


RESEARCH

Open Access



Data classification algorithm for data-intensive computing environments

Tiedong Chen¹, Shifeng Liu¹, Daqing Gong^{1,2*}  and Honghu Gao¹

Abstract

Data-intensive computing has received substantial attention since the arrival of the big data era. Research on data mining in data-intensive computing environments is still in the initial stage. In this paper, a decision tree classification algorithm called MR-DIDC is proposed that is based on the programming framework of MapReduce and the SPRINT algorithm. MR-DIDC inherits the advantages of MapReduce, which make the algorithm more suitable for data-intensive computing applications. The performance of the algorithm is evaluated based on an example. The results of experiments showed that MR-DIDC can shorten the operation time and improve the accuracy in a big data environment.

Keywords: Data-intensive, Data mining, MR-DIDC, MapReduce

1 Introduction

Microblog, personal blog, and photo and video sharing based on Web2.0 have produced large amounts of new data on the Internet and wireless network; among which data stored in semi-structured XML documents, HTML documents, and unstructured photos, audio and video are becoming increasingly abundant. There are massive amounts of data in the fields of web search, commercial sales, biomedicine, natural observation, and scientific computing [1]. Although the data types are diverse, they have common characteristics including massive scale, rapid changes, distributed storage, heterogeneity, and semi-structured or non-structured features. Outlier-mining algorithms based on data flow cannot satisfy the needs [2]; thus, data-intensive computing has emerged to satisfy the needs of obtaining, managing, analyzing, and understanding massive and rapidly changing data effectively [3].

In a data-intensive computing environment, massive amounts of data must be filtered, analyzed, and stored. Algorithm efficiency is not the only aim; distribution, effectiveness, and availability in a heterogeneous data environment are also considered. Massive data sets that change rapidly require high data storage efficiency

[4–10]. In addition to the algorithm efficiency, we consider the effectiveness and the feasibility of the algorithm in distributed and heterogeneous data environments [11–15]. The network transmission speed restricts the free transfer between different computers. The bottleneck of data management and task analysis lies not only in computational capacity but also in data availability, namely, whether the network transfer speed can match the speed of system collection, processing, and analysis [16]. Obtaining the needed information is the current focus of data-intensive computing.

Clustering is one of the most important technologies in data mining. It aims at associating physical or abstract subjects with similar subjects. Jiang et al. proposed the use of the k -means clustering algorithm with MapReduce and realized the transformation of the k -means [17] algorithm by MapReduce [18]. Hong et al. presented the DRICA (Dynamic Rough Increment Clustering Algorithm) [19] as an approach for solving the data updating problem; they ensured the stability of the algorithm and overcame the inefficiency of implementing the algorithm on all data. HDCH (High Dimensional Clustering on Hadoop), which was designed for clustering massive audio data by Liao et al. [20], uses a large cutting granularity in every dimension, thereby implementing clustering efficiently.

Most common classification algorithms are based on prior knowledge and forecast unknown data by extracting

* Correspondence: gongtuipigua@163.com

¹School of Economics and Management, Beijing Jiaotong University, No.3 Shangyuan, Haidian District, Beijing 100044, People's Republic of China

²School of Economics and Management, Tsinghua University, Beijing, China

a data model [21]. Li et al. proposed the analysis of classification research based on distributed data warehousing [22], which is only available in single machines on GAC-RDB. Because the Internet is becoming increasingly complicated, it is difficult to perform mining network classification accurately. Li et al. proposed an active collaborative method that combined feature selection and a link filtering method [23]. To solve the problem of being unable to process machine-learning data in memory, Liu et al. presented the LS-SVM classification algorithm coordinate descent I_2 [24], which involves the optimization problem of improving the objective function mode to transform a multi-objective problem into a single objective problem.

Frequent item set mining is the most basic and important procedure of association rule mining, sequence pattern mining, relevance mining, and multi-layer mining [25, 26]. The Apriori mining algorithm, which is based on cloud computing, implements dual binary encoding on the transaction set and the item set based on the MapReduce model. Because the algorithm requires only Boolean *AND* and *OR* operations, its efficiency is high [27]. Li et al. proposed the CMFS algorithm [28] based on closed-item-set mining algorithms [29], which uses the constrained maximum-frequent-item-set deep priority strategy and a pruning algorithm. Hong et al. proposed the FIMDS [30] algorithm, which can mine frequent item sets from distributed data. The algorithm uses a frequent-mode tree structure to store data, which facilitates the acquisition of the item-set frequency from each partial mode tree (FP-tree) root.

Outlier mining is one of the main approaches used in data mining. An outlier is a data object that is different from other data objects because it is produced by a different mechanism [31]. Since outlier mining was first proposed, it has been researched continuously. Most outlier mining algorithms in data-intensive computing involve expanding and improving classic outlier mining algorithms. Recently, many traditional outlier detection algorithm based on data flow have been proposed; however, research on applying outlier detection algorithms to data is still in the primary stage. Due to high time complexity, liable response speed has not been achieved, result errors are large, and the accuracy is unsatisfactory [32]. Therefore, research on outlier detection in data-intensive computing environments is of great importance. Pan [33] utilized the SPRINT algorithm, which is based on the Hadoop platform, by employing a parallel method of constructing a decision tree and then solving the parallel problem in the Hadoop platform. In this paper, systematic research on an outlier detection algorithm is carried out. It focuses on integrating and improving an existing algorithm by proposing a coding framework based on MapReduce and the decision-tree classification method MR-DIDC of the SPRINT algorithm, which takes advantage of the outstanding features of

MapReduce to make the approach more suitable for data-intensive environments.

2 Basic algorithm analysis

Shafer et al. proposed the SPRINT decision tree algorithm, which is based on SLIQ, in 1996 [34]. The SPRINT algorithm combines the property list and category list. The property list is used to store attribute values, a histogram plot is used to record the category distributions of the partition before and after a specified node, and a hash table is used instead of SLIQ to record the attribute sub-node information of the training tuple. The attribute list and histogram plot data structures do not require storage or memory, which eliminates the size limitation of the memorization capability. The SPRINT algorithm is not only simple, accurate, and fast, it also improves the data structure, which makes mining problems easier to solve.

