A detailed study of the distributed rough set based locality sensitive hashing feature selection technique

Zaineb Chelly Dagdia, Christine Zarges

Fundamenta Informaticae XX (2020) 1–??
DOI 10.3233/FI-2016-0000
IOS Press

A detailed study of the distributed rough set based locality sensitive hashing feature selection technique

Zaineb Chelly Dagdia*
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
Department of Computer Science, Aberystywth University, Aberystwyth, United Kingdom
LARODEC, Institut Supérieur de Gestion de Tunis, Tunis, Tunisia
chelly.zaineb@gmail.com, zaineb.chelly-dagdia@inria.fr

Christine Zarges
Department of Computer Science, Aberystywth University, Aberystwyth, United Kingdom
c.zarges@aber.ac.uk

Abstract. In the context of big data, granular computing has recently been implemented by some mathematical tools, especially Rough Set Theory (RST). As a key topic of rough set theory, feature selection has been investigated to adapt the related granular concepts of RST to deal with large amounts of data, leading to the development of the distributed RST version. However, despite of its scalability, the distributed RST version faces a key challenge tied to the partitioning of the feature search space in the distributed environment while guaranteeing data dependency. Therefore, in this manuscript, we propose a new distributed RST version based on Locality Sensitive Hashing (LSH), named LSH-dRST, for big data feature selection. 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 efficient way. More precisely, in this paper, we perform a detailed analysis of the performance of LSH-dRST by comparing it to the standard distributed RST version, which is based on a random partitioning of the universe. We demonstrate that our LSH-dRST is scalable when dealing with large amounts of data. We also demonstrate

Address for correspondence: INRIA Nancy - Grand Est 615 Rue du Jardin Botanique, 54600 Villers-lès-Nancy, France
*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.

that LSH-dRST ensures the partitioning of the high dimensional feature search space in a more reliable way; hence better preserving data dependency in the distributed environment and ensuring a lower computational cost.

Keywords: Granular Computing; Rough Set Theory; Big Data; Feature Selection; Locality Sensitive Hashing; Distributed Processing.

1. Introduction

Granular computing [1] is a concept covering the family of theories, methodologies, tools, and techniques that use the notion of "granules". Granules can be in the form of groups, sets, classes, or clusters of a universe, which are used in the process of problem solving [2, 3, 4]. These granules are drawn together by indistinguishability, equivalence, similarity, proximity or functionality [5]. In the literature, the basic notion of granular computing, i.e., granules, has appeared in various research approaches among these we mention fuzzy set theory, rough set theory, belief function theory (also called Dempster-Shafer theory), quantization, and many other mathematical approaches [5]. Recently, granular computing has received more attention by researchers from a very different perspective: the challenging big data context. Today, there is a fast growing interest in the study of granular computing in the context of big data, and this manuscript is embedded in this context.

Within the context of big data, granular computing has begun to play important roles in many application domains such as in big data processing [6] where the theory of fuzzy sets was applied to offer a novel promising processing environment, in sentiment analysis where fuzzy set theory was applied on big social data [7], in knowledge acquisition where rough set theory was applied [8], in epidemiology where rough set theory was applied as a big data mining technique [9], in mammography mass classification where rough set theory was applied to analyse deep and hand-crafted features [10], etc.

For these example application domains and many others, there is always the same big challenge in granular computing, which is the task of performing big data pre-processing, specifically feature selection. In this context, the theory of rough sets has recently been adapted to the big data context and successfully been applied as a scalable and effective feature selection technique within a distributed environment [11, 12]. This success is due to many characteristics of the theory, among these the capability of the theory to analyze the facts hidden in data, its independence from the user or expert knowledge as it does not require any supplementary information about the given data, and its ability to find a minimal knowledge representation [13]. These characteristics are gained from the use of the RST granularity structure.

By adapting the basic granular concepts of RST to deal with large amounts of data, a first scalable version of RST within a distributed environment, called Sp-RST, was proposed in [11]. Sp-RST, also detailed in [14], was proposed to avoid the prohibitive complexity of the standard RST, i.e., the sequential/classical RST version, which is caused by the search for an optimal feature subset through the competing of an exponential number of candidate subsets; an exhaustive search, which becomes impractical for big data as it becomes unmanageable to generate the set of all possible feature combinations. In this context, it is important to mention that in the literature there are several RST

heuristics and other RST based methods and algorithms [15, 16] that can find a minimal reduct – which is not globally minimal – in a much faster way. Despite the fact that these approaches avoid such an exhaustive search performed by the classical RST version, the problem of finding a globally minimal reduct remains a big challenge mainly in the context of big data, and specifically if the application domain requires it. For instance, in many real-world applications, specifically in health or medical research, it is essential to have a globally minimal reduct to better guide the decision maker and facilitate their task as revealed in [9, 10]. In this context, it is important to mention that in our research study and in line with our manuscript's goals, we mainly focus on the generation of a globally minimal reduct without calling for heuristics, which makes our task more challenging in the context of big data, and hence we aim to revise Sp-RST. It is important to highlight that the generation of the globally minimal reduct will be performed with respect to the partitioning process, i.e., with respect to the generated smaller data sets that will result from the data partitioning process and where data dependency is guaranteed.

Technically, to perform feature selection in the context of big data, Sp-RST partitions the feature search space in a random manner so that every single partition can hold a random set of attributes. Each of the generated partitions is dealt with in a separate way, i.e., independently in the distributed environment so that at the end of the feature selection process all the selected attributes from every single partition are assembled together to generate the ultimate reduced set of features. However, it can be noticed that in such implementation design, it is very likely that similar attributes will appear in different partitions and hence a cut in data dependency will occur. It is crucial to point out that data dependency is a significant matter in a distributed environment and in parallel computing; and hence considered as a challenge. Based on the defined Sp-RST architecture [11], data dependency will not be guaranteed as the algorithm uses a randomized procedure when partitioning the feature search space. Hence, in this manuscript, we propose to revise the Sp-RST architecture and develop instead a novel efficient distributed algorithm, which is based on the distributed granular concepts of rough set theory and adopts a hashing based technique. Specifically, our proposed solution is based on the Locality Sensitive Hashing (LSH) algorithm [17] for large-scale data pre-processing.

The main motivation behind the choice of LSH among other hashing techniques proposed in the literature [17, 18, 19] is that LSH is often taken as a baseline. But most importantly, the algorithm is widely used in industry, e.g., for image recognition, clustering, and some other tasks, but specifically in database systems for high dimensional similarity search. The choice of LSH is also based on several advantages and characteristics of the algorithm in comparison to other hashing techniques when dealing with high dimensional data sets as demonstrated in [20].

Our proposed solution, dubbed LSH-dRST, adopts LSH, which maps similar data instances based on their feature values into the same bucket in low dimensional cases. Based on this process, LSH-dRST uses the generated buckets to partition the feature search space in a more reliable way, hence better preserving data dependency and a lower computational cost as will be demonstrated in Section 7. Please, note that a preliminary version of LSH-dRST was presented in [21]. In this paper, we will conduct a more detailed study of the work presented in [21] and a deeper analysis of its performance as a distributed feature selection technique. Based on the shortcomings of [11] and the limitations of [21] as it presents a preliminary version not thoroughly studied, a summary of the main contributions of this manuscript are presented as follows:

- Revise the work presented in [11] by developing a novel efficient distributed RST algorithm that adopts a hashing technique to better preserve data dependency in the distributed environment, specifically when it comes to partitioning the universe.
- Execute a thorough analysis of the work presented in [21] by providing (1) a detailed explanation of LSH-dRST in terms of its architecture, its algorithmic design, and its functioning as a distributed algorithm, and (2) by making a detailed analysis of the performance of LSH-dRST by studying the number of features selected and the runtime performance in terms of the different parts of the algorithms as well as its speedup, sizeup and scaleup. Additionally, we present a more detailed model evaluation by additionally considering a simply Naive Bayes classifier (in addition to the random forest classifier originally used).
- Demonstrate that LSH-dRST is not only scalable but also more reliable for feature selection, which will show that it is more relevant to big data pre-processing.
- Demonstrate that LSH-dRST performs the partitioning of the high dimensional feature search space in a more reliable way. This is to show that it better preserves data dependency in the distributed environment, and ensures a lower computational cost.

The rest of this paper is structured as follows. Section 2 presents a review of attribute selection techniques in the context of big data. Section 3 presents preliminaries for Locality Sensitive Hashing and Rough Set Theory for feature selection. Section 4 presents a description of parallel computing frameworks as well as a description of the MapReduce programming model. Section 5 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 6. The results of the performance analysis are given in Section 7 and the work is concluded in Section 8.

2. Review of Feature Selection Techniques in the Context of Big Data

In the literature, several feature selection techniques for non-high dimensional environments have been proposed, which were initially developed in a sequential way [22, 23]. However, their scalability was questioned in the context of big data. In this context, in [24], a detailed study was conducted where authors performed a deep analysis of the scalability of the state-of-the-art feature selection techniques in all their three categories: filter techniques, i.e., approaches, which are independent from any specific induction algorithm, the embedded techniques, i.e., approaches that fuse feature selection and the learning approach into a single process, and the wrapper techniques, i.e., approaches involving a specific learning algorithm when evaluating the attribute subset.

In [24], it was demonstrated that the state-of-the-art feature selection techniques would obviously have scalability issues when dealing with big data. Authors have proved that the existent techniques will be inadequate to handle a high number of attributes in terms of training time and/or effectiveness in selecting the relevant set of features. Therefore, the adaptation of feature selection techniques for big data problems seems essential and it may require the redesign of these algorithms and their incorporation in parallel and distributed environments/frameworks. Among the possible alternatives is the

MapReduce paradigm [25], which was introduced by Google and offers a robust and efficient framework to deal with big data analysis. Several recent works have been concentrated on parallelizing and distributing machine learning techniques using the MapReduce paradigm [26, 27, 28]. Recently, a set of new and more flexible paradigms have been proposed aiming at extending the standard MapReduce approach, mainly Apache Spark[1] [29], which has been applied with success over a number of data mining and machine learning real-world problems [29]. Further details and descriptions of such distributed processing frameworks will be given in Section 4.

With the aim of choosing the most relevant and pertinent subset of features, a variety of feature reduction techniques were proposed within the Apache Spark framework to deal with big data in a distributed way. Among these are several feature extraction methods, i.e., methods that create new attributes from the initial feature set (such as nn-gram, Principal Component Analysis, Discrete Cosine Transform, Tokenizer, PolynomialExpansion, ElementwiseProduct, etc) contrary to feature selection approaches that select a subset from the initial features. There have been very few feature selection techniques, which were proposed, and these are the VectorSlicer, the RFormula and the ChiSqSelector. To further expand this restricted research, i. e., the development of parallel feature selection methods, lately, some other feature selection techniques were proposed in the literature, which are based on evolutionary algorithms [30]. Specifically, the evolutionary algorithms were implemented based on the MapReduce paradigm to obtain subsets of features from big data sets[2]. These include a generic implementation of greedy information theoretic feature selection methods[3], which are based on the common theoretic framework presented in [31], and an improved implementation of the classical Minimum Redundancy and Maximum Relevance feature selection method [31]. This implementation includes several optimizations such as cache marginal probabilities, accumulation of redundancy (greedy approach) and a data-access by columns[4]. However, when studying these distributed techniques, it is noticed that most of them suffer from some limitations. For example, they usually require the user or expert to deal with the algorithms' parameterization, noise levels specification, or simply order the set of attributes and let the user choose their own subset. There are some other feature selection techniques that require the user to indicate how many attributes should be selected, or require the user to give a threshold that determines when the algorithm should stop. This are all considered as significant drawbacks as they require users/experts to make a decision based on their own (possibly subjective) perception.

To overcome the shortcomings of the parallel state-of-the-art techniques, it seems to be crucial to look for a filter approach that does not require any external or supplementary information to function properly. Rough Set Theory (RST) can be used as such a technique. As previously mentioned, most of the classical rough set algorithms are sequential ones, computationally expensive and can only deal with non-large data sets. With a special focus on the generation of a globally minimal feature set, the prohibitive complexity of these algorithms comes from the search for an optimal attribute subset through the computation of an exponential number of candidate subsets. Although it is an exhaustive method, this is quite impractical for most data sets specifically for big data as it becomes clearly

[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

unmanageable to build the set of all possible combinations of features. In order to overcome these weaknesses, a first version presenting a parallel rough set model was presented in [11]. The main idea in [11] is to randomly split the given big data set into several partitions, each with a smaller number of randomly selected features, which are all then processed in a parallel way. This is to minimize the computational effort of the RST computations when dealing with a very large number of features particularly. As it can be noticed, and as previously explained in Section 1, this parallel RST version suffers from one main limitation: it cannot guarantee data dependency within the distributed environment. Our work, which is a revision of [11], is based on a distributed partitioning procedure, within a Spark/MapReduce paradigm, that makes our proposed solution scalable and effective in dealing with big data. The new version that we are proposing in this manuscript, LSH-dRST, adopts a hashing technique that will better preserve data dependency as it will match similar features into the same bucket and map the generated buckets into partitions to enable the splitting of the universe in a more efficient way. As previously mentioned in Section 1, a preliminary version of LSH-dRST was introduced in [21] where a non-detailed description of the algorithm as well as a limited analysis of the algorithm's performance were presented. In this paper, a deep and thorough elucidation and performance investigation of LSH-dRST will be given as highlighted in the contributions presented in Section 1.

3. Preliminary knowledge

In this section, we provide preliminary knowledge about the Locality Sensitive Hashing algorithm and the granulation structure of Rough Set Theory for feature selection.

3.1. Locality Sensitive Hashing

There are several hashing techniques that have been proposed in the literature [17, 18, 19]. Among these the Locality Sensitive Hashing (LSH) algorithm [17] is considered as the most representative and popular one. LSH is presented as a probabilistic similarity-preserving dimensionality reduction method. Based on the adopted distances and similarities, including l_p distance [32], angular distance [33], Hamming distance [34], Jaccard coefficient [35], etc., different types of LSH can be designed, which also depends on the type of the used data [17]. Many variants are developed based on these basic LSH families such as Spectral hashing [36], Kernelized spectral hashing [37], and independent component analysis hashing [38]. These methods aim at learning the hash functions for better fitting the data distribution [39].

In this section, we will mainly introduce LSH, among other hashing techniques, as this algorithm is often taken as a baseline. But most importantly, the algorithm is widely used in database systems for high dimensional similarity search. The choice of LSH is also based on several advantages and characteristics the algorithm has in comparison to other hashing techniques when dealing with high dimensional data sets as demonstrated in [20].

LSH 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 $h \in \mathcal{H} : \mathbb{R}^d \to \mathcal{U}$. Consider a function h that is chosen uniformly at random from $\mathcal{H}$ and a similarity function $sim : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$. 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 means, 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 $\mathcal{G} = \{g : \mathcal{R}^d \to \mathcal{U}^k\}$ 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. 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.

3.2. Rough Sets for Feature Selection

Rough Set Theory (RST) [40] is a formal approximation of the conventional set theory which supports approximations in decision making. It provides a filter-based technique by which knowledge may be extracted from a domain in a concise way; retaining the information content whilst reducing the amount of knowledge involved [16]. The fundamentals of RST for feature selection are as follows:

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. Any pair (x, a), where $x \in U$ and $a \in A$ defines the table entry consisting of the value $a(x)$.

Let $B \subset A$ be a subset of attributes. The indiscernibility relation, denoted by $IND(B)$, is the central concept of RST and is an equivalence relation, which is defined as: $xIND(B)y$ if and only if $a(x) = a(y)$ for every $a \in B$, where $a(x)$ denotes the value of feature a of object x.

The family of all equivalence classes of $IND(B)$, referring to a partition of U determined by B, is denoted by $U/IND(B)$. Each equivalence class may be viewed as a granule consisting of indistinguishable elements. It is also referred to as an equivalence granule. The granulation structure induced by an equivalence relation is a partition of the universe. Each element in $U/IND(B)$ is a set of indiscernible objects with respect to B. The equivalence classes $U/IND(C)$ and $U/IND(D)$ are called *condition* and *decision* equivalence classes, respectively.

For any concept $X \subseteq U$, two operations can be defined: $\underline{B}(X) = \{x \in U : B(x) \subseteq X\}$, and $\overline{B}(X) = \{x \in U : B(x) \cap X \neq \emptyset\}$, assigning to every subset X of the universe U two sets $\underline{B}(X)$ and $\overline{B}(X)$ called the B-*lower* and the B-*upper* approximation of X, respectively. The lower approximation of a set X with respect to B is the set of all objects, which can be for *certain* classified as X using B. The upper approximation of a set X with respect to B is the set of all objects, which can be possibly classified as X using B. The concept defining the set of objects that

can be classified neither as X nor as not-X using B is called the *boundary region*, and is defined as: $BND_B(X) = \overline{B}(X) - \underline{B}(X)$. If the boundary region of X is empty, that is $\overline{B}(X) = \underline{B}(X)$, concept X is said to be B-*definable*; otherwise X is a *rough set* with respect to B.

To discover dependencies between attributes in a given decision system $T = (U, C, D)$, the *dependency degree* is defined. Intuitively, a set of attributes D depends totally on a set of attributes C, denoted $C \Rightarrow D$, if the values of attributes from C uniquely determine the values of attributes from D. This can be formally defined as: we say that D depends on C to a degree k $(0 \leq k \leq 1)$, denoted $C \Rightarrow_k D$ if: $k = \gamma(C, D) = \frac{|POS_C(D)|}{|U|}$, where $POS_C(D) = \bigcup_{X \in U/IND(D)} \underline{C}(X)$ is called the *positive region* of the partition $U/IND(D)$ with respect to condition attributes C. The positive region is a set of objects of U that can be uniquely classified to blocks of the partition $U/IND(D)$, by means of C.

If $k = 1$ we say that D depends totally on C, and if $k < 1$, we say that D depends partially (to degree k) on C. If $k = 0$ then the positive region of the partition $U/IND(D)$ with respect to C is empty. The coefficient k, the dependency degree, expresses the ratio of all elements of the universe, which can be properly classified to blocks of the partition $U/IND(D)$, employing attributes C.

Based on these basics, RST defines the *D-reduct* concept for feature selection. Let $C, D \subseteq A$, be sets of condition and decision attributes respectively. We will say that $C' \subseteq C$ is a *D-reduct* (reduct with respect to D) of C, if C' is a minimal subset of C such that: $\gamma(C, D) = \gamma(C', D)$, and there is no $C'' \subset C'$ such that $\gamma(C', D) = \gamma(C'', D)$. 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 D-reducts, called $RED_D^F(C, D)$, from the given information table. In this case, any reduct from $RED_D^F(C, D)$ can be chosen to replace the initial information table.

