A Distributed Rough Set Theory based Algorithm for an Efficient Big Data Pre-processing under the Spark Framework

Zaineb Chelly Dagdia*†, Christine Zarges*, Gaël Beck‡ and Mustapha Lebbah‡
*Department of Computer Science, Aberystywth University, Aberystwyth, United Kingdom
†LARODEC, Institut Supérieur de Gestion de Tunis, Tunis, Tunisia
‡ Computer Science Laboratory (LIPN), University Paris-North - 13, Villetaneuse, France

Abstract—**Big Data reduction is a main point of interest across a wide variety of fields. This domain was further investigated when the difficulty in quickly acquiring the most useful information from the huge amount of data at hand was encountered. To achieve the task of data reduction, specifically feature selection, several state-of-the-art methods were proposed. However, most of them require additional information about the given data for thresholding, noise levels to be specified or they even need a feature ranking procedure. Thus, it seems necessary to think about a more adequate feature selection technique which can extract features using information contained within the dataset alone. Rough Set Theory (RST) can be used as such a technique to discover data dependencies and to reduce the number of features contained in a dataset using the data alone, requiring no additional information. However, despite being a powerful feature selection technique, RST is computationally expensive and only practical for small datasets. Therefore, in this paper, we present a novel efficient distributed Rough Set Theory based algorithm for large-scale data pre-processing under the Spark framework. Our experimental results show the efficient applicability of our RST solution to Big Data without any significant information loss.**

Index Terms—**Big Data Pre-processing; Feature Selection; Rough Set Theory; Distributed Processing; Scalability.**

I. Introduction

Big Data are extremely valuable, but arise with many challenges—most importantly in dimensionality reduction and specifically for feature selection [1], [2]. The use of Rough Set Theory (RST) [3], [4] for feature selection is one approach that has proved successful and more efficient in comparison to a variety of state-of-the-art feature selection methods [5]. However, despite being a powerful feature selection technique, most of the traditional rough set based algorithms are sequential algorithms, computationally expensive and can only deal with small datasets. The computational intensive nature of RST and its incapacity to deal with high dimensional data arise from the necessity to generate all the possible combinations of features at once, process them in turn to finally select the most relevant set. However, as the number of features is increasing this task becomes challenging and this is where the RST inadequacy arises [6]. It is quite unmanageable to generate the set of all possible feature combinations due to hardware and memory constraints. Thus, in this paper, we present a novel efficient distributed rough set based algorithm, named Sp-RST, for large-scale data pre-processing.

II. Rough Sets for Feature Selection

In Rough Set Theory, an *information table* is defined as a tuple $T = (U, A)$ where U and A are two finite, non-empty sets, U the *universe* of primitive objects and A the set of attributes. Each attribute or feature $a \in A$ is associated with a set V_a of its value, called the *domain* of a. We may partition the attribute set A into two subsets C and D, called *condition* and *decision* attributes, respectively.

Let $P \subset A$ be a subset of attributes. The indiscernibility relation, denoted by $IND(P)$, is the central concept to RST and it is an equivalence relation which is defined as: $IND(P) = \{(x, y) \in U \times U : \forall a \in P, a(x) = a(y)\}$, where $a(x)$ denotes the value of feature a of object x. If $(x, y) \in IND(P)$, x and y are said to be *indiscernible* with respect to P. The family of all equivalence classes of $IND(P)$, referring to a partition of U determined by P, is denoted by $U/IND(P)$. Each element in $U/IND(P)$ is a set of indiscernible objects with respect to P. The equivalence classes $U/IND(C)$ and $U/IND(D)$ are called *condition* and *decision* classes, respectively. For any concept $X \subseteq U$ and attribute subset $R \subseteq A$, X could be approximated by the R-*lower* approximation and R-*upper* approximation using the knowledge of R. The lower approximation of X is the set of objects of U that are surely in X, defined as: $\underline{R}(X) = \bigcup\{E \in U/IND(R) : E \subseteq X\}$. The upper approximation of X is the set of objects of U that are possibly in X, defined as: $\overline{R}(X) = \bigcup\{E \in U/IND(R) : E \cap X \neq \emptyset\}$. The concept defining the set of objects that can possibly, but not certainly, be classified in a specific way is called the *boundary region* which is defined as: $BND_R(X) = \overline{R}(X) - \underline{R}(X)$. If the boundary region is empty, that is $\overline{R}(X) = \underline{R}(X)$, concept X is said to be R-*definable*; otherwise X is a *rough set* with respect to R. The *positive region* of decision classes $U/IND(D)$ with respect to condition attributes C is denoted by $POS_c(D)$ where $POS_c(D) = \bigcup \overline{R}(X)$. The positive region $POS_c(D)$ is a set of objects of U that can be classified with certainty to classes $U/IND(D)$ employing attributes of C. Based on the positive region, the *dependency of attributes* is defined as: $k = \gamma(C, c_i) = \frac{|POS_C(c_i)|}{|U|}$ measuring the degree k of the dependency of an attribute c_i on a set of attributes C.