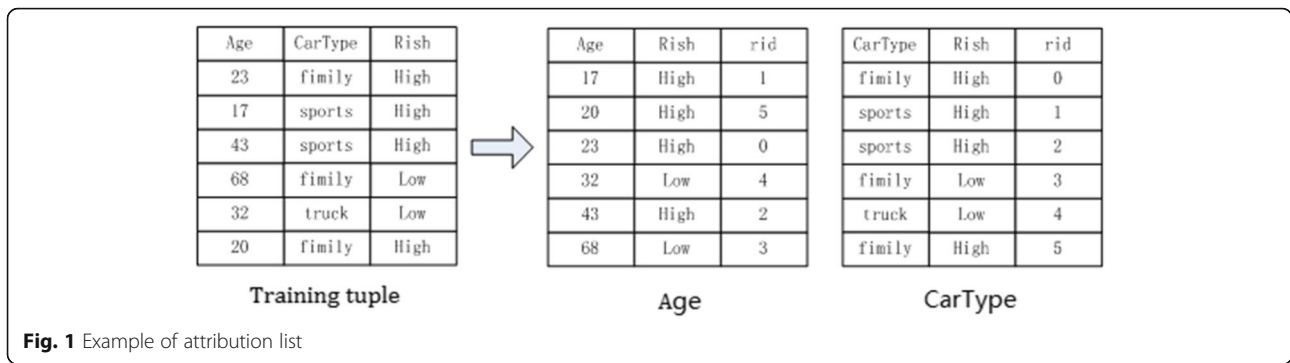
The SPRINT algorithm uses the property list and the histogram to help calculate the best split point and split property (shown in Table 1). Every tuple of the property list consists of a data record index, a property value, and a category ID. In the initialization of the SPRINT algorithm, a property list is created for each property. The property list of continuous attributes needs to be pre-sorted according to its attribute values, which are shown in Fig. 1.

The property list maintained by the current node will expand as the node is divided into sub-nodes in a sub-property list. Every node maintains a histogram, tally matrix, hash table, etc. The histogram is used to describe the classification distribution information of the selected continuous property division, as shown in Fig. 2. The tally matrix is used to determine the classification quantity and property value relationship in the property list of the discrete attribute property chosen by the selected node, as shown in Fig. 3. The histogram and the tally matrix are used to support the calculation of the Gini index. The hash table records the best split property and the available sub-node information after splitting; then, these are used to assist in splitting other property lists.

The calculation of the best property itemization point and the split point of the property list are the core tasks of the SPRINT algorithm. The SPRINT algorithm uses the Gini index as the property measurement standard.

Table 1 SPRINT algorithm flow

```
Function Partition(DataSet S){S:training set}
Begin
If (all  $s \in S$  the same mark) then
    Return;
Foreach  $a \in A$  Do
    calculate column attribute a
    split S into  $S_1$  and  $S_2$  using the best split attribute
    Partition( $S_1$ );
    Partition( $S_2$ );
End SPRINT
```



The Gini index performs better and is easier to calculate than the information gain, and Gini index minimization splitting will yield the minimal information gain. The Gini index method is described as follows:

Suppose the training set contains n categories and m records. The Gini index is defined by Eq. 1, in which P_i is the frequency of type i .

$$Gini(T) = 1 - \sum_{i=1}^n P_i^2 \tag{1}$$

The training set T is split into T_1 and T_2 , with m_1 and m_2 records, respectively. Then, the Gini index is defined by Eq. 2.

$$Gini_{split}(T) = \frac{m_1}{m} Gini(T_1) + \frac{m_2}{m} Gini(T_2) \tag{2}$$

The split with the minimum Gini index is the best split of set T . The inputs for calculating the Gini index are the histogram and the counting matrix.

For a specific node, after determining the best splitting property and its split node, the property list that corresponds to this property can be directly divided among the sub-nodes. Because the sequences of different property lists differ, they cannot be divided directly. Instead, a hash table (shown in Fig. 4) should be produced from the node's best split property and its split-point information before dividing other property lists. The hash table

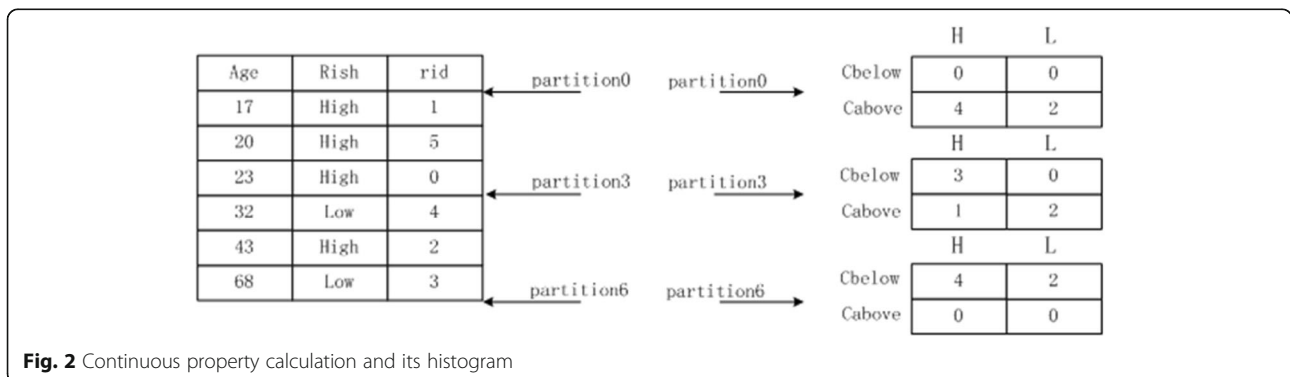
is used to record the sub-node to which the node belongs. The hash table is also used to split other property lists. The hash table for node N0 is shown in Fig. 5.