4. Parallel Computing Frameworks and the MapReduce Programming Model

With the dramatic increase of the amount of data, a specific computing paradigm has been stressed out, which is called parallel computing. Parallel computing is a type of computation in which multiple compute resources are simultaneously used to perform several calculations to solve a computational problem. Within a parallel computing schema, a problem is broken into discrete and smaller parts that can be solved concurrently. Each of these parts is further broken down to a series of instructions, which are executed simultaneously on different CPUs so that they can be solved at the same time.

In the context of big data, it has become necessary to implement a new set of technologies and tools that permit parallel computing in an effective way. Different techniques [41] have been developed to handle high dimensional data sets where most of these proposed tools are based on distributed processing, e.g., the Message Passing Interface (MPI) programming paradigm [42].

The encountered challenges are essentially linked to the access to the given big data, to the transparency of the development process of the software with respect to its prerequisites, as well as to the available programming paradigms [43]. For example, standard techniques require that all the given data should be loaded into the main machine's memory. This obviously presents a technical issue in big data since the data, which is given as input, is usually stored in different locations causing an

intensive communication in the network as well as some supplementary input and output costs. It is true that it is possible to afford this but it is also important to mention that it will be crucial to afford an intensively large main memory to be able to retain all the pre-loaded given data for computing and processing purposes.

To overcome these serious limitations, a new set of highly efficient and fault-tolerant parallel frameworks has been developed and set in the market. These distributed frameworks can be categorized with respect to the nature or type of the data they are able to process. Actually, there are some frameworks that can only process batch data. Within this schema, the parallel processing system functions over a high dimensional and static data set. At a later level of the distributed processing, the system returns the output result(s) when all the processes of computations are successfully achieved. Among the well-known open-source distributed processing frameworks dedicated for batch processing, we mention Hadoop[5]. Hadoop is based on simple programming paradigms that allow a highly scalable and reliable parallel processing of high dimensional data sets. The framework offers a cost-effective solution to store and process different types of data such as structured, semi-structured and unstructured data without any specific format specifications. Technically, Hadoop works on top of the Hadoop Distributed File System (HDFS), which duplicates the input data files in various storage machines (nodes). In this manner, the framework facilitates a fast transfer rate of the data among nodes set in the cluster and allows the system to operate without any interruption if one or a number of nodes fail. MapReduce is the core of the Hadoop framework. This paradigm offers an intensive scalability over a large number of nodes within a Hadoop cluster.

On the other hand, there are some other distributed frameworks that can only deal with streaming data. Within these frameworks' design, the distributed calculations are performed over data (to each individual data item) at the time it enters the parallel framework. Apache Storm[6] and Apache Samza[7] are among the most popular stream processing frameworks. A third category of distributed frameworks can be highlighted, which is considered as hybrid systems. This is because these frameworks are capable of processing not only batch data but also stream data. In these frameworks' designs, similar or some linked elements can be used for both types of data. This makes the diverse processing requirements of the hybrid systems much easier and simpler. Among the well-known streaming processing parallel frameworks, we mention Apache Spark[8] and Apache Flink[9].

In this research, we focus on Apache Spark. This distributed open source framework was initially developed in the UC Berkeley AMP Lab for big data processing. Apache Spark is characterized by its capability of improving the system's effectiveness—which is achieved via the use of intensive memory—, its efficiency, and its high transparency for users. These characteristics allow to perform parallel processing of diverse application domains in a simple and easy way. More precisely and in comparison to Spark, in Hadoop MapReduce multiple jobs would be adjusted together to build a data pipeline. In this process, and in every level of this pipeline, MapReduce will have to read the data from the disk, and then write it back to the disk again. This process was obviously ineffective as it had

[5] http://hadoop.apache.org/
[6] http://storm.apache.org/
[7] http://samza.apache.org/
[8] https://spark.apache.org/
[9] https://flink.apache.org/

to read all the data and write it from and back to the disk at each level of the process. To deal with this issue, Apache Spark comes into play. Based on the same MapReduce paradigm, the Spark framework could offer an immediate 10 times increase in the system's performance. This is explained by the non-necessity to store the given data back to the disk at every stage of the process as all activities remain in the memory [29]. Spark allows a much faster data process in contrast to transferring it through needless Hadoop MapReduce mechanisms. Adding to this specificity, the key concept that Spark offers is a Resilient Distributed Data set (RDD), which is a set of elements that are distributed across the nodes of the used cluster that can be operated on in a parallel way. Indeed, Spark has a number of high-level libraries for working with structured data (Spark SQL[10]), for stream processing (Spark Streaming[11]), for machine learning (MLlib)[12][44], and for graphs and graph-parallel computation (GraphX[13]). Other than that, there are also many R[14] and Python[15] libraries, among others, which allow the programmers to code without writing mappers and reducers themselves.

The choice of Spark to design our proposed algorithm based on rough sets for big data feature selection is based on several reasons, which are as follows: (i) To offer a general solution based on a hybrid parallel framework. (ii) Apache Spark provides high speed benefits with a trade-off in the usage of high memory. (iii) Spark is one of the well-known and certified distributed frameworks and also a mature hybrid system. This is specifically true when comparing it to some other frameworks in the market, which are considered as more niche in terms of their usage but more importantly they are still in their initial periods of adoption[16].

As previously mentioned, Spark is based on MapReduce [25] which is one of the most popular processing techniques and program models for distributed computing to deal with big data. Spark revolves around the concept of a resilient distributed dataset (RDD)[17], which is the Spark programming model. It is a fault-tolerant collection of elements that can be operated on in parallel. RDDs, used as our programming model, support two types of operations: *transformations*, which create a new dataset from an existing one, and *actions*, which return a value to the driver program after running a computation on the dataset. For example, *map* is a transformation that passes each dataset element through a function and returns a new RDD representing the results. On the other hand, *reduce* is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program. The map and the reduce concepts constitute the MapReduce paradigm, which was proposed by Google in 2004 and designed to easily scale data processing over multiple computing nodes. As mentioned, the MapReduce paradigm is composed of two main tasks/phases, namely the map phase and the reduce phase, which will be the main concepts used in our developed approach (see Sections 5 and 5.2.2). At in abstract level, the map process takes as input a set of data and transforms it into a different set where each element is represented in the form of a tuple *key/value* pair, producing some intermediate results. Then, the reduce process collects the output from the map task as an input and combines these given *key/value* tuples into a smaller set of pairs to generate the final output. A

[10] https://spark.apache.org/sql/
[11] https://spark.apache.org/streaming/
[12] https://spark.apache.org/mllib/
[13] https://spark.apache.org/graphx/
[14] https://mapr.com/blog/installation-guide-rhadoop-mapr-now-available/assets/rhadoop_and_mapr.pdf
[15] http://spark.apache.org/docs/latest/api/python/
[16] https://www.digitalocean.com/community/tutorials/hadoop-storm-samza-spark-and-flink-big-data-frameworks-compared
[17] https://spark.apache.org/docs/latest/rdd-programming-guide.html#overview

representation of the MapReduce framework is given in Figure 1.

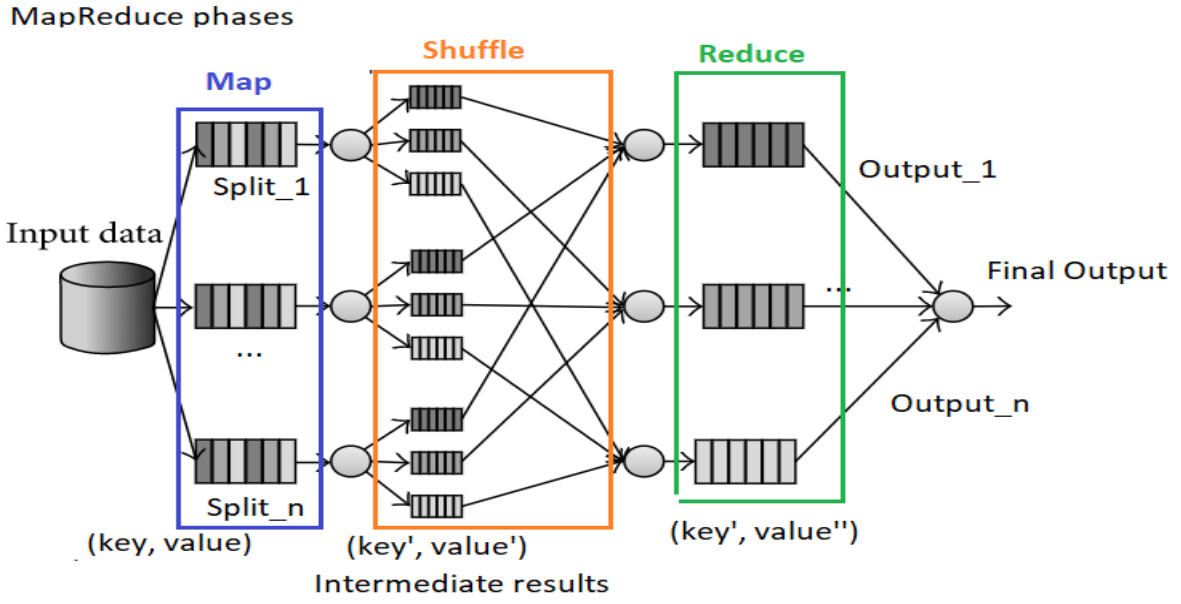


Figure 1. The process of the MapReduce framework.

Technically, the MapReduce paradigm is based on a specific data structure, which is the (key, value) pair. More precisely, during the map phase, on each split of the data the map function gets a unique (key, value) tuple as an input and generates a set of intermediate (key', value') pairs as output. This is represented as follows:

$$map(key, value) \rightarrow \{(key', value'), \ldots\}. \tag{1}$$

After that, the MapReduce paradigm assembles all the intermediate (key', value') pairs by key via the shuffling phase. Finally, the reduce function takes the aggregated (key', value') pairs and generates a new (key', value") pair as output. This is defined as:

$$reduce(key', \{value', \ldots\}) \rightarrow (key', value''). \tag{2}$$

As discussed, a variety of open source parallel computing frameworks are proposed in the market and in this section, we have highlighted the well-known ones. However, it is important to mention that choosing a particular distributed framework is always dependent to the type or kind of the given data that the system will process. The choice also depends on how time bound the specifications of the users are, and on the types of output results that users are looking for. In this paper, we mainly focus on the use of Apache Spark.

5. LSH-dRST: The proposed Solution

In this section, we will make a detailed description of our proposed solution named LSH-dRST. LSH-dRST has a distributed architecture with respect to the Apache Spark framework for a parallel and in-memory processing job. At the beginning of this section, we will present the main motivation behind the development and implementation of the LSH-dRST solution. To do so, we will first explain the limitations of the standard distributed RST version. Next, we will explain LSH-dRST as an effective technique that is capable of performing big data feature selection in a more intelligent and convenient way without any considerable information loss.

5.1. Motivation and Problem Statement

As explained in Section 3.2, the theory of rough sets goes through the calculation of the dependency of attributes, $\gamma(C, D)$, so that it can perform feature selection. With this aim, and as a first process, the indiscernibility relation, defined as $IND(B)$, for all attributes has to be calculated. As previously defined, the $IND(B)$ searches for similar attribute values and gathers the corresponding features to form the set of the equivalence relations. With respect to these fundamental RST notions, it is essential to guarantee data dependency in order to define the most reliable and consistent equivalence relations, so that the most representative reduct set can be determined. However, assuring data dependency is considered as a big challenge when it comes to distributed environments and parallel computing.

The standard distributed RST version [11], named Sp-RST, applies a random process when partitioning the feature search space; a process that does not guarantee data dependency. More precisely, Sp-RST partitions the information table T (the big data set) into a set of m data blocks based on splits from the conditional attribute set C, i.e., m smaller data sets with a fewer number of features instead of using a single data block (T) with an unmanageable number of features $|C|$ (noted as $T(C)$). The key idea is to generate m smaller data sets named T_i, where $i \in \{1, \ldots, m\}$, from the big data set T, where each T_i is defined via a manageable number of features r with $r \lll |C| \in \{c_1, \ldots, c_V\}$ and $r \in \{1, \ldots, V\}$. The resulting data blocks are defined as $T_i(r)$. This is formalized as $T = \bigcup_{i=1}^{m} T_i(r)$. r is a user defined parameter that refers to the number of attributes, which will be considered to build each data block T_i, and V refers to the total number of attributes. Specifically, every T_i is built using r *random* and distinct attributes, which are selected from C, where there are no common attributes between all the built T_i. With respect to the parallel implementation design, and as each constructed partition is processed by a different machine, i.e., node, the distributed Sp-RST algorithm is applied to every $T_i(r)$ while gathering all the intermediate results from the m distinct created partitions (instead of being applied to the complete T that encloses the whole set C of conditional features ($T(C)$)).

Based on this Sp-RST architecture and implementation design, it is very probable that similar attributes will be part of different partitions T_i. Consequently, an erroneous estimation of the constructed $IND(B)$ is more likely to occur. More precisely, the applied random process may mislead the RST feature selection process by generating a non-relevant reduct. Based on these limitations, and aiming at guaranteeing data dependency, in this paper we propose the 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 attribute values into the same bucket. By using the generated buckets, a more intelligent and reliable partitioning of the universe can be applied. In such a way,

LSH-dRST can conserve data dependency within the same buckets and hence it is capable of solving the standard distributed RST limitations.

5.2. LSH-dRST

In order to deal with big data sets and to make use of the adopted LSH technique within a distributed environment, the appropriate set of buckets, which is based on LSH, is generated first. After that, these generated buckets will be mapped into several partitions. Then, the entire rough set feature selection process, with all its granular concepts, will be partitioned into different elementary tasks where each of these will be executed independently on each generated bucket. As a last step, the intermediate results will be conquered to finally acquire the final output, i.e., the reduct set.

5.2.1. General Model Formalization

To perform feature selection, our learning problem has to select a set of high discriminating attributes from the original large set of features that describes the high dimensional input database. The input dataset 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 (defined in Section 4). This design can be formalized as follows: the created RDD reflects the given information table, which we name T_{RDD}. T_{RDD} is defined via a universe $U = \{x_1, \ldots, x_N\}$, which refers to the set of data items, a large conditional attribute set $C = \{c_1, \ldots, c_V\}$ that contains every single attribute of the T_{RDD} information table, and via a decision attribute D of the given learning problem. D corresponds to the class, i.e., label, of each T_{RDD} sample and is defined as $D = \{d_1, \ldots, d_W\}$. The conditional feature set C reflects the pool from where the most convenient attributes will be selected.

To make LSH-dRST scalable with the large number of attributes – instead of applying the algorithm to a single data block (T_{RDD}) defined via its unmanageable feature set C (that we also note as $T_{RDD}(C)$) – and with respect to data dependency, the given T_{RDD} information table is partitioned into B data blocks based on the B generated LSH buckets, named as $T_{RDD_{(b)}}$ where $b \in \{1, \ldots, B\}$. This is achieved by performing a transpose of the feature set C, on which LSH is applied to generate the buckets based on the features. Specifically, LSH is applied based on a family of hash functions $\mathcal{H}$ and based on the p-stable similarity and on a Gaussian distribution as explained in Section 3.1. The different buckets correspond to splits from the conditional feature set C and each bucket covers a definite feature space h incorporating all similar and close data instances based on their attribute values. The resulting data block is denoted as $T_{RDD_{(b)}}(h)$. This can be formalized as: $T_{RDD} = \bigcup_{b=1}^{B} T_{RDD_{(b)}}(h)$, where $h \in \{1, \ldots, V\}$. The parameter h refers to the value, which is generated by LSH, and corresponds to the number of attributes per bucket that will be considered to build each $T_{RDD_{(b)}}$ data block. h is equal to the size of the feature space C divided by B. This part of the LSH-dRST functioning is presented in Figure 2.

Once the data blocks (also referred to as buckets) $T_{RDD_{(b)}}$ are defined, the K nearest neighbors approach is applied on every $T_{RDD_{(b)}}$. This results in automatically creating a set of S sub-information tables that we name as Cl, i.e., $T_{RDD_{(b)}}$ is partitioned into S sub-information tables Cl. K corresponds to the number of attributes per sub-information table Cl_s, with $s \in \{1, \ldots, S\}$, and on which LSH-dRST will be applied. The resulting information table is denoted as $Cl_s(K)$. This can be formalized

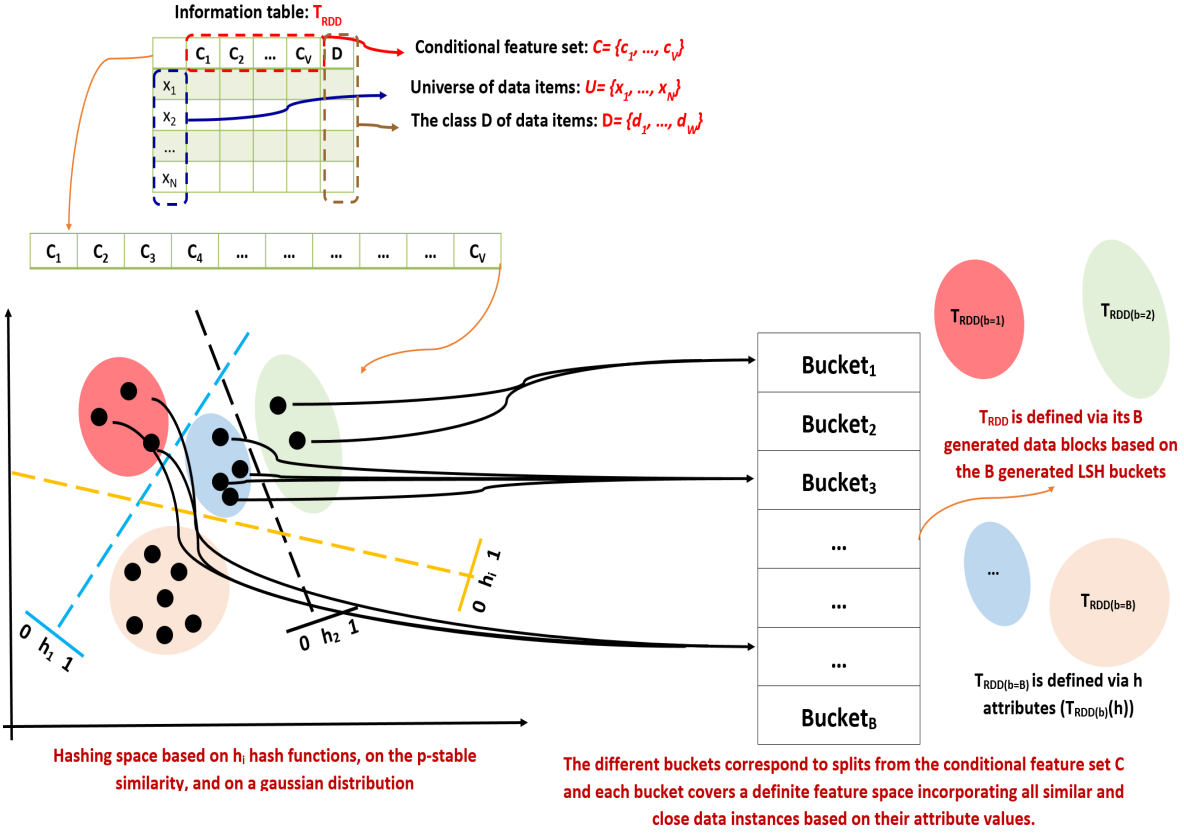


Figure 2. The process of generating the B data blocks based on the B generated LSH buckets.

as: $T_{RDD_{(b)}} = \bigcup_{s=1}^{S} Cl_s(K)$, where $S = h/K$. This part of the LSH-dRST functioning is presented in Figure 3.

