Two Parallelized Filter Methods for Feature Selection Based on Spark

Reine Marie Ndéla Marone[1(✉)], Fodé Camara[2], Samba Ndiaye[1], and Demba Kande[1]

[1] Department of Mathematics, Cheikh Anta Diop University, Dakar, Senegal
reine.marie.marone@ucad.edu.sn
[2] Department of Mathematics, Alioune Diop University, Bambey, Senegal
fode.camara@uadb.edu.sn

Abstract. The goal of feature selection is to reduce computation time, improve prediction performance, build simpler and more comprehensive models and allow a better understanding of the data in machine learning or data mining problems. But the major problem nowadays is that the size of datasets grows larger and larger, both vertically and horizontally. That constitutes challenges to the feature selection, as there is an increasing need for scalable and yet efficient feature selection methods. As an answer to those problems, we present here two effective parallel algorithms developed on Apache Spark, a unified analytics engine for big data processing. One of them is a parallelized algorithm based on the famous feature selection method called mRMR. In the second algorithm we propose a totally novel metric to select the more relevant and less redundant features. To show the superiority of that algorithm we have created its centralized version that we have called CNFS_Spark.

Experimental results demonstrate that our algorithms achieve a great performance improvement in scaling well and take less time than classical feature selection methods.

Keywords: Feature selection · Parallel computing · Apache spark
mRMR · Novel method · Big data · Large scale · High dimensional

1 Introduction

Feature selection has been successfully applied for years as a preprocessing step and has become an active research field in pattern recognition, statistics, and data mining communities.

In supervised learning, the main goal of feature selection is to choose a subset of input variables that produces higher classification accuracy by eliminating features with little or no predictive information. Feature selection can also significantly reduce the learning time of classification.

Feature selection methods are commonly categorized into filter, wrapper and embedded approaches [1].

R. Zitouni and M. Agueh (Eds.): AFRICATEK 2018, LNICST 260, pp. 175–192, 2019.
https://doi.org/10.1007/978-3-030-05198-3_16

In the wrapper methods a subset of features is ranked by the prediction performance of a classifier on the given subset [1]. The Wrapper methods are computationally expensive and over fit on small training sets.

Embedded methods perform feature selection during the modeling algorithm's execution [1]. These methods are thus embedded in the algorithm either as its normal or extended functionality. Embedded methods select features based on criterions that are generated during the modeling algorithm's execution. The chosen features are sensitive to the structures of the underlying classifiers.

Filter methods rely on in various statistical tests to evaluate and select features without involving any mining algorithm [1, 2]. The subset selection procedure is generally a pre-processing step. In our studies we have focused on the filters methods because they offer better computational complexity for datasets having large number of features.

But generally, classical feature selection algorithms are designed to run on a single machine and are not adapted for big-data problems [3]. In fact, algorithms do not scale well when dealing with very large datasets and their efficiency significantly downgrades.

Distributed computing techniques such as MapReduce [3] along with its open-source implementation Apache Hadoop can be a solution to solve this problem.

But the MapReduce parallel programming with Apache Hadoop is not suited for the feature selection because MapReduce programming model reads and writes from disk [4]. As a result, it slows down the processing speed. Spark is lightning fast cluster computing tool because of reducing the number of read/write cycle to disk and storing intermediate data in memory. That explains our preference for apache spark.

On that basis we have developed two novel, parallel and scalable feature selection algorithms named respectively PSFS-mRMR (for Parallel Spark Feature Selection method based on mRMR) and PNFS_Spark (Parallel Neighbor Feature Selection based on Spark) on the Spark framework.

The results that we obtained show that the parallelized methods outperform the centralized ones, in terms of scalability and efficiency which shows that parallelization is better than centralization.

The remainder of this paper is organized as follows:

Section 2 discusses related works.
Section 3 consists of defining the problem.
Section 4 presents the classical mRMR feature selection method.
Section 5 gives the metrics we used in our proposal.
Section 6 deals with the presentation of our methods.
Section 7 describes the experiments.
Section 8 consists of analyzing the experiment results.
Section 9 concludes the paper and gives some future works.

2 Related Works

Filter methods are kind of feature selection methods that attempt to assess the importance of features statically according to a heuristic scoring criteria without any particular classifier. Many filter methods are based on information theory specifically mutual Information but are centralized and do not scale well when dealing with big data. They must be parallelized to gracefully scale with larger datasets [5]. So, in this section we will discuss parallel approach of filter methods that have been proposed to decrease the training time and produce better accuracy.

In [6] authors implement on Apache Spark, a generic feature selection framework that includes a broad group of well-known information theory-based methods like mRMR. The experimental results demonstrate that this framework handles efficiently large-scale data and outperforms the sequential version.

Authors in [7] propose a package named mRMRe that extends mRMR by using an approach to better explore the feature space and build more robust predictors. In mRMRe, the main functions are implemented and parallelized in C using the openMP Application Programming Interface. mRMRe provides less learning time and can identify genes more relevant to the biological context and may lead to richer biological interpretations.