Based on these basics, RST defines two important concepts for feature selection which are the $Core$ and the $Reduct$. In RST, a subset $R \subseteq C$ is said to be a D-*reduct* of C if $\gamma(C, R) = \gamma(C)$ and there is no $R' \subset R$ such that $\gamma(C, R^{'}) = \gamma(C, R)$. In other words, the $Reduct$ is the minimal set of selected attributes preserving the same dependency degree as the whole set of attributes. Meanwhile, RST may generate a set of reducts, $RED_D^F(C)$, from the given information table. In this case, any reduct from $RED_D^F(C)$ can be chosen to replace the initial information table. The second concept, the $Core$, is the set of attributes that are contained by all reducts, defined as $CORE_D(C) = \bigcap RED_D(C)$; where $RED_D(C)$ is the D-reduct of C. Specifically, the $Core$ is the set of attributes that cannot be removed from the information system without causing collapse of the equivalence-class structure. This means that all attributes present in the $Core$ are indispensable.

III. The Proposed Solution

A. General Model Formalization

Sp-RST creates a Resilient Distributed Dataset (RDD) and formalizes it as a given information table defined as T_{RDD}, where $U = \{x_1, \ldots, x_N\}$ is the universe, the conditional attribute set is $C = \{c_1, \ldots, c_V\}$ and the decision attribute $D = \{d_1, \ldots, d_W\}$ corresponds to the class (label) of each T_{RDD} sample. The conditional attribute set C presents the pool from where the most convenient features will be selected.

In order to make our algorithm scalable with the high number of features, we partition the given T_{RDD} into m data blocks based on splits from the conditional attribute set C. Hence, $T_{RDD} = \bigcup_{i=1}^{m} (C_r) T_{RDD_{(i)}}$, where $r \in \{1, \ldots, V\}$. Each $T_{RDD_{(i)}}$ is constructed based on r random features selected from C, where $\forall T_{RDD_{(i)}} : \nexists \{c_r\} = \bigcap_{i=1}^{m} T_{RDD_{(i)}}$. To ensure scalability, rather than applying Sp-RST to T_{RDD} including the whole C set the distributed algorithm will be applied to every single $T_{RDD_{(i)}}$ that at the end all the intermediate results will be gathered from the different m partitions. In such a way, we can guarantee that Sp-RST can be applied to a computable number of features and hence solving the standard RST computational inefficiencies. Algorithm 1 highlights the pseudo-code of our proposed Sp-RST solution. In order to further guarantee the Sp-RST performance while avoiding any significant information loss, we apply the algorithm N times on the T_{RDD} m data blocks. More precisely, through all the iterations, the algorithm will first generate the m random $T_{RDD_{(i)}}$ as previously explained. Then, for each partition the distributed Sp-RST tasks, Algorithm 1 line 5 to 10, will be executed. As seen in Algorithm 1, line 1 presenting the first Sp-RST task is executed before the iteration loop. This is because this task deals with the calculation of the indiscernibility relation of the decision class $IND(D)$ and is independent from the m generated partitions as the result depends on the data items class and not on the features. After the iteration loop, line 12, the output of each partition is either a single reduct $RED_{i_{(D)}}(C_r)$ or a family of reducts $RED_{i_{(D)}}^F(C_r)$. Based on the RST preliminaries, any reduct of $RED_{i_{(D)}}^F(C_r)$ can be used to represent the $T_{RDD_{(i)}}$ information table. Consequently, if Sp-RST generates only one reduct, for a specific $T_{RDD_{(i)}}$ block, then the output of this feature selection phase is the set of the $RED_{i_{(D)}}(C_r)$ features. These features reflect the most informative ones among the C_r attributes resulting a new reduced $T_{RDD_{(i)}}$, $T_{RDD_{(i)}}(RED)$, which preserves nearly the same data quality as its corresponding $T_{RDD_{(i)}}(C_r)$ that is based on the whole feature set C_r. On the other hand, if Sp-RST generates a family of reducts then the algorithm randomly selects one reduct among $RED_{i_{(D)}}^F(C_r)$ to represent the corresponding $T_{RDD_{(i)}}$. This random choice is justified by the same priority of all the reducts in $RED_{i_{(D)}}^F(C_r)$. At this stage, each i data block has its output $RED_{i_{(D)}}(C_r)$ referring to the selected features. However, since each $T_{RDD_{(i)}}$ is based on distinct features and with respect to $T_{RDD} = \bigcup_{i=1}^{m} (C_r) T_{RDD_{(i)}}$ a union of the generated selected features is required to represent the initial T_{RDD}; Algorithm 1, line 12 to 14. As previously mentioned, the process of applying Sp-RST will iterate N times generating N $Reduct_m$. Thus, at the end an intersection of all the obtained $Reduct_m$ is needed. By removing irrelevant and redundant features, Sp-RST can reduce the dimensionality of the data from $T_{RDD}(C)$ to $T_{RDD}(Reduct)$.