Aiming at ensuring scalability, instead of applying LSH-dRST to T_{RDD}, which covers the whole conditional attribute set C ($T_{RDD}(C)$), the distributed algorithm will be applied to every single $Cl_s(K)$. At the end, all the intermediate results will be congregated from the different sub-information tables Cl of every single $T_{RDD_{(b)}}$ partition. Based on such a process, we can ensure that LSH-dRST can be applied to a computable and manageable number of attributes while preserving data dependency. Consequently, the limitations of the standard distributed RST [11], i.e., Sp-RST, can be solved. The pseudo-code of LSH-dRST is presented in Algorithm 1.

Technically, as a first step, LSH-dRST will generate the B partitions using LSH, defined as $T_{RDD_{(b)}}$, while preserving data dependency as previously explained and as presented in Figure 2 (see Algorithm 1, line 1). After that, for each specific partition $T_{RDD_{(b)}}$, a set S of sub-information tables (called Cl) will be built in such a way that the K nearest neighbors from any data point within the $T_{RDD_{(b)}}$ feature search space form a sub-information table called Cl_s, where $s \in \{1, \ldots, S\}$, which is defined via its number of features K (noted as $Cl_s(K)$) (see Algorithm 1, line 4).

The different tasks of the distributed LSH-dRST, Algorithm 1, lines 6 - 11, will be performed on

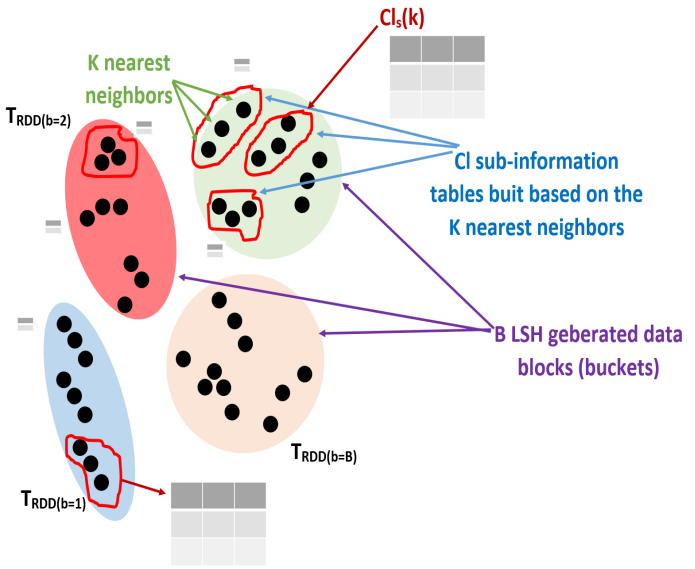


Figure 3. The process of generating the sub-information tables Cl for each $T_{RDD_{(b)}}$ bucket based on the K nearest neighbors approach.

all the S Cl sub-information tables, i.e., on all the $Cl_s(K)$ of all the $T_{RDD_{(b)}}$ partitions. As noticed in Algorithm 1, the task defined in line 2, which is tied to generating the $IND(D)$, is performed outside the $T_{RDD_{(b)}}$ and the $Cl_s(K)$ iteration loops. This implementation is explained by the fact that this specific task deals with the calculation of the indiscernibility relation of the decision class, i.e., $IND(D)$. This task is independent from the B partitions, which are built, as the result depends on the data items class/label and not on the set of attributes.

Now, outside the iteration loops, Algorithm 1, line 14, each $Cl_s(K)$ will have an output, which can be in one of the following two forms: (i) either a single reduct $RED_s(K, D)$ or (ii) a family of reducts $RED_s^F(K, D)$. As previously explained in Section 3.2, based on the RST preliminaries and granular concepts, any reduct of $RED_s^F(K, D)$ can be used to represent the $Cl_s(K)$ sub-information table. Accordingly, in the case where LSH-dRST generates a single reduct for a specific $Cl_s(K)$ sub-information table, the output of this attribute selection phase is the set of the $RED_s(K, D)$ attributes. These selected attributes reflect the most informative features among the initial K features defining $Cl_s(K)$. This results in a new reduced $Cl_s(K)$, defined as $Cl_s(RED)$, which preserves nearly the same data quality as its corresponding $Cl_s(K)$, which is based on the whole feature set K. The second case is when LSH-dRST generates a family of reducts. In this particular case, the algorithm will randomly select one reduct among $RED_s^F(K, D)$ to represent the corresponding $Cl_s(K)$. Let us recall, as seen in Section 3.2, that this random choice is mathematically justified by the different granular concepts of rough set theory, which define the same priority for all the possible generated reducts in $RED_s^F(K, D)$. This means that any reduct, which is included in $RED_s^F(K, D)$, can be used to replace the K features of $Cl_s(K)$. At this algorithmic phase, each Cl_s sub-information table has its output $RED_s(K, D)$, which corresponds to the selected attributes. All the selected

Algorithm 1: LSH-dRST

Inputs: T_{RDD}: information table with:
D: decision class
K: number of nearest neighbors
B: number of buckets

Output: $Reduct$

1: Generate the B LSH buckets: $T_{RDD_{(b)}}$, where $b \in [1, \ldots, B]$
2: Calculate $IND(D)$
3: **For each** $T_{RDD_{(b)}}$ — where $b \in [1, \ldots, B]$
4: Generate the set S of sub-information tables Cl based on K: $Cl_s(K)$ — where $s \in [1, \ldots, S]$
5: **For each** $Cl_s(K)$ — where $s \in [1, \ldots, S]$
6: Generate $AllComb_{(K)}$
7: Calculate $IND(AllComb_{(K)})$
8: Calculate $DEP(AllComb_{(K)})$
9: Select $DEP_{max}(AllComb_{(K)})$
10: Filter $DEP_{max}(AllComb_{(K)})$
11: Filter $NbF_{min}(DEP_{max}(AllComb_{(K)}))$
12: **End For**
13: **End For**
14: Gather the outputs: Each $Cl_s(K)$ will have an output, which can be (i) either a single reduct $RED_s(K, D)$ or (ii) a family of reducts $RED_s^F(K, D)$ from which a single reduct $RED_s(K, D)$ is selected randomly. All the selected reducts are irreducible within their $Cl_s(K)$.
15: Union all the generated reducts $RED_s(K, D)$ from all the Cl_s from all the $T_{RDD_{(b)}}$: $Reduct = \bigcup_{b=1}^{B} \bigcup_{s=1}^{S} RED_s(K, D)$
16: **Return** $(Reduct)$ *//*Reduct is irreducible based on step 14*//*

reducts are irreducible within their $Cl_s(K)$. Nevertheless, as each Cl_s is defined using 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 operation is applied to merge all the $RED_s(K, D)$ from all the Cl_s and from all the $T_{RDD_{(b)}}$. This is defined as $Reduct = \bigcup_{b=1}^{B} \bigcup_{s=1}^{S} RED_s(K, D)$ (Algorithm 1, line 15). The generated $Reduct$ represents the minimal reduced set that represents the initial T_{RDD}. $Reduct$ is indeed irreducible as it is based on the irreducible reducts $RED_s(K, D)$ within their $Cl_s(K)$ (see Algorithm 1, line 14).

By removing the set of irrelevant and redundant attributes, LSH-dRST can reduce the dimensionality of the data, specifically the large number of features, from $T_{RDD}(C)$ to $T_{RDD}(Reduct)$. A high level description of the full functioning of LSH-dRST is given in Figure 4.

In the following subsections, the different elementary distributed tasks of LSH-dRST will be described in more detail.

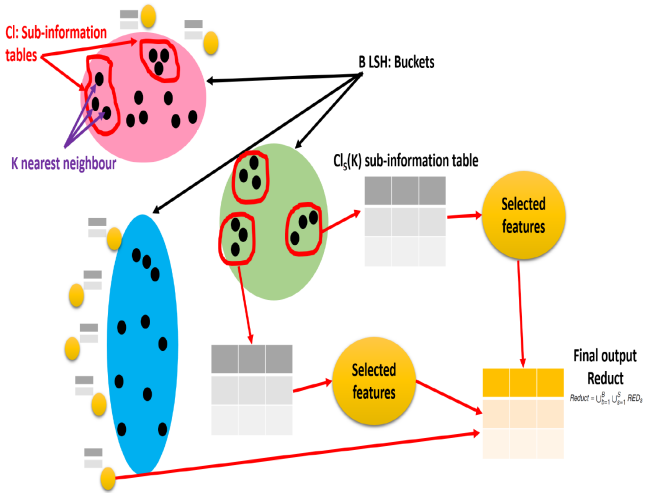


Figure 4. A high level description of LSH-dRST.

5.2.2. Algorithmic Details

As previously mentioned, the different elementary distributed tasks of LSH-dRST feature selection will be performed on every single $Cl_s(K)$ sub-information table, which is defined by its K attributes within the $T_{RDD_{(b)}}$ partitions; except for the task defined in line 2 in Algorithm 1, which deals with $IND(D)$. To generate the final output, $Reduct$, LSH-dRST performs ten main jobs.

Initially, LSH-dRST applies the hashing technique, i.e., LSH, to build and generate the B different buckets based on a hash table as previously explained in Section 3.1. To do so, LSH-dRST creates the hash table based on a set of random vectors following a Gaussian distribution, referred to as the family $\mathcal{H}$ of hash functions. The hash table created is based on the number of the T_{RDD} attributes. As a next step, LSH-dRST maps the T_{RDD} to work on each single partition in a separate and parallel way. On each partition, the algorithm applies a projection for each vector based on the set of the mapped feature vectors in T_{RDD}. Consequently, the buckets are automatically generated, each with a specific index, referred to as a hash code (as explained in Section 3.1). Lastly, the algorithm performs a sort operation/action to order the different buckets with respect to the given number of buckets B. Algorithm 2 presents the pseudo-code of this distributed task.

The next process is that LSH-dRST has to compute the indiscernibility relation for the decision class $D = \{d_1, \ldots, d_W\}$, defined as $IND(D)$: $IND(d_i)$. Technically, the algorithm calculates the indiscernibility relation for every single decision class d_i by collecting the same data items from T_{RDD}, which are defined in the universe $U = \{x_1, \ldots, x_N\}$ and belong to the same class d_i. To achieve this task, LSH-dRST processes a first map transformation operation taking the data in its format of *(id_i of x_i, List of the features of x_i, Class d_i of x_i)* and transforming it to a $\langle key, value \rangle$ pair: $\langle$Class d_i of x_i, List of id_i of $x_i\rangle$. Based on this transformation, the decision class d_i defines the key of the generated output and the data items identifiers id_i of x_i of T_{RDD} define the values. After

Algorithm 2: Generate $Buckets(B)$

Inputs: T_{RDD}: information table
B: number of buckets
Output: $T_{RDD_{(b)}}$: buckets

1: Generate the hash table based on the feature size of T_{RDD}
2: Map the T_{RDD}
3: Apply the LSH projection for each vector
4: Sort the buckets by B

that, the $foldByKey()$[18] transformation operation is applied to merge and reduce all values of each key in the transformed RDD output. This is to represent the sought $IND(D)$: $IND(d_i)$. Algorithm 3 describes the pseudo-code of this distributed job.

Algorithm 3: Calculate $IND(D)$

Input: T_{RDD}
Output: $IND(D)$: [d_i, List of id_i of x_i]

1: Map the T_{RDD} based on its format *(id_i of x_i, List of the features of x_i, Class d_i of x_i)* and generate the new format as a key-value pair $\langle$Class d_i of x_i, List of id_i of $x_i\rangle$
2: Merge/Reduce the values of each generated key using the $foldByKey()$ operation
3: Return $IND(D)$

In the third algorithmic step of LSH-dRST, the algorithm has to generate the set S of the sub-information tables $Cl_s(K)$ based on the number of attributes K. At this stage, let us recall that LSH has already gathered all the instances based on their similar attributes within a same specific bucket. On these similar attributes collected, an additional partitioning is required to build and generate the sub-information tables that can be handled by LSH-dRST. As previously explained in Section 1, and as we are focusing on the generation of a globally minimal reduct, to perform feature selection, the theory of rough sets has to generate all the combination of features at once, process them in turn to finally generate the ultimate reduct. Since it is infeasible to generate all the combinations of attributes based on the large amounts of features, the distributed LSH-dRST algorithm will function on the $Cl_s(K)$ sub-information tables built on K features. K is a manageable size that can be handled by LSH-dRST.

Based on this formalization, and to achieve this distributed job, for every bucket $T_{RDD_{(b)}}$, LSH-dRST performs a $mapPartitionsWithIndex$[19] transformation operation using the buckets indexes, i.e., the already generated hash codes in Algorithm 2. This is to be able to apply the k-nearest neighbors (KNN) separately on each $T_{RDD_{(b)}}$ partition while providing an index of the partition. After that, the output of the applied transformation function is mapped in a way that: for each partition, the K

[18] https://spark.apache.org/docs/0.7.3/api/core/spark/PairRDDFunctions.html
[19] https://spark.apache.org/docs/1.6.2/api/java/org/apache/spark/rdd/RDD.html#mapPartitionsWithIndex(scala.Function2,%20boolean,%20scala.reflect.ClassTag)

features, which are the nearest to a randomly chosen feature within the same $T_{RDD_{(b)}}$ hash code, are selected to build a sub-information table ($Cl_s(K)$). Algorithm 4 presents the pseudo-code related to this distributed task.

Algorithm 4: Generate $Cl_s(K)$

Inputs: $T_{RDD_{(b)}}$: bucket
K: number of features
Outputs: S: set of the sub-information tables $Cl_s(K)$
$Cl_s(K)$: sub-information tables

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

At this stage, and for all the $T_{RDD_{(b)}}$, the set S of the sub-information tables $Cl_s(K)$ is generated. As a next step, the algorithm has to select the most pertinent features from each $Cl_s(K)$. To perform this task, LSH-dRST creates a set of all possible combinations of the sets of K features, defined as the $AllComb_{(K)}$ RDD, by applying the $flatmap()$[20] transformation operation and the $combinations()$ operation. This is shown in Algorithm 5.

Algorithm 5: Generate $AllComb_{(K)}$

Input: K: number of features
Output: $AllComb_{(K)}$: set of all possible combinations of K features

1: Generate the $AllComb_{(K)}$ RDD by applying the $flatmap()$ function and the $combinations()$ operation on each element
2: Return $AllComb_{(K)}$

After that, within the fifth LSH-dRST distributed job, the algorithm computes the indiscernibility relation $IND(AllComb_{(K)})$ for every previously generated combination, i.e., the indiscernibility relation of every element in the output of Algorithm 5, named $AllComb_{(K)_i}$. With respect to the terminologies used in Section 3.2, and to calculate the indiscernibility of a combination of features, the following equations are applied: (i) $IND_C = U/C = \{[o_j]_C | o_j \in U\}$, (ii) $[o_j]_C = \{o_i | C(o_i) = C(o_j)\}$, where o_i and o_j refer to data objects of the universe. In this task and as described in Algorithm 6, the algorithm aims at collecting all the identifiers id_i of the data items x_i that have identical values of the combination of attributes, which are extracted from $AllComb_{(K)}$. To do so, a first map operation is applied, taking the data in its format of *(id_i of x_i, List of the features of x_i, Class d_i of x_i)* and transforming it to a $\langle key, value \rangle$ pair: $\langle (AllComb_{(K)}$, List of the features of $x_i)$, List of id_i of $x_i \rangle$. Based on this transformation, the combination of features and their vector of features define the

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

key and the identifiers id_i of the data items x_i define the value. After that, the $foldByKey()$ operation is applied to merge all values of each key in the transformed RDD output, i.e., all the identifiers id_i of the data items x_i that have the same combination of features with their corresponding vector of features ($AllComb_{(K)}$, List of the features of x_i). This is to represent the sought $IND(AllComb_{(K)})$.

Algorithm 6: Calculate $IND(AllComb_{(K)})$

Inputs: $Cl_s(K)$: sub-information tables
$AllComb_{(K)}$: set of all possible combinations of K
Output: $IND(AllComb_{(K)})$

1: Map the $Cl_s(K)$ based on its format *(id_i of x_i, List of the features of x_i, Class d_i of x_i)* and generate the new format as a key-value pair $\langle$($AllComb_{(K)}$, List of the features of x_i), List of id_i of $x_i\rangle$
2: Merge the values of each generated key using the $foldByKey()$ operation
3: Return $AllComb_{(K)} : IND(AllComb_{(K)})$

At this algorithmic phase, LSH-dRST arranges the set of attributes that will be selected in the next algorithmic distributed jobs/steps. As presented in Algorithm 7, the dependency degree, defined as $\gamma(AllComb_{(K)}, D)$, of every single feature combination is calculated. This is achieved based on the definition of γ given in Section 3.2 and with respect to the definition of the indiscernibility of a combination of features $IND(AllComb_{(K)})$ given above. For this task, the distributed job requires three input parameters, which are the calculated indiscernibility relations $IND(D)$, the $IND(AllComb_{(K)})$ and the set of all attribute combinations $AllComb_{(K)}$. For every element $AllComb_{(K)_i}$ in $AllComb_{(K)}$, and by using the $intersection()$ transformation, the job tests first if the intersection of every $IND(d_i)$ of $IND(d)$ with each element $IND(AllComb_{(K)_i}$ in $IND(AllComb_{(K)})$ holds all the elements in the latter parameter. This process refers to the calculation of the lower approximation as detailed in Section 3.2. We name the length of the resulting intersection as *LengthIntersect*. If the condition is satisfied, then a score, which is equal to the length of the elements resulting from the generated intersection, i.e., *LengthIntersect*, is assigned, else a 0 value is given.