The work in [8] is a combinaison of ReliefF and mRMR consisting on two-steps: apply ReliefF on the first stage to find a candidate subset of gene and then use mRMR method for reducing redundancy and selecting an effective set of gene from the candidate subset. The experimental evaluation demonstrates the effectiveness of this algorithm.

In [9], a parallel version of mRMR called fast-mRMR is proposed to overcome the computationally expensive of mRMR. Authors provide a package with three implementations of this algorithm in several platforms, namely, CPU for sequential execution, GPU (graphics processing units) for parallel computing, and Apache Spark for distributed computing. The experimental results show that fast-mRMR outperforms the original version of mRMR and show a clear improvement when using the parallel and distributed versions over the sequential one.

In [10], authors reimplement four popular feature selection algorithms including RELIEF-F, InfoGain, CFS and SVM-RFE in Weka. Multithreaded implementations previously not included in Weka as well as parallel Spark implementations were developed for each algorithm. Experimental results obtained from tests on real-world datasets show that the new versions offer significant reductions in processing times.

In [11], authors, present a completely redesigned distributed version of the popular ReliefF algorithm based on the novel Spark cluster computing model called DiReliefF.

But the proposed algorithms in the related work include iteratively one or many features into the feature's subset to return. So the number of onward and backwards movements between the workers and the driver increases according to the number of features to select. This can lead to high communication and synchronization costs between the workers and the driver.

In what we propose, the score of each feature is calculated by the workers and sent once to the driver that selects the best features in one pass. That leads to less learning time while keeping a good accuracy.

3 Problem Definition

Our work concerns the binary classification problems. $c \in \{0, 1\}$ denotes the class label. Let F be the input set of attributes $\{f_1, .., f_n\}$ and I an instance represented by a n-dimensional vector $(v_1, .., v_n)$, where v_j represents the value of the attribute f_j in I and n the number of features which can be large. $J(S', D)$ is the objective function that measuring the quality of a subset S' of S using the data D. If $J\left(S'_1, D\right) > J(S'_2, D)$, it means that the subset S'_1 *is* better than S'_2.

So in this work, we proposed a high-dimensional filter method: PSFS-mRMR for Parallel Spark Feature Selection method based on mRMR (Minimum Redundancy and Maximum Relevancy) and another one called PNFS_Spark (Parallel Neighbor Feature Selection based on Spark) using the unified analytics engine for big data processing Apache Spark to implement them.

4 The Classical MRMR

mRMR (minimum Redundancy and Maximum Relevance) is an algorithm frequently used to rank features based on their relevance to the label class, and, at the same time, the redundancy of features is also penalized. The goal is to find the maximum dependency between a subset F' of features, and the class label l, using mutual information (MI) [12].

Let f_i and f_j be two features in F. $MI\left(f_i, f_j\right)$ is the mutual information between f_i and f_j. $MI(l, f_i)$ stands for the mutual information between the class label l and f_i.

The redundancy between features in F, is determined by

$$Q_I(F) \frac{1}{|F|^2} \sum_{f_i, f_j \in F} MI\left(f_i, f_j\right) \tag{1}$$

The relevance of the features in F with the class label l is given by

$$R_I(F) \frac{1}{|F|} \sum_{f_i \in F} MI(l, f_i) \tag{2}$$

Determine the best subset of features F^* in F, which contains maximally relevant and minimally redundant features consist of optimizing (1) and (2) as follows:

$$F^* = \arg\max_{F' \subseteq F} [R_I(F) - Q_I(F)] \tag{3}$$

5 Ours Proposals

- **PSFS-mRMR**

Many authors demonstrate that [13] SVMs (support vector machines) with simple feature rankings are effective on datasets. That's why, in our proposal PSFS-mRMR, we rank the features using a combination of [14] SVM and mRMR for better results. Let $\beta \in [0, 1]$ determines the tradeoff between SVM ranking and mRMR ranking. The relevancy $R_{F,i}$ of feature f_i in the F set in classification is given by

$$R_{F,i} \frac{1}{|F|} \sum_{l} MI(l, f_i) \tag{4}$$

And $Q_{F,i}$ the redundancy of feature f_i in the set F in classification is computed as follows

$$Q_{F,i} \frac{1}{|F|^2} \sum_{f_i, f_j \in F} MI(f_i, f_j) \tag{5}$$

Let ω_i represents the SVM weight of the attribute f_i.

For i-th feature, the ranking measure d_i is calculated as follows

$$d_i = \beta |\omega_i| + (1 - \beta) \frac{R_{F,i}}{Q_{F,i}} \tag{6}$$

- **PNFS_Spark**

In the PNFS_Spark we propose a novel metric called m_e that represents the median value of the relevance with the class label l. In fact, the median gives a satisfactory idea of the general tendency of a statistical serie (*F set* in our case). The outliers of the variable that might be in the series do not influence it.