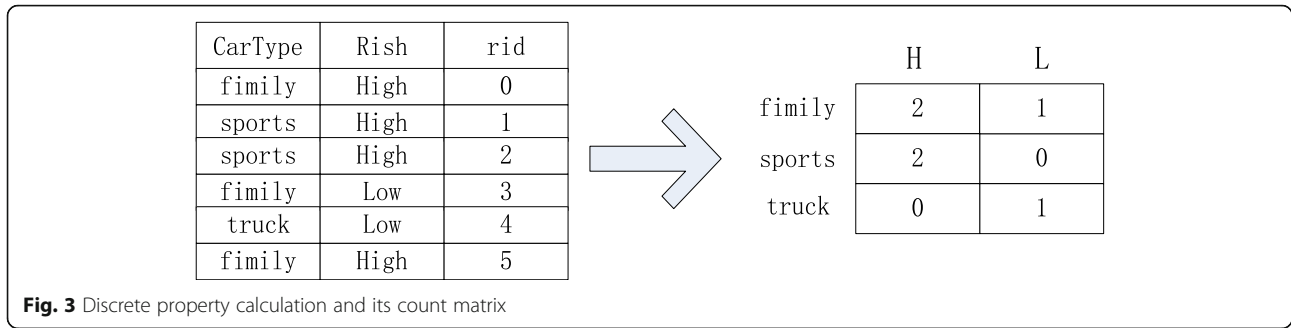
3 Modified algorithm

3.1 Data structure that is used by the MR-DIDC algorithm

The property list has the same function as in the SPRINT algorithm: it is used to record the distribution information. The training data set is decomposed into independent property lists based on the properties during the initialization period. Each property list consists of property values, category values, and index values. The number of property lists depends on the number of properties. Initially, the root node is used to maintain the whole property list. Each property list that corresponds to a continuous property is pre-ranked and the property lists are split as the number of nodes increases. The split property lists belong to the corresponding sub-nodes.

A histogram is used to describe the class distribution and corresponds to the tree node in the decision tree in the SPRINT algorithm; however, it is structured differently than the histogram in the MR-DIDC algorithm. The histogram is used to calculate the best split property and its split nodes that belong to the tree node. However, to utilize the MapReduce programming framework, some changes have been made to the histogram structure. Data are split into several chunks and stored in the clustering environment for data processing with





MapReduce. The histogram is used to record the property value distribution that belongs to each data chunk scan node. For a continuous property, every data chunk has two corresponding histograms, namely, C_{below} and C_{above} , which depend on the scan location in the property list of the current data chunk. C_{below} represents all the records of the class distribution conditions for $a \leq R$ and C_{above} represents all the records of class distribution conditions for $a > R$, in which parameter a is the continuous property of the scan value and D represents the best split value. However, every histogram record consists of a data chunk index and all class records, as shown in part B in Fig. 6.

The piece histogram is a new kind of data structure introduced by this algorithm to assist in the calculation of the split property and split nodes. The form of the piece histogram is similar to that of the histogram; however, the piece histogram is used to record the total record number of data chunks in the property list, which is shown in part A of Fig. 6.

