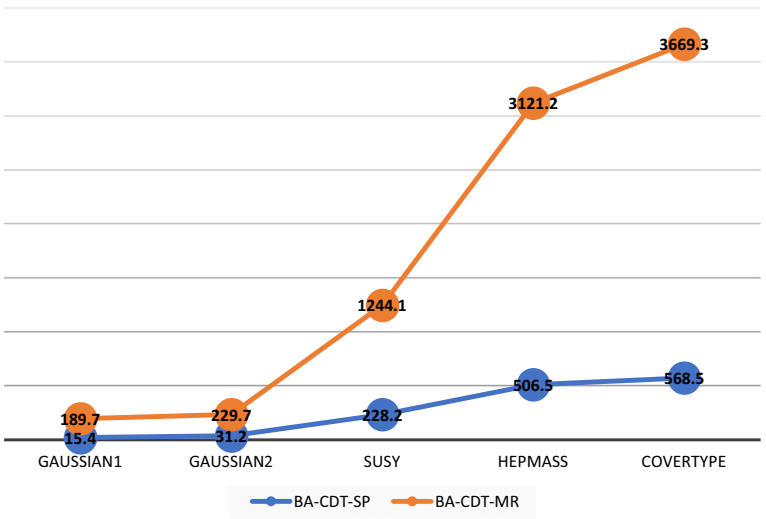

Fig. 5 A comparison of the running time of the solution based on partitioning attribute subsets under two frameworks

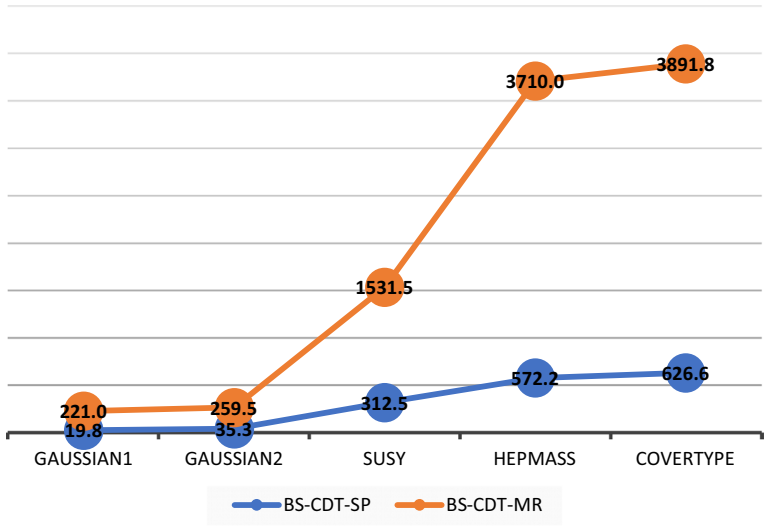

Fig. 6 A comparison of the running time of the solution based on partitioning instance subsets under two frameworks

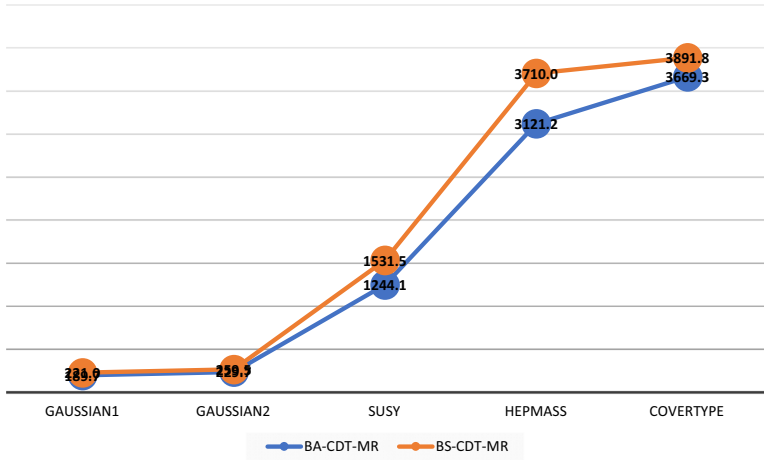

Fig. 7 A comparison of the running time of the two solutions under the framework MapReduce

better fault tolerance, because MapReduce saves the intermediate result first and then processes it. When program errors occur, MapReduce can rectify them in time to ensure the safe running of algorithms. Especially in the processing of big data, a small error may cause the entire training process to fail. The high reliability of MapReduce ensures its secure running.