Let n be the number of features in F and e a number given by applying the formula below if n is pair:

$$\mathrm{e} = (\mathrm{n} + 1)/2$$

or

$$\mathrm{e} = (\mathrm{n}/2 + \mathrm{n}/2 + 1)/2$$

if n is impair.

m_e is obtained by:

$$m_e = R_{F,e} \tag{7}$$

where $R_{F,e}$ represents the relevance of e-th feature.

If the mutual information between 2 features f_i and f_j is greater than the median then f_i is considered a neighbor of f_j. The score of the feature f_i is given by multiplying its relevance $R_{F,i}$ with its number of neighbors.

6 Our Algorithms

Our proposed algorithms, called respectively PSFS_mRMR and PNFS_Spark are feature selection methods that we base on Spark, an open-source distributed cluster-computing framework.

In this section we describe how our algorithms perform.

Let S be the input dataset (composed of n attributes and m instances) and K the number of attributes that must be returned. Let β be a ratio between SVM ranking and mRMR ranking (in the case of PSFS_mRMR), and x the number of partitions for the dataset. F represents the space of attributes. The output S' is the subset of S constituted of K features that have the maximal d_i scores.

PSFS_mRMR method follows seven steps:

- **Step 1: create x partitions of features**

1. Construct *labels*={$l_1, .., l_m$} the set of the class label in each instance.
2. Construct *vals* ={{$v_i^1, .., v_i^m$}, i=1 to n }

vals represents the values v_i^j of each attribute f_i of instance I_j:

3. Construct x subspaces of features SF_t, $t = 1..x$ from the space of feature F.
4. Construct x subspaces sub_t of {{$v_i^1, .., v_i^m$}, i∈SF}
5. Send each sub_t to a worker (between the x workers).

- **Step 2: Combine features by two with class labels**

On each worker t:

6. Map each attribute f_i with each other attribute f_j in F as follows:

$$f_i \Rightarrow \{ f_i, \{v_i^1, .., v_i^m\}, \{v_j^1, .., v_j^m\}, \{l_1, .., l_m\}\}$$

We call the set {f_i, {$v_i^1, .., v_i^m$}, {$v_j^1, .., v_j^m$}, {$l_1, .., l_m$}, i=1 to n, j=1 to n and j≠i} *rdd2*.

- **Step 3: calculate the mutual information among the features and the relevance of each feature**

In this step, we use each element of rdd2 to calculate mutual information M_{ij} between each feature f_i and another feature f_j of F. We compute also the relevance R_i (mutual information with the class label) of f_i. We proceed as follows:

For each element $e \subset$ *rdd2*

7. rdd [(f_i, M_{ij}, R_i)] = *mapToPair* (e=>{ f_i, M_{ij}, R_i })

M_{ij} = *MutualInformation* ({$v_i^1, .., v_i^m$}, {$v_j^1, .., v_j^m$})

R_i= *MutualInformation* ({$v_i^1, .., v_i^m$}, {$l_1, .., l_m$}) /n

where $l_{k \in 1..m}$ *is the class label in the instance* I_k.

End For each

The set constituted of each attribute f_i, mutual information between f_i and another attribute f_j and the relevance R_i between the class label and f_i will be called rdd3.

- **Step 4 : aggregate the mutual information for each feature by summing them.**

Calculate the redundancy of each feature f_i by summing its mutual information with other features. A set constituted of {f_i, $sumM_{ij}$, R_i}, where f_i is the feature, $sumM_{ij}$ the sum of mutual information between f_i and the other features and R_i the mutual information between f_i and the class label, is then obtained. This set is called *rdd4*.

This is done as follows:

For each element (f_i, M_{ij}, R_i) ∈ *rdd3*

8. rdd [(f_i, $sumM_{ij}$, R_i)]= *reduceByKey* (_+_)

$$sumM_{ij} = \sum_{i=1}^{n} M_{ij}$$

End Foreach

- **Step 5: for each feature, compute the SVM weight**

In this step compute for each feature f_i its SVM weight ω_i as follows:

For each $f_i \subset F$

9. rdd [(f_i, ω_i)]= map (f_i =>{ f_i, ω_i})

$\omega_i \subset \omega$ *where* ω *=SVMWeight(F)*

End For each

- **Step 6: for each feature, compute the rapport d_i between relevance and redundancy**

Determine for each feature f_i its score d_i, which represents a tradeoff between the redundancy and the relevance of f_i. Send d_i scores to the master.

This is done as follows:

For each element (f_i, $sumM_{ij}$, R_i) ⊂rdd4

10.rdd [(f_i, d_i)]= mapToPair ({f_i, $sumM_{ij}$, R_i } =>{f_i, d_i })

*Q_i= $sumM_{ij}$ /(n*n);*

$d_i = \beta + \omega_i + ((1-\beta) * (R_i / Q_i))$;

/ ω_i is the SVM weight of attribute f_i*/*

End For each

11.Workers send d_i scores to the master

- **Step 7: Choose the best features in F**

Master collects, orders and returns the *K* attributes that obtained the highest d_i scores. This is done as follows:

On the master:

12.Collect and take ordered

13.Return S': optimal subset of K features in S with highest scores d_i.