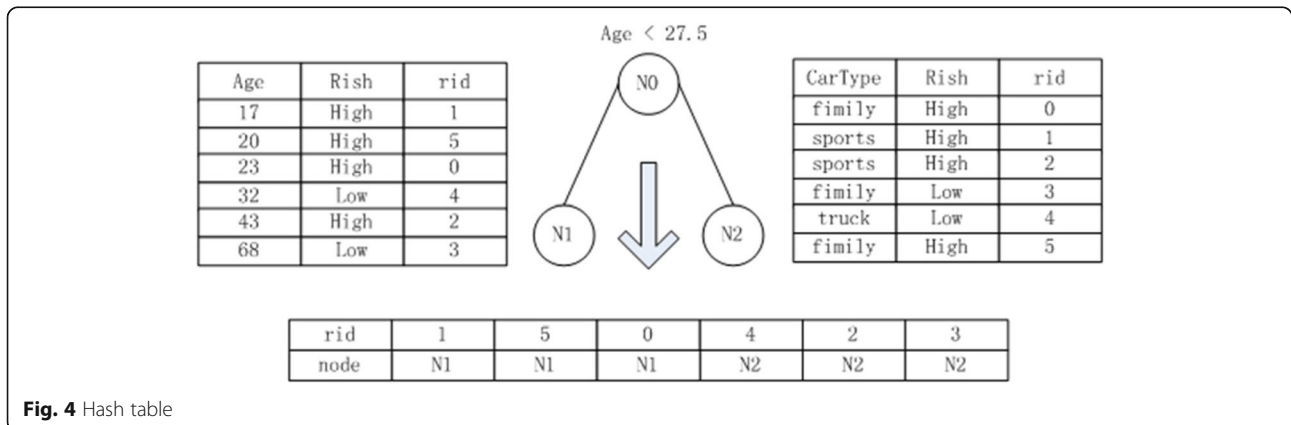
3.2 MR-DIDC algorithm description

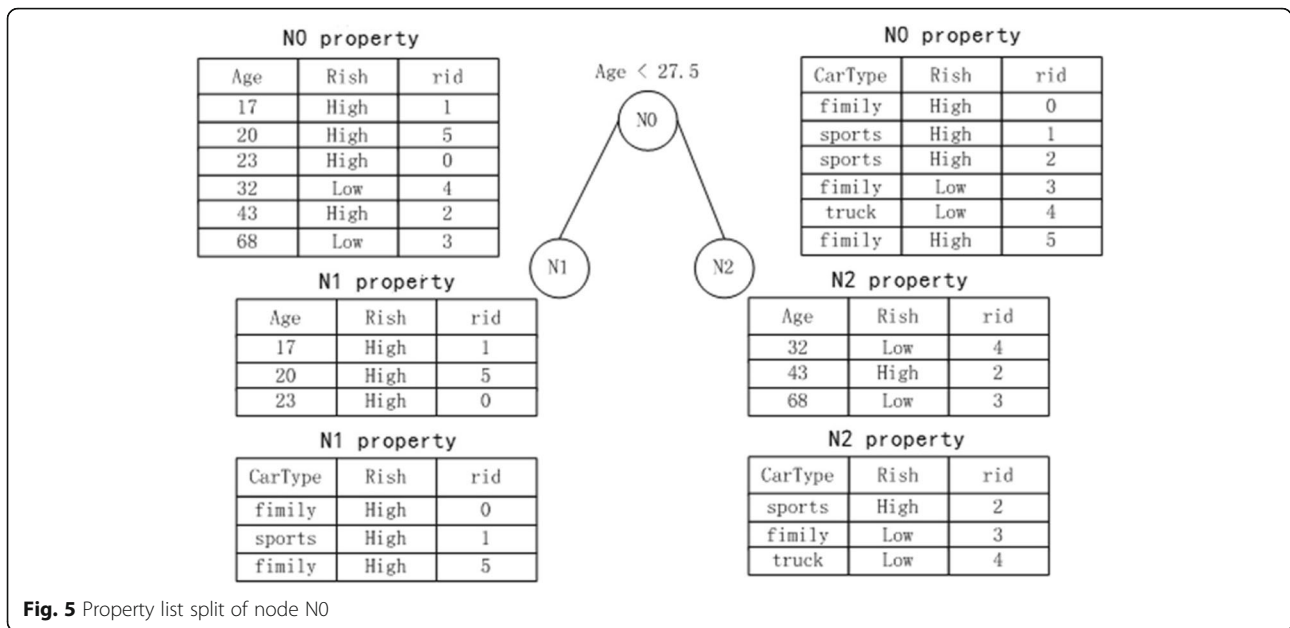
The core program of Build Classifier in Master Server consists of launching, planning, and controlling the whole decision tree model. A series of MapReduce tasks, which are run on Slaves, are used to calculate the split property and the splitting of the property list. The core processes maintain a global decision tree. The global decision tree is

used to save the established part of the built part. The Build Classifier program flow is shown in Table 2.

In the initial stage, the core program builds the root nodes according to information from the pre-processed property list. Then, the root nodes are added to the global nodes queue and the Build Tree function is invoked to create the sub-tree of the queue nodes. The program flow of BuildTree is shown in Table 3.

Each non-leaf sub-node has only two branches, and only binary splitting is performed in the decision tree model. In the course of building the decision tree, T_{curr} denotes the tree nodes that are extracted from the global node queue and processed sequentially. First, we need to check whether T_{curr} is a leaf node. If the training data tuple T_{curr} is “pure” (all three components belong to the same class) or the number of training data tuples reaches the threshold value, which is defined by the user, T_{curr} is labeled a leaf node. Under the first condition, each node is labeled according to the class to which it belongs. Under the second condition, the number of categories to which each node belongs is used as a label. The third to tenth steps comprise the core of the decision tree modeling algorithm, which will be discussed in the next section. The core calculates the split nodes and chooses the best split property. T_{curr} is labeled according to best split information and split property. Training set is used to split the property list of T_{curr} into leftD and rightD, which are used to produce the sub-trees





T_{left} and T_{right} . Then, T_{left} and T_{right} are added into the global node queue. Nodes in queue NQ are removed iteratively. When the queue is empty, the program ends and the decision tree model is complete.

3.3 Calculation of the best split point

The algorithm for calculating the best split point is similar to the SPRINT algorithm and involves scanning the property list of the continuous property calculation split point or the discrete property technology matrix. This algorithm uses MapReduce technology to realize the parallel processing of the best split point, thereby improving the efficiency of the algorithm. Assume that N Mapper tasks are used to process the scan statistics, so that each Mapper task only processes 1/Nth of the training data set. The information aggregation and split-point calculation are performed through the Reducer, which returns the best split information and the split property.

The main advantage of the algorithm is that it needs to calculate only the category distribution of each data

chunk. This algorithm executes a series of MapReduce tasks to build the histogram and block histogram of the current tree nodes. Then, it calculates the Gini index to determine the best split points for every property. The descriptions of the Mapper part and the Reducer part of the FindBestSplit algorithm are shown in Tables 4 and 5, respectively.

The pseudo code above describes the Mapper tasks during the Map phase of FindBestSplit. Each Mapper gathers the category distribution information of each data chunk independently and outputs a key table, in which the key is the property subscript index of the property index and the values consist of the data chunk series number and the category key value.

In the Reduce phase, the output, histogram, and block histogram are collected. Then, the other group of MapReduce tasks calculate the Gini index. In the Map phase, the histogram and block histogram that were obtained in the Reduce phase are used to calculate the best split point of each available property. In the Reduce phase, we need to collect the results of

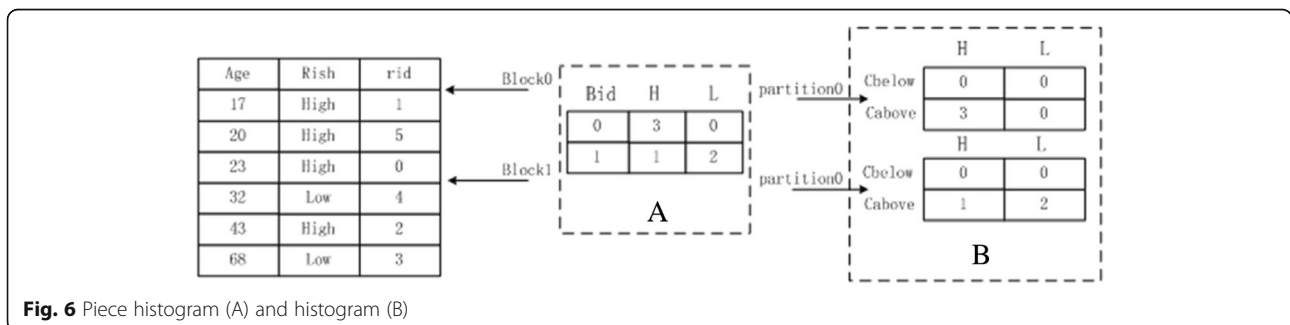


Table 2 Build Classifier program flow

Require: NodeQueue NQ, TreeModel TM, Training record

$(x,y) \in D$, Attribute set Att

1. $T_{root} = \text{new Node}$
2. $\text{Initiate}(T_{root}, D, \text{Att})$
3. $TM = T_{root}$
4. $NQ.\text{push_back}(T_{root})$
5. $\text{BuildTree}(NQ)$

Map phase to select the best split point and split property.