Algorithm 1 Sp-RST

```
    Inputs: $T_{RDD}$ the information table
            $m$ number of partitions
            $N$ the number of iterations
    Output: $Reduct$
 1: Calculate $IND(D)$
 2: for each iteration $n \in [1, \ldots, N]$ do
 3:     Generate $T_{RDD_{(i)}}$ based on the $m$ partitions
 4:     for each $T_{RDD_{(i)}}$ partition, $i \in [1, \ldots, m]$ do
 5:         Generate $AllComb_{(C_r)}$
 6:         Calculate $IND(AllComb_{(C_r)})$
 7:         Calculate $DEP(AllComb_{(C_r)})$
 8:         Select $DEP_{max}(AllComb_{(C_r)})$
 9:         Filter $DEP_{max}(AllComb_{(C_r)})$
10:         Filter $NbF_{min}(DEP_{max}(AllComb_{(C_r)}))$
11:     end for
12:     for each $T_{RDD_{(i)}}$ output do
13:         $Reduct_m = \bigcup_{i=1}^{m} RED_{i_{(D)}}(C_r)$
14:     end for
15: end for
16: $Reduct$ = $\bigcap_{n=1}^{N} Reduct_m$
17: return ($Reduct$)
```

B. Algorithmic Details

The algorithm goes through 7 main jobs in order to generate the final sought $Reduct$. First, Sp-RST has to compute the indiscernibility relation for the decision class $D = \{d_1, \ldots, d_W\}$; defined as $IND(D)$: $IND(d_i)$. More precisely, Sp-RST will calculate the indiscernibility relation for every decision class d_i by gathering the same T_{RDD} data items which are defined in the universe $U = \{x_1, \ldots, x_N\}$

and which belong to the same class d_i. To do so, Sp-RST processes a $foldByKey$ operation where the decision label d_i defines the key and the T_{RDD} data items identifiers id_i of x_i, define the values (see Algorithm 2).

Algorithm 2 Calculate $IND(D)$

```
Input: $T_{RDD}$
Output: $IND(D) : Array[Array[x_i]]$
$IND(d_i) = data.map\{case(id_i, vector, d_i) => (d_i, ArrayBuffer(id_i))\}$
    $.foldByKey(ArrayBuffer.empty[Long])(\_ + + = \_)$
$IND(d_i).map\{case(d_i, x_i) => x_i\}.collect$
```

Once the $IND(D)$ is calculated and within a specific partition, Sp-RST creates all the possible combinations of the C_r set of feature; $AllComb_{(C_r)}$. The third Sp-RST job deals with the indiscernibility relation computation for every previously generated combination. As presented in Algorithm 3, Sp-RST aims at grouping all the data items identifiers id_i sharing the same specific combination of features extracted from $AllComb_{(C_r)}$. In order to achieve this, we use the $foldByKey$ spark operation where the combination of features defines the key and the id_i as value.

