



MiCS-P: Parallel mutual-information computation of big categorical data on spark

Junli Li^a, Chaowei Zhang^c, Jifu Zhang^{b,*}, Xiao Qin^d, Lihua Hu^b

^a Department of Information Technology and Engineering, Jinzhong University, Jinzhong 030619, China

^b School of Computer Science and Technology, Taiyuan University of Science and Technology, Taiyuan 030024, China

^c Department of Computer Science Yangzhou University, Yangzhou 225127, China

^d Department of Computer Science and Software Engineering, Samuel Ginn College of Engineering, Auburn University, AL 36849-5347, United States of America

ARTICLE INFO

Article history:

Received 16 January 2021

Received in revised form 12 December 2021

Accepted 12 December 2021

Available online 16 December 2021

Keywords:

Parallel mutual-information computation

Feature grouping

Data skewness

Big categorical data

Spark

ABSTRACT

Mutual information can effectively measure the correlation between categorical attributes. However, it is found to be quite computationally intensive and time consuming process for enormous size and different distribution data sets. It involves steps for computation of marginal entropies, probability distribution and so on. Spark is a fast, general-purpose parallel framework designed specifically for large-scale data processing. Main motivation of this paper is to provide an intelligent method for parallel mutual information calculation based on Spark computing environment with maintaining the synchronization between different computing nodes. Proposed method named *MiCS-P* has been able to execute with different number of computing nodes, and gives significant speedup working with different dimensions and sizes of data sets. The *MiCS-P* algorithm adopts column-wise transformation scheme, which is conducive to the calculation of mutual information between a large number of feature pairs. And to alleviate imbalanced load causing long execution times, we implement a two-phase virtual partitioning scheme running on Spark.

© 2021 Elsevier Inc. All rights reserved.

1. Introduction

Mutual information is a measure of the amount of information shared between two random variables ([27][37]). Applied in feature grouping problems, mutual information can quantify the similarity between two features. To compute mutual information, one has to compute entropy and joint entropy between two variables; such a computation includes estimating a marginal probability and a joint probability of each sample. Therefore, speeding up mutual information computation of big data is a pressing need of modern data centers ([26]).

With the advent of the era of big data, the amount of data increases at a dramatic pace. The emergence of big-data applications brings huge challenges to data processing ([41][35]). In addition to building large-scale storage systems for big data, there is a growing demand to discover unknown and valuable information hidden behind big data. As such, large-scale data processing has been placed

under a spotlight in recent years. For instance, a growing number of efficient parallel computing platforms, such as Hadoop ([34]) and Spark ([10]), became widely adopted computing systems to cope with big data ([13]).

In this study, we propose a parallel computing system, called *MiCS-P*, for mutual-information computation of categorical data based on Spark. To demonstrate the strength of *MiCS-P*, we apply it to solve the feature grouping problem. In addition, we show how to tackle data skewness to cut back network and computing loads.

1.1. Observations

MiCS-P investigated in this study and its application in feature grouping are motivated by three observations.

Observation 1. Computing mutual information is a highly expensive process due to the enormous number of operations. Because computing mutual information has to compute entropy and joint entropy between two variables; such a computation includes steps for estimating a marginal probability and a joint probability of each sample. It is found to be quite computationally intensive

* Corresponding author.

E-mail addresses: 1468375302@qq.com (J. Li), cwzhang@yzu.edu.cn (C. Zhang), jifuzh@sina.com (J. Zhang), xqin@auburn.edu (X. Qin).

and time consuming process for enormous size and different distribution data sets (e.g., feature grouping in categorical data ([29])). Given big data, conventional mutual-information computation algorithms need more computational and storage resources ([40]). Datasets in modern applications become excessively large and; therefore, speeding up mutual-information computation is a practical way of significantly shortening runtime of applications processing categorical data. In order to wrestle with the contradiction between a massive dataset and the sequential computation of mutual information, we advocate for parallel mutual information calculation of big categorical data on parallel and distributed computing platforms.

Observation 2. Spark ([22]) can run on Hadoop's data source and fit well into the Hadoop ecosystem. Compared to the MapReduce programming model ([7]), Spark has the following three advantages. (1) The Spark framework caches calculation results in main memory, improves the ability to share data among iterative operations, and reduces the number of disk operations. (2) All the data operations in the Spark framework are powered by resilient distributed dataset (RDD). (3) Spark uses the event-driven libraries to start tasks to improve communication efficiency while maintaining low task-scheduling overhead.

Observation 3. In case of uneven data distribution, execution time is slowed down due to the unbalanced amount of data allocated to different partitions. Skewed data not only suppress the performance of our MiCS-P, but also reduce the utilization rate of cluster resources.

1.2. Contributions

The estimation of mutual-information (MI) is undoubtedly the most expensive operations in MiCS-P. Moreover, the column-wise processing is clearly much more manageable for mutual-information computation than row-wise processing. While facing a large-scale of categorical dataset, we first split an original dataset into feature subsets by applying a column-wise transformation. Next, we address the challenging issue of data skewness.

Our main contributions of this study are summarized as follows.

- A Spark-based parallel mutual-information computing system (MiCS-P) is proposed to process big categorical data.
- A column-wise transformation is designed and implemented on Spark to reduce mutual-information computation.
- A virtual partitioning scheme is utilized on Spark to mitigate the data skewness issue caused by uneven data distribution.
- Extensive experiments are carried out to evaluate MiCS-P using the synthetic and real datasets.

1.3. Roadmap

The rest of this study is organized as follows. In Section 2, we survey the related work of this study. In section 3, we summarize the preliminaries of this study. In Section 4, we formulate a problem statement and outline our design goals. Section 5 details the parallel computing mutual information and data skewness problem in MiCS-P on Spark. The Spark-based MiCS-P is implemented in Section 6. Section 7 discusses the experimental settings and the results. Finally, section 8 concludes our work.

2. Related work

2.1. Mutual information

Mutual information, a statistical description of the association degree between two random variables in the information theory,

can be expressed as a function of the probability of these two random variables ([27]).

Recently, mutual information have been widely used in various fields. Miura *et al.* ([23]) redefined the entropy and mutual information in one-dimensional traffic flow from the information theoretical perspective, focusing on the velocity variation of vehicles. Malladi *et al.* ([20]) presented mutual information in Frequency and its application to measure Cross-Frequency coupling in epilepsy. Yang *et al.* ([36]) introduced a new data reconciliation strategy based on mutual information for industrial processes. Coelho *et al.* ([5]) proposed a mutual information estimator for continuous and discrete variables applied to feature selection and classification problems. Yu *et al.* ([38]) introduced the fuzzy information entropy and fuzzy mutual information to furnish computing relevance between numerical or fuzzy features and decision. Hu *et al.* ([12]) proposed an approach to measuring relevance between discrete and continuous features using neighborhood mutual information. Todorov *et al.* ([32]) devised an entropy-based method to calculate correlation measures, which allows exhaustive exploration of variable subsets on real data.

2.2. Mutual-information-based feature grouping

Mutual information is beneficial for feature grouping thanks to its ability in revealing general dependencies between features. To search feature groups, a handful prior studies explored learning algorithms like logistic regression ([8]), K-means ([33]), classification ([6]), and self-organizing map ([28]). The other techniques take full advantage of mutual-information measures ([14] [30]).

Mutual-information-based techniques have attracted growing attention because of their exemplary performance. Lee *et al.* ([15]) proposed a way of dynamically selecting features using the feature-group mutual information; in contrast to static feature selection, Lee's scheme chooses each new feature based on its correlation with the previously selected features. Aki *et al.* ([4]) investigated bankruptcy prediction using a multiple classifier system with mutual-information feature grouping, where the input features for each base classifier are selected using its mutual information with respect to the output. Song *et al.* developed a method ([29]) ranking the mutual information between features; Song advocated for the fuzzy-C-means algorithm to compose groups.

Different from the aforementioned approaches, our MiCS-P is the first of its kind to parallel computing of mutual information on Spark. MiCS-P spearheads an effort towards applying parallel mutual-information computation to feature grouping of categorical data.

2.3. Parallel computation of mutual information

As the scale of the data expands, more and more parallel-computing strategies are proposed ([25][26][1]) to improve the calculation of mutual-information. Unlike these parallel algorithms based on GPU- or multi-core-based computing, the Spark computing platform is utilized for mutual-information calculation in our MiCS-P.

In recent years, some new models and algorithms have been proposed to solve the data skewness in Spark. For example, SCID ([31]) designed a Pond-sampling algorithm to collect data distribution information and to estimate the overall data distribution. In the process of data partitioning, SCID implements the Bin-packing algorithm to barrel the output of Map tasks. Furthermore, during the partitioning process, large partitions are further cut. SP-Partitioner ([18]) takes the arriving batch data as candidate samples and selects samples on the basis of systematic sampling to predict the characteristics of the intermediate data.