4 Environment construction of the MapReduce algorithm

The experiment is carried out on a Hadoop0.20.2 distributed platform, which is made up of nine PC machines, with a 2.93-GHz CPU, 2-GB memory, 150-GB hard disk, and the Ubuntu Linux operating system. Eclipse3.3.2 is a development tool, in which one is the Name node server and the other eight are data nodes. We show them in Table 6.

The installation and allocation of Master and Slave in Hadoop are the same, and it is only necessary to complete the installation and allocation in Master. Then, the results are transferred to every corresponding folder of Slave through SSH no-password public key authentication (in this paper, the decompressions are performed in the administrator folder). The files of Hadoop allocation are stored in the folders of /hadoop/conf. The allocation of core-site.xml, hdfs-site.xml, and mapred-site.xml is necessary for version 0.2 and above. The master file and slave files need to be allocated by writing to the node machine the names of the Master and Slaves. Eventually, hadoop-env.sh is allocated, by writing to the environment variable JAVA_HOME. The value is the

Table 3 Program flow of BuildTree

Require: NodeQueue NQ, TreeModel TM, Training record

$(x,y) \in D$

1. **For** each $T_{curr} \in NQ$ **do**
2. **If** $\text{JudgeLeaf}(T_{curr})$ is false **then**
3. $\text{bestSplit} = \text{FindBestSplit}(T_{curr})$
4. $T_{curr} \rightarrow \text{splitAtt} = \text{bestSplit} \rightarrow \text{splitAtt}$
5. **If** $\text{bestSplit} \rightarrow \text{splitAtt}$ is category **then**
6. $T_{curr} \rightarrow \text{leftAttSet} = \text{bestSplit} \rightarrow \text{leftAttSet}$
7. $T_{curr} \rightarrow \text{rightAttSet} = \text{bestSplit} \rightarrow \text{rightAttSet}$
8. **Else**
9. $T_{curr} \rightarrow \text{splitValue} = \text{bestSplit} \rightarrow \text{splitValue}$
10. $\text{parationTrainingSet}(T_{curr} \rightarrow D, \text{leftD}, \text{rightD})$
11. $\text{remove}(T_{curr} \rightarrow \text{splitAtt})$
12. Create new nodes T_{left}, T_{right}
13. $\text{Initiate}(T_{left}, \text{leftD}, \text{Att})$
14. $\text{Initiate}(T_{right}, \text{rightD}, \text{Att})$
15. $T_{curr} \rightarrow \text{left} = T_{left}$
16. $T_{curr} \rightarrow \text{right} = T_{right}$
17. $NQ.\text{push_back}(T_{left})$
18. $NQ.\text{push_back}(T_{right})$
19. **Else**
20. $T_{curr} \rightarrow \text{isLeaf} = \text{true}$
21. $T_{curr} \rightarrow \text{label} = y // y$ is the most common label

Table 4 Mapper task flow of FindBestSplitRequire: Current node T_{curr} , Attribute set Att, Class set Y

1. **For** each $A \in \text{Att}$ **do**
2. Class Count array countY for Y
3. $\text{Index} = \text{firststercord}(T_{curr} \rightarrow D)$
4. **For** all $(x,y) \in (T_{curr} \rightarrow D, \text{Attribute list of } A)$ **do**
5. $\text{countY}[\text{findY}(Y,y)]++$
6. $\text{Output}((\text{findA}(A)), (\text{Index}, \text{countY}))$

root of node machine JDK. The allocation process of Hadoop for Master nodes is as follows:

After finishing the file allocation process, the Hadoop task of Master node allocation is complete. The Hadoop nodes in Slaves do not need to be allocated independently. It is only necessary to copy the Hadoop files into Master to obtain the customer indices of the Slave nodes through SSH no-password public key verification and complete the allocation process. The specific commands are as follows:

```
~$scp -r /home/ administrator /hadoop-0.20.2 ubuntu@C1:/home/administrator/
```

The above command copies the allocated Hadoop location to SlaveC1 using Master; the allocated Hadoop locations are copied to other Slave machines similarly. In particular, the allocated locations of Hadoop and JDK in Master have a one-to-one correspondence with those of Slave. However, they are not all the same. Therefore, the JAVA_HOME value must be allocated based on the corresponding location.

5 Case study

The UCI database is one of the main data sources for data mining. The KDD Cup 1999 data set in the UCI database is used in this experiment, which is made up of 4,000,000 records and 40 properties, among which 34 are continuous properties, 5 are discrete properties, and 1 is a class-labeled property (discrete property).

To analyze the performance of the MR-DIDC algorithm, we evaluate the time efficiency, scalability, parallelism, and accuracy. The experimental data are the mean values of repeated experiments. The operation time consists of the algorithm operation time, I/O communication time, and data pre-processing time.

In experiment 1, the calculation times of SPRINT and MR-DIDC are compared, to evaluate the time performance and test the scalability of the MR-DIDC algorithm.

Table 5 Reducer task flow of FindBestSplit

Require: Key k, Value Set V, Attribute set Att, Class set Y

1. For All k do
2. If $\text{Att}[k]$ is continuous then
3. For all distinct values $e \in V$ do
4. If $\text{sameBlock}(\text{value}[i])$ then
5. $\text{Output}((k), (\text{value}[i], \text{sumCount}(\text{value}[i])))$

Table 6 Hadoop clustering environment

A. Edit conf/master, by replacing the master hostname (every host has one independent name). The specific commands are as follows:

```
~$cd /home/ administrator /hadoop-0.20.2
~$gedit conf/master
Write the following in the edit window:
C0
```

B. Edit conf/slaves, by adding all hostnames of the slaves. The specific commands are as follows:

```
~$cd /home/ administrator /hadoop-0.20.2
~$gedit conf/slaves
Write the following in the edit window:
C1
C2
C3
C4
C5
C6
C7
C8
```

Save and close the edit window.

