Journal of Parallel and Distributed Computing 138 (2020) 1–14

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

# DQPFS: Distributed quadratic programming based feature selection for big data

Majid Soheili, Amir Masoud Eftekhari-Moghadam [*]

*Faculty of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran*

A R T I C L E  I N F O

*Article history:*
Received 9 April 2019
Received in revised form 23 June 2019
Accepted 1 December 2019
Available online 20 December 2019

*Keywords:*
Big data
Apache Spark
Feature selection
Feature ranking
Quadratic programming

A B S T R A C T

With the advent of the Big data, the scalability of the machine learning algorithms has become more crucial than ever before. Furthermore, Feature selection as an essential preprocessing technique can improve the performance of the learning algorithms in confront with large-scale dataset by removing the irrelevant and redundant features. Owing to the lack of scalability, most of the classical feature selection algorithms are not so proper to deal with the voluminous data in the Big Data era. QPFS is a traditional feature weighting algorithm that has been used in lots of feature selection applications. By inspiring the classical QPFS, in this paper, a scalable algorithm called DQPFS is proposed based on the novel Apache Spark cluster computing model. The experimental study is performed on three big datasets that have a large number of instances and features at the same time. Then some assessment criteria such as accuracy, execution time, speed-up and scale-out are figured. Moreover, to study more deeply, the results of the proposed algorithm are compared with the classical version QPFS and the DiRelief, a distributed feature selection algorithm proposed recently. The empirical results illustrate that proposed method has (a) better scale-out than DiRelief, (b) significantly lower execution time than DiRelief, (c) lower execution time than QPFS, (d) better accuracy of the Naïve Bayes classifier in two of three datasets than DiRelief.

## 1. Introduction

In recent years, the data generation rate has been increased continuously; therefore, nowadays there is a vast amount of data in the world so that there had been no imagination about it before. These data which have a large volume, high complexity, and fast generation rate are widely known by the term Big Data [24]. These data include lots of knowledge and insight that they are so valuable and crucial for most of the prestigious companies [41], but the classical data analytics methods are not suitable for extracting this knowledge because they need to load all data in a single machine that is a major limitation for Big Data analysis [39].

Feature selection (FS) is one of the most crucial preprocessing techniques that it is utilized in a wide range of applications such as machine learning, pattern recognition, data mining and so on [31,32,52,54], and its primary objective is to identify the most informative features from a given dataset. During the feature selection process, irrelevant and redundant features are removed that it can lead to generating a less complicated and more accurate model [18,19]. When the given dataset is massive, FS is more crucial because it reduces feature space and then the computational cost.

According to the output, Feature selection algorithms are categorized into two categories, feature subset selection (FSS) and feature ranking (FR). The output of the FSS is a subset of features that contains discriminating power as same as original features set [56], and the output of the FR is a ranking of features such that features placed on the top of the ranking are more important than the rest of features [51]. It is worth mentioning that this subcategory of feature selection is especially relevant for large-scale context [44].

Quadratic programming feature selection (QPFS) [47] is a feature ranking algorithm that it uses the information theory as the similarity measure, and also it applies an optimization solution to estimate the quality of a given dataset's features. The QPFS assigns a weight to each feature such that the more critical features will have more significant values. As a final result, the features are sorted based on their weights decreasingly and then by applying a threshold value, the top features will be selected as the final selected features.

Due to applying an optimization method, this algorithm has the pretty proper feature ranking as the final output however it has a time execution complexity $\mathcal{O}\left(nd^2\right) + \mathcal{O}(d^3)$, where $d$ is the number of features and $n$ is the number of instances. The first and second components of the execution complexity are related

* Corresponding author.
*E-mail addresses:* majid.soheily@qiau.ac.ir (M. Soheili), eftekhari@qiau.ac.ir (A.M. Eftekhari-Moghadam).

https://doi.org/10.1016/j.jpdc.2019.12.001
0743-7315/© 2019 Elsevier Inc. All rights reserved.

to make a similarity matrix and rank the features, respectively. In confront a big dataset, making similarity matrix is a computational bottleneck, and also it is a significant weakness to cope with large scale datasets. Therefore, a scalable adaptation of the QPFS algorithm is necessary to apply it to large scale dataset.

Apache Hadoop [10] is the most popular open source framework to cope with Big Data that it provides a distributed processing service over a cluster of computers by applying a simple programming model called MapReduce (MR) [9]. The first version of MR implemented in Hadoop does some disk operations (read/write) between each task for running any algorithms. Disk operation is a bottleneck for performing interactive and iterative algorithms that are so common in machine learning and data mining algorithms. In order to amend these weaknesses of the Hadoop MR, Apache Spark [57] is presented which utilized lots of memory-based primitive operations. Thus it has taken lots of attention from the researchers [1,2,5,21,23,33].

MLlib is a well-known machine learning library that includes some standard algorithms based on Apache Spark's operation, but it is not so rich in data preprocessing algorithm. During recent years, preprocessing algorithms on Big Data scope has taken more attention from researchers and some papers about feature selection and attribute reduction based on Apache Spark were published [34,44,45].

In this paper, a distributed and scalable redesign of traditional QPFS algorithm called DQPFS based on Apache Spark computing model is presented such that it can deal with Big Data which has a large number of instances and features simultaneously. Furthermore, some empirical studies on three big datasets and over a real cluster of computers are done. Then, the proposed method is compared with the traditional QPFS and DiRelief algorithms based on some assessment criteria.

The remainder of this paper is organized as follows. The state-of-the-art of the feature selection technique is introduced in Section 2. In Section 3, the background knowledge about the MR paradigm, Apache Hadoop, and Apache Spark are explained. Section 4 describes the traditional QPFS, and in Section 5, the proposed method called DQPFS is explained. In section Section 6, the experimental study will be described, and finally, concluding remarks and interesting issues for future research will be given in Section 7.

## 2. Related work

The feature selection technique is a crucial component in learning algorithms applied in diverse applications [32,36,52–54], and this technique will deal with more challenges such as scalability in big data environment [30]. According to the algorithm structure, all feature selection algorithms can be summarized into three groups: Wrapper [25,27], Embedded [48], and Filter [18]. In the first and second groups, there is a dependency on a specific classifier or learning model, although the third one is based on data related measures such as separability or crowding. Therefore the third group's results have better generalization, and its algorithms are more efficient, which are advantageous in Big Data environment [44]. These advantageous can describe why most of FS methods in the Big Data scope belong to the filter category [15]. In another point of view, feature ranking is a subfamily of feature selection that is highly regarded for large scale context.

In last decades, some papers [3,29,46,59] about parallel or distributed feature selection technique are published, but most of them did not have empirically study over a real cluster of computers to cope with a given large-scale dataset. In this section, just papers are considered, that they were implemented based on the MR programming model and were experimented to handle some large-scale datasets. To this aim, these papers are categorized into two sections. First, papers that their methods are presented based on the MR programming model implemented in the first version of Hadoop that henceforth is known as Hadoop MR. Second, papers that their methods are implemented based on Apache Spark computing model. At the end of this section, the characteristics of these papers are summarized in Table 1.

### *2.1. Feature selection based on Hadoop MR*

Farahat et al. published a paper in about distributed column subset selection based on MR [12]. Their method divides a dataset to some sub-matrices randomly and then it performs a column subset selection algorithm at each machine independently. Finally, in a greedy approach, the algorithm tries to collect the best subset columns among all machines. The proposed method is developed upon the first version of Hadoop MR, and its main objective is minimizing the reconstruction error of the concise representation.

In 2014, Qian et al. proposed a parallel attributed reduction algorithm using MR [39]. In this paper, four algorithms based on the rough set theory proposed in their previous paper [38] are redesigned and developed based on MR paradigm and then the parallelization measures such as scale-out, speed-up, and size-up are studied empirically. Also, in the new version of this work published in 2015 [40], they proposed six hierarchical attribute reduction algorithms based on rough set theory and granular computing. As same as the previous paper, all algorithms built upon Hadoop MR, and the parallelization measures of the algorithms are studied.

### *2.2. Feature selection based on Apache Spark computing model*

Peralta et al. in 2015 [8] proposed an evolutionary feature selection algorithm for Big Data classification. The proposed method splits the given dataset to some chunks such that they are processed in worker nodes independently. In the map phase over each chunk, an evolutionary algorithm is executed and a binary vector is produced as a result. Then in the reduce phase, the binary vectors are averaged to generate a weight vector for the features. All steps of the proposed method build upon Apache Spark computing model.

Eiras et al. in 2016 published a paper [11] in which they performed a comparative study by two approaches, Multi-thread by Weka and Apache Spark parallelization by Apache Spark, for the four popular feature selection algorithms, Information Gain, ReliefF, CFS, and SVM-RFE. The authors studied the execution time and speedup of the algorithms on some real Big Datasets and over a real cluster of computers. The experimental result of the paper showed a significant improvement in execution time for the algorithms implemented by Apache Spark.

In 2016, Ramirez et al. [43] an extended version of the popular mRMR feature selection called fast-mRMR introduced and three implementations of this algorithm proposed in their paper. These implementations are as follows: a sequential version, a parallel version based on GPU's thread-parallelism, and a distributed version based on Apache Spark to handle large dataset. In the paper, the speedup criterion of the proposed algorithm is studied, and the experimental results of the Apache Spark version on three large datasets illustrate that it achieves good speedup values.