Figures 7 and 8 provide the comparison of the running time of the two different solutions under the framework MapReduce and Spark, respectively. One can find from the Fig. 7 that the solution BA-CDT-MR based on partitioning attribute subsets runs less than the solution BS-CDT-MR based on partitioning instance subsets. Although both solutions use the Gini index as a heuristic for the induction of decision tree, BS-CDT-MR requires sorting the current attribute a_j before deploying the data subsets to each compute node, resulting in longer running time than BA-CDT-MR. But the difference

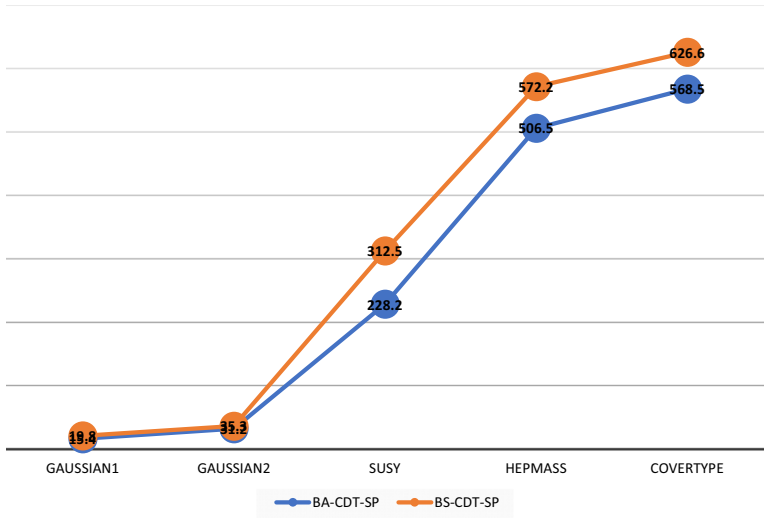

Fig. 8 A comparison of the running time of the two solutions under the framework Spark

is not very significant. In addition, since the calculation of Gini index needs to count the instance number of each class on the whole big data set, for BA-CDT-MR, all the cut points of the same attribute are located on the same computing node, and the statistics on the number of instances for each class can be completed within the same computing node, and the extended attribute can be obtained in one iteration. BS-CDT-MR needs to use inter-node communication to count the number of different instances, resulting in a longer algorithm time than BA-CDT-MR. And as the size of the dataset increases, the time gap becomes more significant. From Fig. 8, one can draw a similar conclusion.

Furthermore, from the experimental results, we found a strange phenomenon, among the five datasets used in the experiment, Covertype had the smallest size, but all the algorithms spent the most time on this dataset. We believe that it is precisely because of their small size, MapReduce and Spark are difficult to fully use their advantages, when processing such data sets, many small files will be generated, and these small files will produce a lot of I/O operations, thus consuming a lot of time.

Conclusions

In this paper, two solutions are proposed to extend the continuous-valued decision tree induction algorithm based on unbalanced cut points to the big data scenarios. The first solution is based on the instance subset partition along the horizontal direction. The technical difficulty of this solution is to calculate the Gini index of the unbalanced cut points across nodes. The second solution is based on attribute subset partition along the vertical direction. The technical difficulty of this solution is that users need to design a big data partition scheme based on attribute subsets. In this scheme, the calculation of the Gini index of unbalanced cut points across nodes is not needed. For these two technical difficulties, this paper gives corresponding perfect solutions, and proposes a continuous-valued big data decision tree induction algorithm based on unbalanced cut points. The proposed algorithm is implemented on two big data platforms, MapReduce and Spark, and the experimental comparison is conducted with five related algorithms in terms of test accuracy and running

time. The experimental results show that (1) the proposed algorithms outperform the five algorithms, and (2) the proposed solutions can efficiently address the problem of scalability of the continuous-valued decision tree algorithm based on unbalanced cut points in big data scenarios.

Acknowledgements
We would like to thank Hebei Key Laboratory of Machine Learning and Computational Intelligence for supporting big data computing platform.