C. Editconf/hadoop-env.sh, by setting variable JAVA_HOME to the JDK installation index. The specific commands are as follows:

```
~$cd /home/ administrator /hadoop-0.20.2
~$gedit conf/hadoop-env.sh
Write the following in the edit window:
export JAVA_HOME /usr/local/java/jdk1.7.0_03
Save and close the edit window.
```

D. Allocatcore-site.xml file. The specific commands are as follows:

```
~$cd /home/ administrator /hadoop-0.20.2
~$gedit conf/core-site.xml
Write the following in the edit window:
Save and close the edit window.
```

E. Allocatohdfs-site.xml file. The specific commands are as follows:

```
~$cd /home/ administrator /hadoop-0.20.2
~$gedit conf/hdfs-site.xml
Write the following in the edit window:
Save and close the edit window.
```

F. Allocatemapred-site.xml file. The specific commands are as follows:

```
~$cd /home/ administrator /hadoop-0.20.2
~$gedit conf/mapred-site.xml
Write the following in the edit window.
```

Table 7 Operation times for training data sets of different sizes (unit: seconds)

Sample size	SPRINT algorithm	MR-DIDC algorithm
4,000,000	102.3	176.5
	197.4	310.3
	305.5	391.8
	412.5	452.6
	504.8	516.9
	611.1	552.4
	766.4	594.7
	977.2	619.8

The line chart of the experimental results is shown in Fig. 7

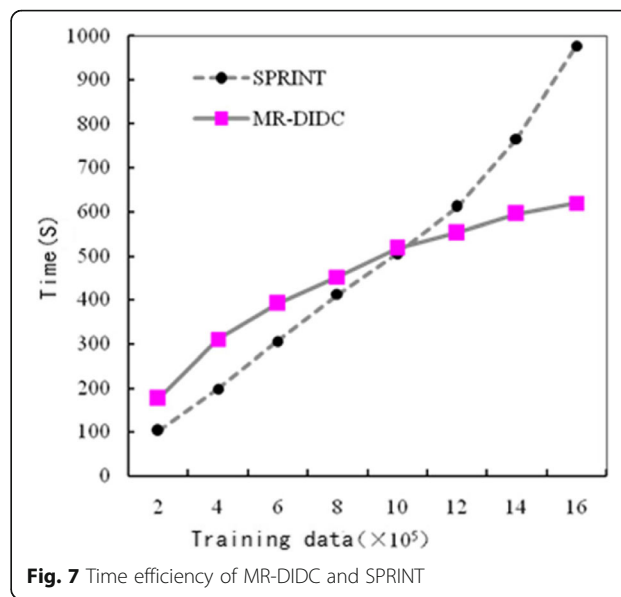


Fig. 7 Time efficiency of MR-DIDC and SPRINT

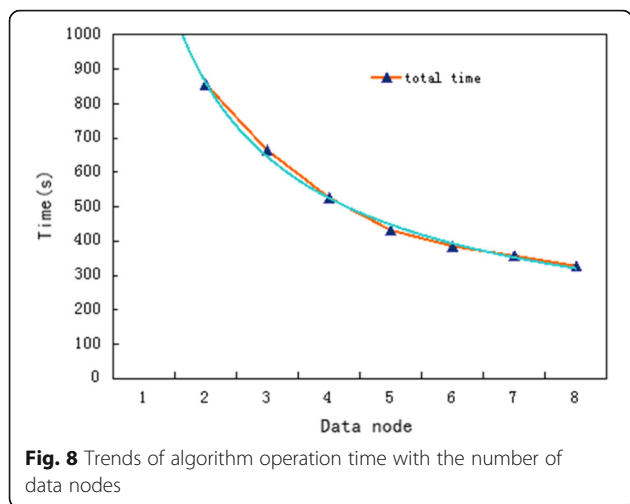
The trends of the operation times of SPRINT and MR-DIDC are compared by using the same training data set and increasing the size of the training set. For every experiment, the parameters remain the same. The operation times are shown in Table 7.

As shown in Fig. 7, the operation efficiency of MR-DIDC gradually becomes superior to that of SPRINT as the size of the training data set increases. When there is only a small amount of data, HDFS needs to split the training data and store the distribution in the data nodes; therefore, the pre-processing time of MR-DIDC is far higher than the operation time of the algorithm and results in the lower total time efficiency of MR-DIDC compared to that of SPRINT. However, after the data set reaches a certain size, the parallelization makes building the decision tree of MR-DIDC more efficient, which shortens the calculation time. Moreover, at the same time, for a large data set, the operating time of the SPRINT algorithm increases rapidly. Although the SPRINT algorithm has good scalability for big data, there are still many restrictions, such as data structure restrictions and restrictions on hash table and histogram storage in memory. The MR-DIDC algorithm uses the

Table 8 Total operation time trends of the MR-DIDC algorithm

Sample size	Number of data nodes	Total operation time
4,000,000	2	857.6
	3	664.0
	4	526.8
	5	431.7
	6	385.1
	7	356.9
	8	327.4

The line chart of the experimental results is shown in Fig. 8



MapReduce framework and the HDFS distributed file system to improve the scalability of the algorithm.

In experiment 2, the parallelization performance of the MR-DIDC algorithm is evaluated.

The trend of the total operation time of the MR-DIDC algorithm as the number of Hadoop clustering data nodes increases using the same training data set is examined. Other parameters of the algorithms stay the same for every experiment. The mean values of multiple observations are shown in Table 8.

According to Fig. 8, as the number of clustering nodes increases, the operating time of MR-DIDC declines exponentially.

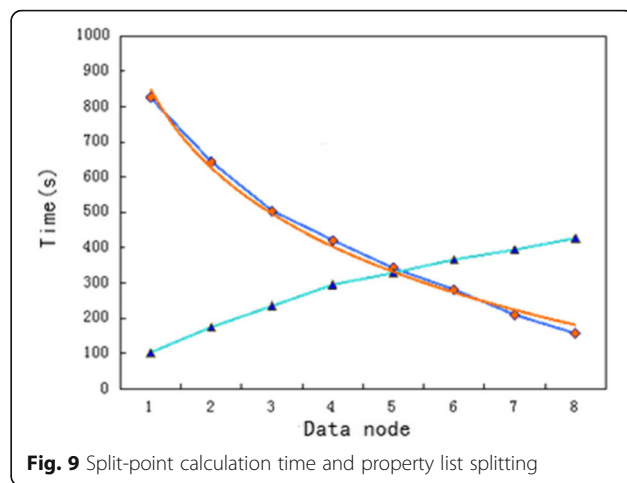
In experiment 3, the trends of test algorithm split-point calculation time and property-list splitting are examined.