Dagida et al. published two papers in 2017 [6] and 2018 [7] about feature selection technique based on roughest theory and Apache Spark computing model. In the first paper, a distributed version of classical rough set theory based feature selection algorithm called dRST was proposed, and it applies a random partitioning of the features space. In the second paper, their

**Table 1**
The characteristics of papers that their methods are implemented based on the MR programming model.

| # | Paper | Year | Method | Output | | Category | Computational model | | Evaluation in classification | Parallel evaluation |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Set | Rank | | Hadoop MR | Apache Spark | | |
| 1 | [12] | 2013 | DistGreedyCSS | ✓ | | Filter | ✓ | | | Execution time |
| 2 | [39] | 2014 | PAAR-DM<br>PAAR-IE<br>PAAR-PR<br>PAAR-BR | ✓ | | Filter | ✓ | | | Speed-up<br>Scale-out<br>Size-up |
| 3 | [40] | 2015 | HARAPOS<br>HARABND<br>HARADIS<br>HARAInfo | ✓ | | Filter | ✓ | | | Speed-up<br>Scale-out<br>Size-up |
| 4 | [8] | 2015 | MR-EFS | | ✓ | Wrapper | | ✓ | True Positive, True Negative, AUC | |
| 5 | [11] | 2016 | Information Gain<br>ReliefF, CFS<br>SVM-RFE | | ✓ | Wrapper<br>Filter | | ✓ | | Execution time<br>Speed-up |
| 6 | [43] | 2016 | fast-mRMR | ✓ | | Filter | | ✓ | | Selection Time<br>Speed-up |
| 7 | [6] | 2017 | dRST | ✓ | | Filter | | ✓ | | Execution time<br>Speed-up, Size-up<br>Stability |
| 8 | [34] | 2018 | DiReliefF | | ✓ | Filter | | ✓ | | Execution time |
| 9 | [35] | 2018 | DiCFS-hp,<br>DiCFS-vp | ✓ | | Filter | | ✓ | | Execution time<br>Speedup |
| 10 | [7] | 2018 | LSH-dRST | | | Filter | | ✓ | Accuracy, Recall, Precision, F1 | Speed-up<br>Scale-out<br>Size-up |
| 11 | [45] | 2018 | DmRMR | ✓ | | Filter | | ✓ | AUC | Feature selection time, Classification time |
| 12 | [44] | 2018 | BELIEF | ✓ | ✓ | Filter | | ✓ | Accuracy, F1 | Execution time |

proposed method called LSH-dRST uses Locality Sensitive Hashing (LSH) to match similar features into the same bucket and maps the generated buckets into partitions to enable the splitting of the universe more appropriately. The reported experimental results illustrated that the second one is more scalable and reliable than the first one. It is worth mentioning that the empirical study was done on a single dataset with 1500 instances and 10 000 features which it does not seem to be a real big dataset. Also, there is no reference and comparison to their previous similar works [39,40].

In 2018, Mendoza et al. proposed a distributed version of the classical and famous ReliefF feature selection algorithm [28] based on Apache Spark computing model called DiReliefF [34]. The authors compared the execution time and memory consumption of the proposed method and the non-distributed version of the RelifF algorithm implemented in Weka. The experimental study is performed on four well-known large datasets and over a real cluster of computers. The results show that DiRelief can process large datasets in a scalable way with much better processing time than the non-distributed version.

As the second work of Mendoza et al. [35], based on the traditional feature selection called CFS [20], they proposed two distributed versions of CFS called DiCFS-hp, DiCFS-vp. These methods are using a horizontal and a vertically partitioning scheme respectively. Then they compared the proposed algorithms with a baseline, represented by the classical non-distributed implementation of CFS in WEKA. Finally, their benefits in terms of reduced execution time are compared with those of the CFS version developed by Eiras-Fanco et al. [11], They examine the proposed methods on four big datasets and the experimental result illustrated that the time-efficiency and scalability of their two versions are an improvement on those of the original version of the CFS.

Ramirez et al. in another paper published in 2018, [45] proposed a framework based on Apache Spark computing model for information theory-based feature selection algorithms. In this paper is claimed, the proposed framework can be used in various well-known information-theory based feature selection methods such as mRMR, Conditional Mutual Information Maximization (CMIM), Joint Mutual Information (JMI), and so on. For experimental study, the distributed version of the mRMR (DmRMR) based on the proposed framework is implemented and three measures, AUC, feature selection time, classification time, are evaluated when it dealt with four large datasets. The authors concluded that the distributed framework could process the large dataset with ultra-high dimensional datasets, although our study shows that this framework to deal with large datasets with the pretty medium dimensional does not have proper performance.

Ramirez et al. proposed as the third work about feature selection techniques in Big Data environment, a new method called BELIEF such that it is inspired by the well-known ReilefF algorithm [44]. This method applied a novel redundancy elimination measure called mCR that is used in a Sequential Forward Selection process [14]. The BELIEF is designed thoroughly based on Apache Spark computing model, and the experimental results over the four popular large scale dataset depict that the algorithm has the smooth scalability on a real cluster of computers.

## 3. MR paradigm, Apache Hadoop, and Apache Spark

MR is a programming model proposed by Google in 2004 for the first time while it applied as a part of Google's search engine [9]. MR is introduced to provide a solution for scalable and flexible processing this data over a cluster of machines. MR has two phases, *Map* and *Reduce*. During *Map* phase, mapper threads transform each chunk of data to a list of <key,value> pairs. During the *Reduce* phase, processed pairs are aggregated to generate summary results by reducer threads. A schema of MR flow is presented in Fig. 1.

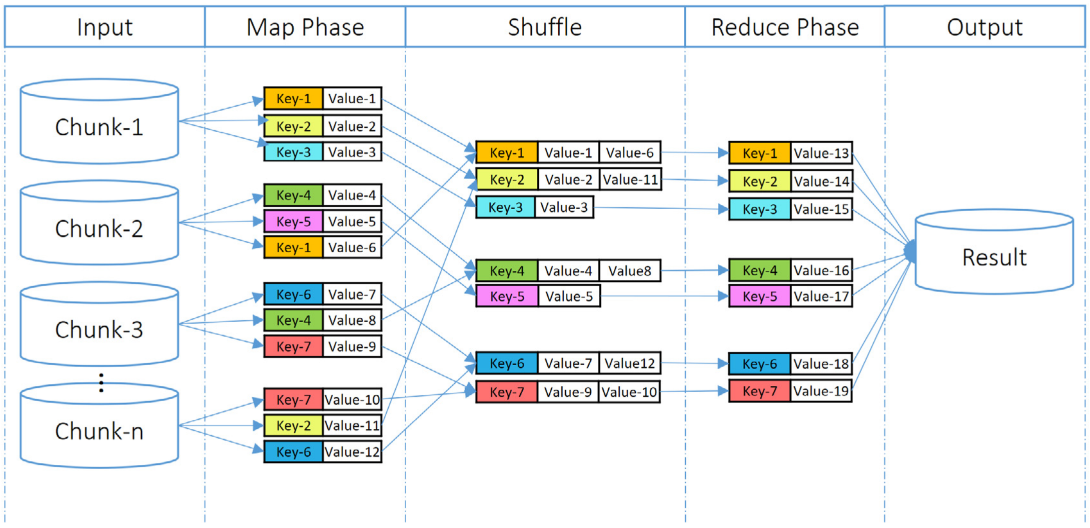

**Fig. 1.** A scheme of MR flow.

Apache Hadoop is the most popular implementation of MR [55] that it facilitates the possibility of deploying a large cluster with commodity hardware which dramatically reduces the cost in comparison with special-purpose parallel machines [41]. Hadoop includes two prominent components: an implementation of the MR and Hadoop Distributed File System (HDFS) [49]. Despite the popularity of the Hadoop, it has some weaknesses such as poorly fitting of the iterative algorithm on MR and the inadequate ability for in-memory processing. For covering these weaknesses, the Apache Spark[1] was introduced in 2012. Apache Spark utilizes a set of memory-based primitives that it causes to perform large-scale processing 100 times faster than Hadoop [57]. Spark is built upon a distributed data structures called Resilient Distributed Dataset (RDD), it is a read-only collection of the object partitioned across the nodes of a cluster [57]. RDD can recover lost data partitions automatically, and it also provides the possibility to extra operations definition over the data without severity adhering to map and reduces functions, however, the MR paradigm still remains as a keystone core in Apache Spark. [1]. There are full explanations of the spark's operators here [26].

Another noticeable aspect of Apache Spark is its cluster architecture and a genuine comprehension of the Apache Spark architecture can lead to understanding the proposed method easier.

Apache Spark applied a master/worker architecture such that the driver program in the master machine drives the user application. Each application includes some job, and each job includes some tasks. The driver program is responsible for assigning the tasks to the executor programs that are run in worker machines. Moreover, once the RDD is created in the master machine it can be distributed in workers and can be cached there as well. Then, the executors execute the task on the partitioned RDD and return the results to the diver program.

