Aberystwyth University

A Distributed Rough Set Theory Algorithm based on Locality Sensitive Hashing for an Efficient Big Data Pre-processing

Chelly Dagdia, Zaineb; Zarges, Christine; Beck, Gaël; Azzag, Hanene; Lebbah, Mustapha

Published in:
2018 IEEE International Conference on BIG DATA

Publication date:
2018

Citation for published version (APA):
Chelly Dagdia, Z., Zarges, C., Beck, G., Azzag, H., & Lebbah, M. (2018). 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* IEEE Press.

General rights
Copyright and moral rights for the publications made accessible in the Aberystwyth Research Portal (the Institutional Repository) are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the Aberystwyth Research Portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the Aberystwyth Research Portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

tel: +44 1970 62 2400
email: is@aber.ac.uk

Download date: 27. Apr. 2024

A Distributed Rough Set Theory Algorithm based on Locality Sensitive Hashing for an Efficient Big Data Pre-processing

Zaineb Chelly Dagdia*†, Christine Zarges*, Gaël Beck‡, Hanene Azzag‡ 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*—A big challenge in the knowledge discovery process is to perform big data pre-processing; specifically feature selection. To handle this challenge, Rough Set Theory (RST) has been considered as one of the most powerful techniques as it has much to offer for feature selection. To extend its applicability to big data, a distributed version of RST was developed. However, one of its key challenges is the partitioning of the feature search space in the distributed environment while guaranteeing data dependency. In this paper, we propose a new distributed version of RST based on Locality Sensitive Hashing (LSH), named LSH-dRST, for big data pre-processing. LSH-dRST uses LSH to match similar features into the same bucket and maps the generated buckets into partitions to enable the splitting of the universe in a more appropriate way. We compare LSH-dRST to the standard distributed RST technique which is based on a random partitioning of the universe and demonstrate that our LSH-dRST is not only scalable but also more reliable for feature selection; making it more relevant to big data pre-processing. We also demonstrate that our LSH-dRST ensures the partitioning of the high dimensional feature search space in a more reliable way. Hence, guarantees data dependency in the distributed environment, and ensures a lower computational cost.**

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

I. Introduction

Big data is extremely valuable and open the gates to ample opportunities for large progress in a number of diverse real-world problems [1]. However, big data arises with many challenges most importantly in dimensionality reduction that comprises feature extraction and feature selection. The former category transforms the original meaning of the features while the latter presents semantic-preserving techniques that attempt to retain the meaning of the original feature set. Within this category a further partitioning can be defined where the techniques are classifies into filter approaches and wrapper approaches. The main difference between the two branches is that wrapper approaches include a learning algorithm in the feature subset evaluation. Although wrappers may produce better results than filters, they are computationally expensive and can break down with a very large number of features. On the other hand, filters tend to be applicable to most domains as they are not tied to any particular induction algorithm and are usually a good option for large-scaled data sets with a large number of features. With the aim of choosing the most relevant and pertinent subset of features, a variety of dimensionality reduction techniques were proposed within the Apache Spark framework[1] to deal with big data in a distributed way. Among these are several feature extraction methods such as nn-gram, Principal Component Analysis, Discrete Cosine Transform, etc., and very few feature selection techniques which are the VectorSlicer, the RFormula and the ChiSqSelector. To further expand this line of research, i. e., the development of parallel feature selection methods, some other feature selection techniques which are based on evolutionary algorithms were proposed recently[2] [2]. These include a generic implementation of greedy information theoretic feature selection methods[3], and an improved implementation of the classical minimum Redundancy and Maximum Relevance feature selection method [3]. This implementation includes several optimizations such as cache marginal probabilities, accumulation of redundancy (greedy approach) and a data-access by columns[4]. However, most of these techniques suffer from some limitations as they involve the user for parameterization, require noise levels to be specified, rank features leaving the user to choose its own subset, require the user to state how many features are to be chosen, or they must supply a threshold that determines when the algorithm should terminate. All of these require the user to make a decision based on its own (possibly faulty) judgment. To overcome these shortcomings, the need for a filter method that does not require any additional information to function properly seems essential. Rough Set Theory (RST) can be used as such a technique [4].

The use of RST for feature selection has proved more efficient in comparison to a variety of state-of-the-art feature selection methods [4]. Over the past years, RST has become a topic of great interest to researchers and has been applied to many domains such as in classification [5] and clustering [6]. This success is due to many aspects of the theory, among them the possibility to analyze the facts hidden in data, it does not require any additional information about the data and it is able to find a minimal knowledge representation [7]. This is achieved by making use of the granularity structure

[1]https://spark.apache.org/docs/2.2.0/ml-features.html
[2]https://github.com/triguero/MR-EFS
[3]https://github.com/sramirez/spark-infotheoretic-feature-selection
[4]https://github.com/sramirez/fast-mRMR

of the provided data only. Although RST has been widely used as a powerful filter feature selection technique, most of the traditional rough set based algorithms are sequential algorithms, computationally expensive and can only deal with non-large data sets. The prohibitive complexity of RST comes from the search for an optimal feature subset through the competing of an exponential number of candidate subsets. Although it is an exhaustive method, this is quite impractical for big data as it becomes unmanageable to generate the set of all possible feature combinations. Therefore, to avoid this prohibitive complexity, recently in [8] a distributed version of RST was proposed and is named Sp-RST. To perform feature selection in the context of big data, Sp-RST partitions the feature search space in a random way where each partition holds a random set of features. Each partition is dealt with separately in the distributed environment so that at the end of the feature selection process all the selected features from each partition are gathered together to generate the final reduced set of features. Eventually, in such an implementation design, it is very likely that similar features will belong to different partitions and hence a cut in data dependency will occur. It is very important to highlight that data dependency is a key issue in a distributed environment and in parallel computing. Nevertheless, based on the Sp-RST architecture, data dependency will not be assured as the algorithm uses an arbitrary procedure in partitioning the feature search space. Hence, in this paper, we propose a novel efficient distributed rough set theory version based on a hashing technique. Specifically, our proposed solution is based on the Locality Sensitive Hashing (LSH) [9] algorithm for large-scale data pre-processing.

Many hashing algorithms have been proposed in literature [9]–[11], and among these LSH is considered as the most representative and popular one. LSH is presented as a probabilistic similarity-preserving dimensionality reduction method, and based on the adopted distances and similarities, including l_p distance [12], angular distance [13], Hamming distance [14], Jaccard coefficient [15], etc., different types of LSH can be designed; which also depends on the types of the used data [9]. Many variants are developed based on these basic LSH families such as Spectral hashing [11], Kernelized spectral hashing [16], and independent component analysis (ICA) Hashing [17]. These methods aim at learning the hash functions for better fitting the data distribution [18].