After that a $reduce$ function is applied over the different $IND(D)$ elements together with a $sum()$ function applied on the calculated scores, which are based on the elements having the same $IND(d_i)$. This operation is followed by a second $reduce$ function, which is applied over the different $IND(AllComb_{(K)})$ elements together with a $sum()$ function applied on the previously calculated results, which are indeed based on the elements having the same $AllComb_{(K)_i}$.

The latter output refers to the dependency degrees: $\gamma(AllComb_{(K)}, D)$. This distributed job generates two outputs namely the set of dependency degrees $\gamma(AllComb_{(D)}, D)$ of the attribute combinations $AllComb_{(K)}$ as well as their associated sizes $Size_{(AllComb_{(K)})}$.

After that, the algorithm searches for the maximum dependency value $DEP_{max}(AllComb_{(K)})$ among all the $\gamma(AllComb_{(K)}, D)$ generated by applying the $max()$ function operated on the given RDD input, referred to as RDD[$AllComb_{(K)}$, $Size_{(AllComb_{(K)})}$, $\gamma(AllComb_{(K)}, D)$]. Specifically, the $max()$ function will be applied on the third argument of the given RDD, i. e.,

Algorithm 7: Generate $DEP(AllComb_{(K)})$

Inputs: $AllComb_{(K)}$: set of all possible combinations of K
$IND(D)$: indiscernibility relation for the decision class D
$IND(AllComb_{(K)})$: indiscernibility relation for every $AllComb_{(K)}$
Outputs: $\gamma(AllComb_{(K)}, D)$: dependency degrees of the feature combinations $AllComb_{(K)}$
$Size_{(AllComb_{(K)})}$: size of the dependency degrees of the feature combinations $AllComb_{(K)}$

```
for each element AllComb_(K)_i in AllComb_(K) do
    for each element IND(d_i) in IND(D) do
        for each element IND(AllComb_(K))_i in IND(AllComb_(K)) do
            Apply the intersection() transformation over IND(d_i) and IND(AllComb_(K))_i
            Get the length of the resulting intersection that we name as LengthIntersect
            If LengthIntersect = length of IND(AllComb_(K))_i
            Score = LengthIntersect
            Else Score = 0
            End if
        end
        Apply a reduce function over IND(D) based on a sum() function on the calculated scores, which are based on the elements having the same IND(d_i)
    end
    Apply a reduce function over AllComb_(K) based on a sum() function on the calculated previous results, which are based on the elements having the same AllComb_(K)_i
end
```

Return $AllComb_{(K)} : \gamma(AllComb_{(K)}, D)$, $Size_{(AllComb_{(K)})}$

$\gamma(AllComb_{(K)}, D)$.

Algorithm 8: Select $DEP_{max}(AllComb_{(K)})$

Input: RDD[$AllComb_{(K)}$, $Size_{(AllComb_{(K)})}$, $\gamma(K, AllComb_{(K)})$]: RDD presenting all $AllComb_{(K)}$ and their characteristics
Output: $MaxDependency$: baseline value for feature selection

1: Apply the $max()$ function on the third argument of the given RDD: $\gamma(AllComb_{(K)}, D)$
2: Return $MaxDependency$

As presented in Algorithm 8, the output of this job, defined as $MaxDependency$, corresponds to (i) the dependency of the whole feature set representing the $Cl_s(K)$ and (ii) the dependency of all the possible feature combinations satisfying the constraint $\gamma(AllComb_{(K)}, D) = \gamma(K, D)$ (outlined in Section 3.2). The output $MaxDependency$ represents the baseline value for feature selection.

Once $MaxDependency$ is defined, the algorithm keeps the set of all combinations having the

same dependency degrees as $MaxDependency$, i.e., $\gamma(AllComb_{(K)}, D) = MaxDependency$. This job is achieved by applying a $filter()$ function. At this stage, the algorithm eliminates in each computation level the useless attributes that may negatively affect the performance of any learning algorithm. This distributed job is presented in Algorithm 9.

Algorithm 9: Filter $DEP_{max}(AllComb_{(K)})$

Inputs: RDD[$AllComb_{(K)}$, $Size_{(AllComb_{(K)})}$, $\gamma(AllComb_{(K)}, D)$]: RDD presenting all $AllComb_{(K)}$ and their characteristics
$MaxDependency$: baseline value for feature selection
Output: Filtered-RDD[$AllComb_{(K)}$, $Size_{(AllComb_{(K)})}$, $\gamma(K, AllComb_{(K)})$]: filtered RDD and its characteristics

1: Apply the $filter()$ function on the input RDD in a way to select all combinations having a dependency that is equal to $MaxDependency$: $\gamma(AllComb_{(K)}) = MaxDependency$
2: Return the filtered RDD: Filtered-RDD[$AllComb_{(K)}$, $Size_{(AllComb_{(K)})}$, $\gamma(AllComb_{(K)}, D)$]

At a final stage, and using the results generated from the previous step, which is the input of Algorithm 10, the algorithm applies first the $min()$ operator to look for the minimum number of features among all the $Size_{(AllComb_{(K)})}$. Specifically, the $min()$ operator will be applied to the second argument of the given RDD. Once determined, a result that we name $minNbF$, the algorithm applies a $filter()$ method to only keep the set of combinations having the same minimum number of features as $minNbF$. This is achieved by satisfying the full reduct constraints highlighted in Section 3.2: $\gamma(AllComb_{(K)}, D) = \gamma(K, D)$ while there is no $AllComb_{(K')} \subset AllComb_{(K)}$ such that $\gamma(AllComb_{(K')}, D) = \gamma(AllComb_{(K)}, D)$. Every combination that satisfies this constraint is evaluated as a possible minimum reduct set. The attributes of the reduct set describe all concepts in the sub-information table $Cl_s(K)$.

Algorithm 10: Filter $NbF_{min}(DEP_{max}(AllComb_{(K)}))$

Input: RDD[$AllComb_{(K)}$, $Size_{(AllComb_{(K)})}$, $\gamma(K, AllComb_{(K)})$]: RDD presenting all $AllComb_{(K)}$ and their characteristics
Output: Filtered-RDD[$AllComb_{(K)}$, $Size_{(AllComb_{(K)})}$, $\gamma(K, AllComb_{(K)})$]: viable minimum reduct set and its characteristics

1: Apply the $min()$ function on the input filtered RDD second argument: $Size_{(AllComb_{(K)})}$ to get $minNbF$
2: Apply a $filter()$ function on the input RDD while satisfying the condition $Size_{(AllComb_{(K)})} = minNbF$
3: Return $Reduct$ = List of selected K

5.3. LSH-dRST: a working example

We apply LSH-dRST to an example of an information table, $T_{RDD}(C)$, which is presented in Table 1. By assuming that the considered $T_{RDD}(C)$ is a big data set, the information table is defined via a universe $U = \{x_1, x_2, \ldots, x_8\}$, which refers to the set of data instances (items), a large conditional feature set C = {a, b, c, d, e, f, g, l} that includes all the features of the information table $T_{RDD}(C)$ and a decision feature X of the given learning problem. X refers to the label (or class) of each $T_{RDD}(C)$ data item and is defined as follows: $X = \{1, 2\}$. C presents the conditional attribute pool from where the most significant attributes will be selected.

Table 1. Toy dataset.

$x \in U$	a	b	c	d	e	f	g	l	X
x_1	1	2	4	0	1	2	5	3	1
x_2	0	3	3	2	1	3	2	2	2
x_3	2	3	1	3	3	1	6	1	2
x_4	1	1	2	1	2	3	5	3	2
x_5	0	2	0	1	2	4	2	3	1
x_6	1	1	2	4	3	1	3	1	2
x_7	2	2	1	3	2	4	2	2	2
x_8	1	2	0	2	2	4	7	0	1

Independently from the set of conditional features C, LSH-dRST computes the indiscernibility relation for the decision class X. We define the indiscernibility relation as $IND(X)$: $IND(X_i)$. LSH-dRST will calculate $IND(X)$ for each decision class X_i by associating the same data items (instances) $T_{RDD}(C)$ that are expressed in the universe U and that belong to the same decision class X_i. Based on the Apache Spark framework and by applying Algorithm 3, line 1, we get the following outputs from the different Apache Spark data splits, which are presented in Table 2 and in Table 3:

$x \in U$	a	b	c	d	e	f	g	l	X
x_1	1	2	4	0	1	2	5	3	1
x_2	0	3	3	2	1	3	2	2	2
x_3	2	3	1	3	3	1	6	1	2
x_4	1	1	2	1	2	3	5	3	2

Table 2. Toy dataset - Split 1.

- **From Split 1:**
  - $\langle 1, x_1 \rangle$
  - $\langle 2, x_2 \rangle$

$x \in U$	a	b	c	d	e	f	g	l	X
x_5	0	2	0	1	2	4	2	3	1
x_6	1	1	2	4	3	1	3	1	2
x_7	2	2	1	3	2	4	2	2	2
x_8	1	2	0	2	2	4	7	0	1

Table 3. Toy dataset - Split 2.

  - $\langle 2, x_3 \rangle$
  - $\langle 2, x_4 \rangle$

- **From Split 2:**
  - $\langle 1, x_5 \rangle$
  - $\langle 2, x_6 \rangle$
  - $\langle 2, x_7 \rangle$
  - $\langle 1, x_8 \rangle$

After that, and by applying Algorithm 3, line 2, we get the following output, which refers to the indiscernibility relation of the class $IND(X)$:

- $1, \{x_1, x_5, x_8\}$
- $2, \{x_2, x_3, x_4, x_6, x_7\}$

In this example, we assume that the number of buckets $B = 2$ and $K = 2$. By applying LSH, Algorithm 2, to the set of features C = {a, b, c, d, e, f, g, l}, the output is a set of data items based on the following similar and close features:

- Bucket 1 ($T_{RDD_{b=1}}$): {d, e, f, g}
- Bucket 2 ($T_{RDD_{b=2}}$): {a, b, c, l}

By applying KNN to each bucket, , Algorithm 4, the following sub-information tables are built:

- Bucket 1 ($T_{RDD_{b=1}}$): {d, e, f, g}
  - $Cl_{s=1}(k = 2)$: {e, f}
  - $Cl_{s=2}(k = 2)$: {d, g}
- Bucket 2 ($T_{RDD_{b=2}}$): {a, b, c, l}
  - $Cl_{s=3}(k = 2)$: {a, b}
  - $Cl_{s=4}(k = 2)$: {c, l}

Based on these assumptions, LSH-dRST will be applied to every $Cl_s(k)$. In what follows, we will only focus on $Cl_{s=1}(k=2)$ as an example. The same process will be applied to the rest of the $Cl_s(k)$, i.e., $Cl_{s=2}(k)$, $Cl_{s=3}(k)$ and $Cl_{s=4}(k)$. The following partitions and splits based on Apache Spark are obtained for $Cl_{s=1}(k=2)$ (Table 4 and Table 5):

$x \in U$	e	f	X
x_1	1	2	1
x_2	1	3	2
x_3	3	1	2
x_4	2	3	2

Table 4. Split 1.

$x \in U$	e	f	X
x_5	2	4	1
x_6	3	1	2
x_7	2	4	2
x_8	2	4	1

Table 5. Split 2.

Based on Split 1, and by applying Algorithm 5, which aims to generate all the possible combinations $AllComb_{(K)}$ of the set of K attributes, the output from both Apache Spark splits is the following:

- e
- f
- e, f

In its third distributed job, LSH-dRST calculates the indiscernibility relation $IND(AllComb_{(K)})$ for every created combination, i.e., the indiscernibility relation of every element in the output of the previous step (Algorithm 5). By applying Algorithm 6 and based on both Apache Spark splits, the output is the following:

- **From Split 1:**
  - e, $\{x_1, x_2\}$, $\{x_3\}$, $\{x_4\}$
  - f, $\{x_1\}, \{x_2, x_4\}, \{x_3\}$
  - e, f, $\{x_1\}$, $\{x_2\}$, $\{x_3\}$, $\{x_4\}$

- **From Split 2:**
  - e, $\{x_5, x_7, x_8\}$, $\{x_6\}$
  - f, $\{x_5, x_7, x_8\}$, $\{x_6\}$
  - e, f, $\{x_5, x_6, x_7\}$, $\{x_6\}$

In a next stage, and by using the previous output as well as $IND(X)$, LSH-dRST computes the dependency degrees $\gamma(AllComb_{(K)}, D)$ of each attribute combination as described in Algorithm 7. This distributed job generates two outputs namely the set of dependency degrees $\gamma(AllComb_{(K)}, D)$ of the attribute combinations $AllComb_{(K)}$ as well as their associated sizes $Size_{(AllComb_{(K)})}$. The output from both splits is the following:

- e, $2, 1$
- f, $5, 1$
- e, f, $5, 2$

Once all the dependencies are calculated, in Algorithm 8, Sp-RST looks for the maximum value of the dependency among all the computed $\gamma(AllComb_{(K)}, D)$. The maximum dependency reflects the baseline value for the feature selection task. The output is the following:

- 5

In a next step, LSH-dRST performs a filtering process to only keep the set of all combinations, which have the same dependency degrees, as the already selected dependency baseline value ($MaxDependency = 5$), i.e., $\gamma(AllComb_{(K)}, D) = MaxDependency = 5$. By applying Algorithm 9, the following output is obtained:

- f, $5, 2$
- e, f, $5, 2$

In fact, through these computations, the algorithm removes in each level the unnecessary attributes that may negatively influence the performance of any learning algorithm. At a final stage, by using the results generated from the previous step and applying Algorithm 10, LSH-dRST looks for the minimum number of features among all the $Size_{(AllComb_{(K)})}$. Once determined ($minNbF = 1$), the algorithm only keeps the set of combinations having the same minimum number of features as $minNbF$. The filtered selected features define the reduct set and describe all concepts in the initial $Cl_{s=1}(K)$ training data set. The output of Algorithm 10, which presents the reduct ($RED_s(K, D)$) for $Cl_{s=1}(K)$, is the following:

- f

The same calculations will be applied to the rest of Cl ($Cl_{s=2}(k)$, $Cl_{s=3}(k)$ and $Cl_{s=4}(k)$). At this stage, different reducts – $Cl_{s=2}(k)$: $RED = \{d\}$, $Cl_{s=3}(k)$: $RED = \{a, b\}$, $Cl_{s=4}(k)$: $RED = \{c\}$

– are generated from the different B partitions and from the different Cl_s. With respect to Algorithm 1, line 15, a union of the obtained results is required to represent the initial big information table $T_{RDD}(C)$, i.e., Table 1. The final output is $Reduct = \{a, b, c, d, f\}$. LSH-dRST could reduce the big information table presented in Table 1 from $T_{RDD}(C)$ to $T_{RDD}(Reduct)$. The output is presented in Table 6.

$x \in U$	a	b	c	d	f	X
x_1	1	2	4	0	2	1
x_2	0	3	3	2	3	2
x_3	2	3	1	3	1	2
x_4	1	1	2	1	3	2
x_5	0	2	0	1	4	1
x_6	1	1	2	4	1	2
x_7	2	2	1	3	4	2
x_8	1	2	0	2	4	1

Table 6. Reduct.

6. Experimental Setup

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, and hence data dependency.

The LSH-dRST algorithm is implemented in Scala 2.11 within the Spark 2.1.1 framework. Based on the experiments conducted in [11], 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 parameter K 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[21], a large-scale testbed for experiment-driven research. Within this testbed, we used dual 8 core Intel Xeon E5-2630v3 CPUs and 128 GB memory. Since the study does not require a scalable version of the classifiers, these experiments are run on a standard

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

laptop configuration with a 3.1 GHz Dual-Core Intel Core i7 CPU, 16 GB 1867 MHz DDR3 RAM, SSD storage, 64-bit, macOS Catalina.

Our analysis first focuses on the number of features selected and the scalability of our algorithm. We evaluate the performance of LSH-dRST using the *speedup*, *sizeup*, and *scaleup* criteria introduced in [45] (see Section 6.2).

We then show that the improvement in performance does not decrease the feature selection ability by performing model evaluation using a Naive Bayes and a Random Forest classifier on the original data set, the reduced data sets produced by LSH-dRST and some other feature selection techniques (see Section 6.3). We use the scikit-learn Random Forest implementation[22] with the following parameters: n_estimators $= 1000$, n_jobs $= -1$, and oob_score $=$ True. A Stratified 10-Folds cross-validator[23] is used for all our conducted experiments. Moreover, we use the Naive Bayes implementation from Weka 3.8.2[24], again with 10-fold cross-validation.

LSH-dRST makes use of randomization 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.

6.1. 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 comparison with the standard distributed RST version [11], we chose the Amazon Commerce reviews data set from the UCI machine learning repository [46]. 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 [11]. 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.

We have discretized the data using the Equal-Width Intervals method. It separates all possible values into k intervals of the same width where width is defined as $(\max \text{ value} - \min \text{ value})/k$. We have used $k = 10$ in our experiments. Note that the number of distinct values for the conditional attributes in the data ranges from 3 (e. g., it+is_an) to 298 (e. g., _).

We demonstrate the scalability of our approach by considering subsets of this data set in terms of attributes. To be more precise, we have created five additional data sets by randomly choosing 1 000, 2 000, 4 000, 6 000, and 8 000 out of the original 10 000 attributes. We use these sets to evaluate our

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

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

[24] http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayes.html

proposed method as discussed in Section 6.2 and refer to them as Amazon1000, Amazon2000, ..., Amazon10000 in the following.

6.2. Evaluation Metrics

To evaluate the scalability of the parallel LSH-dRST, we consider the standard metrics, which are the *speedup*, the *scaleup*, and the *sizeup* from literature [45]. These are defined as follows:

- For the speedup, we keep the size of the data set constant (where size is measured by the number of features, i. e., we use the original data set with 10 000 features) and increase the number of nodes. For a system with m nodes, the speedup is defined as:

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

  An ideal parallel algorithm has linear speedup: the algorithm using m nodes solves the problem in the order of m times faster than the same algorithm using a single node. However, this is difficult to achieve in practice due to startup and communication cost as well as interference and skew [45], which may lead to a sub-linear speedup.

- The sizeup keeps the number of nodes constant and measures how much the runtime increases as the data set is increased by a factor of m:

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

  To measure the sizeup, we use the smaller databases described in Section 6.1. We use 1 000 features as a baseline and consider 2 000, 4 000, 6 000, 8 000, and 10 000 features, respectively. A parallel algorithm with a linear sizeup has a very good sizeup performance: considering a problem that is m times larger than a baseline problem, the algorithm requires in the order of m times more runtime for the larger problem.