Relying on predicted results, this method generates a reference table to guide the uniform distribution of the next batch of data. Cartesian Scheduler ([24]) optimized the Cartesian (i.e., Cartesian product) operator. Because calculating Cartesian product requires join operations, data skewness may occur. SASM (Spark Adaptive Skew Mitigation) ([39]) was proposed to alleviate the data skewness problem by migrating large partitions to the other nodes while balancing the sizes among all the tasks. Unlike these existing schemes, our MiCS-P algorithm explores virtual partitioning, in which virtual prefix is attached in front of all the keys in a large partition, followed by a secondary Hash, so eliminate large partitions.

3. Preliminaries

This section describes the concepts of mutual information first. Then, we introduce the Spark computing framework to facilitate the presentation of MiCS-P. Last, we paint a detailed picture on mutual-information-based feature grouping of categorical data.

3.1. Mutual information

In study ([19]), $H(y_i)$ defines information entropy of feature y_i .

$$H(y_i) = - \sum_{k=1}^{d_i} P_i(y_i = v_{ik}) \times \log P_i(y_i = v_{ik}). \quad (1)$$

The joint entropy $H(y_i, y_j)$ of two features y_i and y_j is defined as

$$H(y_i, y_j) = - \sum_{k=1}^{d_i} \sum_{l=1}^{d_j} P_{ij}(y_i = v_{ik} \wedge y_j = v_{jl}) \times \log P_{ij}(y_i = v_{ik} \wedge y_j = v_{jl}), \quad (2)$$

$MI(y_i : y_j)$ defines Mutual information of two features y_i and y_j .

$$MI(y_i : y_j) = H(y_i) + H(y_j) - H(y_i, y_j) \\ = \sum_{k=1}^{d_i} \sum_{l=1}^{d_j} P_{ij}(y_i = v_{ik} \wedge y_j = v_{jl}) \times \log \frac{P_{ij}(y_i = v_{ik} \wedge y_j = v_{jl})}{P_i(y_i = v_{ik}) P_j(y_j = v_{jl})}. \quad (3)$$

All the notations used in Equations (1)(2)(3) have the same meaning. d_i and d_j are the number of categorical values in features y_i and y_j . P_i and P_{ij} calculate probabilities of single feature and feature pairs. $v_{i1}, \dots, v_{idi}$ and $v_{j1}, \dots, v_{jdj}$ are the domains of feature y_i and y_j . The symbol ' $\wedge$ ' in Equations (2)(3) express a "logical and".

3.2. The spark framework

As a distributed parallel processing framework, *Spark* not only maintains the advantages of the traditional MapReduce computing and its related models, but also resolves the performance bottleneck problems in Hadoop applications to machine learning and graph application (i.e., iterative algorithm and interactive query). *Spark* ([22]) has greatly improved the processing speed of big data through in-memory computing, meeting the demands of most usage scenarios.

Spark provides *Transformation* and *Action* operators that include not only capabilities like Map and Reduce functions in Hadoop, but

Table 1
Symbols.

Symbol	Description
DS	a categorical dataset
m	the number of features in feature set Y
Y	a feature set in DS
y_i	the i th feature in feature set Y
G	a feature group set
k	the number of feature groups
N	a pivot feature set
n_i	the i th pivot feature in feature set N

also RDD related operations. The caching strategy of RDD keeps the intermediate output result of Job in memory, while saving final results to the HDFS (Hadoop Distributed File System). In-memory computing of *Spark* can effectively improve the processing efficiency of iterative operations by reducing the cost of system I/O transmissions.

3.3. Mutual-information-based feature grouping

A handful of prior studies ([21], [16], [29], [4] and [3]) have focused their attention on incorporating mutual information to optimize feature grouping models.

Feature grouping is focused on clustering given features into groups, each of which contains highly relevant features. Clustering algorithms (e.g., K-means) could be used to create feature groups; various clustering methods and metrics could compose different feature groups. Before presenting the problem statement coupled with design goals of MiCS-P, we first introduce mutual-information-based feature-grouping algorithms.

3.3.1. Feature grouping strategies

Ideally, feature grouping strategies should cluster similar features into the same partition. The *K-means* algorithm is a clustering algorithm widely used in the field of data mining. Although K-means achieves exquisite performance in dealing with numeric data, K-means faces a road block in handling categorical data. Unlike numeric data, the values of categorical data are unordered, limited, and non-comparable ([9][17]). Sample categorical variables include gender, educational level, age group, and to name just a few.

We apply the clustering algorithm *K-modes* to create feature groups for categorical data. The K-modes algorithm - an extension of k-means algorithm - substitutes similarity for distance in the k-means algorithm.

3.3.2. Similarity measure

Similarity is a metric to quantitatively measure the correlation strength between two categorical features. We use the mutual information as a similarity metric for quantifying the similarities of categorical features.

Mutual information is incorporated for feature grouping thanks to mutual information's ability in revealing general dependencies between a pair of features.

3.3.3. Specific implementation steps

The mutual-information-based feature grouping algorithm aims to generate k feature subsets from an original dataset using the k-modes clustering approach. The symbols are shown in Table 1.

Let DS be a large categorical dataset, which contains n data objects and m attributes. Let $Y = \{y_1, y_2, \dots, y_m\}$ be a set of m categorical features. Let $G = \{G_1, G_2, \dots, G_k\}$ be a set of k feature groups. The feature grouping problem is addressed in this study as a sample use case of our proposed MiCS-P. The goal of feature grouping is to place each feature (e.g., y_j , $1 \leq j \leq m$) into a group,

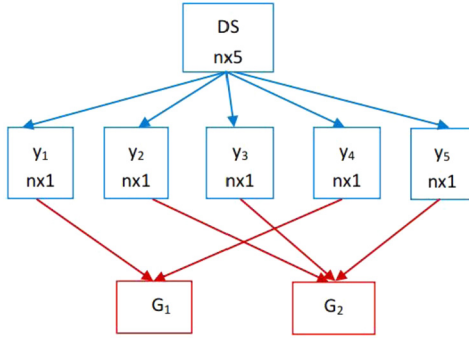


Fig. 1. The feature groups in dataset DS .

thereby forming the number k of feature groups from categorical dataset DS .

There are four specific steps in the feature-grouping algorithm.

Step 1: To choose k pivot features. In this step, k pivot features are selected from original feature set Y in a style reported in ([16]). K pivot features are constructed as a collection of $N = \{n_1, n_2, \dots, n_k\}$.

Step 2: For each feature y_i , calculate the mutual information between y_i and all the other pivot features. If $MI(y_i; n_r) \geq MI(y_i; n_s) (\forall s \in \{1, \dots, k\} - \{r\})$, then y_i is assigned to feature group G_r . After all the features are assigned, k feature groups will be obtained as $G = \{G_1, G_2, \dots, G_k\}$.

Step 3: For each feature group G_i , calculate mutual information between y_i and the other features in this group. The feature with the maximum SMI value in each group is selected; then, the set of pivot features N is updated.

Given feature group $G_r = \{y_j | j = 1, \dots, q\}$ and feature $y_i \in G_r$, SMI of y_i measures mutual information between y_i and the other features in feature group G_r . SMI is expressed as

$$SMI(y_i) = \sum_{j=1}^q MI(y_i; y_j) (i \neq j). \quad (4)$$

$SMI(y_i)$ in (4) is a summation of mutual information (MI) between $y_i \in G_r$ and the other features in G_r . Comparing MI values of all features in feature group G_r , and we choose the feature with the maximal SMI value in G_r as a pivot feature n_r of group G_r .

Step 4: Performing the above Steps 2 and 3 repeatedly until all the number k of pivot features for the feature groups remain unchanged.

Fig. 1 displays the relationship between the features set Y and the feature groups set G in the dataset DS . There are five categorical features in feature set Y , $Y = \{y_1, y_2, \dots, y_5\}$. These features are divided into two feature groups; thus, we have $G = \{G_1, G_2\}$, where $G_1 = \{y_1, y_4\}$, $G_2 = \{y_2, y_3, y_5\}$.

4. Problem statement and design goals

In this section, we first describe the existing problems in the parallel implementation (see Section 4.1). Then, we outline the design goals of our proposed MiCS-P (see Section 4.2).

4.1. Problems

Recall that the mutual-information-based feature grouping approach (see Section 3.3.3) indicates that mutual-information computation (i.e., Step 2 and Step 3 in Section 3.3.3) is a performance bottleneck of the entire feature-grouping process.

We summarize below the two existing problems in parallel mutual information computation of categorical feature grouping.