Using the same training data set and keeping its size constant, the number of Hadoop clustering data nodes is increased and the trends of the MapReduce task split-point calculation time and property-list splitting time of the MR-DIDC algorithm are examined. Other parameters of the algorithm are kept the same. The mean values of the results are shown in Table 9.

Table 9 Trends of algorithm property-list splitting time and split-point calculation time

Sample size	Property-list splitting time	Split-point calculation time
4,000,000	825.44	102.5
	643.57	174.54
	504.45	234.49
	421.86	295.32
	342.26	328.95
	281.61	366.81
	209.74	394.71
	156.65	427.49

The line chart of the experimental results is shown in Fig. 9



As shown in Fig. 9, as the number of nodes increases, the split-point calculation time decreases linearly and the property-list splitting time increases linearly. As the number of nodes increases, the split-point calculation optimization process improves, and the efficiency of the algorithm is improved. However, the property-list splitting methods restrict the improvement of the algorithm in terms of efficiency.

Experiment 4 is the testing and accuracy comparison of the MR-DIDC algorithm and the SPRINT algorithm.

In this experiment, the number of Hadoop clustering data nodes stays the same and the same training data set is used. We change the data set size and observe and compare the accuracy degree between MR-DIDC and SPRINT algorithms. For every experiment, all other parameters stay the same. The mean values of the observed results are shown in Table 10.

As seen in Fig. 10, the accuracy degree of MR-DIDC algorithm does not vary obviously compared with that of the SPRINT algorithm, and as the data set increases, the two algorithms acquire similar accuracy degrees, because the accuracy lookup technology is still adopted in MR-DIDC algorithm. Therefore, the accuracy degree will not be influenced by the change in the algorithmic framework.

Table 10 Compared accuracies of MR-DIDC and SPRINT

Sample size	Number of data nodes	SPRINT	MR-DIDC
3,000,000	2	0.56	0.54
	3	0.65	0.63
	4	0.7	0.69
	5	0.73	0.74
	6	0.76	0.78
	7	0.79	0.8
	8	0.81	0.83
	9	0.82	0.84

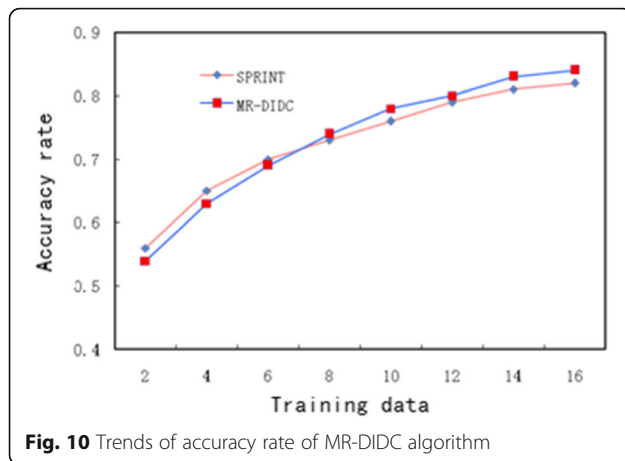


Fig. 10 Trends of accuracy rate of MR-DIDC algorithm

To conclude, from the results, the time efficiency of the algorithm will improve with the same MR-DIDC accuracy. The algorithm has good scalability, and it could satisfy the needs of massive data. However, the structure of the algorithm is complicated and that complexity becomes the performance restriction.

6 Conclusions

With the development of big data, mining useful information has become a subject of interest. Data-intensive environments have been considered only in the context of big data mining research. Current research on data mining algorithms in data-intensive calculation environments have concentrated on improving traditional large-scale clustering algorithms. In this paper, a decision tree classification algorithm called MR-DIDC is introduced that is based on the SPRINT algorithm and the MapReduce calculation framework and that is suitable for data-intensive calculations. We tested the performance of the MR-DIDC algorithm experimentally. The results show that the MR-DIDC algorithm has good scalability and a high level of data availability. The running time for large-scale clustering is reduced when there are large amounts of data.

Acknowledgements

We gratefully acknowledge the International Center for Informatics Research, Beijing Jiaotong University, China, which provided the simulation platform.

Availability of data and materials

Data were collected from the UCI database.

Authors' contributions

The proposed algorithm was designed to be suitable for data-intensive calculations. We tested the performance of the MR-DIDC algorithm experimentally and proved that the MR-DIDC algorithm has good scalability and a high level of data availability. CT collected the data, LS planned and conducted the experiments, and GH analyzed the results. GD wrote the paper and we all approved the paper.

Funding

The study is supported by a project funded by Beijing Natural Science Foundation (041501108), the China Postdoctoral Science Foundation (2016M591194), and the National Natural Science Foundation (71132008,71390334). We greatly appreciate their support.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 29 August 2017 Accepted: 1 December 2017