- The scaleup evaluates the ability to increase the number of nodes and the size of the data set simultaneously:

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

  Again, we use the sub-data set with 1 000 features as a baseline. Here, a scaleup of 1 implies 'linear' scaleup, which similarly to linear speedup is difficult to achieve.

For the model evaluation, we use the standard measures, which are the accuracy, precision, recall, and F1 score to compare the quality of the LSH-dRST selected feature set with other feature selection methods. Note that these metrics are formally defined for binary classification and thus, need to be adjusted to be used with the Amazon dataset, which has 30 classes.

In the following we provide the formal definitions for the binary case. For the case with more than two classes we use a standard weighted average approach. This approach first calculates the metrics

for each class separately. Afterwards it determines the weighted average over all classes using the number of true instances for each class as weights. This has the advantage to take potential class imbalances into account.

The metrics definitions for the binary case are as follows (where TP: True positive, TN: True negative, FP: False positive, and FN: False negative):

- Precision: measures the ratio of correctly predicted positive observations to the total predicted positive observations, and is defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP + FP}}$$

- Recall: measures the ratio of correctly predicted positive observations to the all observations in actual class - yes, and is defined as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP + FN}}$$

- Accuracy: measures the ratio of correctly predicted observation to the total observations, and is defined as follows:

$$\text{Accuracy} = \frac{\text{TP + TN}}{\text{TP + FN + TN + FP}}$$

- F1 score: is the weighted average of Precision and Recall. F1 score is defined as follows:

$$\text{F1 score} = 2 * \frac{\text{Recall * Precision}}{\text{Recall + Precision}}$$

6.3. Other Feature Selection Techniques

We compare LSH-dRST with a number of other feature selection techniques from Weka 3.8.2[25]

- CfsSubsetEval: considers the individual predictive ability of each feature along with the degree of redundancy between them
- ChiSquaredAttributeEval: computes the value of the chi-squared statistic with respect to the class
- ConsistencySubsetEval: considers the level of consistency in the class values when the training instances are projected onto the subset of attributes
- CorrelationAttributeEval: measures the correlation (Pearson's) between it and the class
- CVAttributeEval: first creates a ranking of attributes based on the Variation value, then divides into two groups, last using Verification method to select the best group
- GainRatioAttributeEval: measures the gain ratio with respect to the class

[25] `https://www.cs.waikato.ac.nz/~ml/weka`

- InfoGainAttributeEval: measures the information gain with respect to the class
- ReliefFAttributeEval: repeatedly samples an instance and considers the value of the given attribute for the nearest instance of the same and different class
- SignificanceAttributeEval: computes the Probabilistic Significance as a two-way function (attribute-classes and classes-attribute association)
- SymmetricalUncertAttributeEval: evaluates the symmetrical uncertainty with respect to the class

and Smile[26]

- SumSquaresRatio (AttributeEval): measures the ratio of between-groups to within-groups sum of squares

These methods include both, attribute and subset evaluation methods. For subset evaluation we use a 'Best First' greedy search method. For attribute evaluation, we need to either provide a threshold or a number of features to be selected. 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 and additionally use 0 as a threshold. We determine the sets of features selected by these methods and then perform model evaluation with a Random Forest and Naive Bayes classifier as discussed previously.

7. Results and Analysis

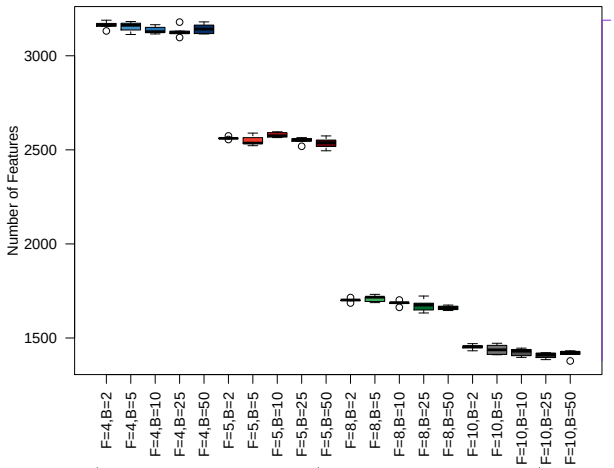
In this section, we will discuss our results. We first consider the features selected by LSH-dRST (Section 7.1). After that we look into the runtime and scalability of LSH-dRST (Section 7.2). Finally, we investigate the quality of the feature selection and compare our results with the previously introduced algorithm creating random partitions (Sp-RST) and other feature selection techniques as discussed above (Section 7.3).

7.1. Selected Features

We plot the number of features selected over 10 runs as boxplots and provide values for average and standard deviation in Figure 5. 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 on average 3 145 features for $F = 4$, 2 555 for $F = 5$, 1 686 for $F = 8$, and 1 427 for $F = 10$. Again, the number of buckets hardly has any impact. For Sp-RST the results reported in [11] were much more erratic with no clear tendency based on the parameter setting and numbers ranging between 1 600 and 6 200. We will use these observed numbers to parameterize other feature selection techniques when performing our model evaluation.

In a second step, we have a closer look at the features selected by LSH-dRST. For this, we perform 5 independent runs for each parameter setting and look at the number of times each of the 10 000

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



	B=2	B=5	B=10	B=25	B=50
F=4	3162.6 (20.94)	3153.4 (28.18)	3136.6 (20.60)	3129.6 (30.31)	3143.8 (27.93)
F=5	2562.6 (6.95)	2549.2 (27.29)	2579.0 (13.84)	2548.2 (17.94)	2535.4 (30.44)
F=8	1700.8 (10.33)	1709.8 (18.28)	1684.8 (14.10)	1673.0 (34.73)	1659.8 (12.11)
F=10	1452.2 (14.04)	1438.8 (27.55)	1423.6 (21.20)	1407.0 (16.00)	1414.2 (21.63)

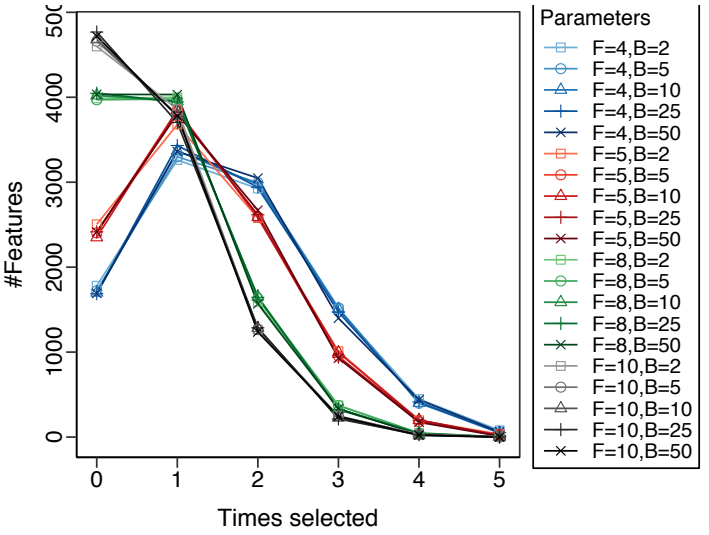
Figure 5. The number of features in the reduced data set. The boxplot visualizes the results over 10 independent runs for all parameter settings. Average and standard deviation for these runs are listed in the table.

features is selected. We plot the results in Figure 6 and see that only very few features are selected in every run while a considerable number of features is never or only once selected. This is a very different picture from our original algorithm Sp-RST where features were typically selected in all or none of the runs. It is also interesting to note that the number of buckets B has hardly any influence on this result—the shape of the curves in Figure 6 is mostly determined by the value of F. We therefore only provide additional averages for all values of F in Figure 6.

We emphasize that selecting different features in different runs is not a weakness of our proposed method. As pointed out in Section 3.2 more than one reduct may exist and selecting an arbitrary one among these is appropriate.

7.2. Runtime and Scalability

As discussed in the previous section, we consider standard measures for parallel algorithms to analyze the scalability of our approach, i. e., speedup (Section 7.2.1), sizeup (Section 7.2.2), and scaleup (Section 7.2.3). In addition, we consider the runtime of LSH-dRST in more detail, e. g., how the runtime is split between the LSH and the RST phase. We provide the actual runtimes for the different phases in Tables 11 and 12 in the appendix. The total runtimes for different numbers of nodes and the datasets



	0	1	2	3	4	5
$F = 4$	1716.4	3344.8	2977	1481	419.8	61
$F = 5$	2417.8	3798.2	2608	965	189.8	21.2
$F = 8$	4011.8	3981.6	1617.4	346	42.2	1
$F = 10$	4685	3785.4	1266.8	235.4	26.4	1

Figure 6. Number of features selected 0, 1, . . . , 5 times in five independent runs of LSH-dRST. Averages for each value of F are listed in the table.

described in Section 6.1 can be found in Tables 7, 8, 9 and 10 in the appendix.

Figures 7 and 8 visualize the runtimes for the different phases of LSH-dRST, both in terms of absolute and relative values. We observe that for some parameter settings the LSH part of LSH-dRST is more time-consuming than the RST part, but for others it is less time-consuming: For $F = 10$, the time for the LSH part of our algorithm is negligible for all settings of B and all numbers of nodes. For $F = 8$, this is only the case for more than 2 buckets (i.e., $B \neq 2$). For $F = 4$ and $F = 5$, the time taken for the LSH part is larger or equal to the time taken for RST in most cases. In general, we can observe that the share of the LSH part is large if more nodes are used, for smaller number of buckets B and for smaller values of F. We also observe that the runtime for the RST part generally peaks if two nodes are used and decreases for larger number of nodes. We conclude that the overhead caused by parallelization is not yet compensated for such a smaller number of nodes.

7.2.1. Speedup Analysis

We plot the speedup in Figure 9 and the actual runtime for the six data sets discussed in Section 6.1 in Figure 10. We see that the speedup for most parameter settings is very similar. However, setting $F = 8$ for the datasets with 8 000 and 10 000 features 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

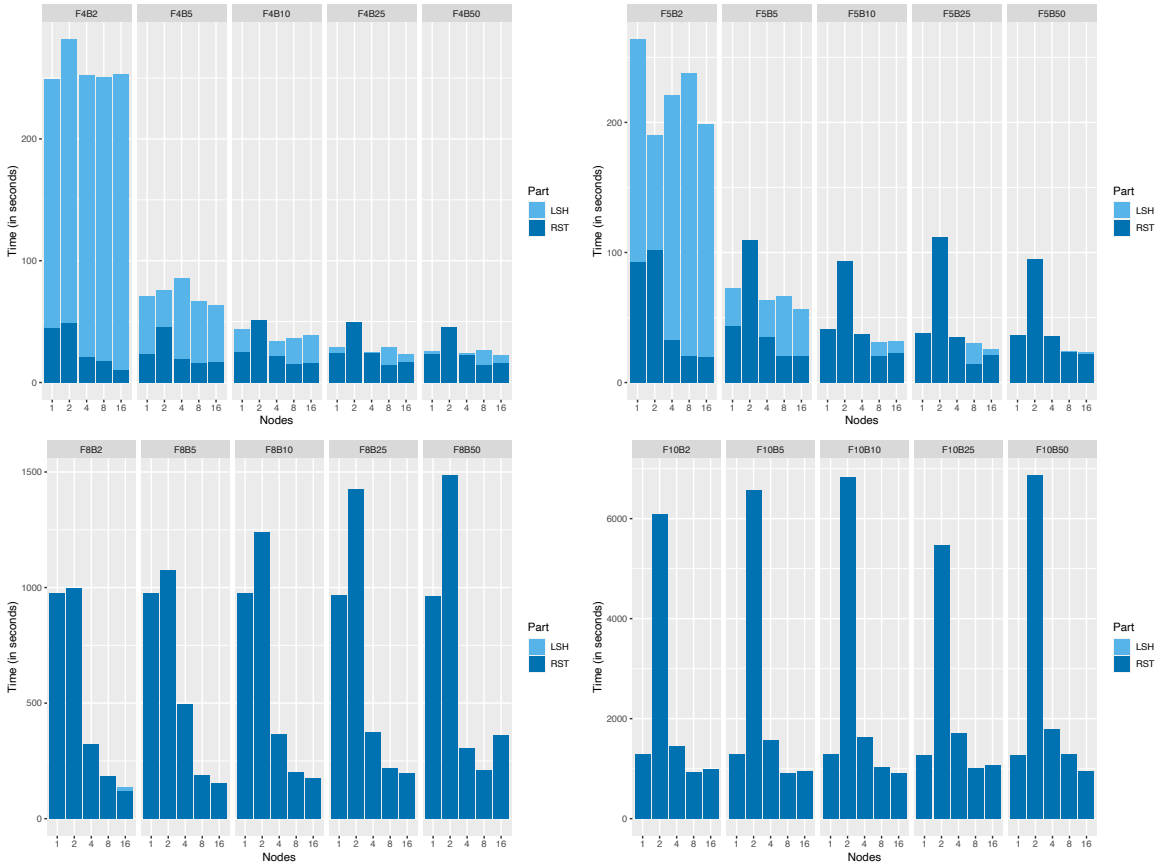


Figure 7. Absolute runtimes for different parameters of LSH-dRST, split by LSH and RST part. Please note the different y-scales for different values of F.

the number of buckets does not have a considerable influence on the speedup, but the number of sub-information tables F does. The latter is expected since the execution time grows exponentially with respect to 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 [45].

7.2.2. Sizeup Analysis

To measure the sizeup 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 1, 2, 4, 8, and 16 nodes in Figure 11 and the actual runtimes for different numbers of nodes in Figure 12. We see that our method has sub-linear sizeup for most parameter settings if at least 4 nodes are used, 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

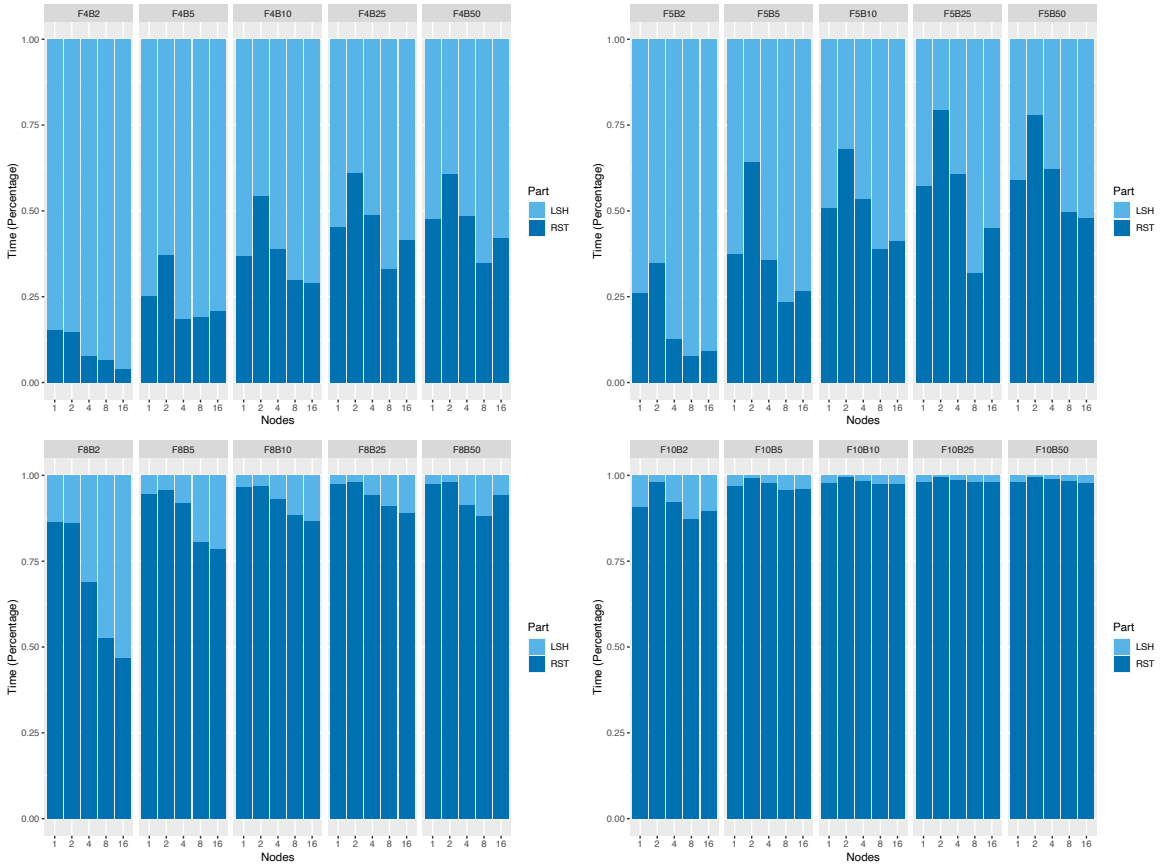


Figure 8. Relative runtimes for different parameters of LSH-dRST, split by LSH and RST part.