Algorithm 3 Calculate $IND(AllComb_{(C_r)})$

```
Inputs: $T_{RDD_i}$, $AllComb_{(C_r)}$
Output: $IND(AllComb_{(C_r)}) : Array[Array[id_i]]$
$IND(AllComb_{(C_r)}) = data.map$
    $\{case(id_i, vector, d_i) => ((AllComb_{(C_r)_i}, vector), ArrayBuffer(id_i))\}.foldByKey(ArrayBuffer.$
    $empty[Long])(\_ + + = \_)$
$IND(AllComb_{(C_r)}).map\{case(ListValues, id_i) => id_i\}.collect$
```

Then, the dependency degrees $\gamma(C_r, AllComb_{(C_r)})$ of each feature combination are computed. To do so, the calculated indiscernibility relations $IND(D)$ and $IND(AllComb_{(C_r)})$ as well as the set of all feature combinations $AllComb_{(C_r)}$ are required. The task is to first test first if the intersection of every $IND(d_i)$ with each $IND(AllComb_{(C_r)})$ keeps all the latter elements; referring to the lower approximation. If so then a score which is equal to the length of $IND(AllComb_{(C_r)})$ is given, zero otherwise. As this process is made in a distributed way where each machine is dealing with some feature combinations, a first sum operation of the $IND(d_i)$ scores is operated followed by a second sum operation to record all the $IND(D)$ scores; referring to the dependency degrees $\gamma(C_r, AllComb_{(C_r)})$. The output of this step is the set of dependency degrees $\gamma(C_r, AllComb_{(C_r)})$ of the feature combinations $AllComb_{(C_r)}$ and their associated sizes $Size_{(AllComb_{(C_r)})}$. At this stage, Sp-RST looks for the maximum dependency value among all $\gamma(C_r, AllComb_{(C_r)})$ using the max function operated on the given RDD. The output $MaxDependency$ reflects in one hand the dependency of the whole feature set (C_r) representing the T_{RDD_i} and on the other hand the dependency of all the possible feature combinations satisfying the constraint $\gamma(C_r, AllComb_{(C_r)}) = \gamma(C_r)$. $MaxDependency$ is the baseline value for feature selection.

Once the $MaxDependency$ is generated, Sp-RST keeps the set of all combinations having the same dependency degrees as $MaxDependency$; $\gamma(C_r, AllComb_{(C_r)}) = MaxDependency$. This is achieved by applying a filter function. In fact, at this stage Sp-RST removes in each computation level the unnecessary features that may affect negatively the performance of any learning algorithm. Finally and based on the output of the previous step, Sp-RST keeps the set of combinations having the minimum number of features, $Size_{(AllComb_{(C_r)})}$, by applying a filter operation and by satisfying the full reduct constraints discussed in Section II; $\gamma(C_r, AllComb_{(C_r)}) = \gamma(C_r)$ while there is no $AllComb^{'}_{(C_r)} \subset AllComb_{(C_r)}$ such that $\gamma(C_r, AllComb^{'}_{(C_r)}) = \gamma(C_r, AllComb_{(C_r)})$. Each combination satisfying this condition is considered as a viable minimum reduct set. The attributes of the reduct set describe all concepts in the original training dataset T_{RDD_i}.

IV. Experimental Setup

A. Used Benchmark

To demonstrate the effectiveness of our proposed approach we chose the Amazon Commerce reviews data set from the UCI machine learning repository [7] as it was the dataset with the largest number of features that still had a sufficiently large number of data items. This data set was derived from customer reviews on the Amazon commerce website by identifying a set of most active users and with the goal to perform authorship identification. It includes 1 500 data items described through 10 000 features and 50 distinct classes (authors). Instances are identically distributed across the different classes.