- **Problem 1.** Mutual-information-based feature grouping is similar to traditional clustering algorithms; nevertheless, the difference is that mutual-information-based feature grouping algorithms are to cluster features rather than data objects. The computation of pairwise mutual information (MI) takes up an excessive amount of time. MI between all the features and each pivot feature must be repeatedly calculated, followed by updating the centers of all the feature groups. The algorithm won't terminate until no pivot feature is updated in the iteration.
- **Problem 2.** In the Spark computing framework, all data exists in the form of key-value pairs (i.e., $[K, V]$); all operations for RDDs are based on partitions of RDDs. Spark needs to perform an all-to-all operation, namely, reading from all partitions to find the values for all keys, and then bring together values across partitions to compute a final result for each key. In Spark's scheduling policy, each task is responsible for a partition. For a Reduce task, imbalance of RDD partitions results in uneven data partitions processed by different tasks. Imbalanced data partitions reduce the utilization rate of Spark cluster resources due to (1) long task execution times and (2) a large number of idle nodes.

4.2. Design goals

The overall design goals of MiCS-P are (1) to reduce the amount of computation caused by repeatedly computing mutual information and (2) to alleviate the data skewness issue induced by uneven data distribution.

To fulfill the first design goal, we propose in MiCS-P a column-wise transformation in accordance with (1) the characteristics of feature variables and (2) the resource requirements of computing tasks. The column-wise transformation module in MiCS-P strives to minimize the computing cost of single-feature-value frequency and feature-pair-value frequency caused by mutual-information calculations. After transformation is accomplished, mutual information is computed in parallel, and transformed data are cached and reuse in the subsequent iterations.

To achieve the second design goal, we design a virtual-partitioning approach to addressing the issue of data skewness. Mitigating the data skewness is expected to dramatically boost the performance of our MiCS-P.

5. Parallel mutual-information computing

We start this section by presenting a column-wise transformation scheme in Section 5.1. Then, we describe the data skewness problem in Spark (see Section 5.2). Last, in Section 5.3, we adopt a virtual partition scheme to mitigate the data communication cost and workload imbalance problems caused by skewed data.

5.1. Column-wise transformation

Recall that we use mutual information to measure the similarities of feature variables (see Section 3.3). Equation(3) calculates the frequencies of each feature variable and feature variable pair, respectively. Intuitively, it is easier to manage feature variables by column than by row. In order to calculate MI in parallel, the MiCS-P decomposes data set DS into several feature subsets.

Fig. 2 shows the column-wise transformation. In data set DS , n is the size of data objects, $\{x_0, x_1, \dots, x_{n-1}\}$ are all data objects. m is the number of features of each data object, $\{y_1, y_2, \dots, y_m\}$ are features. By using column-wise transformation, $\{y_1, y_2, \dots, y_m\}$ are divided into different subsets. Each subset forms an RDD.

Parallel mutual information calculating in the MiCS-P mainly involves the following two operations.

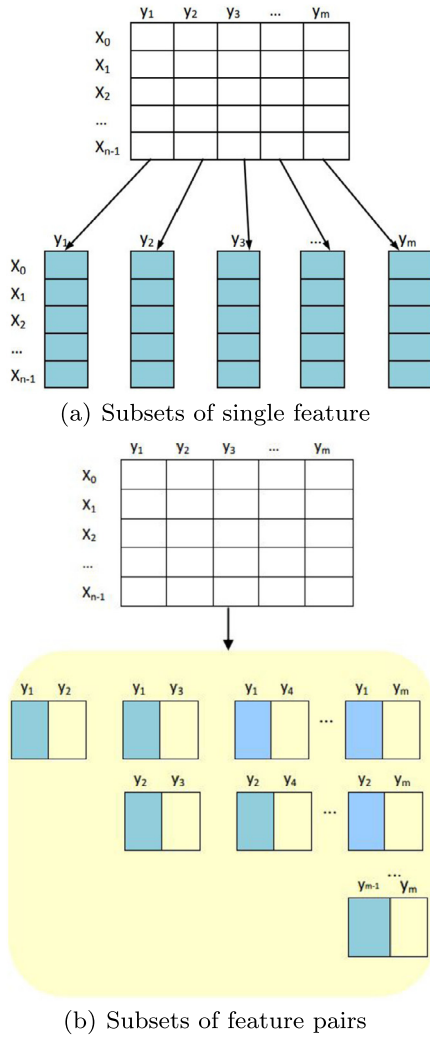


Fig. 2. The column-wise transformation.

Table 2
The *singleCol* of MiCS-P.

Indexes	1	2	3	4	5	...
value	(e,15)	(d,27)	(a,13)	(c,45)	(f,16)	...

- **Map Operation.** The operator Map converts RDD into a key-value pair format. The key means the value of feature; The value represents the number of occurrences. In order to compute MI (see Equation(2)) of feature pair, we need to track all occurrences of each feature.
- **CountByKey or reduceByKey Operation.** In a Spark cluster, each node independently counts local occurrences; Then, the local occurrences are aggregated by the aggregation operator.

To reduce the computational burden caused by an increase in the number of iterations during feature grouping, the frequency values of single features and feature pairs are computed and buffered into variable-length arrays called *singleCol* and *doubleCol*. The computing task then loads the data from the two arrays, avoiding the repetition of calculations of single-feature and feature-pair, thereby reducing computing load.

Table 2 and Table 3 give an example of *singleCol* and *doubleCol* in our MiCS-P.

Mutual information captures the correlation between a pair of features; thus, the values of $MI(y_i; y_j)$ and $MI(y_j; y_i)$ are the

Table 3
The *doubleCol* of MiCS-P.

ln	1	2	3	4	5	...
1		((b,c),9)	((z,n),63)	((a,b),65)	((e,d),76)	...
2			((f,e),12)	((n,c),34)	((m,f),24)	...
3				((d,e),87)	((z,n),19)	...
4					((f,c),92)	...
5						...
...						...

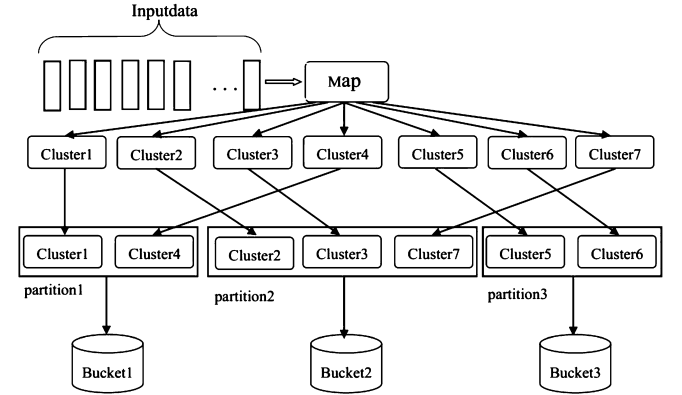


Fig. 3. The default hash distribution in the Spark Shuffle phase, in which intermediate results are grouped into partitions in accordance with keys. Skewed data leads to uneven-sized partitions.

identical. While calculating the frequency of feature pair values, we simply compute each feature pair once (see Table 3).

5.2. Data skewness

We first give an overview of skewed data in Spark shuffle (see Section 5.2.1). Next, Section 5.2.2 specifically describes the data skewness model.

5.2.1. An overview

In the shuffle phase, Spark must pull the same key from each node to a task on a node. Such a process is likely to burden a single node with heavy load. As shown in Fig. 3.

While encountering the data skewness issue, the execution time of an entire Spark job is dominated by a task with the longest running time, making the Spark job run fairly slow. In the worst case scenario, the Spark job may run out of memory due to an excessive amount of data processed by the slowest task.

5.2.2. The data-skewness model

A data-skewness model is built to quantify the problems caused by data skewness among data partitions in Spark. Prior to presenting the model, we elaborate the three major steps carried out in Spark (see Fig. 3). Next, we introduce the notation of key set, value set, domains, domain size, and average domain size, which give rise to the definition of skewness degree in (7).

Assuming data are aggregated to p unique keys, we use K to represent all the keys, $K = \{k_1, \dots, k_p\}$. And we present V as the values of keys in the set K . Thus, the value set V is presented as

$$V = \{ \underbrace{v_{11}, \dots, v_{1l_1}}_{dom(k_1)}, \dots, \underbrace{v_{i1}, \dots, v_{il_i}}_{dom(k_i)}, \dots, \underbrace{v_{p1}, \dots, v_{pl_p}}_{dom(k_p)} \}, \quad (5)$$

where there are p partitions; The values of every partition share a key, each key has different domain. For example, $dom(k_i)$ represents the domain of key k_i , $dom(k_i) = \{v_{i1}, \dots, v_{il_i}\}$, $i \in 1, \dots, p$. It is worth noting that all partitions may have different sizes (i.e.,

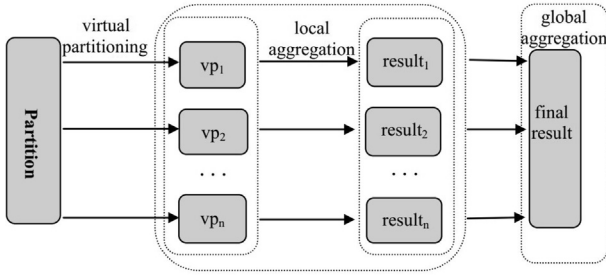


Fig. 4. The process of virtual partitioning.