## 4. QPFS

In the feature selection algorithms presented in the last decade, two concepts redundancy and relevancy are applied. The redundancy concept refers to the replication of a specific feature or the tight correlation between features, and the relevancy concept refers to the dependency between a feature and target class. Therefore, redundancy and relevancy measures are harmful and useful factors respectively. In the feature selection algorithm, a subset of features is appropriate such that it has the least redundancy between selected features and the most relevancy between selected features and target class at the same time. As a result, in the feature selection technique, Eq. (1) will be maximizing and selected features belong to the set $S$. In Eq. (1), $Rel(F_i)$ refers to relevancy between feature $F_i$ and class label, and also $Red(F_i|S)$ refers to redundancy between feature $F_i$ and the selected features belong to the set $S$.

$$\max_{F_i \in F} \{Rel\,(F_i) - Red(F_i|S)\} \tag{1}$$

Quadratic programming is an optimization method used to minimize a multivariable function with some linear constraints. This method has been utilized in various ranges of problems successfully; also it has been applied in feature selection technique in 2010 [47]. The original form of QP was introduced as the equation below.

$$\min_{w} \left\{ \frac{1}{2} w^T H w - F^T w \right\} \tag{2}$$

Assuming that the given dataset has been defined as $\mathcal{DS} \in R^{n\times d}$, in Eq. (2), $w$ is a $d$-dimensional vector, $H$ is a symmetric positive semidefinite matrix, and $F \in R^d$ is another $d$-dimensional vector with non-negative entries. As mentioned in [47], in order to apply Eq. (2) in feature selection technique and optimizing Eq. (1), the $H \in R^{d\times d}$ matrix represents the similarity or redundancy among features and the $F$ vector represents the correlation or relevancy between each feature and the target class. Once Eq. (2) is solved, the $w$ will be a feature weights vector such that the features with higher values will be more effective features in the learning process. Since $w_i$ represents the weight of $i$th feature, the following linear constraints must be considered in the optimization process.

$$w_i \geq 0 \text{ for all } i = 1 \ldots q \tag{3}$$

$$\sum_{i=1}^{q} w_i = 1 \tag{4}$$

As mentioned before, quadratic (relevancy) and linear (redundancy) terms can be emphasized differently in various object functions [47]. Thus, the $\alpha \in [0, 1]$, a scalar parameter, is added

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

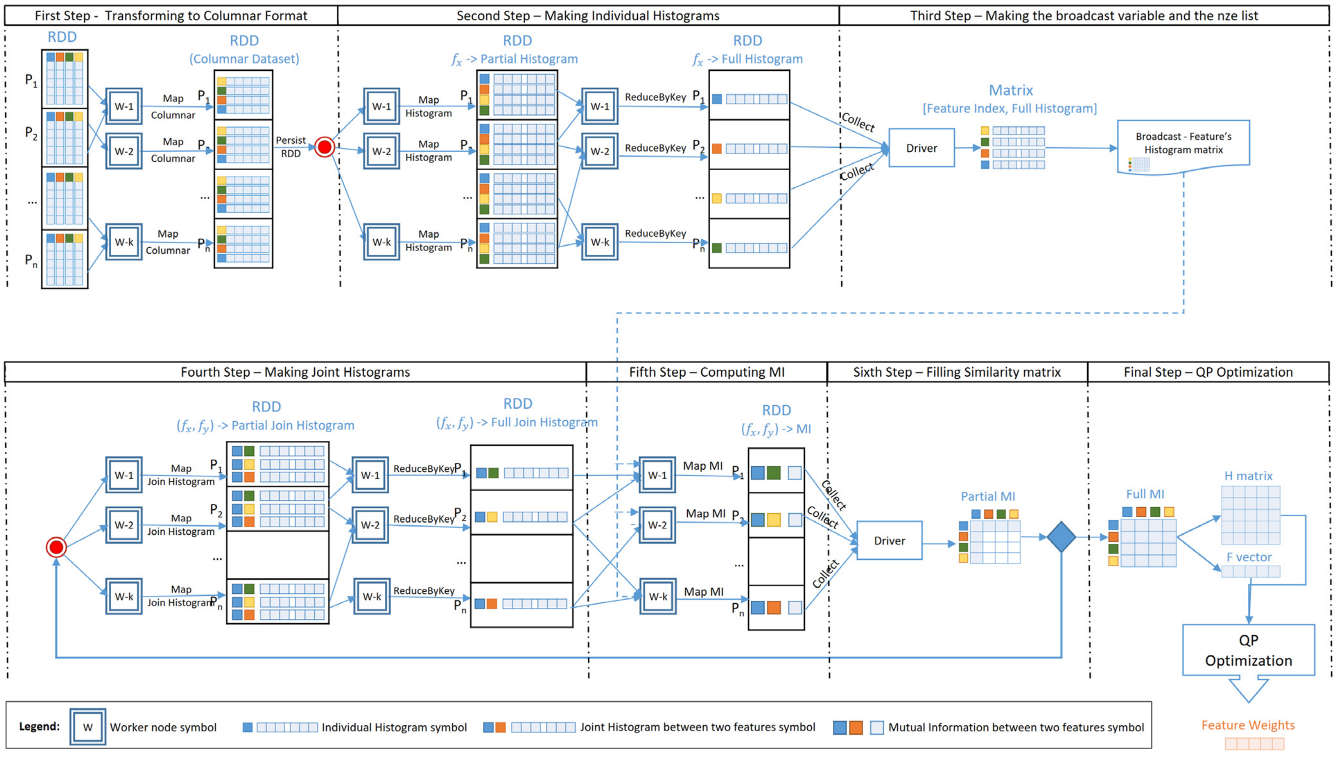
First Step - Transforming to Columnar Format
Second Step – Making Individual Histograms
Third Step – Making the broadcast variable and the nze list
RDD
RDD (Columnar Dataset)
$f_x$ -> Partial Histogram
$f_x$ -> Full Histogram
Map Columnar
Persist RDD
Map Histogram
ReduceByKey
Collect
Driver
Matrix [Feature Index, Full Histogram]
Broadcast - Feature's Histogram matrix
Fourth Step – Making Joint Histograms
Fifth Step – Computing MI
Sixth Step – Filling Similarity matrix
Final Step – QP Optimization
$(f_x, f_y)$ -> Partial Join Histogram
$(f_x, f_y)$ -> Full Join Histogram
Map Join Histogram
$(f_x, f_y)$ -> MI
Map MI
Partial MI
Full MI
H matrix
F vector
QP Optimization
Feature Weights
Legend:
Worker node symbol
Individual Histogram symbol
Joint Histogram between two features symbol
Mutual Information between two features symbol

**Fig. 2.** A scheme of the proposed method.

to Eq. (2), and it is transformed into Eq. (5). If the $\alpha$ parameter is assumed to 1, Eq. (5) will have just linear part thus just the relevancy between features and class label will be considered, on the contrary, if the $\alpha$ parameter is assumed to 0, the quadratic part represented redundancy will be retained so that just the independence among features will be considered. In feature selection technique both relevancy and redundancy have the same importance, so the default value of the parameter $\alpha$ is set by Eq. (6) [47]. In Eq. (6), $\hat{h}$ and $\hat{f}$ are the mean values of the matrix $H$ and vector $F$ respectively.

$$\min_{w}\left\{\frac{1}{2}(1-\alpha)w^{T}Hw-\alpha F^{T}W\right\} \tag{5}$$

$$(1-\alpha)*\hat{h}=\alpha*\hat{f}\rightarrow\alpha=\frac{\hat{h}}{\hat{h}+\hat{f}} \tag{6}$$

In this paper, the similarity measure applied in the QPFS algorithm presented in [47] has been changed to make a better outcome, and also some necessary details are added to the classical version. These changes are listed below.

**Redundancy and Relevancy Measures:** In the original method, the similarity measure was mutual information ($MI$) used to compute the relevancy and redundancy concepts [47]. $MI$ is ranged as $[0,\infty]$ so that it is not a proper quantization for pairwise correlation [58]. When the features of a given dataset have been discretized in variety bins, the weaknesses of the MI are more obvious. Therefore, in this paper, the normal form of mutual information called the symmetric uncertainty ($SU$) [37] is applied as the similarity measure. SU measure compensates the mutual information's bias towards features having a large number of different values and normalizes within the range [0, 1] [50]. SU measure is defined by Eqs. (7) and (8). In Eqs. (8) and (10), $H(X)$ is the entropy of the arbitrary variable $X$, and $H(X|Y)$ is the conditional entropy between two variables $X$ and $Y$.

$$SU(X;Y)=\frac{2*MI(X;Y)}{H(X)+H(Y)} \tag{7}$$

$$MI(X;Y)=H(X)-H(X|Y)=\sum_{x\epsilon X}\sum_{y\epsilon Y}p(x;y)\log_{2}\frac{p(x;y)}{p(x)\,p(y)} \tag{8}$$

$$PMI\,(\text{Pointwise mutual information})=\log_{2}\frac{p(x;y)}{p(x)\,p(y)} \tag{9}$$

$$H(x)=-\sum p(x)\log_{2}(p(x)) \tag{10}$$

**Symmetric positive semidefinite matrix**: As mentioned before in article [47], the matrix $H$ in Eq. (5) must be a symmetric positive semidefinite matrix. A square matrix $H\in R^{n\times n}$ is positive semi-definite (PSD) if $\upsilon^{T}H\upsilon\ \geq\ 0,\forall\upsilon\ \in\ R^{n}$ [22]. There is no guarantee to the $H$ matrix created by $SU$ or $MI$ similarity measure always be a PSD. Therefore, in this paper, before optimizing Eq. (5), the $H$ matrix is tested. If the $H$ matrix is not a PSD matrix, the nearest PSD matrix to the $H$ will be found and utilized instead of the original matrix $H$. It is worth mentioning that, for testing the matrix $H$, the Cholesky decomposition [16,17] method and for finding the nearest PSD matrix, Higham proposed method are applied [22].

**Not zero entropy:** If the entropy of a feature of a given dataset, $\mathcal{DS}\ \in\ R^{n\times d}$, is zero, it should be excluded before creating the redundancy matrix, the matrix $H$ in Eq. (5). Excluding this feature reduces the number of required mutual information computing to generate the matrix H as much as d times. Not excluding this feature causes to generate a row and a column with zero values in the matrix $H$; in this situation, the redundancy matrix is not a PSD. As a result, removing features with zero entropy is necessary.