B. Experimental plan, testbed and tools

Our experiments are performed on the Grid5000 testbed which is a French national large-scale and versatile platform. Under this testbed, we used dual 8 core Intel Xeon E5-2630v3 CPUs and 128 GB memory to test the performance of our Sp-RST algorithm which is implemented in Scala 2.11 within the Spark 2.1.1 framework [8].

The main aim of our experimentation is to demonstrate how our proposed approach Sp-RST speeds up the execution time for large data sets without introducing too much information loss. We investigate different parameters of Sp-RST and analyze how these affect execution time and stability of the feature selection. We then show that the improvement in performance does not decrease the feature selection ability by using a Random Forest classifier on the original dataset and the reduced datasets produced by Sp-RST. We use the Random Forest implementation provided in the Spark framework (org.apache.spark.mllib.tree.RandomForest) with the following parameters: maxDepth=6, numTrees=300, featureSubsetStrategy='all' and impurity='gini'. The algorithm automatically identifies categorical features and indexes them. Preliminary results revealed that a maximum of 10 features per partition is the limit that can be processed by Sp-RST. We therefore perform experiments for 1000, 1200, 1250, 1330, 1500, 2000 and 2500 partitions in Algorithm 1. We run all settings on 1, 4, 8, 16, and 32 nodes on Grid5000. For the purpose of this study we set the number of iterations in Algorithm 1 to 10 (based on preliminary experiments).

Our analysis first focuses on the scalability of the algorithm that allows it to solve the standard rough set feature selection inadequacy to be applied to Big Data. To do so, we will evaluate the performance of Sp-RST using the *speedup* and *sizeup* criteria introduced in [9]. It should be noted that the authors introduce a third criterion (*scaleup*), however, given the above experimental setup and the overall execution times we are currently unable to obtain sufficient data to perform a meaningful analysis based on scaleup. For the evaluation of the Random Forest classifier we use the standard set based performance measures which are the time, measured in seconds, and the classification error.

Since both, Sp-RST and Random Forest are randomized, we need to use appropriate statistics in our analysis. For Sp-RST, randomization is used during partitioning and the selection of one reduct among the generated family of reducts. We reduce the effect of the first by performing several iterations of the main part of the algorithm and keeping only features that are selected in all iterations. The second is justified by the fundamentals of Rough Set Theory as discussed in Section II.

We perform an analysis of the stability of the selected feature sets by considering several runs of Algorithm 1 and report averages and standard deviations for the classification error of 100 independent runs of the Random Forest classifier. To investigate the significance of any observed difference in classification error between the Random Forest classifier on the original dataset and a reduced set produced by Sp-RST we perform Wilcoxon signed rank tests.

V. Results and Analysis

A. Speedup

We first consider the speedup [9] of Sp-RST: We keep the size of the dataset constant (where size is measured by the number of features in each partition) and increase the number of nodes. We plot the average time needed to run a single iteration within Algorithm 1 (over the 10 iterations executed) and the respective speedups in Figures 1 and 2, respectively.

As discussed in [9], an ideal parallel algorithm has linear speedup, which is, however, difficult to achieve in practice due to communication cost and the fact that the slowest slave dominates the total execution time. From Figure 1 we see that our method has a good speedup for settings with fewer partitions. The more the size of the database, i. e., the number of attributes per partition, increases, the closer the speedup gets to linear. This can be explained by the fact that fewer partitions imply that each partition has more features. As discussed previously the execution time grows exponentially in the number of features and thus, using more nodes is more beneficial in cases with many features. We obtain good speedup if the number of features per partition is between 7 and 10 (1000 to 1330 partitions), but for 4 or 5 features (2000 and 2500 partitions, respectively) the speedup quickly stagnates. This observation is also supported by the execution times (Figure 2). For few partitions the execution time quickly decreases with increasing number of nodes while for many partitions we observe hardly any improvement.