The main motivation of choosing LSH among other hashing techniques is that LSH is often taken as the baseline and is widely used in industry, for image recognition, clustering, and for some other tasks as well; but specifically in database systems for high dimensional similarity search. The choice of LSH is also argued by its advantages in comparison to other hashing techniques when dealing with high dimensional data sets as presented in the detailed study conducted in [19].

Our proposed solution, dubbed LSH-dRST, uses LSH which maps similar data instances based on their feature values into the same bucket in low dimensional cases and based on this process, LSH-dRST uses the generated buckets to partition the feature search space in a more reliable way and hence guaranteeing data dependency and a lower computational cost.

The rest of this paper is structured as follows. Section II presents preliminaries about the Locality Sensitive Hashing and Rough Set Theory for feature selection. Section III formalizes the motivation of this work and introduces our novel distributed LSH-dRST algorithm for large-scale data pre-processing. The experimental setup is introduced in Section IV. The results of the performance analysis are given in Section V and the conclusion is given in Section VI.

II. Preliminaries

A. Locality Sensitive Hashing

The Locality Sensitive Hashing (LSH) algorithm [9] was introduced as a probabilistic technique suitable for solving the approximate K nearest neighbors (K-NN) problem in a high dimensional space. It is based on the definition of an LSH family ($\mathcal{H}$), a family of hash functions mapping similar input items to the same hash code with higher probability than dissimilar items. Formally, an LSH family is defined as follows: Let $\mathcal{H}$ be a family of hash functions such that $h \in \mathcal{H} : \mathbb{R}^d \rightarrow \mathcal{U}$. Consider a function h that is chosen uniformly at random from $\mathcal{H}$ and a similarity function $sim\colon \mathbb{R}^d \times \mathbb{R}^d \rightarrow [0, 1]$. The family $\mathcal{H}$ is called locality sensitive if for any vectors u, $v \in \mathbb{R}^d$, it satisfies the property: $P(h(u) = h(v)) = sim(u, v)$

That is, the more similar a pair of vectors is, the higher the collision probability is. The LSH scheme indexes all items in hash tables and searches for near items via hash table lookup. Formally, for an integer k, we define a function family $\mathcal{G} = \{g : \mathcal{R}^d \rightarrow \mathcal{U}^k\}$ such that $g(v) = (h_1(v), \ldots, h_k(v))$ where $h_i \in \mathcal{H}$, i. e., g is the concatenation of k LSH functions. For an integer ℓ, we choose ℓ functions $\mathcal{G} = \{g_1, \ldots, g_\ell\}$ from $\mathcal{G}$ independently and uniformly at random. Each g_i, $1 \leq i \leq \ell$ effectively constructs a hash table denoted by D_{g_i}. The hash table is a data structure that is composed of buckets, each of which is indexed by a hash code. A bucket in D_{g_i} stores all $v \in V$ that have the same g_i values. For space, only the nonempty buckets are retained using standard hashing. $\mathcal{G}$ defines a collection of ℓ tables $I_G = \{D_{g_1}, \ldots, D_{g_\ell}\}$ and we call it an LSH index.

As previously mentioned, there are different kinds of LSH families for different (dis)similarity measures including Hamming distance, Jaccard similarity, and cosine similarity. In this paper, we rely on the LSH scheme that supports the p-stable similarity.

B. Rough Sets for Feature Selection

Rough Set Theory (RST) [20] is a formal approximation of the conventional set theory which supports approximations in decision making. The fundamentals of RST for feature selection are as follows:

1) Basic concepts: In RST, 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.

2) Reduction process: 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. LSH-dRST: The proposed Solution

In this section, we present our proposed solution, dubbed "LSH-dRST" which is characterized by its distributed implementation design with respect to the Spark framework for a parallel and in-memory processing task. First, we will highlight the motivation behind the development of our proposed LSH-dRST solution by pointing out the limitations of the standard distributed RST. Secondly, we will elucidate our LSH-dRST solution as an efficient approach capable of performing big data feature selection in a more intelligent and convenient manner without any significant information loss.

A. Motivation and Problem Statement

As previously explained in section II-B, to perform feature selection, RST has to calculate the dependency of attributes $\gamma(C, c_i)$. To do so and as a first step, the indiscernibility relation $IND(P)$ for all the features has to be calculated. Let us recall that $IND(P)$ looks for similar feature values and gathers them together to form the set of the equivalence relations. Based on these fundamental RST concepts, it is crucial to guarantee data dependency in order to set the most reliable equivalence relations and hence to guarantee the most representative reduct set. However, data dependency is a key issue in a distributed environment and in parallel computing. The standard distributed RST version proposed in [8] partitions the feature search space in a random manner which does not assure data dependency. More precisely, the work in [8] partitions the information table T into m data blocks T_i based on splits from the conditional attribute set C in a way that: $T = \bigcup_{i=1}^{m}(C_r)T_i$; where $r \in \{1, \ldots, V\}$. r is a user-defined parameter of the algorithm referring to the number of features that will be considered to constitute each T_i data block and V is the total number of features. Indeed, each T_i is constructed based on r random features selected from the conditional attribute set C. Subsequently, each partition is handled by a different machine (node) in the distributed environment so that at the end all the intermediate results will be gathered from the different m partitions. In such architecture, it is likely that similar features will belong to different partitions T_i and hence a false estimation of the constructed $IND(P)$ is more likely to occur. Consequently, such a random process may mislead the RST feature selection process by generating a non-relevant reduct. These are the main motivations for our proposed LSH-dRST solution that makes use of the locality sensitive hashing algorithm. Using LSH can guarantee the process of gathering similar or close data instances based on their feature values into the same bucket and by using the generated buckets a more intelligent partitioning of the universe can be applied. In such a way, we can guarantee that LSH-dRST preserves data dependency within the same buckets and hence solving the limitations of the standard distributed RST.

B. LSH-dRST

Technically, to deal with high dimensional data sets and to make use of the LSH technique, within a distributed environment, we first generate the appropriate buckets based on LSH, map them into partitions, partition the entire rough set feature selection process into elementary tasks, each executed independently on each generated bucket, and then conquer the intermediate results to finally acquire the ultimate output; the reduct set.