Algorithm 1- Main procedure of QPFS

```
Input: $\mathcal{DS} \in R^{N \times D}$ - Dataset
Output: $\mathcal{FR}$ – Feature Ranking
1  $\mathcal{D}$ – Features and class label
2  Compute entropy for all member of $\mathcal{D}$
3  Remove features of $\mathcal{D}$ that their entropy is zero to make the not zero entropy ($nze$) list
4  $MI = \sum_{i \in nze} \sum_{j \in nze} MI(i,j)$        // According to equation(6)
5  Weights = QP(MI)                  // According to Algorithm 2
6  $\mathcal{FR}$ = Order weights decreasingly and append zero entropy features to the end of the lis
```

Algorithm 2- QP (Quadratic Programming Optimization based on symmetric uncertainty)

```
Input: $\mathcal{MI} \in R^{n \times n}$ – A Symmetric Matrix of Mutual information among features
Output: $\mathcal{W}$ – Feature Weights
1  $SU = \sum_{i \in nze} \sum_{j \in nze} \frac{2 * MI(i,j)}{H(i)+H(j)}$
2  $H = SU(1 \cdots n-1, 1 \cdots n-1)$
3  $F = SU(1 \cdots n-1, n)$
4  IF $H$ is not a PSD according to Cholesky decomposition
       $\bar{H}$ = nearest PSD of $H$ by using the Higham method
   ELSE
       $\bar{H} = H$
5  $\bar{h} = \frac{1}{n*n} \sum_{i=1}^{n} \sum_{j=1}^{n} \bar{H}_{i,j}$      $\bar{f} = \frac{1}{n} \sum_{i-1}^{n} F_i$      $\bar{\alpha} = \frac{\bar{h}}{\bar{h}+\bar{f}}$
6  $\hat{H} = (1-\bar{\alpha})$      $\hat{F} = \bar{\alpha} * F$
7  Weights = Minimize equation (1) so that the required constrains are set
```

According to the explanation above, the pseudo-code of QPFS algorithm is redesigned and presented in two Algorithm 1 and Algorithm 2. In Algorithm 1 the main procedure of the QPFS algorithm and in Algorithm 2 the optimization procedure are presented. It is worth mentioning that, in the proposed algorithm, there is no change in Algorithm 2 because it will be run just in the driver node as a final step.

## 5. DQPFS

In this section, the proposed algorithm, Distributed Quadratic Programming Feature Selection (DQPFS) based on Apache Spark computing model, is presented. According to Algorithm 1, the highest computation cost operation in the QPFS algorithm is to create the matrix *MI* in line 4 such that its complexity is $\mathcal{O}(nd^2)$ on a given dataset with $d$ features and $n$ instances. Indeed, $\frac{d*(d-1)}{2}$ times of *MI* operations for generating the matrix $H$ and $d$ times of *MI* operation for generating the vector $F$. Also according to Eqs. (8) and (9), the main part of *MI* operation is *PMI*. *PMI*, between two arbitrary features $X$ and $Y$, needs to histogram of their values to compute $P(x)$ and $P(y)$ and needs to the joint histogram of their joint values to compute $P(x; y)$. As a result, DQPFS algorithm concentrates on computing mutual information and pointwise mutual information in a distributed process based on Apache Spark computing model.

The Pseudo-code of the DQPFS is presented in Algorithm 3, and also it is shown graphically in Fig. 2 in seven steps. Notice that in the algorithm, the corresponding steps in Fig. 2 are mentioned as well. In sum up, in the first step, the given dataset is transformed to the columnar format for improving the speed and the performance of the algorithm in next steps, and in third till sixth steps, the matrix *MI* is computed and in the seventh step the weights of the features are determined based on the matrix *MI*. The details of these steps are described as follows

**The First Step — Transforming to columnar format**: In the feature selection technique, a feature is accessed multi-time usually. So in the first step, the data in each partition is transformed into the columnar format, and the new format of the data is cached in the memory of each worker machine to use in next steps. This matter provides access to a specific feature without the need to read all data. This idea was proposed in [45] formerly. Algorithm 4 explains the transforming method to the columnar format.

**The Second Step — Making Individual Histograms:** In the second step, the entropy of each feature is computed. For achieving this aim, in map phase, the value frequency of each partial feature, a row in columnar format RDD, is calculated and this process is repeated for all features. Consequently, for each partial feature, a pair of $\langle key, value \rangle$ is generated such that the key is the index of the feature and the value is a value frequency vector of the feature, that is known as the partial-histogram in Fig. 2. Thus, for each feature, a set of value frequency vectors as much as the number of the data partition will be generated that they need to be aggregated to a vector in reduce phase.

Algorithm 3- Main Procedure of DQPFS

```
Input: $\mathcal{DS} \in R^{N \times D}$ - RDD
Output: $\mathcal{FR}$ – Feature Ranking
1  $\mathcal{D}_c$= Transform $\mathcal{DS}$ to Columnar format and persist it on the memory   ▶ The first step
2  $\mathcal{H}_x$ = Compute the histogram of each feature individually   ▶ The second step
3  $\mathcal{NZ}_e$= Compute the list of not zero entropy features based on $\mathcal{H}_x$   ▶ The third step
4  $bc\mathcal{H}_x$= Broadcast $\mathcal{H}_x$ to all worker to compute $PMI$
5  $\mathcal{MI}$ = new Matrix ( length of $\mathcal{NZ}_e$, length of $\mathcal{NZ}_e$)
6  $\mathcal{BH}$ = Dived $\mathcal{NZ}_e$ features list into few subgroups called batch
7  For all $b$ in $\mathcal{BH}$
8      $\mathcal{H}_{xy}$ = Compute the joint histogram of each two features included in $b$   ▶ The fourth step
9      $\mathcal{PW}$ = Compute pointwise mutual information
10     Compute mutual information for all features included in $b$   ▶ The fifth step
11     Update $\mathcal{MI}$ matrix   ▶ The sixth step
12 Next
13 Weights = QP($\mathcal{MI}$)   ▶ The final step
```

Algorithm 4 - Transform to Columnar RDD

```
Input: $\mathcal{DS}$ – Dataset , an RDD
Input: $\mathcal{NR}$ – Number of row
Input: $\mathcal{NC}$ – Number of columns
Output: $\mathcal{DS}_c$ – Columnar form of $\mathcal{DS}$, an RDD
1.  $\mathcal{DS}_c$ =
2.  map partition $part \in \mathcal{DS}$
3.      $matrix = new\ Matrix(\mathcal{NC}, \mathcal{NR})$
4.      for $rIndex = 0$ until $part.length$
5.          for $cIndex$= 0 until $\mathcal{NC}$
6.              $matrix(cIndex)(rIndex) = part(rIndex)(cIndex)$
7.          next
8.      next
9.  end map
10. return ($\mathcal{DS}_c$.persist)
```

Therefore, in the reduce phase, the partial histograms that have the same feature index are aggregated to a single value frequency vector, that is known as full-histogram in Fig. 2, by a simple sum operation. This step is detailed in the pseudocode presented in Algorithm 5.

**The Third Step — Making the Broadcast Variable**: In this step, the full histograms generated in the previous step are gathered in the driver node, and they are applied for two purposes. Firstly, to filter features with zero entropy and create the *nze* list. Secondly, generate the broadcast variable that is an essential component for computing the joint histogram in the next steps.

**The Fourth Step — Computing MI:** In the fourth step, the joint entropy between every two features in *nze* will be computed. For computing the joint entropy, calculating the joint histogram is crucial. To calculate the joint histogram, for every two partial features in each partition, a pair of $\langle key, value\rangle$ is generated such that the key is a pair of the feature indexes $(f_x, f_y)$ and the value is a joint value frequency of those two partial features, that is known as the partial joint histogram in Fig. 2. Totally, for every two arbitrary features in the *nze* list, few partial joint histograms are generated, as much as the number of the data partition, that they should be aggregated in reduce phase. Therefore, as the same as the third step, the partial joint histograms with the same keys will be summed with each other, and then generate the full joint histogram for each feature. The pseudocode of Algorithm 6 is explaining the joint histogram method.

**The Fifth Step — Filling the Similarity Matrix:** In the fifth step, mutual information between every two features in the *nze* will be computed. The histograms of all features computed in step 3 are broadcasted to all worker nodes, line 4 in Algorithm 3. Consequently, all joint histograms computed in the fourth step can be transformed into their mutual information value independently. To this aim, for each joint histogram, the corresponding individual histograms are found in the broadcasted variable, then pointwise mutual information values are produced, and finally, these values are summed up to generate the mutual information value. The details of the way of transforming a joint histogram RDD to a mutual information RDD are described in Algorithm 7.

**The Sixth Step:** In the sixth step, the mutual information RDD that is computed in the previous step is collected in the driver node and are filled in the similarity matrix that it is the fundamental information to compute feature weights.

It is worth mentioning that, when the number of the features in a given dataset is so large, the number of the joint histogram vectors for all features will be increased exponentially such that it can lead to lack of free memory space and to increase the garbage collection time during the process. Thus, to prevent this issue, the features are divided into some subgroups and in each iteration, the fourth, fifth, and sixth steps are performed over one of these groups.

**The Final Step — QP Optimization:** After Computing the similarity matrix based on the Apache Spark distributed model, the mutual information matrix is sent to Algorithm 2, and the weights of the features are acquired as a result such that more informative features have a more significant value.