For PNFS_Spark:

PNFS_Spark algorithm follows seven steps:

- **Step 1: create x partitions of features**

1. Construct *labels*={$l_1, .., l_m$} the set of the class label of the different instance in S.
2. Construct *vals*={{$v_i^1, .., v_i^m$}, i=1 to n }

vals represents the values v_i^j of each feature f_i in each instance I_j:

3. Construct x subspaces of features SF_t, $t = 1..x$ from the feature space F.
4. Construct x subspaces sub_t of $\{\{v_i^1, .., v_i^m\}, i \in SF\}$
5. Send each sub_t to a worker.

- **Step 2: Combine features by two with labels**

On each worker x:

6. Map each attribute f_i with each other attribute f_j in F as follows:

$$f_i \Rightarrow \{f_i, \{v_i^1, .., v_i^m\}, \{v_j^1, .., v_j^m\}, \{l_1, .., l_m\}\}$$

We call the set $\{f_i, \{v_i^1, .., v_i^m\}, \{v_j^1, .., v_j^m\}, \{l_1, .., l_m\}, i=1 \text{ to } n, j=1 \text{ to } n \text{ and } j \neq i\}$ obtained *rdd2*.

- **Step 3: calculate the mutual information between features and the relevance of each feature**

In this step, we use each element of rdd2 to calculate mutual information M_{ij} between each feature f_i and another feature f_j of F. We compute also the relevance R_i (mutual information with the class label) of f_i. We proceed as follows:

For each element $e \subset rdd2$

7. $rdd\ [(f_i, M_{ij}, R_i)] = mapToPair\ (e \Rightarrow \{f_i, M_{ij}, R_i\})$

$M_{ij} = MutualInformation\ (\{v_i^1, .., v_i^m\}, \{v_j^1, .., v_j^m\})$

$R_i = MutualInformation\ (\{v_i^1, .., v_i^m\}, \{l_1, .., l_m\})\ /n$

where $l_{k \in 1..m}$ *is the class label in different instances.*

End For each

The set constituted of each attribute f_i, mutual information between f_i and another attribute f_j and the relevance R_i between the class label and f_i will be called rdd3.

- **Step 4 : calculer la valeur medianne de la relevance des attributs avec l'etiquette de classe.**

Calculate R_{fe} the median value of the set of values of relevance R_i.

Let f_e be the index of the median value in the list of relevance values between features and class labels.

If (n%2==0)

$$f_e = (n+1)/2$$

else

$$f_e = (n/2 + n/2+1)/2$$

$R_{fe} = rdd3\ [f_e].\ R_e$

- **Step 5: for each feature, determine the list of his neighbor**

In this step calculate for each feature f_i the ranking measure d_i that represents a tradeoff between the redundancy and the relevance of f_i. Then send all d_i values to the master.

This is done as follows:

On each worker x:

For each $f_i \subset sub_x$

For each $f_j \subset F$

/ M_{ij} represents the mutual information between f_i and f_j */*

If ($M_{ij} > R_{fe}$)

rdd $[(f_i, Vois_{ij})]$ = *mapToPair* ({f_i} **=>**{f_i, f_j})

EndForeach

EndForeach

8. All workers send d_i to the master

- **Step 6: for each feature, compute the ranking measure d_i which represents the score of the feature**

Determine for each feature f_i its score d_i, which represents a tradeoff between the redundancy and the relevance of f_i. Send d_i scores to the master.

This is done as follows:

For each element $f_i \subset$ *rdd* $[(f_i, Vois_i)]$

For each element $f_j \subset$ *rdd* $[f_i].Vois_i)$

rdd $[(f_i, d_i)]$ = *mapToPair* ({f_i} **=>**{f_i, d_i})

/ determines nV number of neighbor of f_i */*

nV=$Vois_i$*.length*

/ the score d_i of f_i is obtained by multiplying its relevance R_i with its number of neighbor nV */*

$$d_i = \sum_{i=1}^{nV} R_i$$

End For each

End For each

9. Workers send d_i scores to the master

- **Step 7: Choose the best features in F**

Master collects, orders and returns the *K* attributes that obtained the highest d_i scores. This is done as follows:

On the master:

10.Collect and take ordered

Return S′: optimal subset of K features in S with highest d_i ***scores.***

7 Experiment Setup

The classifier that we used in our experiments is support vector machine.

The datasets used are from mldata.org [15] and are in LibSVM format. Table 1 describes those datasets.

Table 1. Characteristics of datasets

Name	Number of Features	Number of Instances
Colon-cancer	2000	62
Colon-Tumor	2000	60

We perform ours experiments on a cluster of 4 nodes, then on a cluster of 6 nodes. Each node runs at the linux and consists of 8 cores. Each core runs at 2.60 GHz and has 56 GB of memory and 382 GB of disk.

8 Experimental Results

We conducted the experiment with different number of selected features and number of nodes to measure the scalability of our propositions and compare the running time of ours methods in comparison with the centralized ones.