1) General Model Formalization: For feature selection, our learning problem has to select high discriminating features from the original high dimensional input database which corresponds to the data stored in the given Distributed File System (DFS). To operate on the given DFS, a Resilient Distributed Data set (RDD) is created. We may formalize the latter as a given information table defined as T_{RDD}, where universe $U = \{x_1, \dots, x_N\}$ is the set of data items, the conditional attribute set $C = \{c_1, \dots, c_V\}$ contains every single feature of the T_{RDD} information table and the decision attribute D of our learning problem corresponds to the class (label) of each T_{RDD} sample and is defined as $D = \{d_1, \dots, d_W\}$. The conditional attribute set C presents the pool from where the most convenient features will be selected.

Algorithm 1 LSH-dRST

```
     Inputs: $T_{RDD}$: information table with $D$ as decision class,
     $K$: number of nearest neighbors, $B$: number of buckets
     Output: $Reduct$
 1:  Generate the $B$ LSH buckets
 2:  Calculate $IND(D)$
 3:  for each $T_{RDD_{(b)}}$ where $b \in [1, \dots, B]$ do
 4:      Generate the set $S$ sub-information tables $Cl$ based on $K$
 5:      for each $Cl_s(K)$ where $s \in [1, \dots, S]$ do
 6:          Generate $AllComb_{(K)}$, Calculate $IND(AllComb_{(K)})$
 7:          Calculate $DEP(AllComb_{(K)})$, Select $DEP_{max}(AllComb_{(K)})$
 8:          Filter $DEP_{max}(AllComb_{(K)})$, Filter $NbF_{min}(DEP_{max}(AllComb_{(K)}))$
 9:      end for
10:  end for
11:  for each $T_{RDD_{(b)}}$ do
12:      for each $Cl_s(K)$ output do
13:          $Reduct = \bigcup_{b=1}^{B} \bigcup_{s=1}^{S} RED_s$
14:      end for
15:  end for
16:  return ($Reduct$)
```

In order to make our algorithm scalable with the high number of features and with respect to data dependency, we partition the given T_{RDD} information table into B data blocks based on the B generated LSH buckets. The buckets are splits from the conditional attribute set C and each bucket covers a specific feature space enclosing all similar and close instances based on their feature values. Hence, $T_{RDD} = \bigcup_{b=1}^{B} (C_h) T_{RDD_{(b)}}$; where $h \in \{1, \dots, V\}$. h is a value generated by LSH referring to the number of features per bucket that will be considered to create each $T_{RDD_{(b)}}$ data block; and is equal to the size of the feature space C divided by B. Once the buckets are defined, each $T_{RDD_{(b)}}$ is divided into S automatically created sub-information tables Cl based on the K nearest neighbors approach; where K refers to the number of features per sub-information table and on which LSH-dRST will be applied. Hence, $T_{RDD_{(b)}} = \bigcup_{s=1}^{S} Cl_s(K)$; where $S = C_h/K$.

To ensure scalability, rather than applying LSH-dRST to T_{RDD} including the whole conditional feature set C the distributed algorithm will be applied to every single $Cl_s(K)$, where $s \in \{1, \dots, S\}$, so that at the end all the intermediate results will be gathered from the different Cl sub-information tables of every $T_{RDD_{(b)}}$ partition. In such a way, we can guarantee that LSH-dRST can be applied to a computable number of features while preserving data dependency and hence solving the standard distributed RST limitations. Algorithm 1 highlights the pseudo-code of our proposed LSH-dRST solution.

More precisely, the algorithm will first generate the B partitions using LSH, $T_{RDD_{(b)}}$ while preserving data dependency as previously highlighted. Then, for each partition, the Cl sub-information table will be created in a way that the K nearest neighbors from any data point within the $T_{RDD_{(b)}}$ feature search space form a sub-information table $Cl_s(K)$ (Algorithm 1, line 4). Through all the S Cl sub-information tables, the distributed LSH-dRST tasks, line 5 to 9, will be executed. As seen in Algorithm 1, line 2 is executed out of the $T_{RDD_{(b)}}$ and the $Cl_s(K)$ iteration loops. The main reason for this implementation is that this task deals with the calculation of the indiscernibility relation of the decision class $IND(D)$. This task is independent from the B generated partitions as the result depends on the data items class and not on the features. Out from the iteration loops, line 10, the output of each $Cl_s(K)$ is either a single reduct $RED_{s_{(D)}}(K)$ or a family of reducts $RED^F_{s_{(D)}}(K)$. Based on the RST preliminaries previously mentioned in Section II-B, any reduct of $RED^F_{s_{(D)}}(K)$ can be used to represent the $Cl_s(K)$ sub-information table.

Consequently, if LSH-dRST generates only one reduct, for a specific $Cl_s(K)$ sub-information table, then the output of this feature selection phase is the set of the $RED_{s_{(D)}}(K)$ features. These features reflect the most informative ones among the K attributes of $Cl_s(K)$ resulting in a new reduced $Cl_s(K)$, $Cl_s(RED)$, which preserves nearly the same data quality as its corresponding $Cl_s(K)$ which is based on the whole feature set K. On the other hand, if LSH-dRST generates a family of reducts then the algorithm randomly selects one reduct among $RED^F_{s_{(D)}}(K)$ to represent the corresponding $Cl_s(K)$. This random choice is argued by the same priority of all the reducts in $RED^F_{s_{(D)}}(K)$. In other words, any reduct included in $RED^F_{s_{(D)}}(K)$ can be used to replace the K attributes of $Cl_s(K)$. At this stage, each Cl_s sub-information table has its output $RED_{s_{(D)}}(K)$ referring to the selected features. However, since each Cl_s is based on distinct features within different $T_{RDD_{(b)}}$ feature search spaces and with respect to $T_{RDD_{(b)}} = \bigcup_{s=1}^{S} Cl_s(K)$ a union of the generated selected features is required to represent the initial T_{RDD}; defined as $Reduct = \bigcup_{b=1}^{B} \bigcup_{s=1}^{S} RED_s$ (Algorithm 1, line 11 to 15).

By removing irrelevant and redundant features, LSH-dRST can reduce the dimensionality of the data from $T_{RDD}(C)$ to $T_{RDD}(Reduct)$. In what follows, we will elucidate the different LSH-dRST elementary distributed tasks.