$l_i \neq l_j$). We use $|dom(k_i)|$ to denote the domain $dom(k_i)$'s size. $|dom(K)|_{avg}$ represents the average partition size, which is presented as follows:

$$|dom(K)|_{avg} = \frac{\sum_{i=1}^p |dom(k_i)|}{p} = \frac{\sum_{i=1}^p \sum_{j=1}^{l_i} v_{ij}}{p} \quad (6)$$

The degree of skewed data among p partitions is defined as the deviation of the partition size, which is also the size of the domain (a.k.a., $|dom(k_i)|$). We let $s(k_i)$ represent skewness degree of the i th partition, $s(k_i)$ relative to the domain $dom(k_i)$ is formally defined as follows

$$s(k_i) = \frac{|dom(k_i)|}{|dom(K)|_{avg}} = \frac{p \times \sum_{j=1}^{l_i} v_{ij}}{\sum_{i=1}^p (\sum_{j=1}^{l_i} v_{ij})}, \quad (7)$$

5.3. Handling skewed data in MiCS-P

In this section, we describe the problem triggered by skewed data in the parallel process of MiCS-P (see Section 5.3.1) and give the solution (see Section 5.3.2).

5.3.1. Data skewness: a problem

Now we are positioned to shed light on data skewness tackled by our MiCS-P running on Spark.

The aggregation operator is a performance bottleneck in the Spark Shuffle phase. A key challenge in parallel computing mutual information lies in the *countByKey* or *reduceByKey* operators (see Section 5.1), which kick in shuffle that embraces two stages. During the course of shuffling, the first stage performs the *Shuffle Write* operation partitioning data. Processed data with the same key are written to the same disk file.

As soon as the *countByKey* or *reduceByKey* operator executes, each task in the second stage executes the *Shuffle Read* operation. Tasks performing this operation pull out keys that belongs to the task nodes in the previous stage, followed by performing a global aggregation or join operation on the same key. In this scenario, the keys' values are accumulated. If the data distribution is uneven, skewed data will occur.

5.3.2. Virtual partitioning: a solution

Virtual partitioning is a mechanism to shuffle operators (e.g., *reduceByKey*).

As shown in Fig. 4, virtual partitioning is used to slash the excessive amount of data processed by a single task. After the prefix of each key is removed, the global aggregation operation (a.k.a., *reduceByKey*) is carried out again to obtain a final result.

6. Algorithm design

We start to address the design issue by focusing on column-wise transformation and virtual partitioning in Section 6.1. Next, we discuss the implementation of computing mutual information

in Section 6.2. Finally, section 6.3 presents MiCS-P's main algorithm grouping features of big categorical data on Spark.

6.1. Column-wise transformation and partitioning

Algorithm 1 depicts the column-wise transformation (see Section 5.1) and virtual partitioning (see Section 5.3.2).

Algorithm 1 Transformation and Partitioning.

```

1: Input: Dataset DS ( $n \times m$ )
2: Input: Datapre, RDD built from DS
3: Output: two arrays
4: val doubleCol
5:   =new Array[ArrayBuffer[Map[(String,String),Long]]](dim)
6: for ( $m \leftarrow 0$  until  $dim$ ) /* 'dim' is the dimension of data set */
7:   for ( $n \leftarrow 0$  until  $dim$ ) /* 'dim' is the dimension of data set */
8:     doubleCol(m)(n) += datapre.map(x => ((x(m), x(n)), 1))
9:     .countByKey().toMap
10:  end for
11: end for
12: val singlecol = ArrayBuffer[Map[String,Long]]()
13: for ( $k \leftarrow 0$  until  $dim$ )
14:   singleCol += datapre.map(
15:     val random:Random = new Random()
16:     val prefix: Int = random.nextInt(10)
17:     prefix += "x(k), 1)
18:     .reduceByKey(_+_ )
19:     .map(line => (line._1.split("-")(1), line._2))
20:     .reduceByKey(_+_ )
21:     .collectAsMap().toMap
22:  end for

```

Step 1. The *map* transformation converts RDD *datapre* into the key-value pair format, for example, $((x(m), x(n)), 1)$. The key (i.e., $((x(m), x(n)))$ means the value of feature; The value 1 represents the number of occurrences.

Step 2. In order to compute MI of feature y_m and feature y_n , we need to track all occurrences of each feature (i.e., $x(m), x(n)$) and feature pair (e.g., $(x(m), x(n))$).

Given a feature-value pair $(x(m), x(n))$, we articulate how to calculate the overall occurrences of $(x(m), x(n))$ stored across all the data nodes in Spark Cluster, which is a parallel computing in distributed manner. Each computing node independently counts $(x(m), x(n))$'s local occurrences in Spark Cluster. Therefore, multiple computing nodes are measuring the occurrences of $(x(m), x(n))$ in parallel.

Array *doublecol* saves the number of times feature-value pair occurs, all the data stored in *doublecol* are in the form of key-value (e.g., $((x(m), x(n)), 5)$).

Step 3. The variable-length array *singleCol* is similar to *doubleCol*, except that *singleCol* tracks the single-feature values that appear on one dimension.

Step 4. Virtual partitioning (see lines 15–20) is used to alleviate data skewness problems that are prone to occur when calculating a single-feature value (see Section 5.3.2).

Line 16 and 17 add a random prefix to each key in RDD. The *reduceByKey* aggregation is implemented in Line 18. After Line 19 remove the prefix of each key, Line 20 executes the global aggregation operation.

6.2. Mutual-information computing

Algorithm 2 computes mutual information of feature pairs using the frequency of single-feature values and feature-pair values.

Two arrays *singleCol* and *doubleCol* produced by Algorithm 1 are the input of Algorithm 2, which carries out the following three steps.

Step 1. Function *Pr1* calculates the probability of occurrence of any single-feature values (see Lines 5–12); function *Pr2* computes

Algorithm 2 Computing Mutual information (MI).

```

1: Input: Dataset DS ( $n \times m$ )
2: Input: array of singleCol, array of doubleCol
3: Input: column number of two features
4: Output: MI between feature pair
5: def pr1(t:Map[String,Long],k:String):Double
6:   var p1: Double = 0
7:   for ( $i \leftarrow t.keys$ )
8:     if ( $i.equals(k)$ )
9:        $p1 = t(i)/n$ 
10:   endif
11: endfor
12: end def
13: def pr2(t:Map[(String,String),Long],k(String,String)):Double
14:   var p2: Double = 0
15:   for ( $i \leftarrow t.keys$ )
16:     if ( $i.equals(k)$ )
17:        $p2 = t(i)/n$ 
18:     endif
19:   endfor
20: end def
21: def CalculateMI( $y_1 : Int, y_2 : Int$ ) : Double
22:   val t1 = doubleCol( $y_1$ )( $y_2$ )
23:   val t2 = singleCol( $y_1$ )
24:   val t3 = singleCol( $y_2$ )
25:   var MI : Double = 0
26:   for ( $i \leftarrow t1.keys$ )
27:     val  $p_1 = pr2(t1, i)$ 
28:     val  $p_2 = pr1(t2, i...1)$ 
29:     val  $p_3 = pr1(t3, i...2)$ 
30:      $MI += pr2 * Math.log(p_1 / (p_2 * p_3))$ 
31:   end for
32:   return MI
33: end def

```

the probability of occurrence of any feature- pair values (see Lines 13–20).

Step 2. The column numbers of the feature pair (e.g., (y_1, y_2)) are retrieved from the input. $t1$ extracts all values corresponding to feature pair (y_1, y_2) from *doubleCol* (see Line 23), $t2$ and $t3$ extract all values corresponding to single feature y_1 and y_2 from *singleCol*, respectively (see Lines 23 and 24).

Step 3. Governed by (3), Lines 27–29 compute three probabilities (i.e., p_1, p_2 , and p_3) - three building blocks for measuring mutual information finalized in Line 30. Line 32 returns the mutual-information value MI .

6.3. Main MiCS-P algorithm

Algorithm 3 Main function.

```

1: Input: Dataset DS ( $n \times m$ )
2: Input:  $k$  /* the Number of groups */
3: Output:  $k$  feature groups
4: def main(args: Array[String])
5:   val conf = new SparkConf().setAppName("MiCS-P Application")
6:   val sc = new SparkContext(conf)
7:   val data = sc.textFile("hdfs://SparkMaster:9000
8:     /input/input.txt")
9:   val len:Double = data.count()
10:  val dimension: Int = data.first().split(',').length
11:  val datapre=data.map(lines => lines.split(','))
12:  .persist(StorageLevel.MEMORY_AND_DISK)
13:  val fm  $\leftarrow k$  pivot features
14:  for ( $r = 1; r \leq k; r++$ )
15:    for ( $i = 1; i \leq m; i++$ )
16:      Calculate MI ( $y_i : n_r$ ) of  $y_i$  and  $n_r$  according to Algorithm 2;
17:      if  $MI(y_i : n_r) \geq MI(y_i : n_s)$  ( $s \in \{1, \dots, k\} - \{r\}$ )
18:        assign  $y_i$  to  $G_r$ 
19:      end if
20:    end for
21:    Updates fm;
22:  endfor
23:  Steps 13–21 are repeatedly performed until fm remains unchanged.