Figures 1, 2 and 3 show respectively the running time of PSFS_mRMR method and classical mRMR algorithm for selecting 25%, 50% or 75% of the colon-cancer dataset with various numbers of nodes.

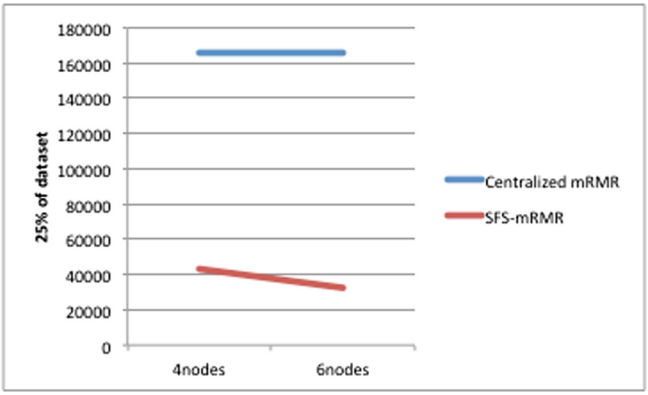

Fig. 1. Scalability of P**SFS_mRMR** and classical **mRMR** with 25%.

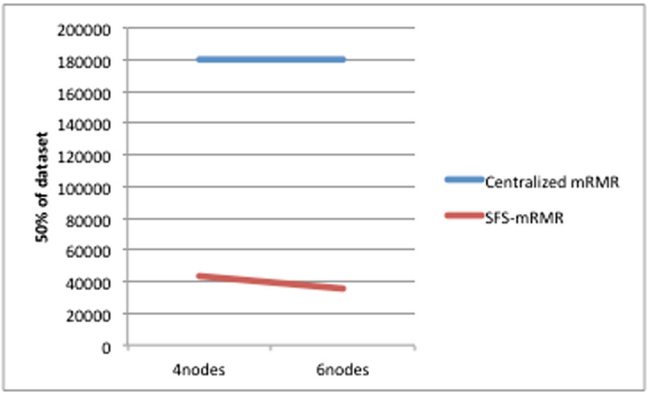

Fig. 2. Scalability of P**SFS_mRMR** and classical **mRMR** with 50%.

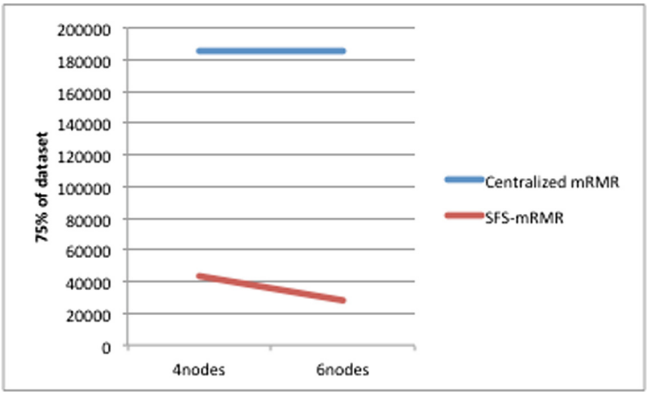

Fig. 3. Scalability of P**SFS_mRMR** and classical **mRMR** with 75%.

Figures 4, 5 and 6 show respectively the running time of **PNFS_Spark** method and its centralized version for selecting 25%, 50% or 75% of the colon-cancer dataset various numbers of nodes.

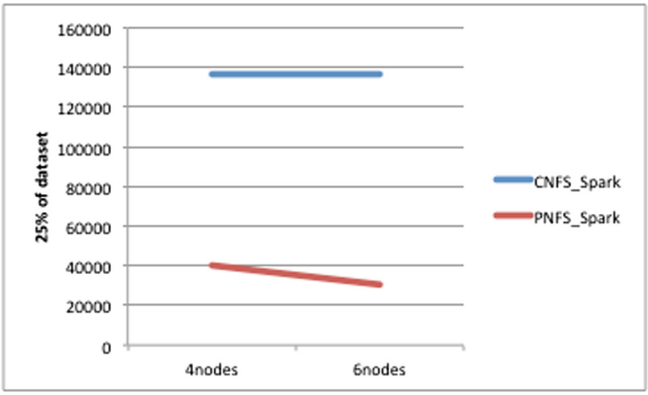

Fig. 4. Scalability of **PNFS_Spark** and **CNFS_Spark** with 25%.

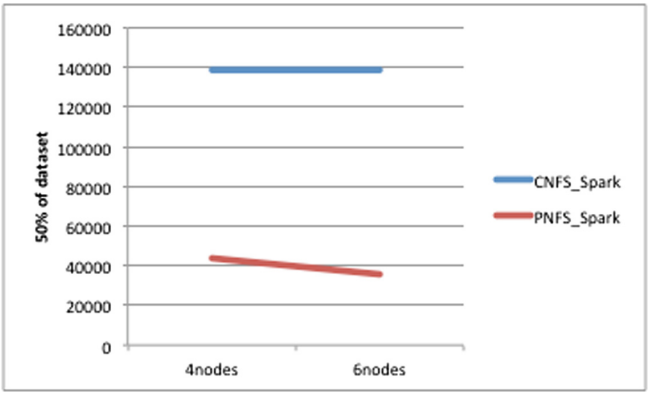

Fig. 5. Scalability of **PNFS_Spark** and **CNFS_Spark** with 50%.