2) Algorithmic Details: As previously highlighted, the elementary feature selection LSH-dRST distributed tasks will be executed on every $Cl_s(K)$ sub-information table defined by its K features along the $T_{RDD_{(b)}}$ partitions; except for task 2 in Algorithm 1 dealing with $IND(D)$. The algorithm goes through 10 main jobs in order to generate the final sought $Reduct$. The first step, is to apply LSH to generate the B buckets based on a hash table as explained in Section II-A. To do so, LSH-dRST creates the hash table based on a set of random vectors following a Gaussian distribution (referred as the $\mathcal{H}$ family of hash functions). The constructed hash table is based on the size of the features of T_{RDD}. After that, the algorithm maps the T_{RDD} to work on each partition separately and on each partition it applies a projection for each vector based on the set of the T_{RDD} mapped feature vectors. As a result the buckets are automatically created; each with a specific index (referred as a hash code). Finally, LSH-dRST performs a sort action to order the buckets with respect to the given number of buckets B. The pseudo-code of this distributed task is given in Algorithm 2.

Algorithm 2 Generate $Buckets(B)$

```
    Inputs: $T_{RDD}$, $B$
    Output: $T_{RDD_{(b)}}$
1:  Generate the hash table based on the $T_{RDD}$ feature size
2:  Map the $T_{RDD}$
3:  Apply the LSH projection for each vector
4:  Sort the buckets by $B$
```

After that, LSH-dRST has to compute the indiscernibility relation for the decision class $D = \{d_1, \dots, d_W\}$; defined as $IND(D)$: $IND(d_i)$. More precisely, LSH-dRST 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 $U = \{x_1, \dots, x_N\}$ and which belong to the same class d_i. To do so, LSH-dRST processes a $foldByKey$[5] 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. The set of gathered data items is only kept as it represents $IND(D)$: $IND(d_i)$. The pseudo-code related to this distributed job is highlighted in Algorithm 3.

Algorithm 3 Calculate $IND(D)$

```
    Input: $T_{RDD}$
    Output: $IND(D)$: $Array[Array[x_i]]$
1:  $IND(d_i) =$
    Map the $T_{RDD}$
    Perform a foldByKey operation based on $\langle d_i, id_i, x_i \rangle$
2:  Map the result of step (1) to keep $x_i$ as follows:
    $IND(d_i).map\{case(d_i, x_i) => x_i\}.collect$
```

At its third step, LSH-dRST has to generate the set S of the $Cl(K)$ sub-information tables based on the number of features K. Let us recall at this stage that LSH gathered all the similar features already within a same specific bucket. On these similar attributes, a further partitioning is required to generate the sub-information tables that can be handled by the LSH-dRST algorithm. As mentioned in Section I, the standard rough set theory has to generate all the combinations of features at once, process them in turn to finally generate the reduct. As it is infeasible to generate all the combinations of features within the big data context, then the distributed LSH-dRST will operate on the sub-information tables constructed on K number of features; where K is a manageable size that can be handled by the algorithm. Therefore and to achieve this distributed task, for every bucket $T_{RDD_{(b)}}$, LSH-dRST performs a $mapPartitionsWithIndex$[6] operation using the buckets indexes; the already generated hash codes in Algorithm 2. Then, the later result is mapped; where on each partition, the K nearest features to a randomly chosen attribute within the same $T_{RDD_{(b)}}$ hash code, are selected to form a sub-information table. The pseudo-code related to this distributed job is highlighted in Algorithm 4.

Algorithm 4 Generate $Cl(K)$

```
    Inputs: $T_{RDD_{(b)}}$, $K$
    Outputs: $S$, $Cl(K)$
1:  Perform a $mapPartitionsWithIndex$ on every $T_{RDD_{(b)}}$
    using its index
2:  Map the result of step (1)
    Perform a KNN by looking for the K nearest features
    within a randomly selected attribute within each $T_{RDD_{(b)}}$
3:  Generate the set $S$ of the $Cl(K)$ sub-information tables
```

Once the set S of the $Cl(K)$ sub-information tables is generated for all the $T_{RDD_{(b)}}$, LSH-dRST has to perform feature selection for each single $Cl_s(K)$. To do so, first, the algorithm has to create all the possible combinations of the K set of feature, $AllComb_{(K)}$, using the $flatMap$[7] spark function. Then, the fifth LSH-dRST job deals with the indiscernibility relation computation for every previously generated combination. As presented in Algorithm 5, LSH-dRST aims at grouping all the data items identifiers id_i sharing the same specific combination of features extracted from $AllComb_{(K)}$. In order to achieve this, we use the $foldByKey$ spark operation where the combination of features defines the key and the id_i mapped as value.

At this phase, LSH-dRST prepares the set of features that will be selected in the coming steps. In Algorithm 6, the dependency degrees $\gamma(K, AllComb_{(K)})$ of each feature combination are computed. To do so, the calculated indiscernibility

[5]https://spark.apache.org/docs/0.7.2/api/core/spark/PairRDDFunctions.html

[6]https://spark.apache.org/docs/1.6.2/api/java/org/apache/spark/rdd/RDD.html#mapPartitionsWithIndex(scala.Function2,%20boolean,%20scala.reflect.ClassTag)

[7]https://spark.apache.org/docs/latest/rdd-programming-guide.html

Algorithm 5 Calculate $IND(AllComb_{(K)})$

```
    Inputs: $Cl_s(K)$, $AllComb_{(K)}$
    Output: $IND(AllComb_{(K)}) : Array[Array[id_i]]$
1:  $IND(AllComb_{(K)})$ =
    Map the $Cl_s(K)$ sub-information table
    Perfom a $foldByKey$ operation based on $\langle AllComb_{(K)}, id_i \rangle$
2:  Map the result of step (1) to keep $id_i$ as follows:
    $IND(AllComb_{(K)}).map\{case(ListValues, id_i) => id_i\}.collect$
```

relations $IND(D)$ and $IND(AllComb_{(K)})$ as well as the set of all feature combinations $AllComb_{(K)}$ are required. The task is to test, first, if the intersection of every $IND(d_i)$ with each $IND(AllComb_{(K)})$ keeps all the latter elements; referring to the lower approximation. If so then a score which is equal to the length of $IND(AllComb_{(K)})$ 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 $\gamma(K, AllComb_{(K)})$.