## 6. Experimental study

In order to evaluate the performance of the proposed algorithm, the empirical study is performed over a cluster of machines by using different large datasets. To apply information theory, the proposed algorithm needs to discretize the continuous features from the categorical features. Thus a distributed discretizing method based on Apache Spark called Distributed-MDLP is

Algorithm 5 – Individual Histogram

```
Input: $\mathcal{DS}_c$– Dataset in columnar format , an RDD[(Index, vector)]
Input: maxBin – maximum value of a discretized feature
Output: $\mathcal{H}_x$ -  A histogram for each columns individually
1.    $\mathcal{H}_x =$
2.    map partition $part \in \mathcal{DS}_c$
3.      // all data placed in a given partition belongs to a specific column
4.      $frequencies = new\ Array(maxBin)$
5.      for all $vec$ in $part.vector$
6.        for all $v$ in $vec$
7.          $frequencies(v) += 1$
8.        end for
9.      end for
10.   $(part.Index, new\ SparseVector\ (frequencies))$
11.   end map
12.   reducebykey vect1 and vect2
13.     // vect1 and vect2 have the same feature index
14.     Aggregate two vectors, vect1 and vect2 and generate a new sparsevector
15.   end reduce
```

Algorithm 6 - Joint Histogram

```
Input: $\mathcal{DS}_c$– Dataset in columnar format , an RDD[(Index, vector)]
Input: $nze$– not zero entropy features list
Input: maxBin – maximum value of a discretized feature
Output :$\mathcal{H}_{xy}$-  A Joint Histogram between $f_y$ and other features
1.    $\mathcal{H}_{xy} =$
2.    map partition $part \in \mathcal{DS}_c$
3.      for all $(idx_1, vec_1) \in part$
4.        for all $(idx_2, vec_2) \in part$
5.          for all $(v_1, v_2)$ in $vec_1.zip(vec_2)$
6.              $vv = v_1 \times maxBin + v_2$
7.              $frequencies(vv) += 1$
8.          end for
9.          $(idx_1, idx_2) \to new\ sparceVector(frequencies)$
10.       end for
11.     end for
12.   end map
13.   reducebykey $\boldsymbol{vec_1}$and $\boldsymbol{vec_2}$
14.     // $\boldsymbol{vec_1}$and $\boldsymbol{vec_2}$ have the same feature index
15.     Aggregate two vectors, $\boldsymbol{vec_1}$and $\boldsymbol{vec_2}$ and generate a new sparsevector
16.   end reduce
```

applied as the primary step [42]. The Distributed-MDLP library can be found in Ramirez repository,[2] and it is a distributed version of the Minimum Description Length Principle (MDLP) method proposed by Fayyad and Irani [13].

In this section, the results obtained from various executions of the proposed algorithm are presented, and the results are compared with DiRelief and traditional version of QPFS. DiRelief algorithm is the most recent feature ranking algorithm proposed in 2018, and it is implemented based on Apache Spark, and its source code can be found in Mendoza repository.[3] For executing the DiRelief Algorithm, the number of neighbors and the number of samples are set to 20. Also, the traditional version of QPFS is implanted by Scala, and it was executed on one node in the cluster. In the following, the datasets characteristics, the setting of the cluster, and the evaluation measures will be described.

2 https://github.com/sramirez/spark-MDLP-discretization.

3 https://github.com/rauljosepalma/DiReliefF.

Algorithm 7 - Computing Mutual Information based on joint histograms and individual histograms

```
Input: $\mathcal{H}_{xy}$= An RDD[$((idx_1, idx_2), vector)$] – Joint Histogram
Input: $\mathcal{H}_x$= A broadcast variable, Collection[$(index, vector)$] - Histogram
Input: maxBin – maximum value of a discretized feature
Input: nInstances – the number of instances
Output: $mMI$- - An RDD[$((idx_1, idx_2), double)$] - Mutual information
1.  map $(idx_1, idx_2), f_{xy} \in \mathcal{H}_{xy}$
2.      $MI = 0, \quad f_x = \mathcal{H}_x(idx_1), \quad f_y = \mathcal{H}_x(idx_2)$
3.      for each active $(value, frequency) \in f_{xy}$
4.          $v_1 = value/maxBin$
5.          $v_2 = value \% maxBin$
6.          $p_{xy} = freqency/nInstances$
7.          $p_x = f_x(v_1)/nInstances$
8.          $p_y = f_y(v_2)/nInstances$
9.          $MI = MI + p_{xy} \times \log_2(p_{xy}/(p_x \times p_y))$
10.     end for
11.     $(idx_1, idx_2) \rightarrow MI$
12. end map
```

### 6.1. Datasets

To generate the experimental results, the proposed algorithm is executed on three well-known large datasets which have a large number of the instances and a large number of the features at the same time. All of these datasets can be found in the Pascal Large Scale Learning Challenge Repository.[4] The main characteristics of these datasets are summarized in Table 2.

**Table 2**
Summary description of three large datasets that are used in experimental study.

| # | Dataset name | #Instances | #Features | Format | #Class |
|---|---|---|---|---|---|
| 1 | Alpha | 500,000 | 500 | ASCII | 2 |
| 2 | OCR (sampled 50%) | 1,7500,000 | 1156 | Binary | 2 |
| 3 | Epsilon | 500,000 | 2000 | ASCII | 2 |

### 6.2. Cluster configuration

For performing a real evaluation, an 8-node cluster of virtual machines was applied such that one of these nodes is used as a master and the rest are used as slaves. The cluster runs over the hardware and software that their configurations are explained in Table 3.

### 6.3. Assessment criteria and results

For evaluating the performance of the proposed algorithm some criteria such as (a) the execution time, (b) speed-up, (c) scale-out, and (d) the classification accuracy are assessed, and then the results are compared with DiRelief, the last recently published feature ranking algorithm [34]. Moreover, the execution time of the DQPFS in comparison of the classical version QPFS is studied. In the following measures and results are explained.

#### 6.3.1. Execution time

For applying a feature selection algorithm, the execution time is such a vital factor. So, in this section the execution time of the proposed method in comparison to the execution time of DiRelief and the execution time of QPFS is studied. As the line charts in Fig. 3 show, the execution time of the DQPFS in all of three datasets is significantly lower than the DiRelief however this difference is decreasing while the number of cores is increasing. As mentioned before, according to [34], the number of samples as an input parameter is set to 20, however, it is a tiny sampling to confront with a big dataset [44]. As an example, for the OCR dataset, the number of sampled instances of the dataset is 0.0011% that is so insufficient. Therefore, the execution time of the DiReliefF in the experimental study is near to minimum value, and it would be  more significant value in the real application. The execution time of the QPFS algorithm is a horizontal line because it uses the sequential approach, and the number of cores does not influence in processing algorithm. Owing to some overheads such as the RDD lineage graph, the DQPFS has more execution time than QPFS when the cores participating in the processing are too small. Then these overheads can be negligible with increasing the number of executors. Consequently, the execution time of DQPFS is decreased to the lower value of the execution time of QPFS when the number of participated cores is bigger than three.

#### 6.3.2. Speed-up

According to (11), for calculating the speed-up, the size of the dataset is kept constant, but the number of slave nodes which participated in the execution of the algorithm is increased. The linear speedup demonstrates a perfect parallel algorithm. It is worth mentioning that in (11) $T(n.p)$, $n$ refers to the size of dataset and $p$ refers to the number of slave nodes.

$$SpeedUp\ (p) = \frac{T(n.1)}{T(n.p)} \tag{11}$$

As it can be observed, in all three Fig. 4 charts, the speedup lines are near to the linear at the beginning however the distance between these lines is spreading by increasing of the number of participated cores. This behavior is typical in the distributed algorithms. The noticeable matter is that the speed line of DQPFS is going away from the linear chart with a higher speed than DiRelief, and this matter should be analyzed in relation to the communication cost and serial dependency between the steps of the algorithm.

[4] ftp://largescale.ml.tu-berlin.de/largescale/.

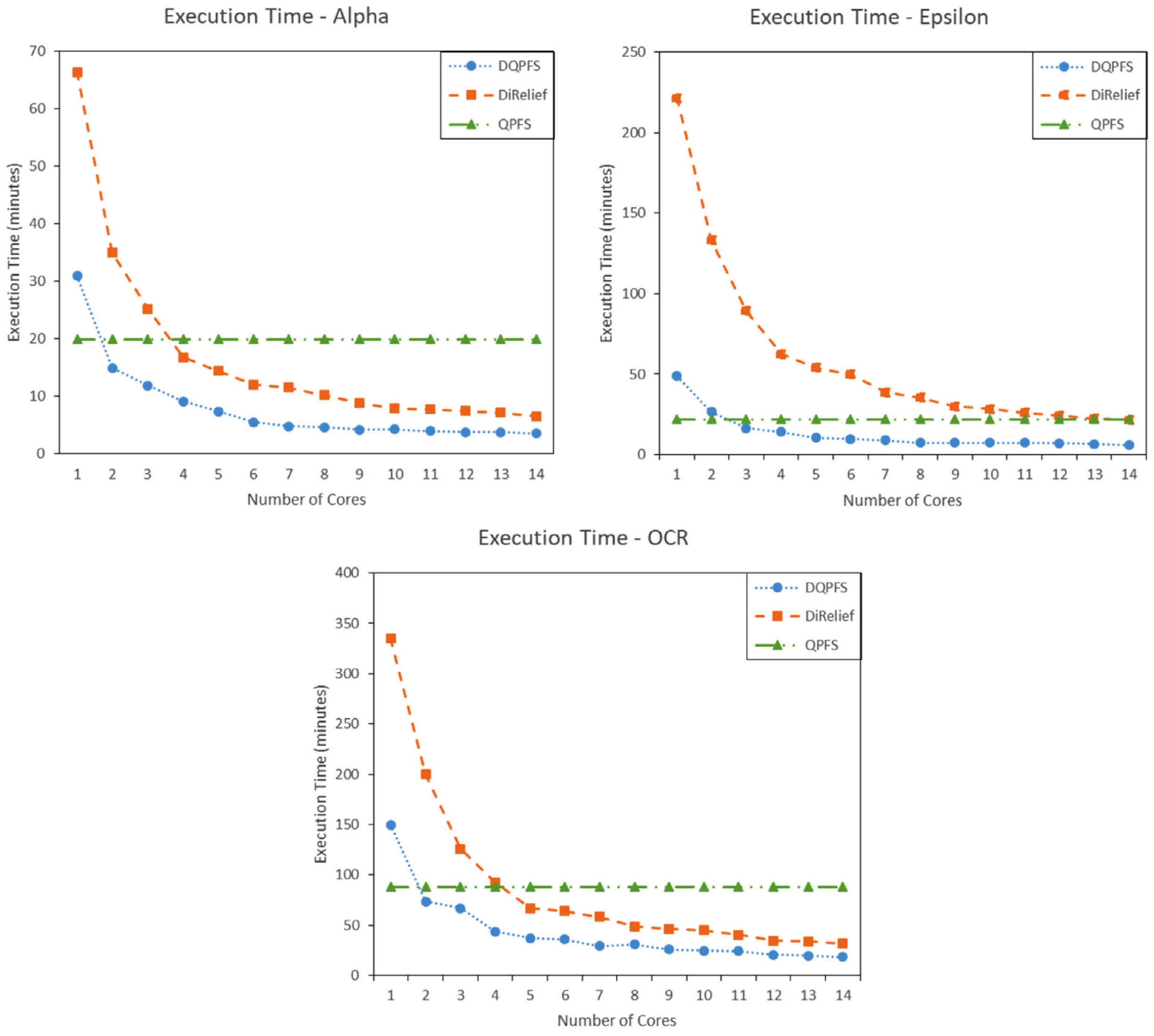

**Fig. 3.** The execution time of the proposed method in comparison DiRelief and classical version of QPFS.

**Table 3**
The configuration of the cluster of machines.

| | | | | | |
|---|---|---|---|---|---|
| Node OS | Ubuntu 14.0 | Node storage | 30 GB | Hadoop version | 2.7.0 |
| Node processors | 2 | Java version | Open JDK 1.8 | Resource manager | Yarn |
| Node memory | 4 GB | Apache spark | 2.3.0 | HDFS replication | 1 |