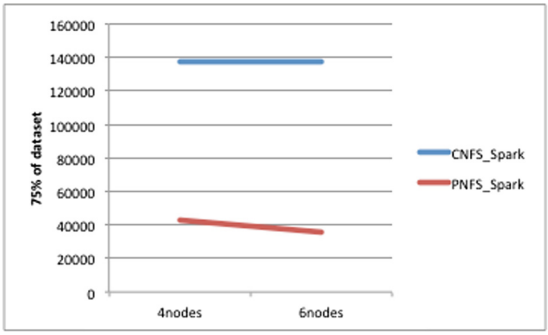

Fig. 6. Scalability of **PNFS_Spark** and **CNFS_Spark** with 75%.

From the figure, when we add more nodes, the running time of **PSFS_mRMR** and **PNFS_Spark** considerably decreases, whereas the time taken by **classical mRMR** and **CNFS_Spark** remains the same.

We have used 4 then 6 nodes for the scalability. And for every case we have run the tests using the same environment.

For every dataset we first select 25% then 50% and after 75% of features.

After having studied the scalability of our propositions we will study the time taken by the feature selection for our parallel methods in comparison with the central methods. We start first with the colon-cancer dataset then with colon-tumor.

- **Colon-cancer**

Figures 7 and 8 show the running time of PSFS_mRMR method and the classical mRMR.

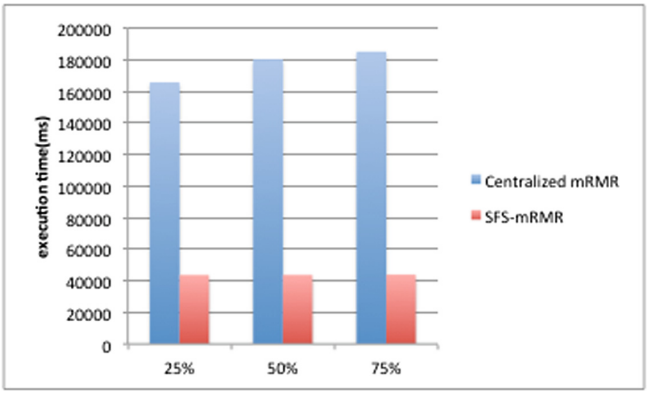

Fig. 7. Time taken for colon-cancer with 4nodes

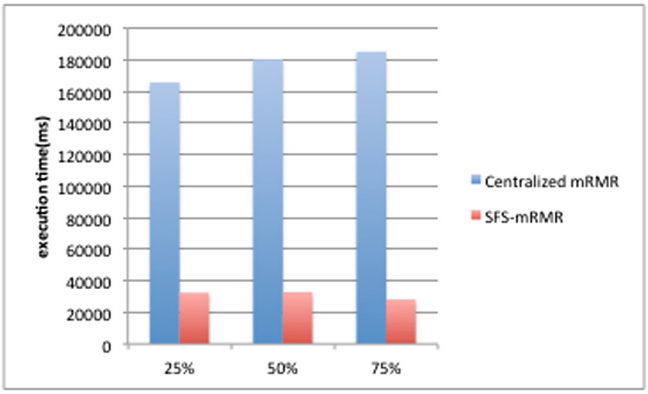

Fig. 8. Time taken for colon-cancer with 6nodes

Figures 9 and 10 show the running time of PNFS_Spark method in comparison with its centralized one for respectively 4 and 6 nodes.

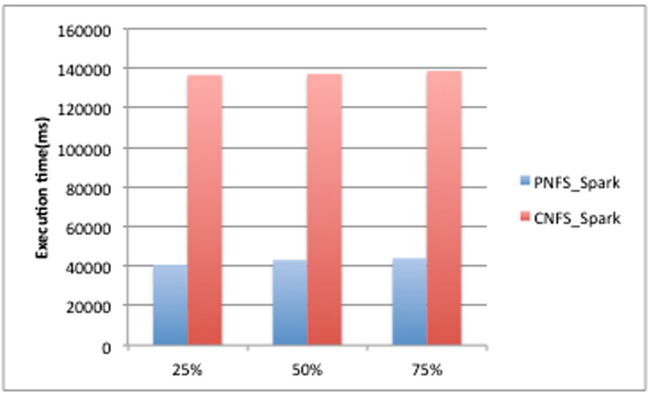

Fig. 9. Time taken for colon-cancer with 4nodes

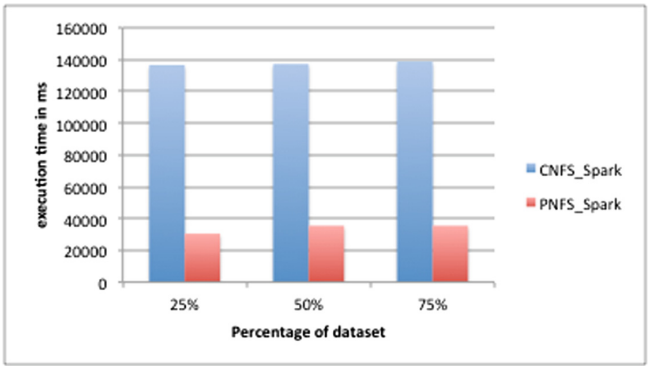

Fig. 10. Time taken for colon-cancer with 6nodes

From the results, we can see that, the running time of ours parallel methods is at least 4 times shorter than the classical **mRMR** and **CNFS_Spark** algorithm.