Algorithm 6 Generate $DEP(AllComb_{(K)})$

```
    Inputs: $AllComb_{(K)}$, $IND(D)$, $IND(AllComb_{(K)})$
    Outputs: $\gamma(K, AllComb_{(K)})$, $Size_{(AllComb_{(K)})}$
1:  for i := $AllComb_{(K)}$ do
2:      for j := $IND(D)$ do
3:          for v := $IND(AllComb_{(K)})$ do
4:              if j.intersect(v).length == v.length then v.length
5:              else 0
6:          Reduce(_ + _)
7:      Reduce(_ + _)
```

The output of this step is the set of dependency degrees $\gamma(K, AllComb_{(K)})$ of the feature combinations $AllComb_{(K)}$ and their associated sizes $Size_{(AllComb_{(K)})}$. Then, LSH-dRST looks for the maximum dependency value $DEP_{max}(AllComb_{(K)})$ among all $\gamma(K, AllComb_{(K)})$ using the max function operated on the given RDD. The output $MaxDependency$ reflects in one hand the dependency of the whole feature set K representing the $Cl_s(K)$ and on the other hand the dependency of all the possible feature combinations satisfying the constraint $\gamma(K, AllComb_{(K)}) = \gamma(K)$. $MaxDependency$ is the baseline value for feature selection. Once the $MaxDependency$ is generated, LSH-dRST keeps the set of all combinations having the same dependency degrees as $MaxDependency$; i. e., $\gamma(K, AllComb_{(K)}) = MaxDependency$. This is achieved by applying a $filter$ function. In fact, at this stage LSH-dRST 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, LSH-dRST keeps the set of combinations having the minimum number of features, $Size_{(AllComb_{(K)})}$, by applying a $filter$ operation ($NbF_{min}(DEP_{max}(AllComb_{(K)}))$) and by satisfying the full reduct constraints discussed in Section II-B; $\gamma(K, AllComb_{(K)}) = \gamma(K)$ while there is no $AllComb'_{(K)} \subset AllComb_{(K)}$ such that $\gamma(K, AllComb'_{(K)}) = \gamma(K, AllComb_{(K)})$. Each combination satisfying this condition is considered as a viable minimum reduct set. The attributes of the reduct set describe all concepts in the sub-information table $Cl_s(K)$.

IV. Experimental Setup

A. Used Benchmark

To validate the efficiency of the LSH-dRST algorithm we require a data set with a large number of attributes as the advantage of the data partitioning scheme and the fact of looking on data dependencies via LSH will become more pronounced for data sets with a large set of features. Therefore, and for the sake of comparisons with the standard distributed RST version [8], we chose the Amazon Commerce reviews data set from the UCI machine learning repository [21]. This choice is based on the fact that this data set was the one with the largest number of features that still had a sufficiently large number of data items and as it was used in [8]. 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. The database includes 1 500 data items described through 10 000 features (linguistic style such punctuation, length of words, sentences, etc.) and 50 distinct classes (authors). Instances are identically distributed across the different classes, i. e., for each class there are 30 items.

B. Experimental plan, testbed and tools

The main aim of our experimentation is to demonstrate that our proposed approach LSH-dRST preserves data dependency within the same generated buckets and within the distributed environment. We will show that by using a more intelligent partitioning of the universe, via the use of LSH, a more reliable process of gathering similar data instances based on their feature values can be reached; and hence better classification results can be obtained. Indeed, we will show that LSH-dRST is not only scalable but also more reliable for feature selection; making it more relevant to big data pre-processing. We, therefore, investigate different parameters of LSH-dRST and analyze how these affect execution time and stability of the feature selection; hence data dependency. We then show that the improvement in performance does not decrease the feature selection ability by using a Random Forest classifier on the original data set, the reduced data sets produced by LSH-dRST and some other feature selection techniques as described below. We use the scikit-learn Random Forest implementation[8] with the following parameters: n_estimators = 1000,

[8]http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

n_jobs $= -1$, and oob_score $=$ True. A Stratified 10-Folds cross-validator[9] is used for all our conducted experiments.

Based on the conducted experiments in [8], a maximum of 10 features per sub-information table Cl is used that can be processed by LSH-dRST. We therefore perform experiments for 2, 5, 10, 25 and 50 buckets (B), in Algorithm 1, each comprising sub-information tables of 4, 5, 8 and 10 features (F); where F refers to the K parameter in Algorithm 1. For instance, for bucket $(B = 2)$ and for a number of 4 features $(F = 4)$ per sub-information table the algorithm generates 1250 Cl. We run all settings on 1, 2, 4, 8, and 16 nodes on the Grid5000 testbed[10] which is a French national large-scale and versatile platform. Our analysis first focuses on the scalability of the algorithm. We evaluate the performance of LSH-dRST using the *speedup*, *sizeup*, and *scaleup* criteria introduced in [22] and define the combined runtime for LSH and dRST as the execution time of our method.

We perform model evaluation using a Random Forest classifier to evaluate the quality of our feature selection and compare it with other common techniques, namely the Sum Squares Ratio as implemented in Smile[11] as well as Information Gain, Gain Ratio and Chi Squared as provided in Weka 3.6.15[12]. For the Sum Squares Ratio, we set the number of features to be selected to a value comparable with LSH-dRST, i. e., the average number of features selected for each parameter setting of F. All other methods were run with 'Ranker' as search method and a threshold of 0. We determine the sets of features selected by these methods and then perform 10 runs of the above Random Forest implementation on the new feature set. We use the standard measures which are the precision, the recall, the accuracy and the F1 score to report our results.

LSH-dRST makes use of randomisation in several places, e. g., LSH uses random projections, the construction of the sub-information tables starts with a randomly selected feature, and we select one reduct among the generated family of reducts randomly. For this reason, we always perform multiple runs of the algorithm and report appropriate statistics.

V. Results and Analysis

In this section, we will discuss our results. We first consider the scalability of LSH-dRST. We then investigate the quality of the feature selection and compare our results with the previously introduced algorithm creating random partitions (Sp-RST).

A. Scalability

1) Speedup Analysis: We first consider the speedup of LSH-dRST: We keep the size of the data set constant (where size is measured by the number of features, i. e., 10 000 features in our case) and increase the number of nodes. The speedup of a system with m nodes is defined as [22]:

$$\text{Speedup(m)} = \frac{\text{runtime on one node}}{\text{runtime on } m \text{ nodes}}$$

We plot the speedup in Figure 1 and see that the speedup for most parameter settings is very similar. However, setting $F = 8$ improves the speedup considerably—independently of the setting for B (where $B = 5$ and $B = 10$ yield the best overall results). Overall, we conclude that the number of buckets does not have a significant influence on the speedup, but the number of sub-information tables F does. The latter is expected since the execution time grows exponentially in the number of features and thus, using more nodes is more beneficial in cases with many features. It is therefore somewhat surprising that $F = 10$ does not exhibit a larger speedup. Note that an ideal parallel algorithm has linear speedup, which is, however, difficult to achieve in practice due to startup and communication cost as well as interference and skew [22].