Author contributions
SM: Methodology, validation, data preprocessing, visualization. JZ: Conceptualization, methodology, investigation, visualization, funding acquisition, supervision, project administration, writing—review & editing.

Funding
This research is supported by the key R &D program of science and technology foundation of Hebei Province (19210310D), and by the natural science foundation of Hebei Province (F2021201020).

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 28 December 2022 Accepted: 22 August 2023
Published online: 31 August 2023

References

1. Roh Y, Heo G, Whang SE. A survey on data collection for machine learning: a Big Data-AI integration perspective. IEEE Trans Knowl Data Eng. 2021;33(4):1328–47.
2. Chu CT, Kim SK, Lin YA, et al. Map-reduce for machine learning on multicore. In: Proceedings of the 2006 conference, advances in neural information processing systems 19. MIT Press; 2007. p.281–8.
3. He Q, Zhuang FZ, Li JC, et al. Parallel implementation of classification algorithms based on MapReduce. RSKT 2010, lecture notes in computer science (LNAI,volume 6401). p. 655–62.
4. Xu Y, Qu W, Li Z, et al. Efficient K-means++ approximation with MapReduce. IEEE Trans Parallel Distrib Syst. 2014;25(12):3135–44.
5. Duan M, Li K, Liao X, et al. A parallel multiclassification algorithm for big data using an extreme learning machine. IEEE Trans Neural Netw Learn Syst. 2018;29(6):2337–51.
6. Wang HB, Gao YJ. Research on C4.5 algorithm improvement strategy based on MapReduce. Procedia Comput Sci. 2021;183:160–5.
7. Mu YS, Liu XD, Yang ZH, et al. A parallel C4.5 decision tree algorithm based on MapReduce. Concurr Comput Pract Exp. 2017. https://doi.org/10.1002/cpe.4015.
8. Dai W, Ji W. A MapReduce implementation of C4.5 decision tree algorithm. Int J Database Theory Appl. 2014;7(1):49–60.
9. Wang S, Jia Z, Cao N. Research on optimization and application of Spark decision tree algorithm under cloud-edge collaboration. Int J Intell Syst. 2022;37(11):8833–54.
10. Yuan F, Lian F, Xu X, et al. Decision tree algorithm optimization research based on MapReduce. In: 2015 6th IEEE international conference on software engineering and service science (ICSESS), Beijing, China; 2015. p. 1010–3. https://doi.org/10.1109/ICSESS.2015.7339225.
11. Chern CC, Lei WU, Huang KL, et al. A decision tree classifier for credit assessment problems in big data environments. Inf Syst e-Bus Manag. 2021;19:363–86.
12. Sabah S, Anwar SZB, Afroze S, et al. Big data with decision tree induction. In: 2019 13th international conference on software, knowledge, information management and applications (SKIMA), Island of Ulkulhas, Maldives; 2019. p. 1–6. https://doi.org/10.1109/SKIMA47702.2019.8982419.
13. Wang M, Fu W, He X, et al. A survey on large-scale machine learning. IEEE Trans Knowl Data Eng. 2022;34(6):2574–94.
14. Nti IK, Quarcoo JA, Aning J, et al. A mini-review of machine learning in big data analytics: Applications, challenges, and prospects. Big Data Min Anal. 2022;5(2):81–97.
15. Segatori A, Marcelloni F, Pedrycz W. On distributed fuzzy decision trees for Big Data. IEEE Trans Fuzzy Syst. 2018;26(1):174–92.
16. L'Heureux A, Grolinger K, Elyamany HF, et al. Machine learning with big data: challenges and approaches. IEEE Access. 2017;5:7776–97.