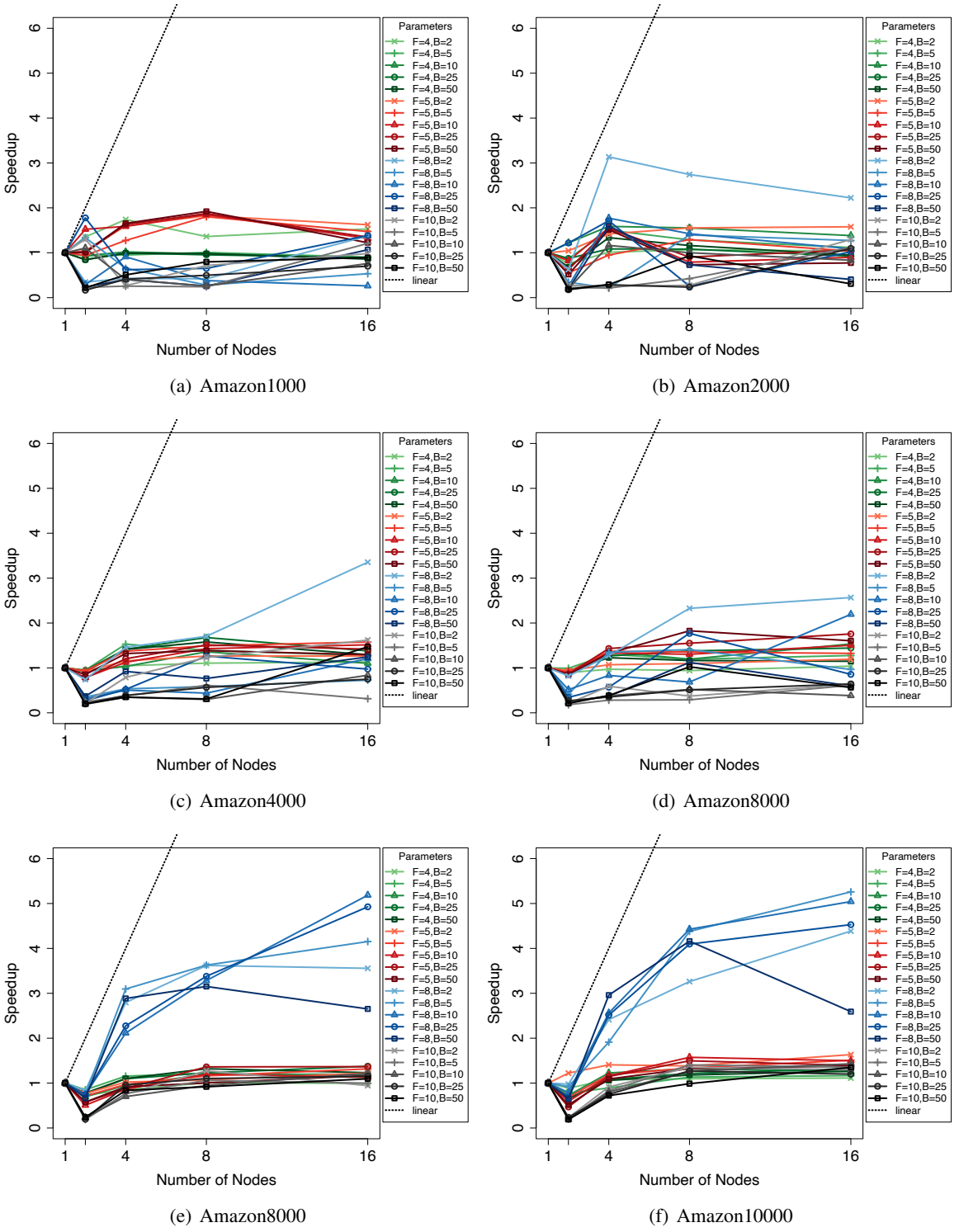
of features) with only two buckets ($B = 2$). Results for 1 or 2 nodes look different, but it should be noted that these settings are less relevant in the context of parallel algorithms.

7.2.3. Scaleup Analysis

We use the sub-data sets previously described with 1 000 features as a baseline and plot the results in Figure 13. 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.

7.3. Model Evaluation

We perform model evaluation with two classifiers, Random Forest and Naive Bayes as described previously. We present boxplots of the results for all metrics (accuracy, precision, recall, F1 score)



(a) Amazon1000

(b) Amazon2000

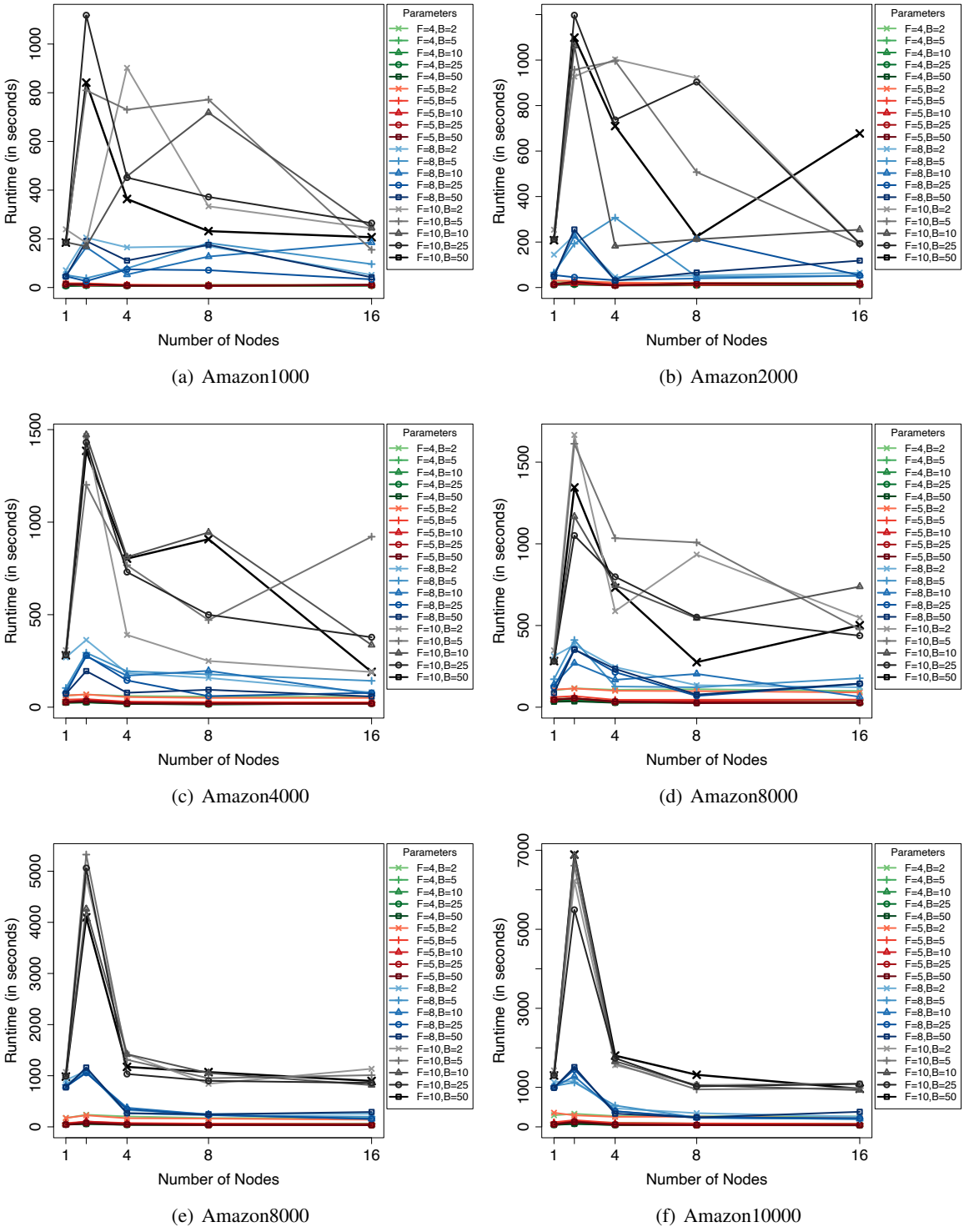
(c) Amazon4000

(d) Amazon8000

(e) Amazon8000

(f) Amazon10000

Figure 9. Speedup.



(a) Amazon1000

(b) Amazon2000

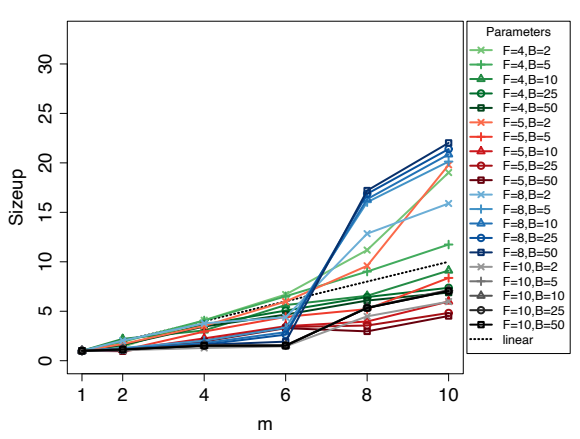
(c) Amazon4000

(d) Amazon8000

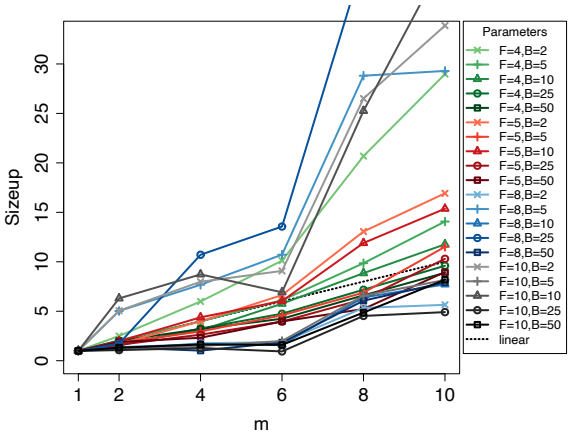
(e) Amazon8000

(f) Amazon10000

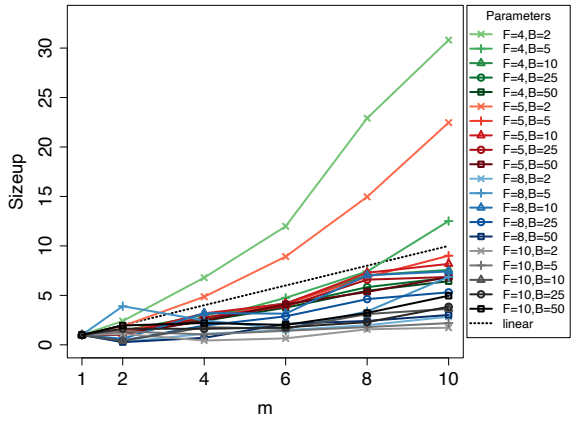
Figure 10. Runtimes (in seconds) for the six data sets discussed in Section 6.1.



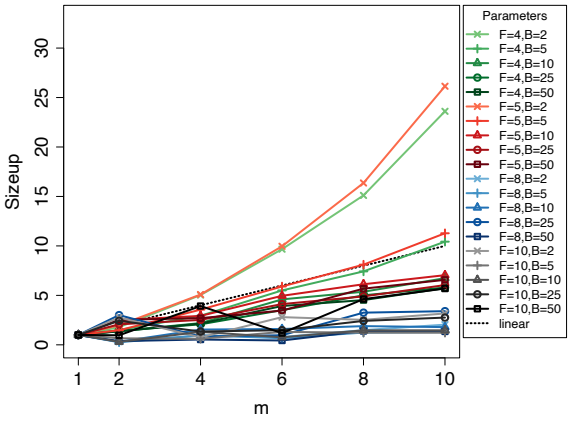
(a) 1 node



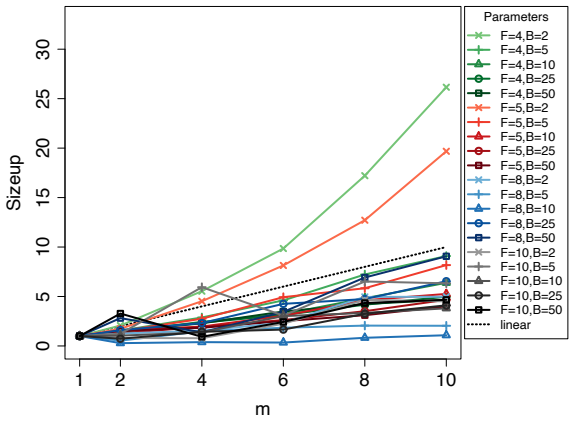
(b) 2 nodes



(c) 4 nodes

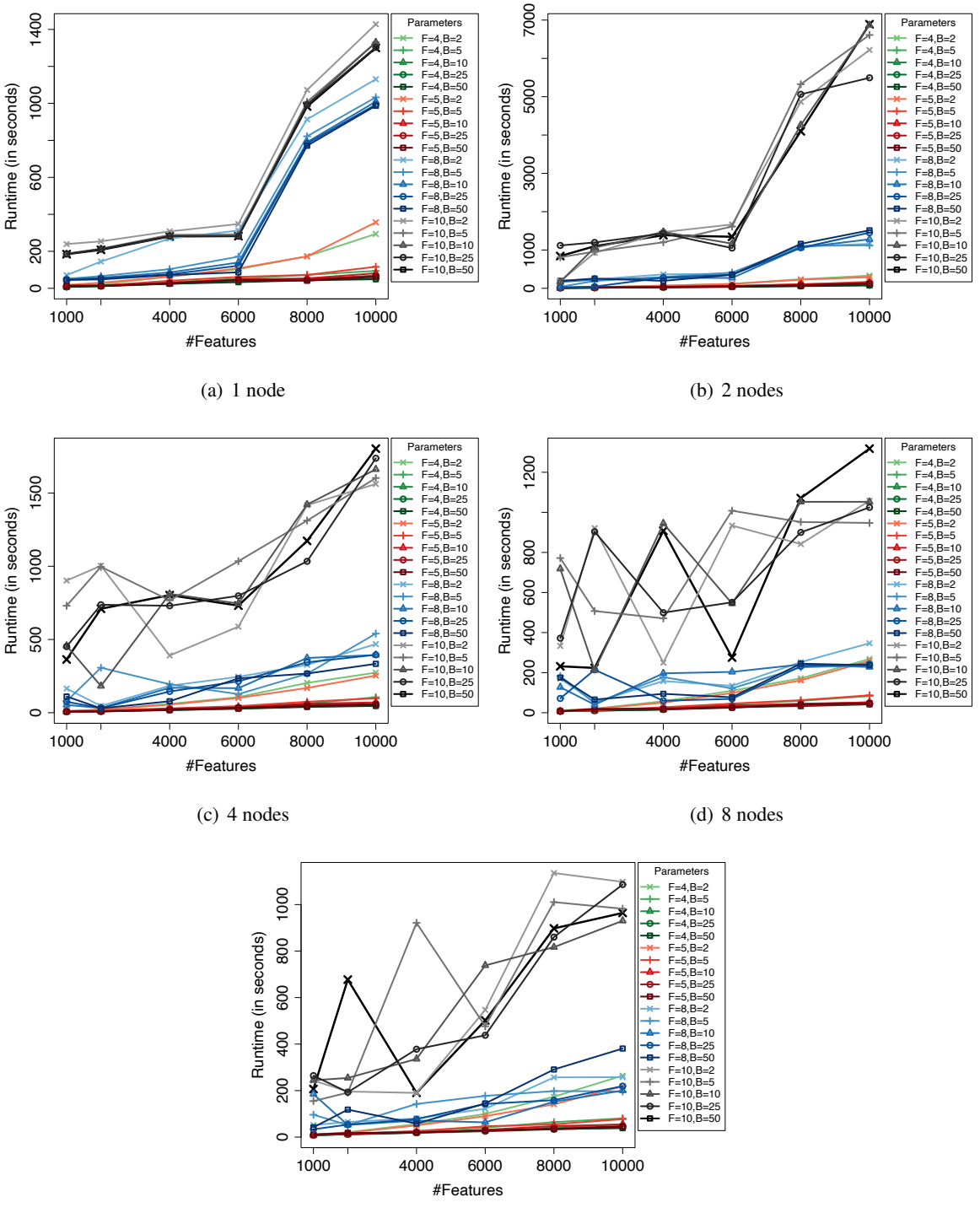


(d) 8 nodes



(e) 16 nodes

Figure 11. Sizeup.



(a) 1 node

(b) 2 nodes

(c) 4 nodes

(d) 8 nodes

(e) 16 nodes

Figure 12. Runtimes (in seconds) for different numbers of nodes.

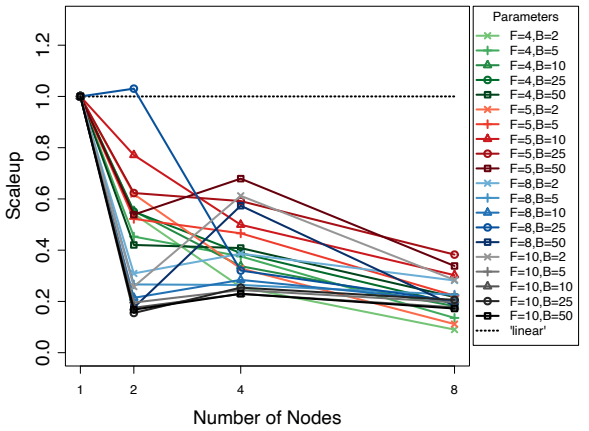


Figure 13. Scaleup.

as well as runtimes in the main text. The corresponding raw data can be found in the appendix: Section 9.1 for runtimes and Sections 9.3 and 9.4 for classification metrics of Random Forest and Naive Bayes, respectively.

As discussed in Section 6.3 some of the other feature selection methods have to be parameterized by either defining a threshold or the number of features to be selected. For these methods we have used five settings: a threshold of 0 (the standard setting) and four different numbers of features. The latter numbers are motivated by the number of features selected by LSH-dRST with different settings of F (see Section 7.1), i. e., 1 427, 1 686, 2 555, and 3 145 features. The relevant parameter settings are displayed in brackets after the name of the corresponding method.

7.3.1. Random Forest Classifier

We show the results of 10 runs of random forest on the different reduced data sets in Figures 14, 15, 16 and 17 (classification metrics) as well as Figure 18 (runtime). The corresponding raw data for classification metrics can be found in Tables 13, 14 and 15 in the appendix. The raw data for runtimes is displayed in Tables 7, 8, 9 and 10 in the appendix.

We observe that LSH-dRST has comparable performance to most other feature selection techniques and outperforms some of them.[27] The only exception is the SumSquaresRatio method, which outperforms all other methods for 1 686, 2 555, and 3 145 features. The overall best result is achieved by SumSquaresRatio with 3 145 features (0.9513 accuracy, 0.9534 precision, 0.9577 recall, and 0.9345 F1 score). The best parameter setting for LSH-dRST is $F = 4$ and $B = 50$: 0.7402 accuracy, 0.7626 precision, 0.7402 recall, 0.7326 F1 score.

We also see that the classification result is quite stable with respect to the parameter settings in LSH-dRST. The slightly better results are generally achieved for smaller values of F, with all results for $F = 4$ (independent of the value for B) outperforming all other parameter settings. We conclude

[27] All observations hold for all evaluation metrics used, i.e., accuracy, recall, precision, and F1 score.