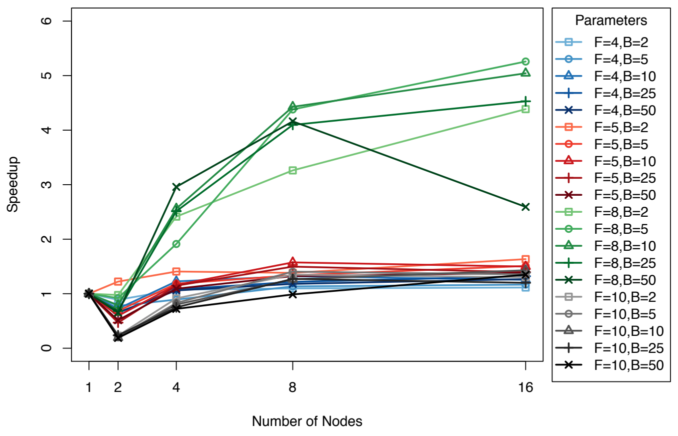


Fig. 1. Speedup for Amazon with 10 000 features for 16 nodes.

2) Sizeup Analysis: The sizeup keeps the number of nodes constant and measures how much the execution time increases

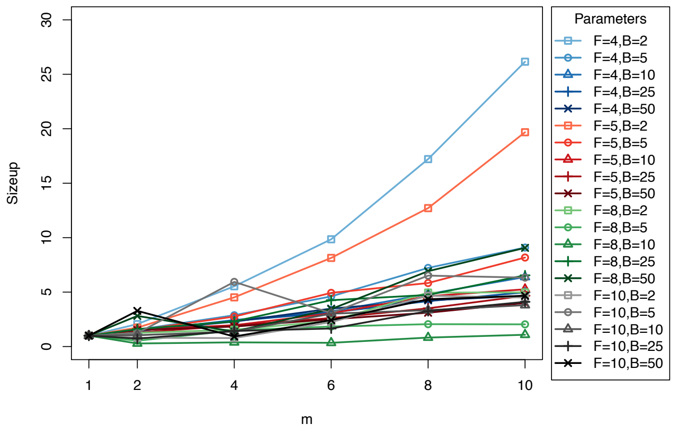


Fig. 2. Sizeup for Amazon with 10 000 features for 16 nodes.

[9]http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

[10]https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home

[11]https://haifengl.github.io/smile/feature.html

[12]https://www.cs.waikato.ac.nz/~ml/weka: classes ChiSquaredAttributeEval, GainRatioAttributeEval, InfoGainAttributeEval

as the data set is increased by a factor of m [22]:

$$\text{Sizeup(m)} = \frac{\text{runtime for data set of size } m \cdot s}{\text{runtime for baseline data set of size } s}$$

To measure the sizeup we have created smaller databases by selecting random features from the original Amazon 10 000 database. We use 1 000 features as a baseline and consider 2 000, 4 000, 6 000, 8 000, and 10 000 features, respectively. We plot the sizeup for 16 nodes (the largest number of nodes we consider) in Figure 2 and see that our method has sub-linear sizeup for most parameter settings, i. e., for a 10-times larger data set it requires less than 10 times more time. The only two exceptions are $F = 4$ and $F = 5$ (i. e., small numbers of features) with only two buckets ($B = 2$). Looking closer into our results we can observe that for some parameter settings the LSH part of LSH-dRST is more time-consuming than the rough set part, but for others it is less time-consuming.

3) Scaleup Analysis: The scaleup evaluates the ability to increase the number of nodes and the size of the data set simultaneously and is defined as [22]:

$$\text{Scaleup(m)} = \frac{\text{runtime for data set of size } s \text{ on 1 node}}{\text{runtime for data set of size } s \cdot m \text{ on } m \text{ nodes}}$$

We use the sub-data sets previously described with 1 000 features as a baseline and plot the results in Figure 3. It should be noted that a scaleup of 1 implies linear scaleup, which similarly to linear speedup is difficult to achieve. Our scaleup is clearly smaller than 1 for all parameter settings, but fluctuates between 0.2 and 0.4 for most settings and 8 nodes, including the ones that exhibit the best speedup. The best scaleup is achieved for $F = 5$ and large values for B.

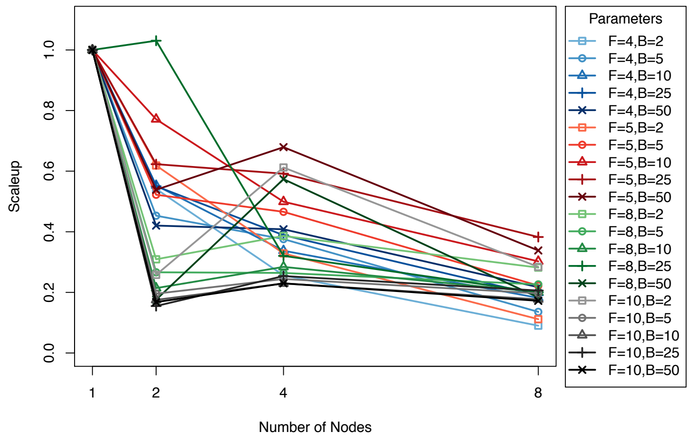


Fig. 3. Scaleup.

B. Comparison with Other Feature Selection Techniques

We show the results of 10 runs of random forest on the different reduced data sets in Figures 4 and 5. We observe that LSH-dRST outperforms the Chi Squared, Information Gain and Gain Ratio selection methods and has comparable performance to the other methods. Moreover, we see that the classification result is quite stable with respect to the parameter settings in LSH-dRST. This is observed for all evaluation metrics, i. e., accuracy, recall, precision, and F1 score.

We plot the number of features selected over 10 runs as boxplots and the corresponding mean execution time of the random forest classifier in Figure 6. We see that the number of features is very concentrated around its median, implying a low variance in the number of features selected. We select around 3 300 features for $F = 4$, 2 700 for $F = 5$, 1 800 for $F = 8$, and 1 500 for $F = 10$. Again, the number of buckets hardly has any impact. For Sp-RST the results reported in [8] were much more erratic with no clear tendency based on the parameter setting and numbers ranging between 1 600 and 6 200.