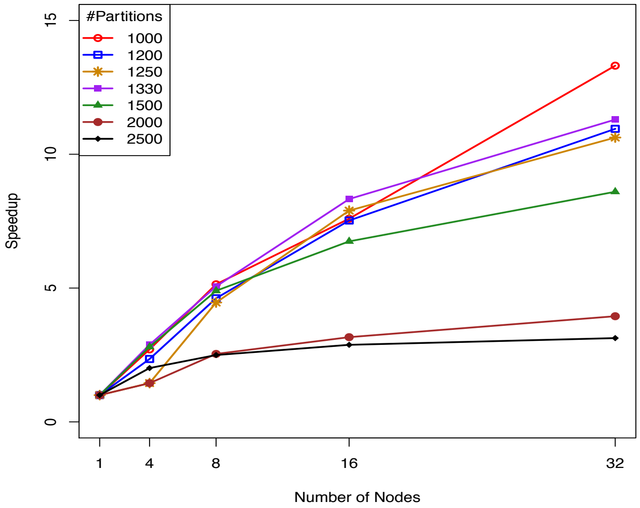

Fig. 1. Speedup for different numbers of nodes and partitions.

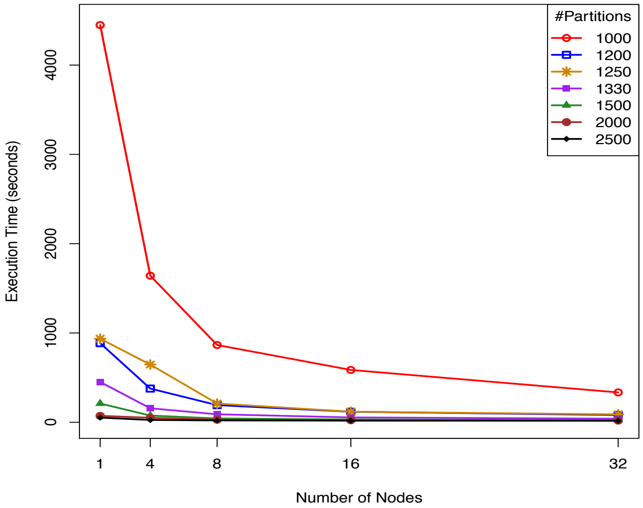

Fig. 2. Average execution times for a single iteration of Algorithm 1 and for different numbers of nodes and partitions.

B. Sizeup

We use Sp-RST with 2500 partitions as baseline as this is the smallest dataset in our experiments and calculate the m values for the other settings based on the number of features per partition. We then plot the sizeup [9] in Figure 3. We see that the sizeup of Sp-RST grows very quickly as m increases, but gets better as the number of nodes increases. Recall that we define the size of the database as the number of features per partition. Thus, this behavior was expected due to the runtime

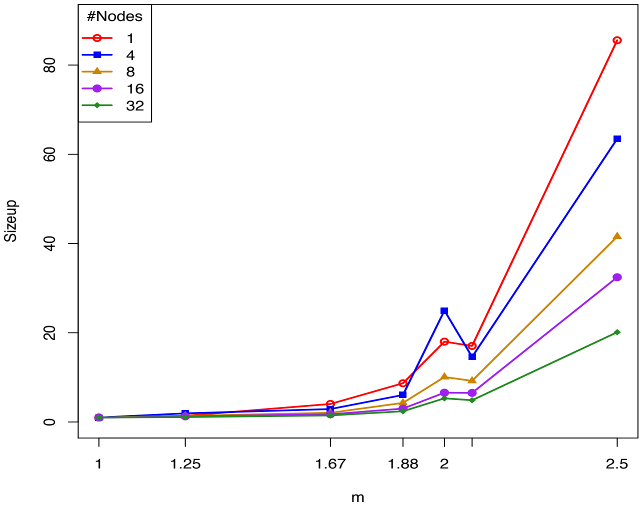

Fig. 3. Sizeup for different numbers of nodes and values of m.

properties of Sp-RST previously discussed. We conjecture that using the classic definition of size as the number of features in the whole database would yield a good sizeup, i. e., that our method is able to process large datasets efficiently while keeping the number of nodes constant and increasing the size of the data.

Together with the above discussion of the speedup we see that there is some trade-off between the number of partitions and the number of nodes used. If only few nodes are available, it may be advisable to use a larger number of partitions to reduce execution times while the number of partitions becomes less important if a high degree of parallelization can be afforded. It should be noted that using a larger number of partitions has the potential to increase information loss during the data-preprocessing step. Our following experimentation and analysis shows that this, however, is not the case.