During computing individual and joint histogram values, the $\langle key, value\rangle$ sets should be dispatched and aggregated among all data partitions, this matter is known as data shuffling based on Apache Spark computing model, and it causes the communication cost across the cluster. For depicting the speed-up curve, on the one hand, the shuffle size would be fixed because the problem size is kept constant. On the other hand, the number of executors will be increased that it causes raising the communication cost.

When a few executors would perform a constant size of data shuffling, it makes a situation in which a large portion of the shuffled data would be transferred in the same nodes that is a fast data transferring. When the number of executors would be increased, the data shuffling would occur among different nodes that it takes more time and raises the communication time. In sum up, by increasing the executors, the general producing time of partial histograms would be decreased, in other words, computational work per executors would be decreased, although the transfer time of the shuffled data among the cluster to generate full histogram would be raised. Thus the communication time would be increased. This situation reduces the granularity [4], time for local computation vs. time for communication, and generally degrades parallel performance. The DQPFS algorithm has larger data shuffled size than DiRelief. Consequently, its communication cost is higher than DiRelief ones. Therefore, by increasing the number of executors, the granularity will be decreased by a higher rate.

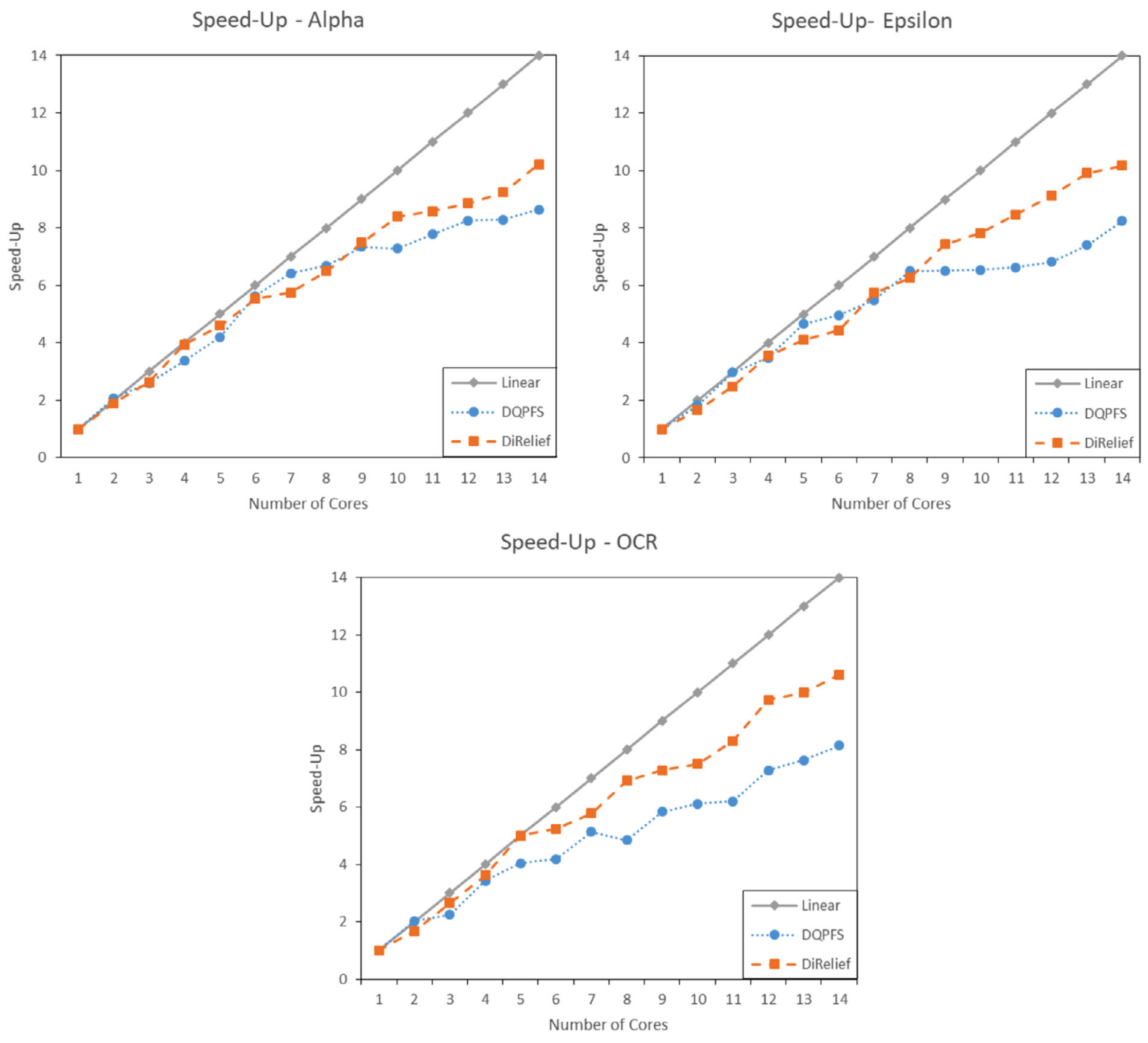

**Fig. 4.** The speedup of DQPFS and DiRelief.

### 6.3.3. *Scale-out*

The Scale-out is defined as the ability of a p-times larger cluster to perform an m-times larger job in the same run-time as the original system [39]. The scale-out criterion shows how the proposed algorithm handles a larger dataset when more slave nodes are available. According to (12), and in order to measure scale-out, the size of the dataset grows in portion to the number of slave nodes that participated in the execution of the proposed algorithm.

$$ScaleOut\ (p) = \frac{T(n.1)}{T(n \times p.p)} \tag{12}$$

The ideal scale-out chart is a horizontal line with 1 value, but this matter is not reachable in a real application. In a real application, the scale-out graph declines with increasing the number of executors. The charts in Fig. 5 illustrate that both algorithm DQPFS and DiRelief have good scale-out lines, but the scale-out of the first graph kept falling more slightly than the second graph. This behavior is repeated in all three datasets.

In opposite of the Speed-Up, in the Scale-Out, the size of the dataset is not constant though computational work per executors is constant. Thus the shuffle size explained in the previous step would not be fixed, and it will be increased by the same rate of the number of the executors participated in the execution, however, the communication time would be approximately constant per executors. Therefore, the granularity only decreases due to increased communication time.

### 6.3.4. *Accuracy*

In Feature ranking Algorithm, to determine the selected features a threshold value is applied usually, such that the features placed on top of the threshold will be known as the selected subset of features. In this paper the various threshold values 5%, 10%, 15%, and 20% are considered. Then in order to assess the performance of the selected feature algorithms, the selected subset of features is passed to the Naive Bayes classification algorithm, and the accuracy of the classifier is accounted as the performance value of the feature selection algorithm. It is worth mentioning that, in this paper, the Naive Bayes classification that is implemented by default in MLIB library in Apache Spark is utilized.

As the bar charts depict in Fig. 6, the DQPFS has better accuracy in all threshold values in two datasets Epsilon and OCR. In the Alpha dataset, however, the DiRelief accuracy is better than