VI. Conclusion and Emerging Trends

We have introduced a distributed algorithm for feature selection based on rough set theory that uses locality sensitive hashing to determine appropriate partitions of the feature set; called LSH-dRST. Our approach improves a previously introduced method using random partitions (Sp-RST). Our experiments demonstrate that LSH-dRST scales well in terms of three commonly used evaluation criteria: speedup, sizeup, and scaleup. We investigate different parameter settings and show that LSH-dRST is robust with respect to the number of buckets used in LSH while there is a clear trade-off between quality of the feature selection and speedup with respect to the second parameter (the number of features in each sub-information table). Configured appropriately the mean accuracy achieved by a random forest classifier on the reduced data sets is better than for the unreduced data set and comparable to the results obtained by Sp-RST, however, LSH-dRST exhibits a much smaller variance in the feature selection process and thus, is considered more reliable. Based on the conducted experiments and results, we can clearly see the benefit and impact of using the locality sensitive hashing in our proposed solution as it partitions the high dimensional feature search space in a more reliable and intelligent way and hence guaranteeing data dependency in the distributed environment, and ensuring a lower computational cost.

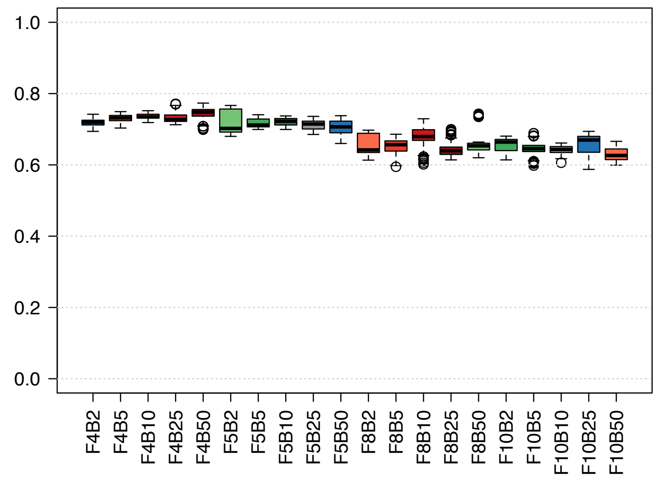
Future research will further investigate the interplay of the different components of LSH-dRST, e. g., the trade-off between the execution times of the LSH and the RST elements of the algorithm and their influence on the reduced data set. Ultimately, we want to use our methods on real-world data sets with more features and data items than the used Amazon data set.

Acknowledgment

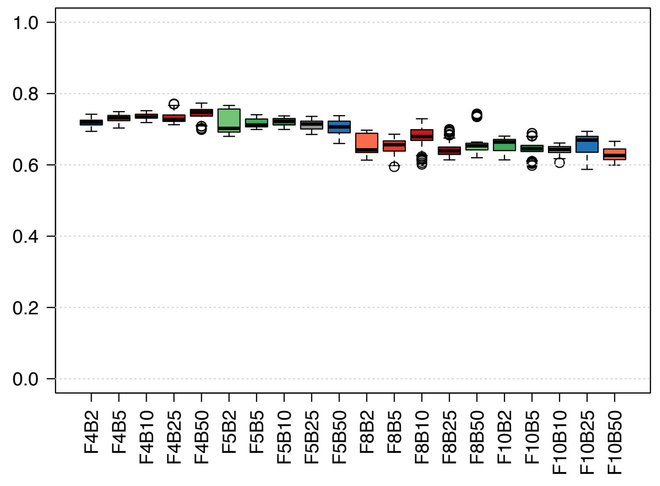
This work is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 702527.

References

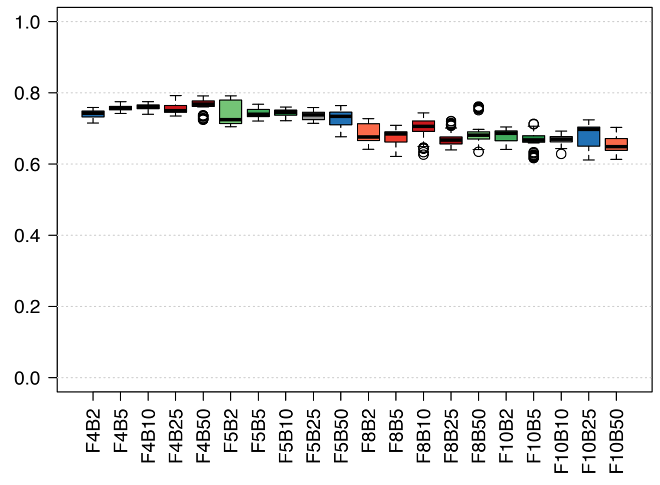
[1] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.



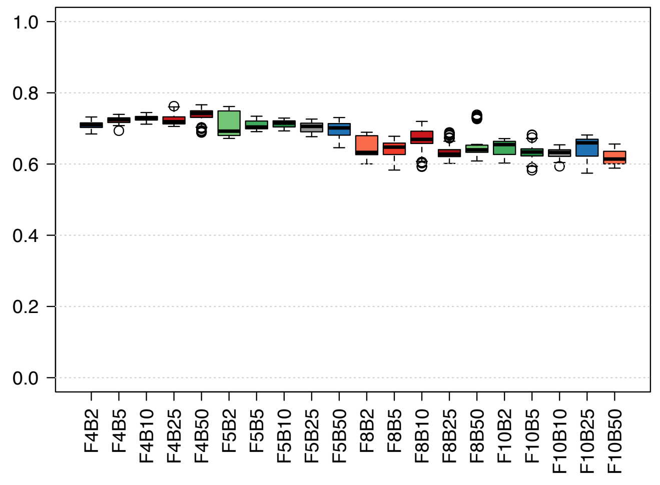
(a) Accuracy



(b) Recall



(c) Precision



(d) F1 Score

Fig. 4. The classification results for different parameters of LSH-dRST.

[2] D. Peralta, S. del Río, S. Ramírez-Gallego, I. Triguero, J. M. Benitez, and F. Herrera, "Evolutionary feature selection for big data classification: A MapReduce approach," *Mathematical Problems in Engineering*, vol. 2015, Article ID 246139, 2015.

[3] H. Peng, F. Long, and C. H. Q. Ding, "Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 8, pp. 1226–1238, 2005.

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

[5] P. Lingras, "Unsupervised rough set classification using gas," *J. Intell. Inf. Syst.*, vol. 16, no. 3, pp. 215–228, 2001.

[6] ——, "Rough set clustering for web mining," in *The 11th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'02)*. IEEE, 2002, pp. 1039–1044.