```

Table 4

Synthetic and real categorical datasets.

Datasets	Type	# Objects	# Dims
uniform(8GB)	Synthetic	2×10^8	100
uniform(16GB)	Synthetic	4×10^8	100
uniform(24GB)	Synthetic	6×10^8	100
uniform(32GB)	Synthetic	8×10^8	100
normal(8GB)	Synthetic	2×10^8	100
normal(16GB)	Synthetic	4×10^8	100
normal(24GB)	Synthetic	6×10^8	100
normal(32GB)	Synthetic	8×10^8	100
Internet-adv	Real(UCI)	3279	1558
Connect-4	Real(UCI)	44916	42
Mushroom	Real(UCI)	3551	22
Chess	Real(UCI)	1699	36
Breast-cancer	Real(UCI)	451	9
Splice	Real(UCI)	1677	61

The main function of MiCS-P creates a context for Spark, reading a dataset and partitioning features of the dataset into multiple groups. Algorithm 3 outlines the pseudocode of MiCS-P's main function, which consists of three steps.

Step 1. Initializing the system.

Step 2. Converting and caching RDD.

Step 3. Grouping features.

We select k initial features to list fm . Algorithm 2 measures the relations between features. If $MI(y_i : n_r) \geq MI(y_i : n_s)$ ($s \in \{1, \dots, k\} - \{r\}$), y_i is assigned to feature group G_r .

After all the features are assigned, we obtain k feature groups $G = \{G_1, G_2, \dots, G_k\}$. And, we updates list fm . Finally, we identify the final feature groups until fm remains unchanged.

7. Experimental results

7.1. Experimental setup

We implement and evaluate the MiCS-P system on Spark Cluster with 24 computing nodes. Each computing node has an Intel processor (i.e., E5-1620 v2 series 3.7 GHz) with four cores and 16GB RAM. We connect all the data nodes via the *Gigabit Ethernet* in the Spark Cluster; the communication among the nodes are facilitated by the *SSH* protocol. The Master of Spark Cluster controls the *HDFS* file system.

7.1.1. Spark cluster configuration

We use the *Scala* programming language in the implementation of MiCS-P. *Scala*, a pure object-oriented programming language, seamlessly combines imperative and functional programming styles. The *IntelliJ IDEA* development environment is used to develop our MiCS-P algorithm.

7.1.2. Data sets

Real-world datasets. All the real-world tested datasets are obtained from the UCI datasets, which is shown in Table 4.

Synthetic datasets. To evaluate our MiCS-P, we construct two types of synthetic datasets: uniform datasets and normal datasets. All the synthetic datasets are generated in the following steps. First, we use the random data generator to generate a relatively small data set. Next, to increase the size of the data set, we copy the data set that was generated in the first step; the data sizes are respectively configured as 8GB, 16GB, 24GB and 32GB, which all have 100 features. We use the synthetic categorical datasets to evaluate the scalability and extensibility of our MiCS-P algorithm.

7.1.3. Alternative algorithms

MiCS-P is compared against three alternative algorithms, the descriptions of which are listed below.

Table 5
The feature groups generated by the MiCS-P, WATCH, and ACA algorithms.

Groups	The MiCS-P Algorithm	The ACA Algorithm	The WATCH Algorithm
G ₁	Y ₁ ,Y ₂ ,Y ₃ ,Y ₄ ,Y ₅ ,Y ₆	Y ₁ ,Y ₂ ,Y ₃ ,Y ₄ ,Y ₅ ,Y ₆	Y ₁ ,Y ₂ ,Y ₃ ,Y ₄ ,Y ₅ ,Y ₆
G ₂	Y ₇ ,Y ₈ ,Y ₉ ,Y ₁₀ ,Y ₁₁ ,Y ₁₂	Y ₉ ,Y ₁₀ ,Y ₁₁ ,Y ₁₂	Y ₇ ,Y ₈ ,Y ₉ ,Y ₁₀ ,Y ₁₁ ,Y ₁₂
G ₃	Y ₁₃ ,Y ₁₄ ,Y ₁₅ ,Y ₁₆ ,Y ₁₇ ,Y ₁₈	Y ₁₃ ,Y ₁₄ ,Y ₁₅ ,Y ₁₆ ,Y ₁₇ ,Y ₁₈	Y ₁₃ ,Y ₁₄ ,Y ₁₅ ,Y ₁₆ ,Y ₁₇ ,Y ₁₈
G ₄	Y ₁₉ ,Y ₂₀ ,Y ₂₁ ,Y ₂₂ ,Y ₂₃ ,Y ₂₄	Y ₁₉ ,Y ₂₀ ,Y ₂₁ ,Y ₂₂ ,Y ₂₃ ,Y ₂₄ ,Y ₂₆ ,Y ₂₇ ,Y ₂₈	Y ₁₉ ,Y ₂₀ ,Y ₂₁ ,Y ₂₂ ,Y ₂₃ ,Y ₂₄
G ₅	Y ₂₅ ,Y ₂₆ ,Y ₂₇ ,Y ₂₈ ,Y ₂₉ ,Y ₃₀	Y ₇ ,Y ₈ ,Y ₃₀	Y ₂₅ ,Y ₂₆ ,Y ₂₇ ,Y ₂₈ ,Y ₂₉ ,Y ₃₀
G ₆	Y ₃₁ ,Y ₃₂ ,Y ₃₃ ,Y ₃₄ ,Y ₃₅ ,Y ₃₆	Y ₂₅ ,Y ₃₁ ,Y ₃₂ ,Y ₃₃ ,Y ₃₄ ,Y ₃₅ ,Y ₃₆	Y ₃₁ ,Y ₃₂ ,Y ₃₃ ,Y ₃₄ ,Y ₃₅ ,Y ₃₆
G ₇	Y ₃₇ ,Y ₃₈ ,Y ₃₉ ,Y ₄₀ ,Y ₄₁ ,Y ₄₂	Y ₂₉ ,Y ₃₇ ,Y ₃₈ ,Y ₃₉ ,Y ₄₀ ,Y ₄₁ ,Y ₄₂	Y ₃₇ ,Y ₃₈ ,Y ₃₉ ,Y ₄₀ ,Y ₄₁ ,Y ₄₂

ACA ([2]): The ACA algorithm is a feature grouping method, which can be used for feature selection, classification and so on. And the method is able to group features in accordance with the features' interdependence.

WATCH ([16]): WATCH is a feature grouping technique, which is adopted to detect outliers. And the method can group features according to feature correlation.

DEFH ([11]): DEFH is the hash algorithm used by default in Spark. It achieves a good performance when keys exhibit a uniform distribution.

ACA and WATCH are used to compare feature grouping results with MiCS-P (see Section 7.2), and DEFH is used to compare parallel performance with MiCS-P in Spark Cluster (see Section 7.4, 7.5, 7.6).

7.2. Assessment of feature grouping

To evaluate grouping results, we apply MiCS-P to a UCI dataset named *Connect-4*. Each object in the dataset consists of 42 categorical features pre-classified into seven feature groups - G₁, G₂, G₃, G₄, G₅, G₆ and G₇. We denote the features as y₁, ..., y₄₂, where G₁ = {y₁, ..., y₆}, G₂ = {y₇, ..., y₁₂}, G₃ = {y₁₃, ..., y₁₈}, G₄ = {y₁₉, ..., y₂₄}, G₅ = {y₂₅, ..., y₃₀}, G₆ = {y₃₁, ..., y₃₆}, G₇ = {y₃₇, ..., y₄₂}.

We apply the ACA, WATCH and MiCS-P algorithms on the *Connect-4* dataset for producing feature groups G₁ – G₇ in this experiment.

Table 5 summarizes the grouping results yielded by ACA, WATCH and our MiCS-P. The results show that the feature groups obtained by WATCH and MiCS-P are superior to that of ACA algorithm. WATCH and our MiCS-P all create seven feature groups, which are consistent with the pre-classified feature groups of the original dataset. In contrast, the ACA algorithm created seven groups with a significant difference from the ideal group. This difference is due to initial features that are randomly selected by the ACA algorithm. Different from ACA, MiCS-P and WATCH choose the initial pivot features to minimize the correlation between the selected pivot features.

7.3. Impacts of column-wise transformation

We evaluate the impacts of column-wise transformation on efficiency of MiCS-P in this group of experiments.

Fig. 5 shows that the column-wise transformation significantly optimizes the performance of the MiCS-P under six conditions. For example, when it comes to the Internet-adv and Connect-4 datasets, the column-wise transformation boosts the feature-grouping performance obviously. Comparing the speedup performance among all the datasets, we can see that as the data size and dimension increases, the performance improvement for column-wise transformation becomes more significant.