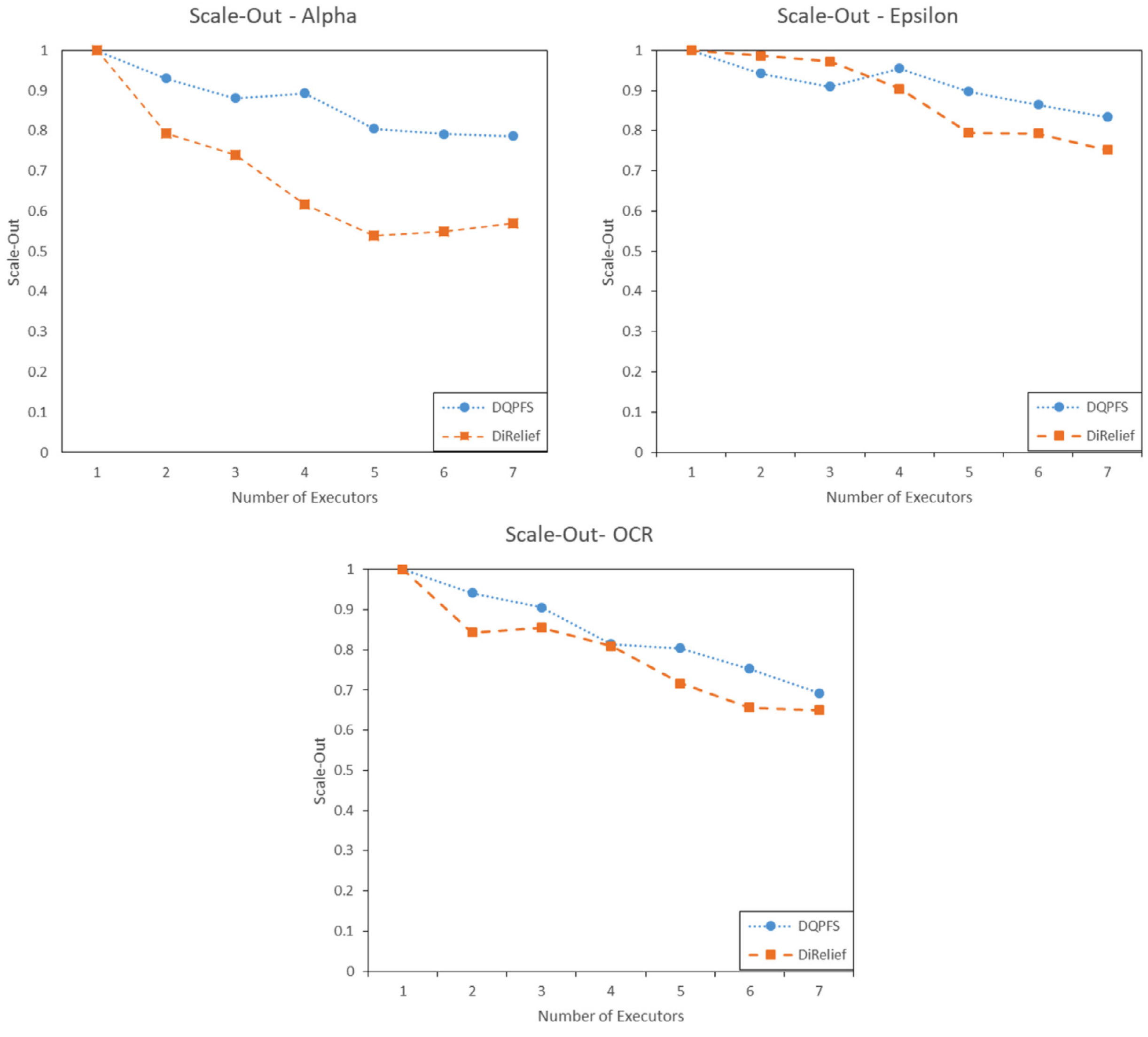

**Fig. 5.** Scale out chart of DQPFS and DiRelief.

the DQPFS, but the difference is not so substantial. The accuracy graphs in Fig. 6 cannot prove an absolute superiority of DQPFS to DiRelief, however, they illustrate the DQPFS deserves much attention for feature ranking in Big Data scope.

## 7. Conclusion

Feature selection is a crucial part in the majority of learning algorithms, and it is more imperative when the dataset is so voluminous like as it is common in the Big Data era. Also, Feature ranking as a subfamily of the feature selection is more relevant when the number of features is substantial.

In this paper, a feature ranking algorithm called DQPFS based on the computing model of Apache Spark had been presented to handle large datasets and execute over a cluster of commodity hardware. Also, in this paper, the large dataset means that a dataset with a large number of instances and a large number of features simultaneously. The proposed method is thoroughly redesigned of the classical version of QPFS algorithm, In QPFS making the redundancy matrix and the relevancy vector are the computational bottlenecks. The proposed method does not suffer from the problem, and it can make the matrix and the vector by some independent task and in different worker nodes.

For experimental study, some criteria such as speedup, scale-out and execution time were figured then the results were compared with DiRelief algorithm that was presented recently and implemented based on the computation model of Apache Spark. The empirical result illustrated that DQPFS has a slightly superior scale-out and a slightly poor speed-up than DiRelief, however, the execution time of DQPFS was significantly lower than the DiRelief.

Also, the execution time of the proposed method was compared with traditional QPFS. The result showed that the execution time of DQPFS was pretty lower than the traditional QPFS when the number of participated cores was bigger than three. Thus in the opposite of QPFS, DQPFS was thoroughly scalable, and it can process the large dataset in a short time. In addition to speed-up, scale-out and execution time, a comparison of the accuracy was performed between the final outputs of DQPFS and DiRelief by applying the Naive Bayes classifier. The results could not prove the accuracy of the classifier was generally better than the DiRelief however they showed that DQPFS could be a reasonable choice to feature selection for the large dataset.

In summary, the experimental study depicts that the proposed method is adequately scalable and it can handle the large dataset and its execution time is significantly lower than both DiRelief and QPFS.

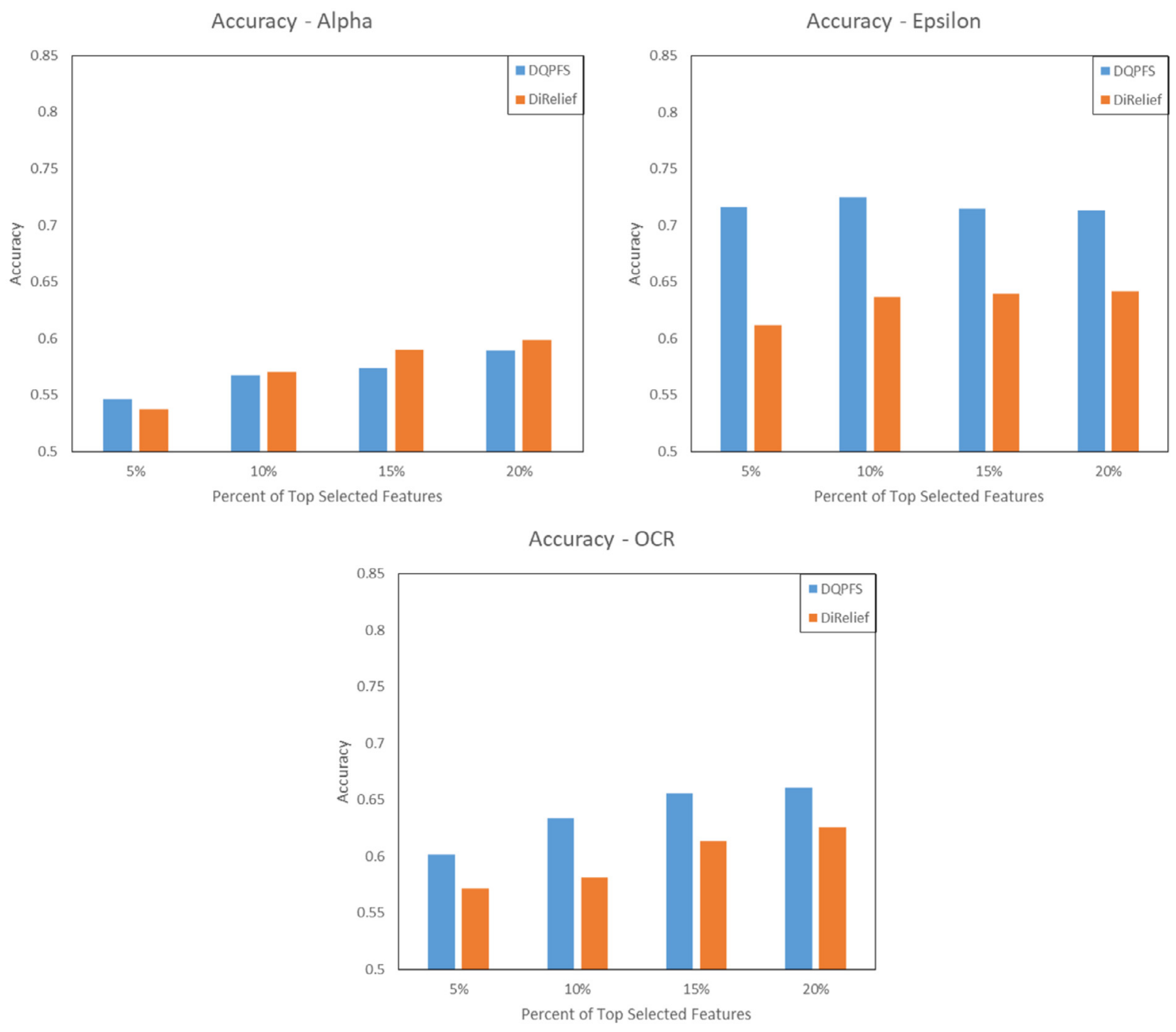

**Fig. 6.** The accuracy of Bayesian classifier when the selected features are determined by DQPFS and DiRelief.

## Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.jpdc.2019.12.001.