C. Stability of Feature Selection

To measure the stability of the feature selection process we consider the number of features selected over several runs as well as the overlap of the resulting feature sets (keeping in mind that according to Rough Set Theory different sets may be equivalent). Since the number of nodes used has no influence on the concrete feature selection, we compare the results for each number of partitions (7 different settings) based on the experiments performed on different number of nodes (5 settings), generating 35 different feature sets. A summary of our observations is shown in Table I. We see that the number of features selected is very similar in all cases (less than 100 features difference), but varies significantly based on the number of partitions used. Comparing the overlap for each pair of runs, we see that this is also very similar (again less than 100 features). We consider the two extreme cases of features that were always selected and features that were never selected and observe that our method is very reliably in identifying features for removal. There is more variation regarding selection, particularly in cases where only few features survive: Here the set of features that are always selected is relatively small and it is not that likely that always the same features are selected. This can be explained by the fact that features are only included in the final set if they were selected in all 10 iterations of the algorithm. We will show in the following sections that this property of our algorithm is not detrimental for the classification task performed and conjecture that this can be explained by the fundamentals of Rough Set Theory as discussed in Section II.

D. Classification Error With and Without Sp-RST

To validate the suitability of our method with respect to classification, we investigate the influence of Sp-RST on the classification error of the Random Forest classifier. We present results of 100 independent runs on the original dataset with 10 000 features and the datasets derived by Sp-RST with different numbers of partitions in Table II. While the overall error seems high, it should be noted that the used database contains 50 distinct classes. Thus, a naive baseline classifier making random guesses would have a classification error of 98%, which is significantly higher than for our approach. The median error of Random Forest on the original dataset is larger than the corresponding value for all Sp-RST settings. Considering averages, only 1000 partitions and 2000 partitions are slightly higher than Random Forest without Sp-RST. We perform statistical tests as described previously and find that the error difference by Sp-RST with 1330, 1500, and 2500 partitions is statistically significant at confidence level 0.05 while for the other settings no statistically significant difference could be found. We, therefore, conclude that Sp-RST introduces no significant information loss as results are at least comparable.

E. Execution Times With and Without Sp-RST

Table III shows execution times for Sp-RST and Random Forest for different numbers of partitions as well as the number of selected features in the considered cases. We see that the overall execution time is decreasing for increasing number of partitions (with the exception of 1200 and 1250 where it should be noted that the number of features selected by the latter is significantly larger, resulting in larger times for Random Forest). For 2000 and 2500 partitions the overall execution time is smaller than the execution time of Random Forest without Sp-RST on the original data set. Together with the analysis of the classification error this clearly demonstrates the strength of our proposed method.

We remark that Sp-RST can easily be further parallelized by executing the n iterations in Algorithm 1 in parallel. Assuming we are able to use n nodes, the times for Sp-RST can be reduced significantly and the overall execution times for Sp-RST and Random Forest become smaller in the majority of settings (see Table III; exceptions are the smallest number of partitions and 1250 which was already pointed out as an outlier with respect to the number of features selected).

Partitions	1000	1200	1250	1330	1500	2000	2500
# of selected features	[6131, 6197]	[3644, 3735]	[5549, 5598]	[2490, 2514]	[1616, 1691]	[4093, 4134]	[3178, 3245]
pairwise overlap	[5508, 5561]	[2129, 2221]	[4899, 4959]	[1049, 1097]	[466, 533]	[3501, 3566]	[2477, 2522]
features never selected	23.93%	34.83%	29.91%	43.41%	54.28%	46.01%	53.44%
features always selected	49.42%	5.61%	43.8%	1.08%	0.12%	28.9%	17.09%
share of features in the returned set that were selected in all runs	[79.75%80.61]	[15.02%15.40]	[78.24%78.93]	[4.30%4.33]	[0.71%0.74]	[69.90%70.61]	[52.67%53.78]

TABLE I

COMPARISON OF THE SETS OF SELECTED FEATURES IN DIFFERENT SETTINGS. [] INDICATES MINIMAL AND MAXIMAL VALUES OBSERVED.