7.4. Impacts of virtual partitioning

To evaluate the virtual partitioning impacts on the efficiency of MiCS-P, we compare the efficiency of MiCS-P with DEFH processing

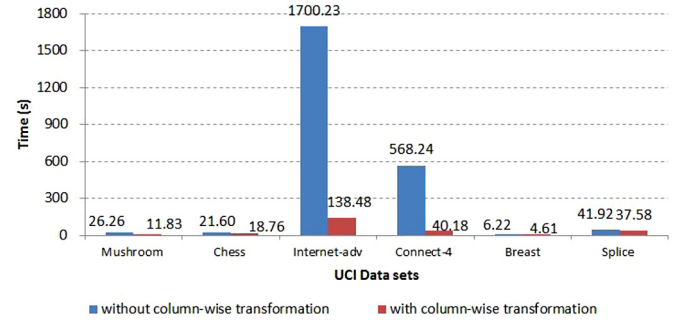
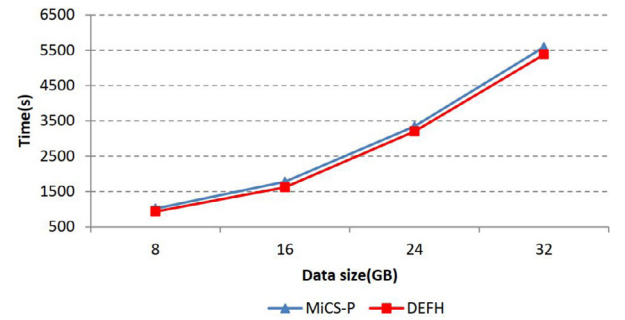
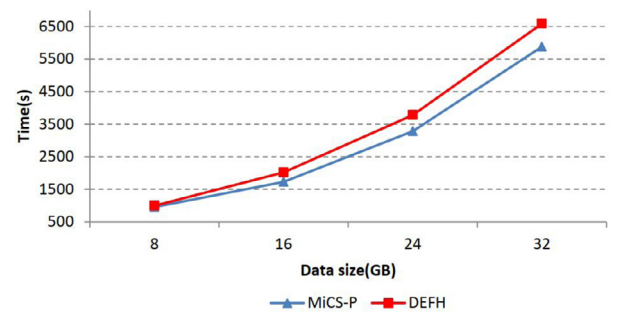


Fig. 5. The running time of MiCS-P on the six UCI datasets. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)



(a) uniform data



(b) normal data

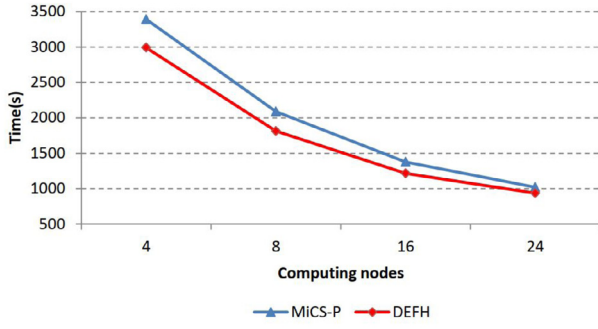
Fig. 6. Execution time of uniform data and normal data under various data sizes.

two types of synthetic datasets (i.e., uniform datasets and normal datasets).

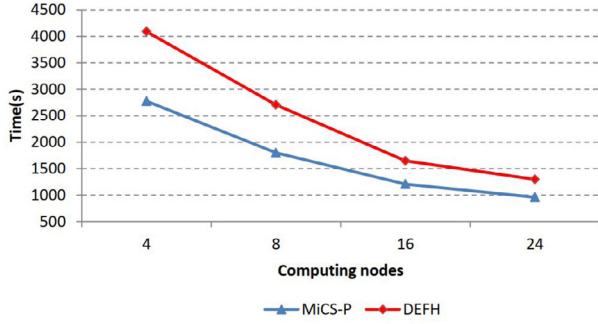
7.4.1. Execution time under various data size

Fig. 6 shows the execution time of MiCS-P and DEFH handling data with uniform and normal distributions. We set data size to 8GB, 16GB, 24GB and 32GB, and computing nodes number to 24.

Fig. 6(a) reveals that in the datasets with uniform distributions, the running times of MiCS-P are longer than those of DEFH due to the side-effect of virtual partitioning. Nevertheless, Fig. 6(b)



(a) uniform data



(b) normal data

Fig. 7. Execution time under the uniform and normal data on various computing nodes.

shows that MiCS-P is superior to DEFH in the case of datasets with normal distributions. This result is expected, because the datasets contain skewed data - judiciously handled by MiCS-P powered by virtual partitioning.

The second observation drawn from Fig. 6(a) and 6(b) is that raising data size leads to increased running times of all the algorithms. Intuitively, it takes long time period to process a large amount of data.

7.4.2. Execution time under various computing nodes

Fig. 7 unravels the execution time of MiCS-P and DEFH processing the uniform and normal datasets on various number of computing nodes. The number of nodes is set to 4, 8, 16, and 24. The data size is set to 8GB.

Fig. 7(a) reveals that the running times of our MiCS-P in the uniform datasets are longer than those of DEFH due to the overhead of virtual partitioning. This trend is consistent with that plotted in Fig. 6(a).

Fig. 7(b) shows that MiCS-P outperforms DEFH in the case of normal datasets exhibiting data skewness. The performance improvement of MiCS-P over DEFH is attributed to virtual partitioning, which effectively alleviates data skewness. Again, these results are confirmed by Fig. 6(b).

We conclude from Figs. 7(a) and 7(b) that when the number of nodes increases, the running times of all the algorithms drop thanks to the expanded computing capacity.

7.5. Impacts of data-skewness degree

Fig. 8 shows the difference in average processing time between MiCS-P and DEFH when the data-skewness increases from 1 to 3. As can be seen from the Fig. 8, compared with DEFH, the average processing time of MICS-P changes little with the increase of data-skewness degree and is less sensitive to data skewness. For example, when we lift the degree from 1.5 to 3, MiCS-P and DEFH's

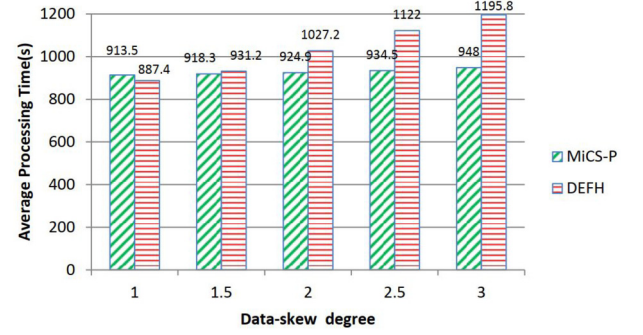


Fig. 8. The impacts of data-skewness degree on the processing times of MiCS-P and DEFH.

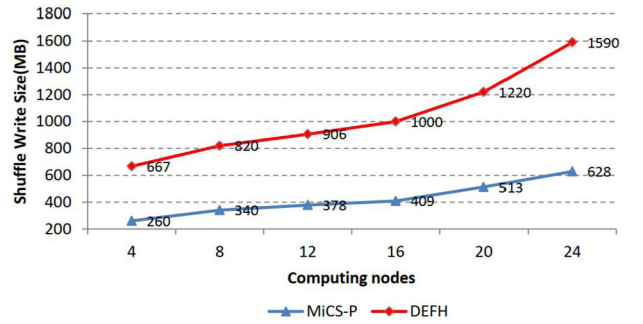


Fig. 9. The shuffling cost of MiCS-P and DEFH running on various number of computing nodes.

processing times go up by 7.2% and 28.4%, respectively. The results show that MiCS-P effectively alleviates the performance problem caused by data skewness using virtual partitioning. Consequently, MiCS-P is superior to DEFH in unbalance datasets; The higher the data skewness, the better the performance of DEFH can be improved by MiCS-P.

As the data-skew degree increases, due to the performance improvement by virtual partitioning mechanism, the growth rate of the processing time of MiCS-P is smaller than DEFH within a certain scope. This experiment demonstrates that MiCS-P can achieve a relatively better performance with skew degree varies from 1 to 3 with an increment 0.5. The larger skew degree of data, the more improvements can be achieved.

7.6. Shuffling-cost analysis

We compare the shuffling-cost of MiCS-P and DEFH algorithms by changing the number of computing nodes. The *shuffle write size* of nodes is used to monitor the shuffling-cost of the algorithms. Take the normal data (8G) as an example, Fig. 9 shows the comparison diagram of two algorithms.