## References

[1] J. Arias, J.A. Gamez, J.M. Puerta, Learning distributed discrete Bayesian network classifiers under MapReduce with Apache Spark, Knowl.-Based Syst. 117 (2017) 16–26.
[2] N. Bharill, A. Tiwari, A. Malviya, Fuzzy based scalable clustering algorithms for handling big data using apache spark, IEEE Trans. Big Data 2 (4) (2016) 339–352.
[3] V. Bolón-Canedo, N. Sánchez-Maroño, A. Alonso-Betanzos, Distributed feature selection: An application to microarray data classification, Appl. Soft Comput. 30 (2015) 136–150.
[4] R. Chandra, et al., Parallel Programming in OpenMP, Morgan Kaufmann Publishers Inc., 2001, p. 230.
[5] J. Chen, et al., A parallel random forest algorithm for big data in a spark cloud computing environment, IEEE Trans. Parallel Distrib. Syst. 28 (4) (2017) 919–933.
[6] Z.C. Dagdia, et al., A distributed rough set theory based algorithm for an efficient big data pre-processing under the spark framework, in: 2017 IEEE International Conference on Big Data (Big Data), IEEE, 2017.
[7] Z.C. Dagdia, et al., A distributed rough set theory algorithm based on locality sensitive hashing for an efficient big data pre-processing, in: 2018 IEEE International Conference on Big Data (Big Data), IEEE, 2018.
[8] S.d.R. Daniel Peralta, Sergio Ramírez-Gallego, Isaac Triguero, Jose M. Benitez, Francisco Herrera, Evolutionary feature selection for big data classification: a MapReduce Approach, Math. Probl. Eng. (2015) 11.
[9] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[10] W. Dhifli, S. Aridhi, E.M. Nguifo, MR-Simlab: Scalable subgraph selection with label similarity for big data, Inf. Syst. 69 (2017) 155–163.
[11] C. Eiras-Franco, et al., Multithreaded and Spark parallelization of feature selection filters, J. Comput. Sci. 17 (Part 3) (2016) 609–619.
[12] A.K. Farahat, et al., Distributed column subset selection on mapreduce, in: 2013 IEEE 13th International Conference on Data Mining, 2013.
[13] U.M. Fayyad, K.B. Irani, Multi-interval discretization of continuous-valued attributes for classification learning, 1993, 1022-1029.
[14] S. Garcia, Data Preprocessing in Data Mining, in: [Place of publication not identified], SPRINGER INTERNATIONAL PU, 2016.
[15] S. García, et al., Big data preprocessing: methods and prospects, Big Data Anal. 1 (1) (2016) 9.
[16] J.E. Gentle, Numerical Linear Algebra for Applications in Statistics, Springer, New York, N.Y, 1998.
[17] J.E. Gentle, Matrix algebra : theory, computations and applications in statistics, 2017.
[18] I. Guyon, et al., An introduction to variable and feature selection, J. Mach. Learn. Res. 3 (2003) 1157–1182.
[19] I. Guyon, et al., Feature Extraction: Foundations and Applications, in: Studies in Fuzziness and Soft Computing, Springer-Verlag New York, Inc., 2006.
[20] M.A. Hall, Correlation-based feature selection for discrete and numeric class machine learning, in: Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc, 2000, pp. 359–366.

[21] D. Harnie, et al., Scaling machine learning for target prediction in drug discovery using Apache Spark, Future Gener. Comput. Syst. 67 (2017) 409–417.
[22] N.J. Higham, Computing a nearest symmetric positive semidefinite matrix, Linear Algebra Appl. 103 (1988) 103–118.
[23] B. Hosseini, K. Kiani, A big data driven distributed density based hesitant fuzzy clustering using apache spark with application to gene expression microarray, Eng. Appl. Artif. Intell. 79 (2019) 100–113.
[24] M.G. Institute, et al., Big Data: The Next Frontier for Innovation, Competition, and Productivity, McKinsey Global Institute, 2011.
[25] G.H. John, R. Kohavi, K. Pfleger, Irrelevant features and the subset selection problem, in: Machine Learning: Proceedings of the Eleventh International Conference, 1994.
[26] H. Karau, Learning Spark, in: [Lightning-Fast Big Data Analysis], O'Reilly, Beijing, 2015.
[27] R. Kohavi, G.H. John, Wrappers for feature subset selection, Artificial Intelligence 97 (1) (1997) 273–324.
[28] I. Kononenko, Estimating Attributes: Analysis and Extensions of RELIEF, Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.
[29] J. Kubica, S. Singh, D. Sorokina, Parallel large-scale feature selection, in: J. Langford, M. Bilenko, R. Bekkerman (Eds.), Scaling Up Machine Learning: Parallel and Distributed Approaches, Cambridge University Press, Cambridge, 2011, pp. 352–370.
[30] J. Li, H. Liu, Challenges of feature selection for big data analytics, IEEE Intell. Syst. 32 (2) (2017) 9–15.
[31] S. Maldonado, R. Montoya, R. Weber, Advanced conjoint analysis using feature selection via support vector machines, European J. Oper. Res. 241 (2) (2015) 564–574.
[32] S. Maldonado, R. Weber, F. Famili, Feature selection for high-dimensional class-imbalanced data sets using Support Vector Machines, Inform. Sci. 286 (2014) 228–246.
[33] I. Mavridis, H. Karatza, Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark, J. Syst. Softw. 125 (2017) 133–151.
[34] R.-J. Palma-Mendoza, D. Rodriguez, L. de Marcos, Distributed Relieff-based feature selection in Spark, Knowl. Inf. Syst. 57 (1) (2018) 1–20.
[35] R.-J. Palma-Mendoza, et al., Distributed correlation-based feature selection in spark, Inform. Sci. 496 (2019) 287–299.
[36] R.H.W. Pinheiro, G.D.C. Cavalcanti, T.I. Ren, Data-driven global-ranking local feature selection methods for text categorization, Expert Syst. Appl. 42 (4) (2015) 1941–1949.
[37] W.H. Press, et al., Numerical Recipes in C, second ed., in: The Art of Scientific Computing, Cambridge University Press, 1992, p. 994.
[38] J. Qian, et al., Hybrid approaches to attribute reduction based on indiscernibility and discernibility relation, Internat. J. Approx. Reason. 52 (2) (2011) 212–230.
[39] J. Qian, et al., Parallel attribute reduction algorithms using MapReduce, Inform. Sci. 279 (2014) 671–690.
[40] J. Qian, et al., Hierarchical attribute reduction algorithms for big data using mapreduce, Knowl.-Based Syst. 73 (2015) 18–31.
[41] A. Rajaraman, J. Ullman, Mining of Massive Datasets, Cambridge University Press, 2011.
[42] S. Ramírez-Gallego, et al., Data discretization: taxonomy and big data challenge, Wiley Interdiscipl. Rev.: Data Min. Knowl. Discov. 6 (1) (2016) 5–21.
[43] S. Ramírez-Gallego, et al., Fast-mrmr: Fast minimum redundancy maximum relevance algorithm for high-dimensional big data, Int. J. Intell. Syst. 32 (2) (2017) 134–152.
[44] S. Ramírez-Gallego, et al., BELIEF: A distance-based redundancy-proof feature selection method for big data, 2018, CoRR.
[45] S. Ramírez-Gallego, et al., An information theory-based feature selection framework for big data under apache spark, IEEE Trans. Syst. Man Cybern.: Syst. 48 (9) (2018) 1441–1453.
[46] M.S. Raza, U. Qamar, A parallel rough set based dependency calculation method for efficient feature selection, Appl. Soft Comput. 71 (2018) 1020–1034.
[47] I. Rodriguez-Lujan, et al., Quadratic programming feature selection, J. Mach. Learn. Res.: JMLR. 11 (1) (2011) 1491–1516.
[48] Y. Saeys, et al., A review of feature selection techniques in bioinformatics, Bioinformatics 23 (19) (2007) 2507–2517.
[49] K. Shvachko, et al., The hadoop distributed file system, in: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010.
[50] B. Singh, N. Kushwaha, O.P. Vyas, A feature subset selection technique for high dimensional data using symmetric uncertainty, JDAIP J. Data Anal. Inf. Process. 02 (04) (2014) 95–105.
[51] M. Soheili, A.M.E. Moghadam, Feature selection in multi-label classification through MLQPFS, in: 2016 4th International Conference on Control, Instrumentation, and Automation (ICCIA), 2016.
[52] H.A.L. Thi, X.T. Vo, T.P. Dinh, Feature selection for linear SVMs under uncertain data: Robust optimization based on difference of convex functions algorithms, Neural Netw. 59 (2014) 36–50.
[53] N.S.-M. Verónica Bolón-Canedo, Amparo Alonso-Betanzos, Feature Selection for High-Dimensional Data, in: Artificial Intelligence: Foundations, Theory, and Algorithms, vol. XV, Springer International Publishing, 2015, p. 147.
[54] K.-J. Wang, K.-H. Chen, M.-A. Angelia, An improved artificial immune recognition system with the opposite sign test for feature selection, Knowl.-Based Syst. 71 (2014) 126–145.
[55] T. White, Hadoop : The Definitive Guide, in: [Storage and Analysis at Internet Scale], O'Reilly et Associates, Beijing, 2015.
[56] Y. Yang, J.O. Pedersen, A comparative study on feature selection in text Categorization, in: Proceedings of the Fourteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., 1997, pp. 412–420.
[57] M. Zaharia, et al., Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, USENIX Association, San Jose, CA, 2012, p. 2.
[58] Y. Zhai, Y. Ong, I.W. Tsang, Making trillion correlations feasible in feature grouping and selection, IEEE Trans. Pattern Anal. Mach. Intell. 38 (12) (2016) 2472–2486.
[59] Z. Zhao, et al., Massively parallel feature selection: an approach based on variance preservation, Mach. Learn. 92 (1) (2013) 195–220.

**Majid Soheili** is currently a lecturer at the Islamic Azad University, Neka Branch, Neka, Iran, where he has been working at the Computer Engineering Department since 2011. He is also a Ph.D. candidate at the Computer and Information Technology Engineering Department of the Islamic Azad University, Qazvin Branch, Qazvin, Iran. He earned his M.Sc. degree in Computer-Software Engineering at the Islamic Azad University, Qazvin Branch, Qazvin, Iran as well. His research interests include Datamining, Artificial Intelligence, Big Data.

**Amir-Masoud Eftekhari Moghadam** is currently an associate professor at the Computer and Information Technology Engineering Department of the Islamic Azad University, Qazvin Branch, Qazvin, Iran since 1996. He earned his B.Sc. (1992) in hardware engineering at Iran University of Science and Technology, M.Sc. (1995) in computer architecture at Iran University of Science and Technology, and Ph.D. (2001) in computer architecture at Islamic Azad University, Science and Research Branch, Tehran, Iran. His research interests include Image Processing, Information Retrieval, Machine Learning, Data Mining, Distributed Processing, Big Data.