[7] I. Düntsch and G. Gediga, "Rough set data analysis," *Encyclopedia of Computer Science and Technology*, vol. 43, no. 28, pp. 281–301, 2000.

[8] Z. C. Dagdia, C. Zarges, G. Beck, and M. Lebbah, "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 (BigData 2017)*. IEEE, 2017, pp. 911–916.

[9] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of the 25th VLDB Conference*. Morgan Kaufmann, 1999, pp. 518–529.

[10] W. Liu, J. Wang, R. Ji, Y. Jiang, and S. Chang, "Supervised hashing with kernels," in *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2012, pp. 2074–2081.

[11] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Advances in Neural Information Processing Systems 21 (NIPS 2008)*. Curran Associates, Inc., 2009, pp. 1753–1760.

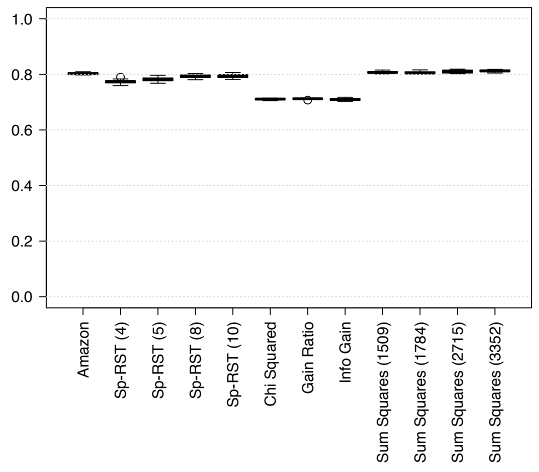
[12] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the 20th ACM Symposium on Computational Geometry (SCG'04)*. ACM, 2004, pp. 253–262.

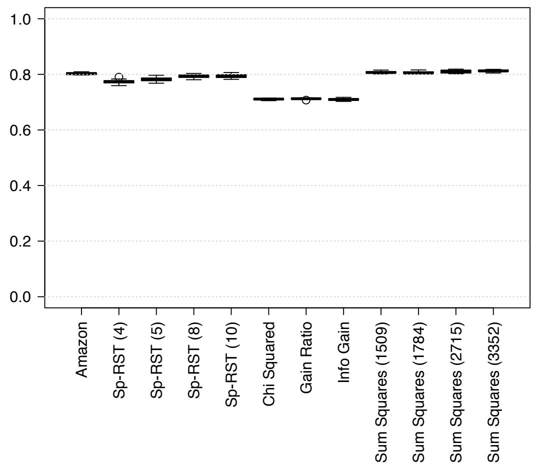
[13] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC'02)*. ACM, 2002, pp. 380–388.

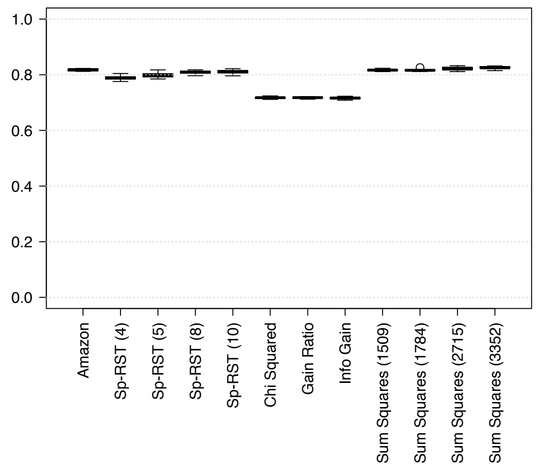
[14] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing (STOC'98)*. ACM, 1998, pp. 604–613.

[15] A. Z. Broder, "On the resemblance and containment of documents," in *Proceedings. Compression and Complexity of SEQUENCES 1997*. IEEE Computer Society, 1997, pp. 21–29.

[16] J. He, W. Liu, and S. Chang, "Scalable similarity search with optimized


(a) Accuracy


(b) Recall


(c) Precision

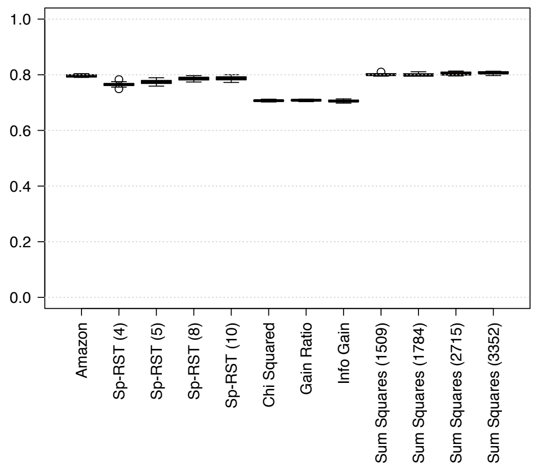

(d) F1 Score

Fig. 5. The classification results for the original data set and other feature selection methods.

kernel hashing," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'10)*. ACM, 2010, pp. 1129–1138.

[17] J. He, S. Chang, R. Radhakrishnan, and C. Bauer, "Compact hashing with joint optimization of search accuracy and time," in *2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2011, pp. 753–760.

[18] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," *ArXiv e-prints*, vol. abs/1408.2927, 2014.

[19] D. Cai, "A revisit of hashing algorithms for approximate nearest neighbor search," *ArXiv e-prints*, vol. abs/1612.07545, 2016.

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

[21] D. Dheeru and E. K. Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[22] 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.

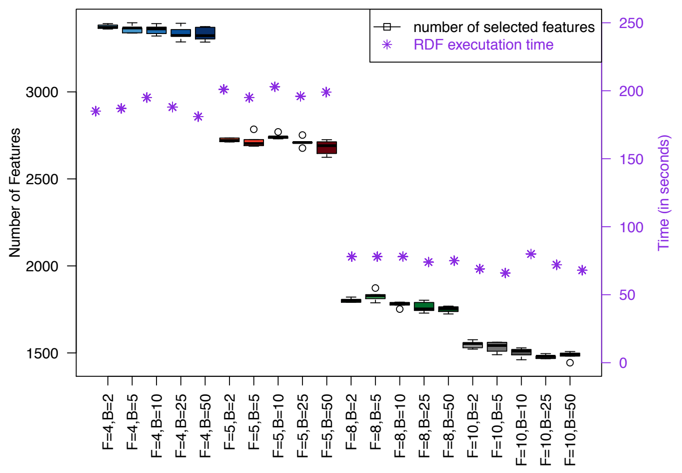


Fig. 6. The number of features in the reduced data set together with the average execution time of the classifier.