Published online: 20 December 2017

References

1. R.E. Bryant. Data-intensive supercomputing: the case for DISC. Technical report CMU-CS-07-128, Available as <http://repository.cmu.edu/compsci/258/>.
2. C Liu, H Jin, W Jiang, H Li, Performance optimization based on MapReduce. *J. Wuhan Univ. Technol.* **20**(32) (2010)
3. Pacific Northwest National Laboratory. Data intensive computing project overview. <https://www.pnnl.gov/publications/results.asp>.
4. Yang, Z., Awasthi, M., Ghosh, M., & Mi, N. (2016). A fresh perspective on total cost of ownership models for flash storage in datacenters. In *Cloud computing technology and science (CloudCom)*, 2016 IEEE International Conference on (pp. 245–252). IEEE
5. Yang, Z., Tai, J., Bhimani, J., Wang, J., Mi, N., & Sheng, B. (2016). GRem: dynamic SSD resource allocation in virtualized storage systems with heterogeneous IO workloads. In *Performance Computing and Communications Conference (IPCCC)*, 2016 IEEE 35th International (pp. 1–8). IEEE.
6. Roemer, J., Groman, M., Yang, Z., Wang, Y., Tan, C. C., & Mi, N. (2014). Improving virtual machine migration via deduplication. In *Mobile Ad Hoc and Sensor Systems (MASS)*, 2014 IEEE 11th International Conference on (pp. 702–707). IEEE.
7. J Tai et al., Improving flash resource utilization at minimal management cost in virtualized flash-based storage systems. *IEEE Trans. Cloud Comp.* **5**(3), 537–549 (2017)
8. Yang, Z., Wang, J., Evans, D., & Mi, N. (2016). AutoReplica: automatic data replica manager in distributed caching and data processing systems. In *Performance Computing and Communications Conference (IPCCC)*, 2016 IEEE 35th International (pp. 1–6). IEEE.
9. Gong D, Liu S. A holographic-based model for logistics resources integration. *Studies in Informatics and Control.* **22**(4):367-376 (2013)
10. Bhimani, J., Mi, N., Leaser, M., & Yang, Z. (2017). FiM: performance prediction for parallel computation in iterative data processing applications. In *Cloud Computing (CLOUD)*, 2017 IEEE 10th International Conference on (pp. 359–366). IEEE.
11. Bhimani, J., Yang, Z., Leaser, M., & Mi, N. (2017). Accelerating big data applications using lightweight virtualization framework on enterprise cloud. In *High Performance Extreme Computing Conference (HPEC)*, 2017 IEEE (pp. 1–7). IEEE.
12. WANG, Jiayin, et al. eSplash: efficient speculation in large scale heterogeneous computing systems. In: *Performance Computing and Communications Conference (IPCCC)*, 2016 IEEE 35th International. IEEE, 2016. p. 1-8.
13. WANG, Jiayin, et al. SEINA: a stealthy and effective internal attack in Hadoop systems. In: *Computing, Networking and Communications (ICNC)*, 2017 International Conference on. IEEE, 2017. p. 525-530.
14. GAO, Han, et al. AutoPath: harnessing parallel execution paths for efficient resource allocation in multi-stage big data frameworks. In: *Computer Communication and Networks (ICCCN)*, 2017 26th International Conference on. IEEE, 2017. p. 1-9.
15. WANG, Teng, et al. EA2S2: an efficient application-aware storage system for big data processing in heterogeneous clusters. In: *Computer Communication and Networks (ICCCN)*, 2017 26th International Conference on. IEEE, 2017. p. 1-9.
16. RT Kouzes, GA Anderson, ST Elbert, et al., The changing paradigm of data-intensive computing. *Computer* **42**(1), 26–34 (2009)

17. Rajashree Dash, Debahuti Mishra, Amiya Kumar Rath, Milu Achrua. "A hybridized K-means clustering approach for high dimensional dataset", *Int. J. Eng. Sci. Technol.*, vol 2(2), (2010), pp.59-66.
18. J Dean, S Ghemawat, MapReduce: a flexible data processing tool[J]. *Commun. ACM* **53**(1), 72–77 (2010)
19. L Hong, K Luo, Rough incremental dynamic clustering method. *Comp. Eng. Appl.* **47**(24), 106–110 (2011)
20. Liao S, He Z. HDCH: audio data clustering system in the MapReduce platform. *Comput. Res. Dev.*. 2011,48(Suppl.):472-475.
21. Lee S D, Kao B, Cheng R. Reducing UK-means to K-means: data mining workshops, 2007. *ICDM Workshops 2007. Seventh IEEE International Conference on, 2007*[C]. IEEE.
22. Li W, Li M, Zhang Y, etc. Classification analysis based on distributed data warehouse. *Comp. Appl. Res.*, 2013,30(10):2936-2943.
23. Li L, Ouyang J, Liu D etc. Active collaboration classification combining characteristics selecting and link filter. *Computer Comput. Res. Dev.* 2013,50(11):2349-2357.
24. Liu J, Fu J, Wang S etc. Coordinate descend l2normLS-SVM classification algorithm. *Mode Identification and Artificial Intelligence.*
25. R Agrawal, T Imielinski, A Swami, in *Proceeding of the ACM SIG-MOD International Conference Management of Date. Mining association rules between sets of items in large databases* (Washington DC, 1993), pp. 207–216
26. Yan Y, Li Z, Chen H. Frequent items set mining algorithm. *Computer Science*, 2004,31(3):112-114.
27. Q Wu, Apriori mining algorithm based on cloud computing. *Comput. Measuring Control.* **20**(6), 1653–1165 (2012)
28. Y Li, Q Li, Maximal frequent itemsets mining algorithm based on constraints. *Comput. Eng. Appl.* **43**(17), 160–163 (2007)
29. I Zak, H Siao, in *Proc 2002 SIAM Int Conf Data Mining(SDM02)*. CHARM: an efficient algorithm for closed itemset mining (Arlington,VA, 2002), pp. 457–473
30. Hong Y. Distributed sensor network data flow mining algorithm of frequent itemsets. *Computer Science* 2013, 40(2):58-94.
31. Su L, Han W,Zou P, et al. Continuous kernel-based outlier detection over distributed data streams [C]. *Proc of Berlin:Springer*, 2007:74–85.
32. P Wang, D Meng, J Yan, B Tu, Research development of computer programming model of data intensive computing. *Comput. Res. Dev.* **47**(11) (2010)
33. TM PAN, in *Advances in future computer and control systems. The performance improvements of SPRINT algorithm based on the Hadoop platform* (Springer, Berlin Heidelberg, 2012), pp. 63–68
34. Shafer, J., Agrawal, R., & Mehta, M. (1996). SPRINT: a scalable parallel classifier for data mining. In *Proc. 1996 Int. Conf. Very Large Data Bases* (pp. 544–555).

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- Convenient online submission
- Rigorous peer review
- Open access: articles freely available online
- High visibility within the field
- Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com