- **Colon-Tumor**

  Results obtained for colon-tumor with 4 nodes are shown in Figs. 11 and 12: For a cluster of 6 nodes the running time is stated in Figs. 13 and 14.

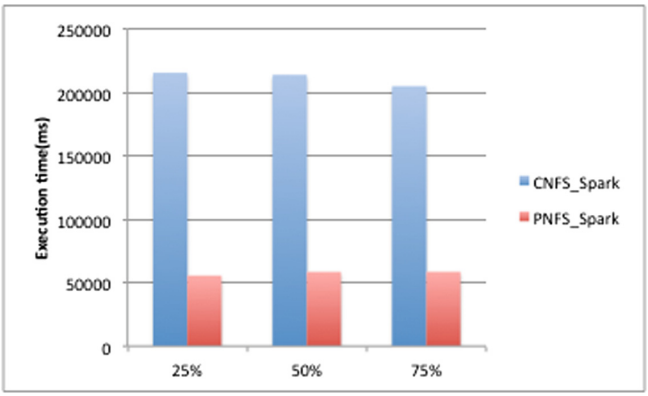

Fig. 11. Time taken per PNFS_Spark for colon-tumor with 4 nodes

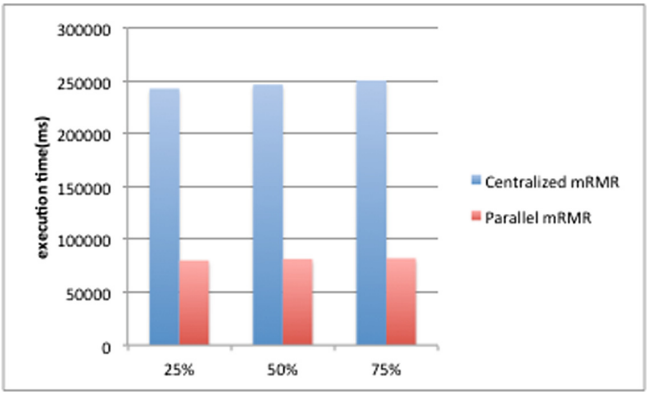

Fig. 12. Time taken per PSFS_mRMR for colon-tumor with 4 nodes

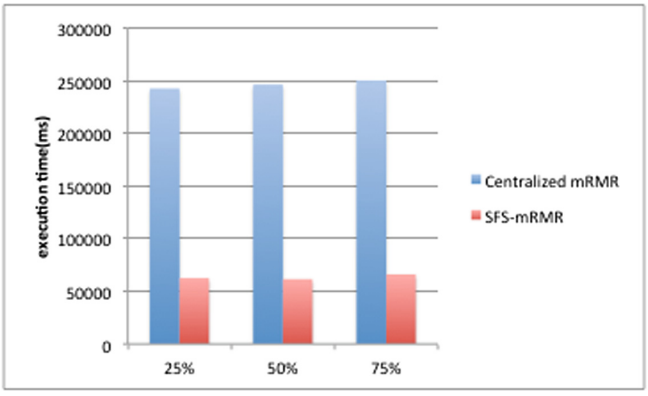

Fig. 13. Time taken per PSFS_mRMR for colon-tumor with 6 nodes

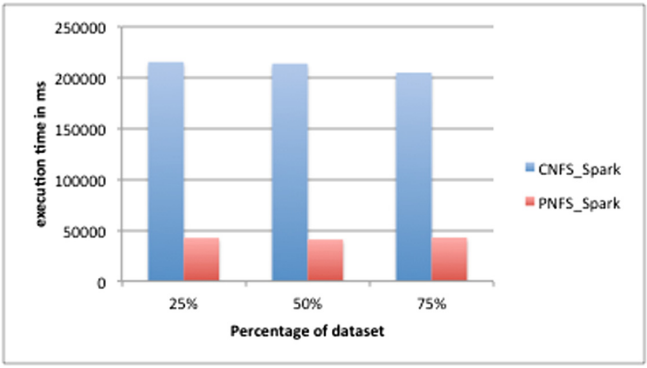

Fig. 14. Time taken per **PNFS_Spark** for colon-tumor with 6 nodes

As for the colon-cancer we can notice that the execution time of PSFS-mRMR and PNFS_Spark is also 4 times shorter at least.