17. Zhang QC, Yang LT, Chen ZK, et al. A survey on deep learning for big data. Inf Fus. 2018;42:146–57.
18. Wu X, Kumar V, Quinlan JR, et al. Top 10 algorithms in data mining. Knowl Inf Syst. 2008;14(1):1–37.
19. Genuer R, Poggi JM, Tuleau-Malot C. Random forests for big data. Big Data Res. 2017;9:28–46.
20. Juez-Gil M, Arnaiz-González Á, Rodríguez JJ, et al. Rotation forest for big data. Inf Fus. 2021;74:39–49.
21. Shivaraju N, Kadappa V, Guggari S. A MapReduce model of decision tree classifier using attribute partitioning. In: 2017 international conference on current trends in computer, electrical, electronics and communication (CT-CEEC), Mysore. New York: IEEE; 2017. p. 207–11.
22. Yuan ZW, Wang CZ. An improved network traffic classification algorithm based on Hadoop decision tree. In: IEEE international conference of online analysis and computing science (ICOACS), Chongqing. New York: IEEE; 2016. p. 53–6.
23. Desai A, Chaudhary S. Distributed decision tree. In: Proceedings of the 9th annual ACM India conference, October 2016. p. 43–50.
24. Desai A, Chaudhary S. Distributed decision tree v.2.0. In: 2017 IEEE international conference on big data (Big Data); 2017. p. 929–34
25. Chen JG, Li KL, Tang Z, et al. A parallel random forest algorithm for big data in a Spark cloud computing environment. IEEE Trans Parallel Distrib Syst. 2017;28(4):919–33.
26. Es-sabery F, Es-sabery K, Hair A. A MapReduce improved ID3 decision tree for classifying twitter data. In: Fakir, M., Baslam, M., El Ayachi, R. (eds) Business intelligence. CBI 2021. Lecture notes in business information processing, vol 416. Cham: Springer; 2021. https://doi.org/10.1007/978-3-030-76508-8_13.
27. Jurczuk K, Czajkowski M, Kretowski M. Multi-GPU approach to global induction of classification trees for large-scale data mining. Appl Intell. 2021;51:5683–700.
28. Abuzaid F, Bradley JK, Liang FT, et al. Yggdrasil: an optimized system for training deep decision trees at scale. Advances in neural information processing systems 29. MIT Press; 2016. p. 3817–25.
29. Chen J, Wang T, Abbey R, et al. A distributed decision tree algorithm and its implementation on big data platforms. In: 2016 IEEE international conference on data science and advanced analytics (DSAA). New York: IEEE; 2016. p. 752–61.
30. En-nattouh Y, El fahssi K, Yahyaouy A, et al. The decision trees and the optimization of resources in Big Data solutions. In: 2020 fourth international conference on intelligent computing in data sciences (ICDS), Fez, Morocco; 2020. p. 1–4. https://doi.org/10.1109/ICDS50568.2020.9268727.
31. Liu K, Chen L, Huang J, et al. Revisiting RFID missing tag identification. In: IEEE INFOCOM 2022—IEEE conference on computer communications, London, United Kingdom; 2022. p. 710–9. https://doi.org/10.1109/INFOCOM48880.2022.9796971.
32. Jin CX, Li FC, Ma SJ, et al. Sampling scheme-based classification rule mining method using decision tree in big data environment. Knowl Based Syst. 2022;244:108522.
33. Lin Z, Sinha S, Towards Zhang W. Efficient and scalable acceleration of online decision tree learning on FPGA. In: IEEE 27th annual international symposium on field-programmable custom computing machines (FCCM). San Diego, CA, USA. 2019; 2019. p. 172–80. https://doi.org/10.1109/FCCM.2019.00032.
34. Weinberg AI, Last M. Selecting a representative decision tree from an ensemble of decision-tree models for fast big data classification. J Big Data. 2019;6:23. https://doi.org/10.1186/s40537-019-0186-3.
35. Mu YS, Liu XD, Wang LD, et al. A parallel fuzzy rule-base based decision tree in the framework of map-reduce. Pattern Recogn. 2020;103: 107326.
36. Wu JMT, Srivastava G, Wei M, et al. Fuzzy high-utility pattern mining in parallel and distributed Hadoop framework. Inf Sci. 2021;553:31–48.
37. Fernandez-Basso C, Ruiz MD, Martin-Bautista MJ. Spark solutions for discovering fuzzy association rules in Big Data. Int J Approx Reason. 2021;137:94–112.
38. Fayyad Usama M, Irani Keki B. On the handling of continuous-valued attributes in decision tree generation. Mach Learn. 1992;8(1):87–102.

Publisher's Note

Shixiang Ma is a M.S. candidate of College of Mathematics and Information Science, Hebei University, Baoding, China. His main research interests include big data processing and machine learning.

Junhai Zhai is a Professor and Ph.D. Supervisor with College of Mathematics and Information Science, Hebei University, Baoding, China. His main research interests include big data processing, machine learning, and deep learning.