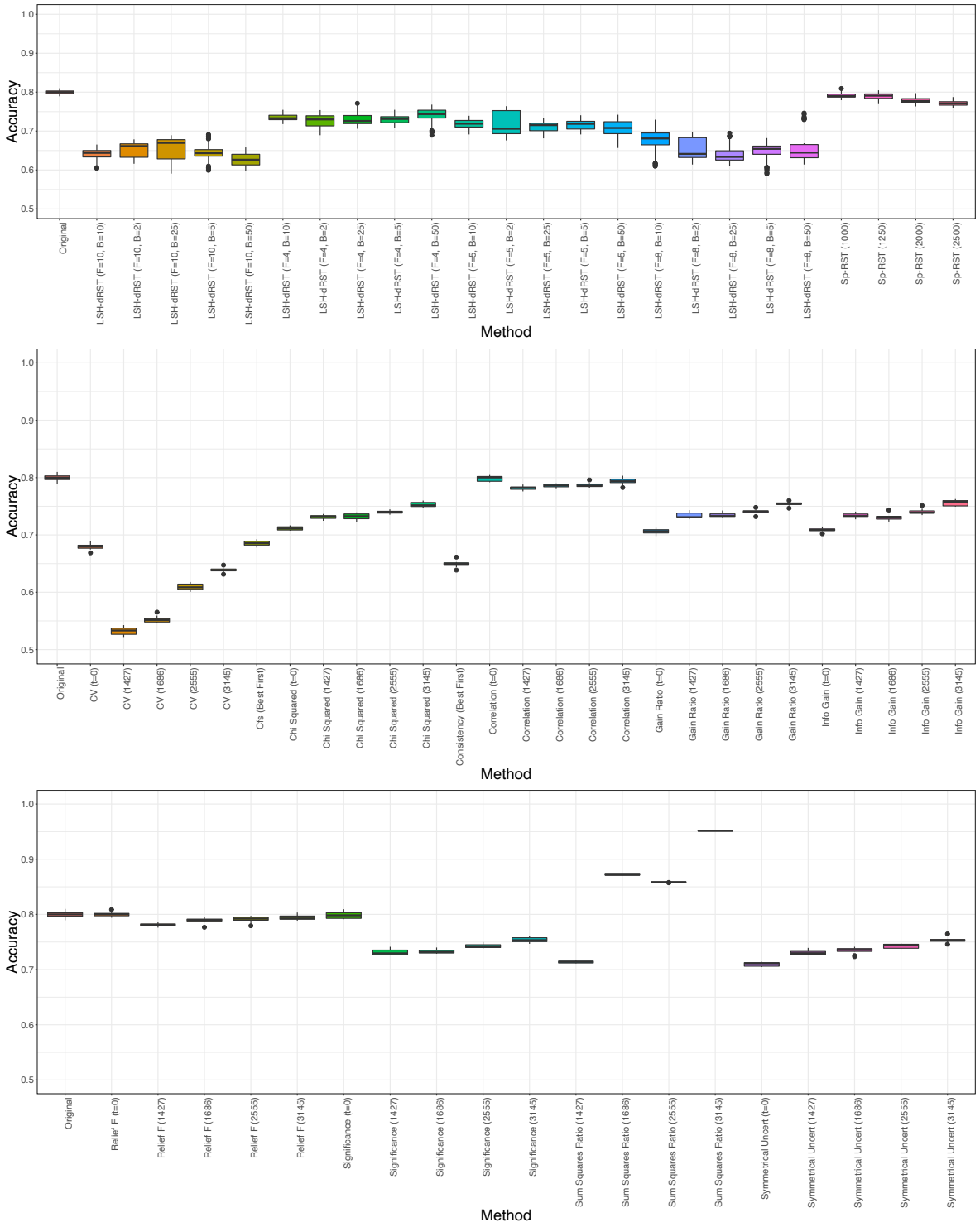


Figure 14. Random Forest: Accuracy for different parameters of LSH-dRST, Sp-RST and other feature selection techniques.

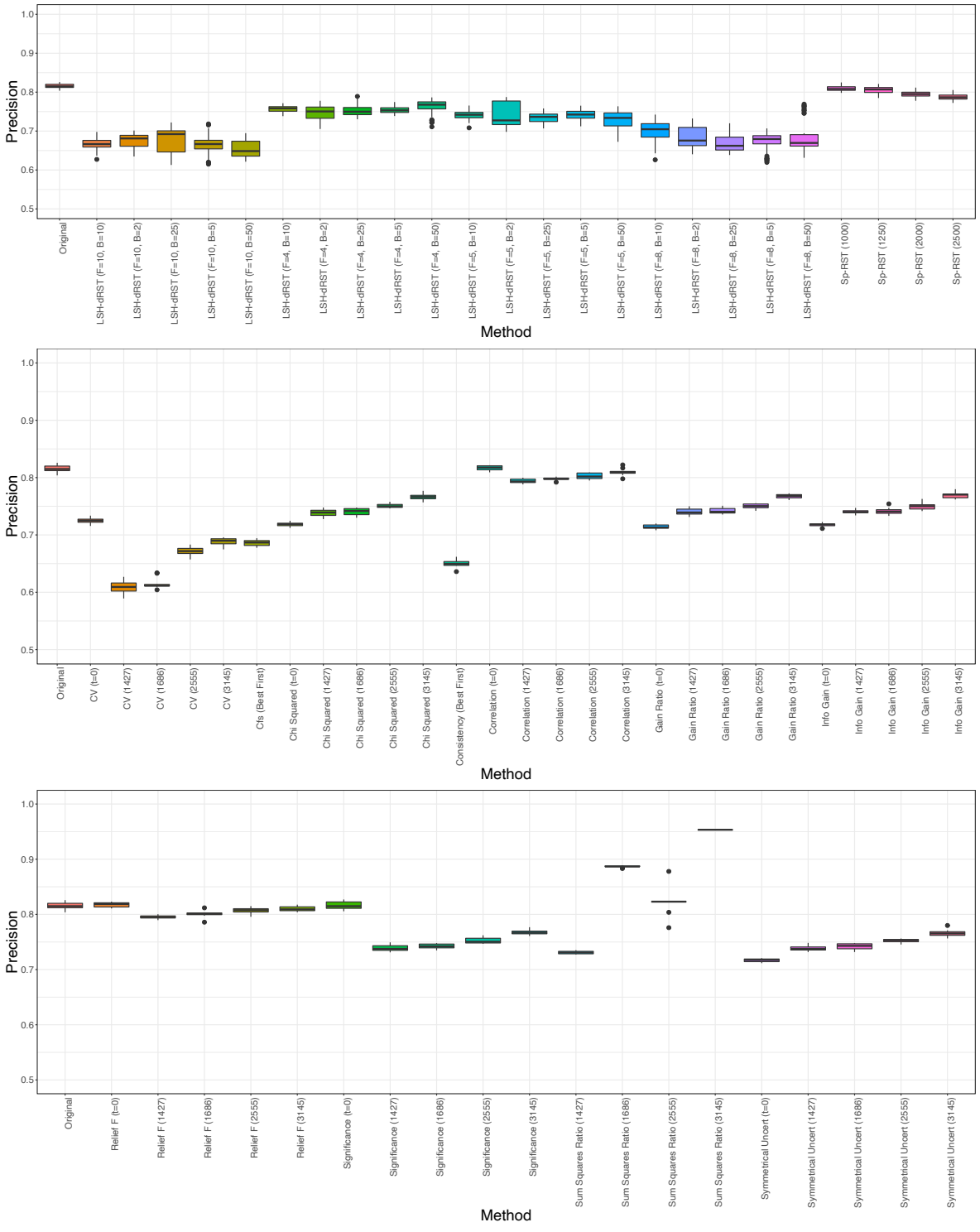


Figure 15. Random Forest: Precision for different parameters of LSH-dRST, Sp-RST and other feature selection techniques.

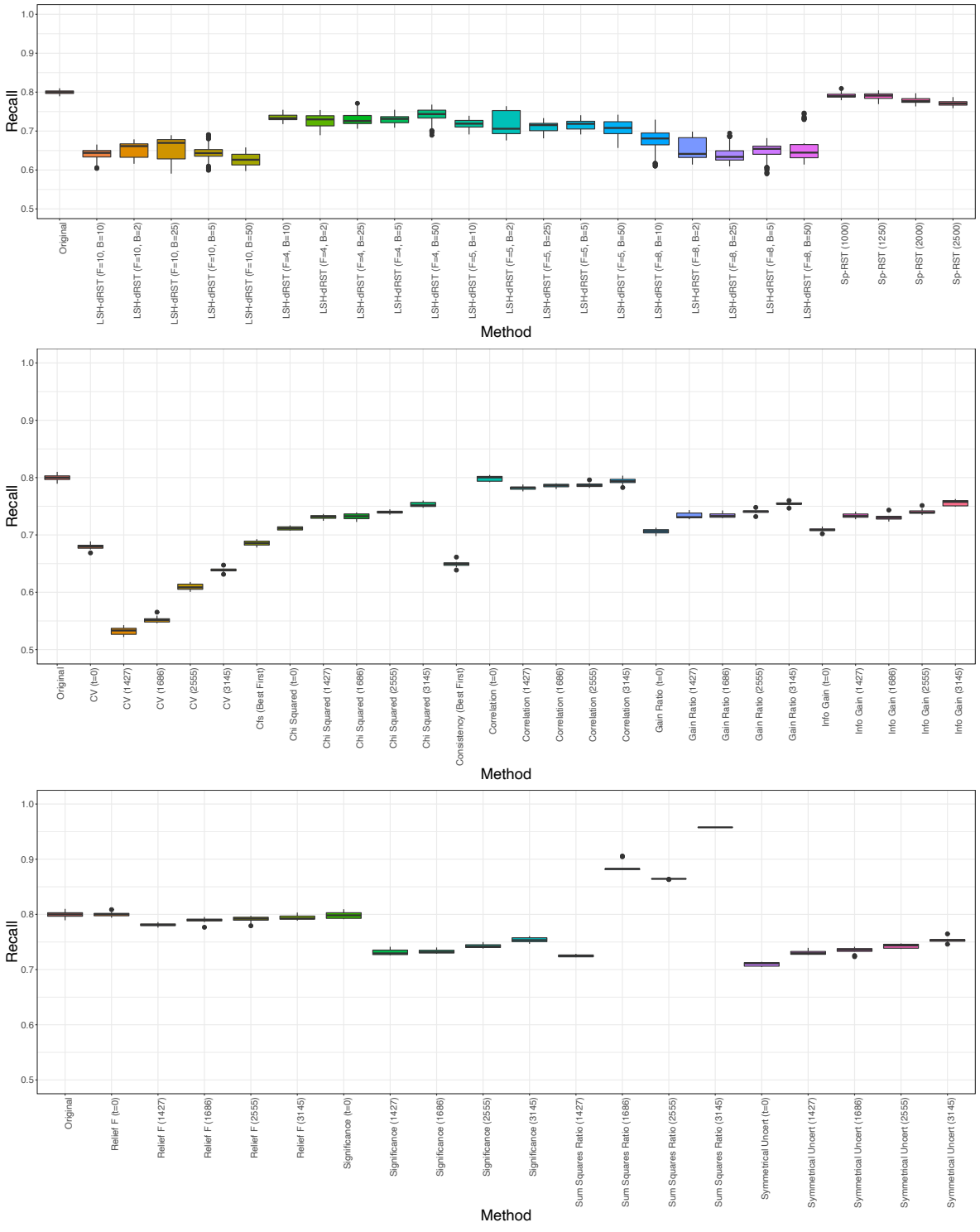


Figure 16. Random Forest: Recall for different parameters of LSH-dRST, Sp-RST and other feature selection techniques.

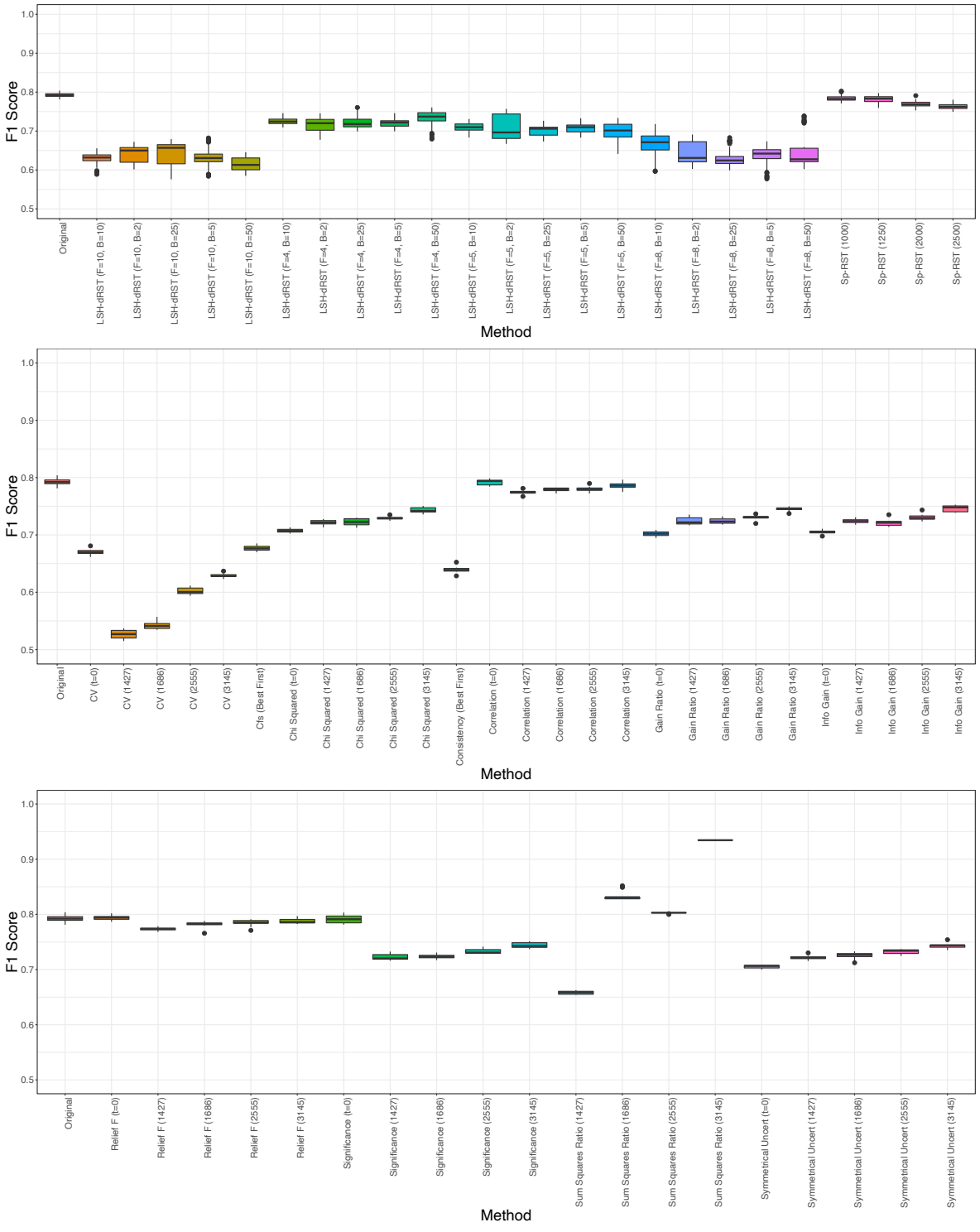


Figure 17. Random Forest: F1 score for different parameters of LSH-dRST, Sp-RST and other feature selection techniques.

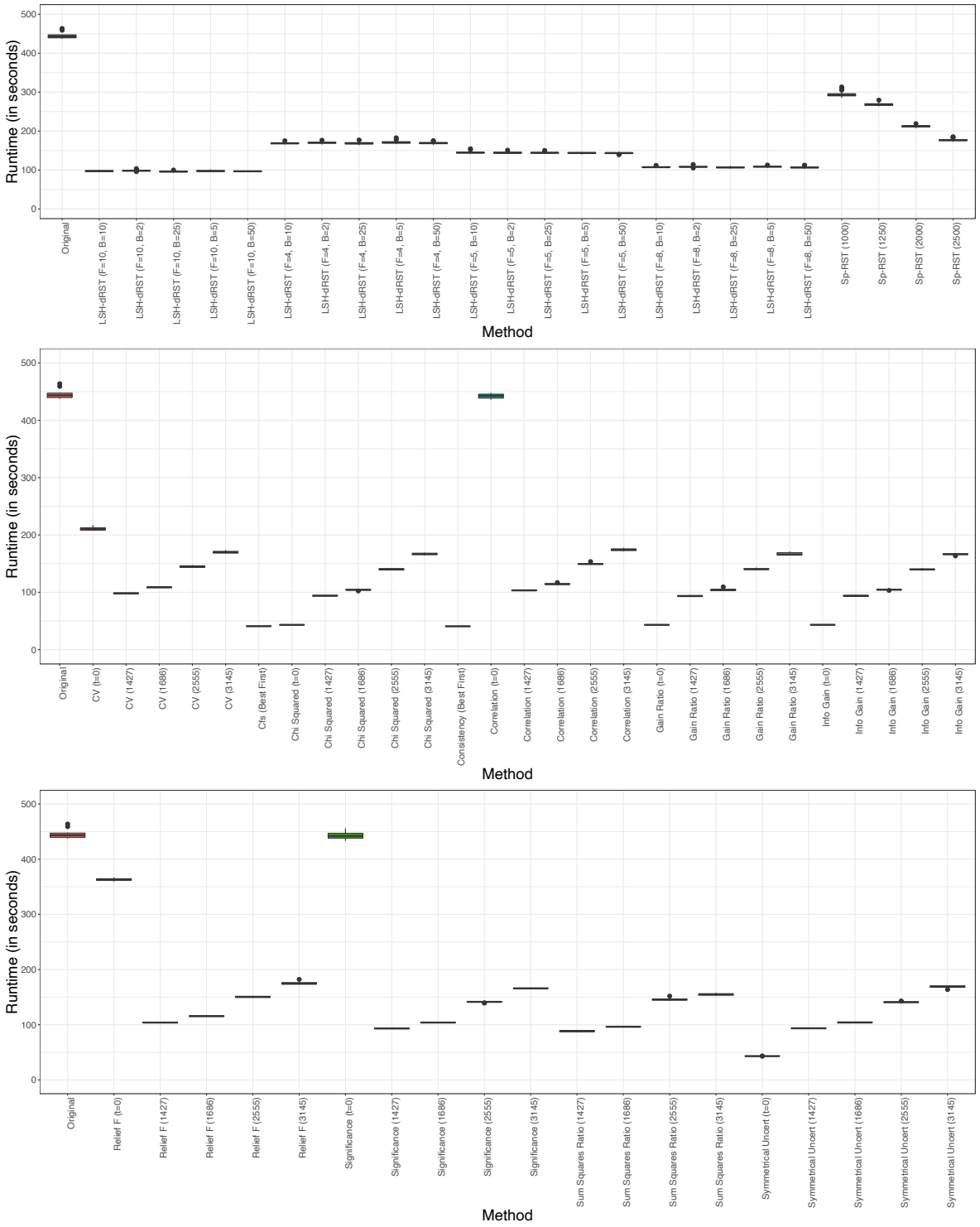


Figure 18. Random Forest: Runtimes for different parameters of LSH-dRST, Sp-RST and other feature selection techniques.

that for a Random Forest classifier small values of F are clearly preferable while the concrete value of B makes less of a difference.