Obviously, the MiCS-P algorithm's *shuffle write sizes* are significantly smaller than DEFH in all test cases. What's more, as the number of computing nodes increases, so does shuffle-write-size gap between MiCS-P and DEFH. DEFH relies on Spark's default hash partition, causing tasks to frequently access data across multiple nodes. As an example, when the number of nodes changes from 4 to 24, the *shuffle write size* of DEFH climbs from 667.0 MB to 1590.0 MB. Unlike the DEFH algorithm, the MiCS-P algorithm utilizes a virtual-partitioning technique to reduce the amount of data transfer in the Spark environment. As a result, the *shuffle write size* of MiCS-P merely jumps from 260.0 MB to 628.0 MB - a much better scenario in terms of shuffling costs than that of DEFH.

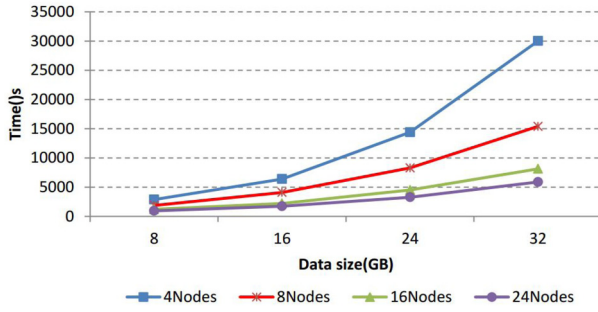


Fig. 10. The execution time of MiCS-P under various data size.

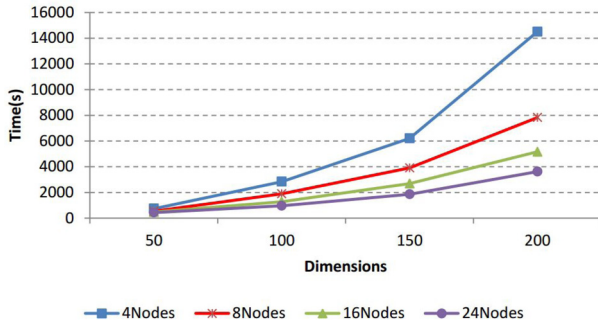


Fig. 11. The execution time of MiCS-P under various dimensions.

7.7. Extensibility

Now, we verify the performance of MiCS-P running on various computing nodes with increasing data size and dimensions.

7.7.1. Impacts of data size

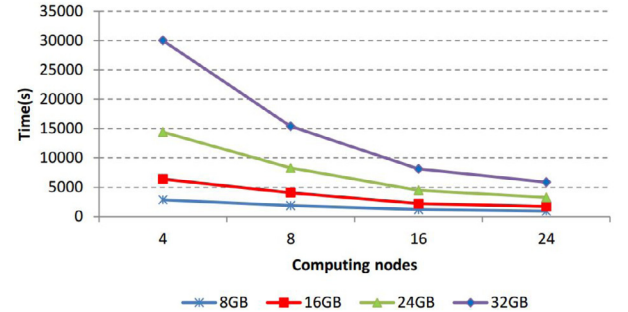
Fig. 10 plots the influences of data size on the performance of MiCS-P. When the data size expands from 8GB to 32GB with an increment of 8G, the execution time rapidly increases in the case of four computing nodes. This trend is predictable because processing a large dataset on a small number of nodes takes longer time. When the number of nodes is relatively small, the execution time of MiCS-P is quite sensitive to data size. The number of nodes increase, the computing capacity expands, which leads to a decrease in processing time. Such comparison indicates that computing mutual information on a large Spark cluster is a practical method to shorten the processing time of big categorical datasets.

7.7.2. Impacts of dimensions

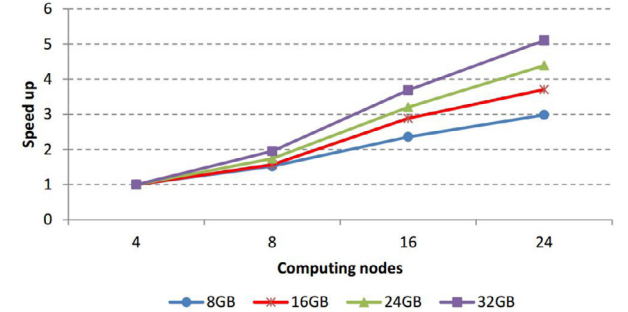
Fig. 11 presents the performance trend when the dimension changes from 50 to 200 with the increase of 50. Due to the calculation of mutual information between feature pairs in MiCS-P, the execution time is enlarged with the increasing dimensions.

Keeping the dimensions as a constant, we reduce the execution time by extending the number of computing nodes. This trend makes sense because the computation of mutual information is speeded up by increasing the number of computing nodes.

Also, when the number of nodes is relatively small, the MiCS-P's run time is sensitive to the number of dimensions. For example, on the 4-node Spark cluster, the dimensions vary from 50 to 200, the running time dramatically increases. In contrast, on the 24-node cluster, the running time marginally changes with the growing number of dimensions. These experimental results suggest that it is prudent to speed up the mutual-information computation of high-dimensional data on a large Spark cluster.



(a) run time



(b) speed up

Fig. 12. The scalability of the MiCS-P.

7.8. Scalability

The scalability of MiCS-P is evaluated by augmenting computing nodes into the Spark cluster (i.e., the number of node is set to 4, 8, 16, and 24, respectively). Then, the size of datasets are 8GB, 16GB, 24GB, and 32GB. Fig. 12(a) illustrates the effects of the number of nodes on the execution time.

Fig. 12(a) unveils that with the increase of the number of computing nodes, the runtime of MiCS-P decreases significantly. Large datasets (e.g., 32GB) have a significant downward trend. Small datasets (e.g., 8GB) have a marginal performance improvement when extending the cluster. These experimental results confirm that parallel mutual-information computing in MiCS-P is highly scalable for large datasets.

Fig. 12(b) demonstrates the influences of number of computing nodes on the MiCS-P's speedup. The speedup of MiCS-P is nearly linear for most datasets. These experimental results demonstrate that our parallel mutual-information computation is capable of maintaining high performance for big categorical datasets. Two factors contribute to the high scalability of MiCS-P. First, the expensive calculation of mutual information between feature pairs is reduced by the column-wise transformation. Second, MiCS-P maintains good load-balancing performance among all the nodes thanks to virtual partitioning.

8. Conclusions and future work

In this paper, we investigated MiCS-P - Spark-based approach to parallel computing mutual-information in the context of big categorical data. During the mutual-information computing, the original data set is divided into subsets through the column-wise transformation, which dramatically slashes the cost of mutual information computing. And, virtual partitioning is utilized to resolve the data skewness problem occurred in the shuffle process is alleviated.

For future work, Shuffle Memory optimization will be our future research direction. When the data distribution is uneven, the

amount of data allocated to different partitions is different, and the memory space required for calculation is also different. If each task can only apply for a fixed proportion of memory space at most, it will lead to frequent overflow of tasks with large memory requirements and cannot be completed for a long time, affecting the overall efficiency of the program. There are no clear instructions on how much memory the Shuffle process should allocate. Efficient use of memory is so important.

And, we argue that the general idea of MiCS-P proposed in this study can be extended to other variants of mutual information computing. The MiCS-P can be used to solve the problem of large amount and strong repeatability of computations related to mutual information.

CRedit authorship contribution statement

- Contributions of Junli Li include designing a column-wise transformation scheme, a twophase virtual partitioning approach in MiCS-P, and experimental validation.
- Contributions of Chaowei Zhang includes improving the solution of MiCS-P approach and some experimental validation.
- Contribution of Jifu Zhang includes proposing a solution of MiCS-P approach.
- Contribution of Xiao Qin includes improving the solution of MiCS-P approach.
- Contribution of Lihua Hu includes experimental validation in MiCS-P.

Declaration of competing interest

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

Acknowledgments

This work is partially supported by the National Natural Science Foundation of P.R. China (62172293), Science and Technology Innovation Project of College and University in Shanxi Province (2021L491), and Xiao Qin's work is supported by the U.S. National Science Foundation under Grants IIS-1618669, OAC-1642133, CCF-0845257 (CAREER).