Therefore, the conclusion of ours experiments is that parallel solutions outperform centralized methods in terms of running time. Moreover, when we add more nodes, the running time of ours methods becomes shorter, whereas the time taken by the centralized one remains constant.

Our empirical analysis confirms the scalability of ours algorithms with respect to the number of features and processing cores.

Our experiment is limited to the datasets up to 2000 features since beyond that number, the centralized algorithms takes too much time to run.

9 Conclusion

In this paper we have proposed two feature selection methods, which are capable of scaling feature selection to large datasets. One of them is a parallel version of a famous feature selection method called mRMR. Our proposal was developed on Spark, a unified analytics engine for large datasets processing.

Our methods consist of computing the score of each feature relatively to its redundancy with the others features and its relevance with the class label. Then return the features with the best scores.

Experimental results demonstrate that our parallel algorithms achieve a great performance improvement in scaling well and reducing the running time. In the case of mRMR our parallel method PSFS_mRMR have led to better performance in terms of selecting relevant and non redundant features.

In the future, we plan to parallelize many other classical feature selection methods like RELIEF or RFE-SVM.

Acknowledgment. We would like to express our sincere thanks to the CEA-MITIC (**A**frican **C**enter of **E**xcellence in **M**athematics, Computer Science and **Tic**) who financed our research by paying the publication fees of the 2 papers that we published in Africatek2018.

The CEA-MITIC, located at the UFR of Applied Sciences and Technology (UFR SAT) of the Gaston Berger University (UGB) of Saint-Louis in Senegal, is a consortium of university institutions in Senegal and subregion of Senegal, research institutions and national, regional and international companies involved in the ICT sector.

In this work Microsoft Azure sponsored us by putting at our disposal a cluster of machines for the tests and we send them our warm thanks as well.

References

1. Liu, C., Wang, W., Zhao, Q., Konan, M.: A new feature selection method based on a validity index of feature subset. Pattern Recognit. Lett. **92**, 1–8 (2017)
2. Wenyan, Z., Xuewen, L., Jingjing, W.: Feature selection for cancer classification using microarray gene expression data. Biostat. Biom. Open Acc. J. **1**(2), 555557 (2017)
3. Zhao, Z., Cox, J., Duling, D., Sarle, W.: Massively parallel feature selection: an approach based on variance preservation. In: Flach, Peter A., De Bie, T., Cristianini, N. (eds.) ECML PKDD 2012. LNCS (LNAI), vol. 7523, pp. 237–252. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33460-3_21
4. Singh, D., Reddy, C.K.: A survey on platforms for big data analytics. J. Big Data **2**(1), 8 (2015). Published online 9 October 2014
5. Jaseena, K.U., David, J.M.: Issues, challenges, and solutions: big data mining. In: Sixth International Conference on Networks and Communications (2014). https://doi.org/10.5121/csit.2014.41311
6. Ramırez-Gallego, S., et al.: An information theory-based feature selection framework for big data under apache spark. J. Latex Class Files **13**(9) (2014)
7. De Jay, N., Papillon, S., Olsen, C., El-Hachem, N., Bontempi, G., Haibe-Kains, B.: mRMRe: an R package for parallelized mRMR ensemble feature selection. Bioinformatics **29**, 2365–2368 (2013). https://doi.org/10.1093/bioinformatics/btt383
8. Zhang, Y., Ding, C., Li, T.: Gene selection algorithm by combining reliefF and mRMR. BMC Genom. **9**(Suppl 2), S27 (2008). https://doi.org/10.1186/1471-2164-9-S2-S27
9. Ramírez-Gallego, S., et al.: Fast-mRMR: fast minimum redundancy maximum relevance algorithm for high-dimensional big data: FAST-mRMR ALGORITHM FOR BIG DATA. Int. J. Intell. Syst. **32**, 134–152 (2016). https://doi.org/10.1002/int.21833
10. Eiras-Franco, C., Bolón-Canedo, V., Ramos, S., González-Domínguez, J., Alonso-Betanzos, A., Touriño, J.: Multithreaded and spark parallelization of feature selection filters. Journal of Computational Science **17**(Part 3), 609–619 (2016)
11. Palma-Mendoza, R.J., Rodriguez, D., de-Marcos, L.: Distributed ReliefF-based feature selection in spark. Knowl. Inf. Syst. **57**, 1–20 (2018)
12. Mandal, M., Mukhopadhyay, A.: An improved minimum redundancy maximum relevance approach for feature selection in gene expression data. In: IEEE/ACM Transactions on Computational Biology and Bioinformatics, July 2016
13. Chang, Y.-W., Lin, C.-J.: Feature ranking using linear SVM. In: Proceedings of the Workshop on the Causation and Prediction Challenge at WCCI 2008 (2008). PMLR **3**, 53–64
14. Mundra, P.A., Rajapakse, J.C.: SVM-RFE with MRMR filter for gene selection. IEEE Trans. Nanobiosci. **9**(1), 31–37 (2010)
15. http://mldata.org/repository/data/viewslug/ovarian-cancer-nci-pbsii-data/