Partitions	1	1000	1200	1250	1330	1500	2000	2500
Average	**50.50%**	51.00%	**49.69%**	**49.25%**	**47.14%**	**42.18%**	51.44%	**45.16%**
Median	**52.53%**	**51.38%**	**50.77%**	**49.83%**	**46.59%**	**41.51%**	**51.51%**	**45.40%**
Standard deviation	11.09%	11.13%	9.98%	9.79%	11.77%	11.65%	10.71%	10.36%
p-value (comparison with '1')	-	0.4897	0.1852	0.1945	**0.009471**	**7.221e-07**	0.6261	**0.000485**

TABLE II

AVERAGE, MEDIAN AND STANDARD DEVIATION OF THE CLASSIFICATION ERROR OVER 100 INDEPENDENT RUNS OF RANDOM FOREST DEPENDING ON THE NUMBER OF PARTITIONS (ROUNDED TO 2 DECIMAL PLACES). THE CASE OF '1 PARTITION' CORRESPONDS TO THE ORIGINAL DATASET WITHOUT PERFORMING SP-RST.

Partitions	1	1000	1200	1250	1330	1500	2000	2500
Sp-RST (average per iteration)	-	4448	886	936	451	210	72	52
Sp-RST	-	44481	8861	9359	4514	2099	723	520
Random Forest	1811	1309	837	1200	648	325	895	717
Total Time	**1811**	45790	9698	10559	5161	2424	**1618**	**1237**
Total Time (based on average)	**1811**	5757	**1723**	2136	**1099**	**535**	**967**	**769**
# of selected Features	10000	6184	3665	5556	2490	1647	4129	3209

TABLE III

EXECUTION TIMES ON A SINGLE NODE DEPENDING ON THE NUMBER OF PARTITIONS (IN SECONDS, ROUNDED). THE CASE OF '1 PARTITION' CORRESPONDS TO THE ORIGINAL DATASET WITHOUT PERFORMING SP-RST.

VI. CONCLUSION AND EMERGING TRENDS

We have presented a novel efficient distributed algorithm based on Rough Set Theory for large-scale data pre-processing under the Spark framework. To reduce the computational effort of the rough set computations, our approach splits the given dataset into partitions with smaller numbers of features which are then processed in parallel. We have demonstrated its effectiveness using the Amazon Commerce reviews data set from the UCI machine learning repository, a dataset with 10 000 features and 1 500 data items equally spread over 50 classes. A detailed experimentation reveals that our proposed Sp-RST method achieves a good speedup, but in order to also achieve good sizeup a large number of partitions is necessary. Indeed, our method performs feature selection without any significant information loss while decreasing the run time of the learning process. Our study provides many ideas for future research directions with particular focus on further analysis of the sizeup and the parameterization of the algorithm. Moreover, tests on other real-world applications will demonstrate the wider applicability of our method.

ACKNOWLEDGMENT

This work is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 702527. The authors would additionally like to thank Beck's main supervisor Hanene Azzag.

REFERENCES

[1] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.

[2] W. Fan and A. Bifet, "Mining big data: current status, and forecast to the future," *ACM sIGKDD Explorations Newsletter*, vol. 14, no. 2, pp. 1–5, 2013.

[3] Z. Pawlak and A. Skowron, "Rudiments of rough sets," *Information sciences*, vol. 177, no. 1, pp. 3–27, 2007.

[4] Z. Pawlak, *Rough sets: Theoretical aspects of reasoning about data*. Springer Science & Business Media, 2012, vol. 9.

[5] K. Thangavel and A. Pethalakshmi, "Dimensionality reduction based on rough set theory: A review," *Applied Soft Computing*, vol. 9, no. 1, pp. 1–12, 2009.

[6] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[7] A. Asuncion and D. Newman, "Uci machine learning repository," 2007.

[8] J. G. Shanahan and L. Dai, "Large scale distributed data science using apache spark," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 2323–2324.

[9] X. Xu, J. Jäger, and H.-P. Kriegel, "A fast parallel clustering algorithm for large spatial databases," in *High Performance Data Mining*. Springer, 1999, pp. 263–290.