References

- [1] A. Adinetz, J. Kraus, M. Axer, M. Huysegoms, S. Köhnen, D. Pleiter, Computation of mutual information metric for image registration on multiple gpus, *IEEE Aerosp. Conf. Proc.* (2013).
- [2] W.H. Au, K.C. Chan, A.K. Wong, Y. Wang, Attribute clustering for grouping, selection, and classification of gene expression data, *IEEE/ACM Trans. Comput. Biol. Bioinform.* 2 (2) (2005) 83–101.
- [3] A.P.F. Chan, W.W.Y. Ng, D.S. Yeung, E.C.C. Tsang, Multiple classifier system with feature grouping for intrusion detection: mutual information approach, in: *International Conference on Knowledge-Based Intelligent Information and Engineering Systems*, 2005, pp. 141–148.
- [4] A.P.F. Chan, W.W.Y. Ng, D.S. Yeung, E.C.C. Tsang, M. Firth, Bankruptcy prediction using multiple classifier system with mutual information feature grouping, in: *2006 IEEE International Conference on Systems, Man and Cybernetics*, Vol. 1, 2006, pp. 845–850.
- [5] F. Coelho, A.P. Braga, M. Verleysen, A mutual information estimator for continuous and discrete variables applied to feature selection and classification problems, *Int. J. Comput. Intell. Syst.* 9 (4) (2016) 726–733.
- [6] A. Dasgupta, P. Drineas, B. Harb, V. Josifovski, M.W. Mahoney, Feature selection methods for text classification, in: *Acm Sigkdd International Conference on Knowledge Discovery and Data Mining*, 2007, pp. 230–239.
- [7] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, *ACM*, 2008.
- [8] M. Dettling, P. Bühlmann, Finding predictive gene groups from microarray data, *J. Multivar. Anal.* 90 (1) (2004) 106–131.
- [9] S. Fienberg, B. Junker, Categorical data analysis, *Technometrics* 45 (1) (1998) 109.
- [10] S. Gupta, N. Dutt, R. Gupta, A. Nicolau, Spark: a high-level synthesis framework for applying parallelizing compiler transformations, in: *International Conference on VLSI Design*, 2003.
- [11] H. Karau, A. Konwinski, P. Wendell, M. Zaharia, *Learning Spark: Lightning-Fast Big Data Analytics*, O'Reilly Media, Incorporated, 2015.
- [12] Q. Hu, L. Zhang, D. Zhang, W. Pan, S. An, W. Pedrycz, Measuring relevance between discrete and continuous features based on neighborhood mutual information, *Expert Syst. Appl. Int. J.* 38 (9) (2011) 10,737–10,750.
- [13] M. Iliard, M. Budiu, Y. Yuan, Dryad: distributed data-parallel programs from sequential building blocks, *ACM SIGOPS Oper. Syst. Rev.* 41 (3) (2007) 59–72.
- [14] C. Krier, D. François, F. Rossi, M. Verleysen, Feature clustering and mutual information for the selection of variables in spectral data, in: *European Symposium on Esann*, 2009, pp. 157–162.
- [15] K.C. Lee, A technique of dynamic feature selection using the feature group mutual information, in: *Proceedings of the Third Pacific-Asia Conference on Methodologies for Knowledge Discovery and Data Mining*, 1999, pp. 138–142.
- [16] J. Li, J. Zhang, N. Pang, X. Qin, Weighted outlier detection of high-dimensional categorical data using feature grouping, *IEEE Trans. Syst. Man Cybern. Syst.* 50 (11) (2020) 4295–4308.
- [17] L. Lin, A.S. Hedayat, W. Wu, *Categorical Data*, Springer, New York, 2012.
- [18] G. Liu, X. Zhu, W. Ji, D. Guo, W. Bao, G. Hui, Sp-partitioner: a novel partition method to handle intermediate data skew in spark streaming, *Future Gener. Comput. Syst.* 86 (2018) 1054–1063.
- [19] David J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.
- [20] Rakesh Malladi, Don H. Johnson, Giridhar P. Kalamangalam, Nitin Tandon, Mutual information in frequency and its application to measure cross-frequency coupling in epilepsy, *IEEE Trans. Signal Process.* 66 (11) (2018) 3008–3023.
- [21] J. Maltar, I. Markovi, I. Petrovi, Visual place recognition using directed acyclic graph association measures and mutual information-based feature selection, *Robot. Auton. Syst.* 132 (2020) 103,598.
- [22] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, Mlib: machine learning in apache spark, *J. Mach. Learn. Res.* 17 (1) (2015) 1235–1241.
- [23] A. Miura, A. Tomoeda, K. Nishinari, Formulation of entropy and anticipation of metastable states using mutual information in one-dimensional traffic flow, *Phys. A, Stat. Mech. Appl.* 125 (2020) 152.
- [24] M. Phinney, S. Lander, M. Spencer, C.R. Shyu, Cartesian operations on distributed datasets using virtual partitioning, in: *IEEE Second International Conference on Big Data Computing Service and Applications*, 2016, pp. 1–9.
- [25] S. Ramtin, S. Parastoo, K. Rodney, H. Richard, Parallel computation of mutual information on the gpu with application to real-time registration of 3d medical images, *Comput. Methods Programs Biomed.* 99 (2) (2010) 133–146.
- [26] S. Saxena, S. Sharma, N. Sharma, Parallel computation of mutual information in multicore environment and its applications in medical image registration, in: *International Conference on Medical Imaging*, 2014.
- [27] C.E. Shannon, A mathematical theory of communication, *Bell Syst. Tech. J.* 27 (4) (1948) 623–656.
- [28] B. Silva, N. Marques, Feature clustering with self-organizing maps and an application to financial time-series for portfolio selection, in: *International Conference on Fuzzy Computation and International Conference on Neural Computation*, 2010, pp. 301–309.
- [29] J. Song, Z. Zhu, C. Price, Feature Grouping for Intrusion Detection System Based on Hierarchical Clustering, *Springer International Publishing*, 2014.
- [30] Q. Song, J. Ni, G. Wang, A fast clustering-based feature subset selection algorithm for high-dimensional data, *IEEE Trans. Knowl. Data Eng.* 25 (1) (2013) 1–14, <https://doi.org/10.1109/TKDE.2011.181>.
- [31] Z. Tang, W. Lv, K. Li, K. Li, An intermediate data partition algorithm for skew mitigation in spark computing environment, *IEEE Trans. Cloud Comput.* 9 (2) (2018) 461–474.
- [32] D. Todorov, R. Setchi, Time-efficient estimation of conditional mutual information for variable selection in classification, *Comput. Stat. Data Anal.* 72 (3) (2014) 105–127.
- [33] L. Wang, F. Chu, W. Xie, Accurate cancer classification using expressions of very few genes, *IEEE/ACM Trans. Comput. Biol. Bioinform.* 4 (1) (2007) 40–53.
- [34] T. White, *Hadoop: The Definitive Guide*, 2011.
- [35] Y. Xun, J. Zhang, X. Qin, Fidoop: parallel mining of frequent itemsets using mapreduce, *IEEE Trans. Syst. Man Cybern. Syst.* 46 (3) (2017) 313–325.
- [36] C. Yang, S. Xie, X. Yuan, X. Wang, Y. Xie, A new data reconciliation strategy based on mutual information for industrial processes, *Ind. Eng. Chem. Res.* 57 (38) (2018) 12861–12870.
- [37] Y. Yang, Elements of information theory, *Publ. Am. Stat. Assoc.* 103 (481) (1992) 429.
- [38] D. Yu, A. Shuang, Q. Hu, Fuzzy mutual information based min-redundancy and max-relevance heterogeneous feature selection, *Int. J. Comput. Intell. Syst.* 4 (4) (2011) 619–633.
- [39] J. Yu, H. Chen, F. Hu, Sasm: improving spark performance with adaptive skew mitigation, in: *IEEE International Conference on Progress in Information and Computing*, 2015, pp. 102–107.
- [40] M.J. Zaki, Parallel and distributed association mining: a survey, *IEEE Concurr.* 7 (4) (1999) 14–25.

- [41] J. Zhang, X. Yu, Y. Xun, S. Zhang, X. Qin, Scalable mining of contextual outliers using relevant subspace, *IEEE Trans. Syst. Man Cybern. Syst.* 50 (3) (2020) 988–1002.



Junli Li received the M.S. degree in computer science and technology from the Taiyuan University of Science and Technology, Taiyuan, China, in 2011, and the Ph.D. degree in computer science and Technology from the Taiyuan University of Science and Technology in 2020. Her current research interests include data mining and parallel computing.



Chaowei Zhang received the BS degree in software engineering from the North University of China, China, in 2014, and the Ph.D. degree in computer science from Auburn University, AL, USA, in 2021. He is currently an assistant Professor with the Department of Computer, Yangzhou University, China. His research interests include natural language processing, data mining, and parallel computing.



Jifu Zhang (M'10) received the B.S. and M.S. degrees in computer science and technology from the Hefei University of Technology, Hefei, China, in 1983 and 1989, respectively, and the Ph.D. degree in pattern recognition and intelligence systems from the Beijing Institute of Technology, Beijing, China, in 2005. He is currently a Professor with the School of Computer Science and Technology, Taiyuan University of Science and Technology, Taiyuan, China. His current research interests include data mining, parallel computing, and celestial spectrum data analysis.



Xiao Qin (SM'08) received the B.S. and M.S. degrees in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 1992 and 1999, respectively, and the Ph.D. degree in computer science from the University of Nebraska–Lincoln, Lincoln, NE, USA, in 2004. He is currently a Professor with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL, USA. His current research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation. Dr. Qin was a recipient of the U.S. NSF Computing Processes and Artifacts Award and the NSF Computer System Research Award in 2007 and the NSF CAREER Award in 2009.



Lihua Hu received the PH.D. degree in computer science and technology from the Taiyuan University of Science and Technology, Taiyuan, China, in 2015. Her current research interests include data mining and Artificial Intelligence.