Concerning the runtime we have to consider both, the runtime for the actual feature selection and the runtime for the Random Forest on the reduced dataset. In general, the runtime for the feature selection is faster for small values of F, with the best runtime achieved for $F = 4$ and $B = 50$. Note that this is independent of the number of nodes used and that this setting also achieved the best classification result. The runtime for the Random Forest classifier ranges from 109.4952 seconds ($F = 8$, $B = 50$) to 196.8976 seconds ($F = 4$, $B = 25$), with the fastest times obtained for large values of F where fewer features are selected. For $F = 4$ and $B = 50$ (the fast setting for the feature selection) the runtime is 156.7633, a medium value. Given the considerable increase in runtime for the actual feature selection for larger values of F, we conclude that small values of F should be preferred as the overall runtime (selection+classification) is smaller and the classification performance is better. $F = 4$ and $B = 50$ appears to be a good choice for LSH-dRST if used with a Random Forest classifier.

Looking at the other feature selection methods, the runtimes for the classifiers vary more drastically (40.6593 to 443.0512), with the fast time achieved for the ConsistencySubsetEval (40.6593 seconds). However, this method also takes considerably longer than any of the other methods used for comparison to perform the feature selection task (608 seconds) and performs worse in terms of classification (0.6493 accuracy, 0.6500 precision, 0.6493 recall, 0.6395 F1 score). The fastest methods in terms of feature selection are ReliefF with threshold 0 (1 second), ChiSquared (2-3 seconds) and SumSquaresRatio (2 seconds). Note that the latter one was also the best in terms of classification while the other two achieve results comparable to LSH-dRST.

Finally, in comparison to our previous method (Sp-RST), runtimes for the classification after using LSH-dRST appear to be smaller (for some parameters much smaller) with slightly worse classification results (accuracy 0.7402 vs 0.7912 in the best parameter settings). Sp-RST in general selects more features than LSH-dRST, which is a possible explanation for this behavior. In terms of runtime for the feature selection, the runtime for both methods varies largely with the parameterization and the number of nodes used. For Sp-RST, we observe that the slower the method, the better the classification. This does not seem to be the case for LSH-dRST where the faster parameters also yield the better classification results. The runtimes for the fastest parameterization for both algorithms are comparable ($F = 4$, $B = 50$ for LSH-dRST and 2500 for SP-RST). In this case we achieve a classification accuracy of 0.7402 (LSH-dRST) vs 0.7710 (Sp-RST) and runtimes of less than 75 seconds (depending on the number of nodes used).

7.3.2. Naive Bayes Classifier

Again, we show the results of 10 runs of Naive Bayes on the different reduced data sets: Figures 14, 15, 16 and 17 (classification metrics) as well as Table 18 (runtime). The corresponding raw data for classification metrics can be found in Tables 16, 17 and 18. The raw data for runtimes is again displayed in Tables 7, 8, 9 and 10 in the appendix.

We first observe that the overall classification performance of the Naive Bayes classifier is worse than for the Random Forest classifier. While this observation is independent from the feature selection method used, the overall results for LSH-dRST are somewhat disappointing as our method is outper-

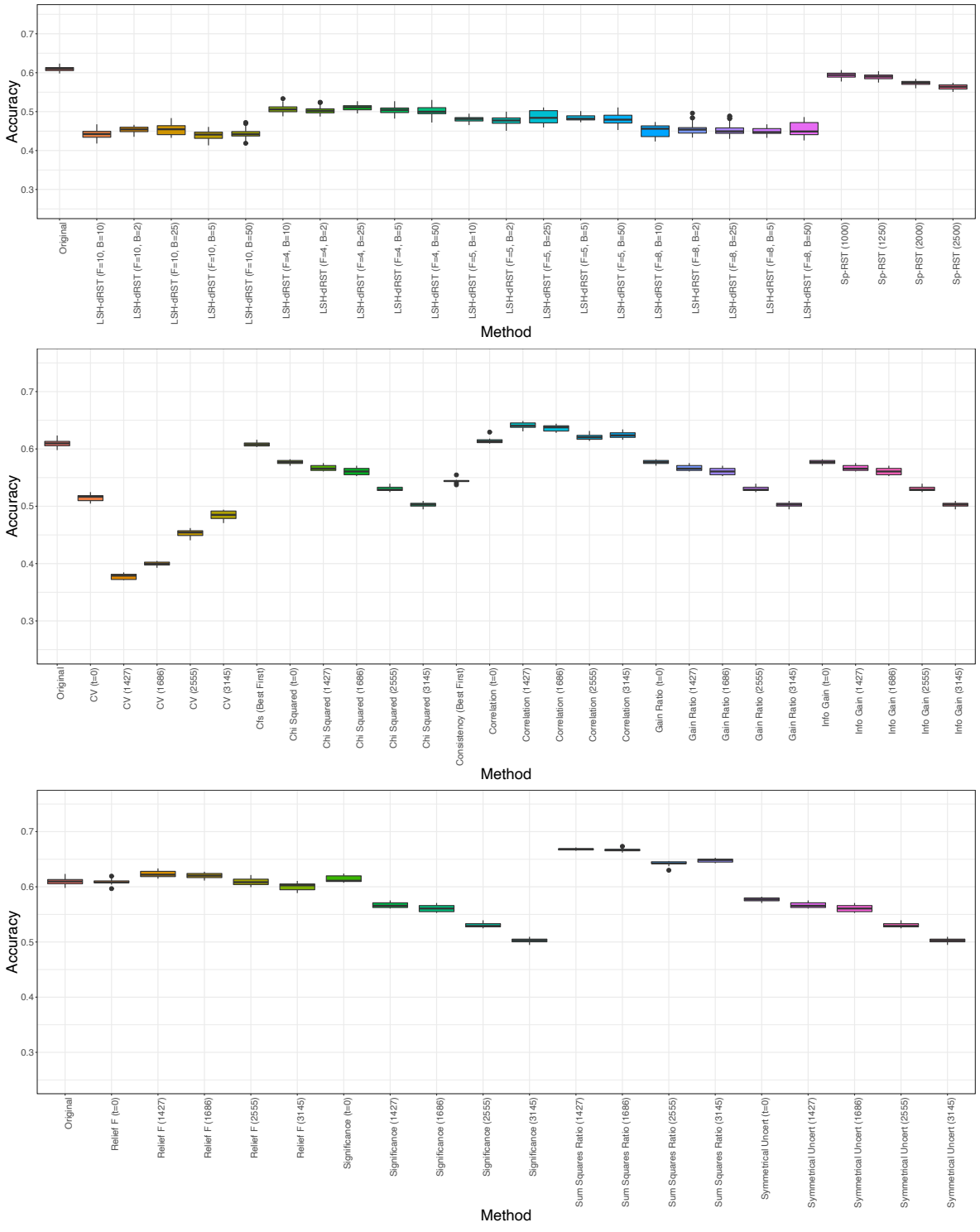


Figure 19. Naive Bayes: Accuracy for different parameters of LSH-dRST, Sp-RST and other feature selection techniques.

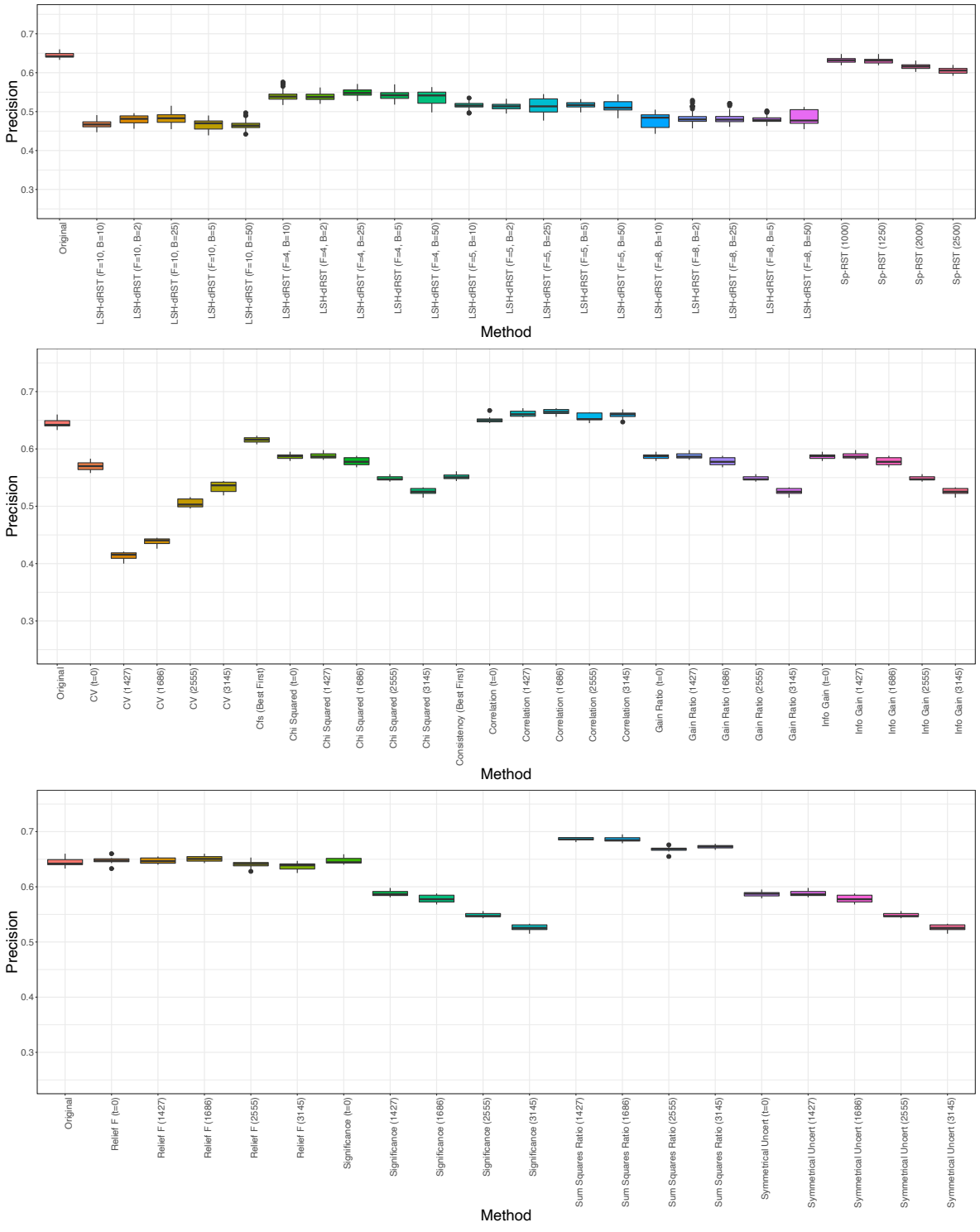


Figure 20. Naive Bayes: Precision for different parameters of LSH-dRST, Sp-RST and other feature selection techniques.

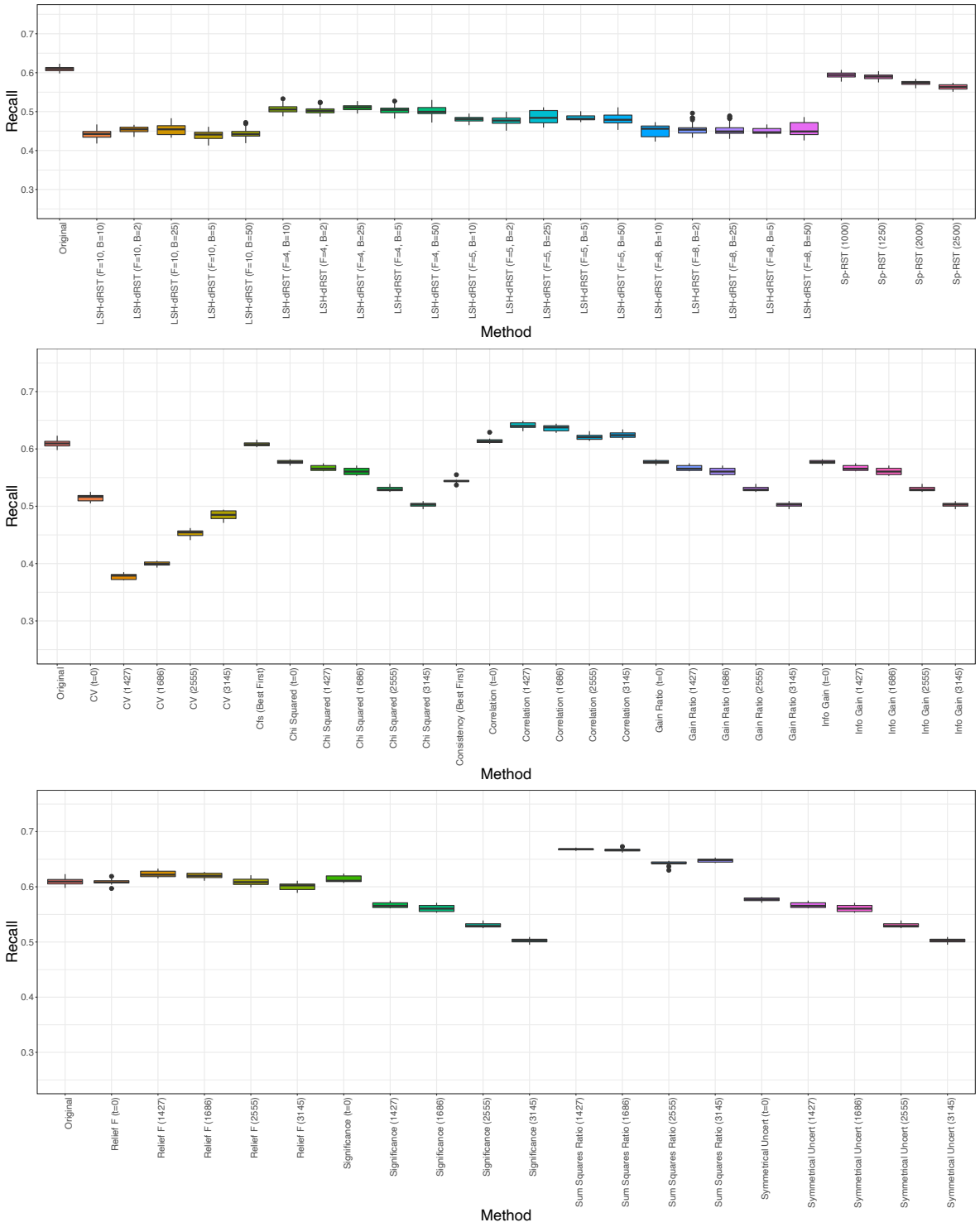


Figure 21. Naive Bayes: Recall for different parameters of LSH-dRST, Sp-RST and other feature selection techniques.

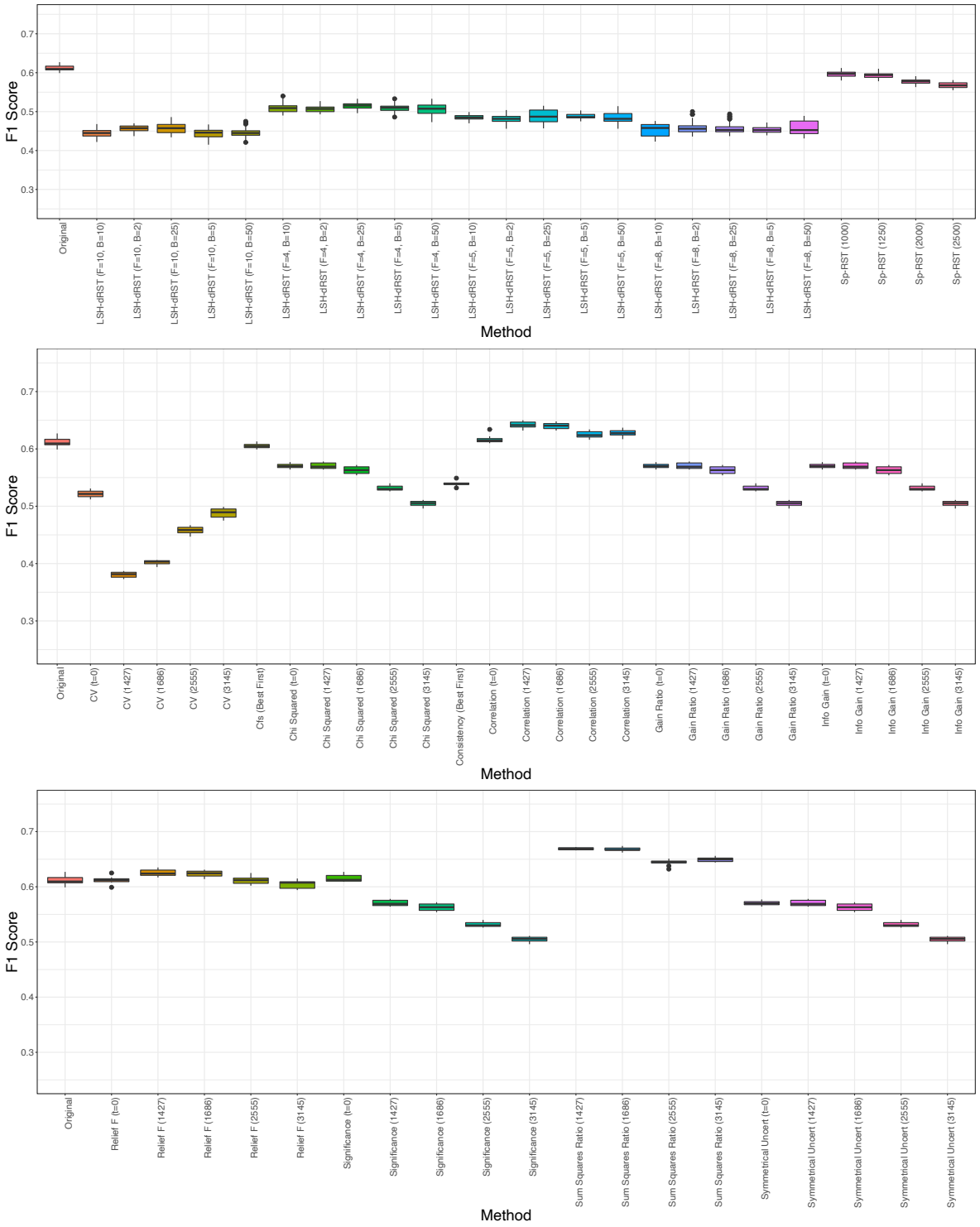


Figure 22. Naive Bayes: F1 score for different parameters of LSH-dRST, Sp-RST and other feature selection techniques.

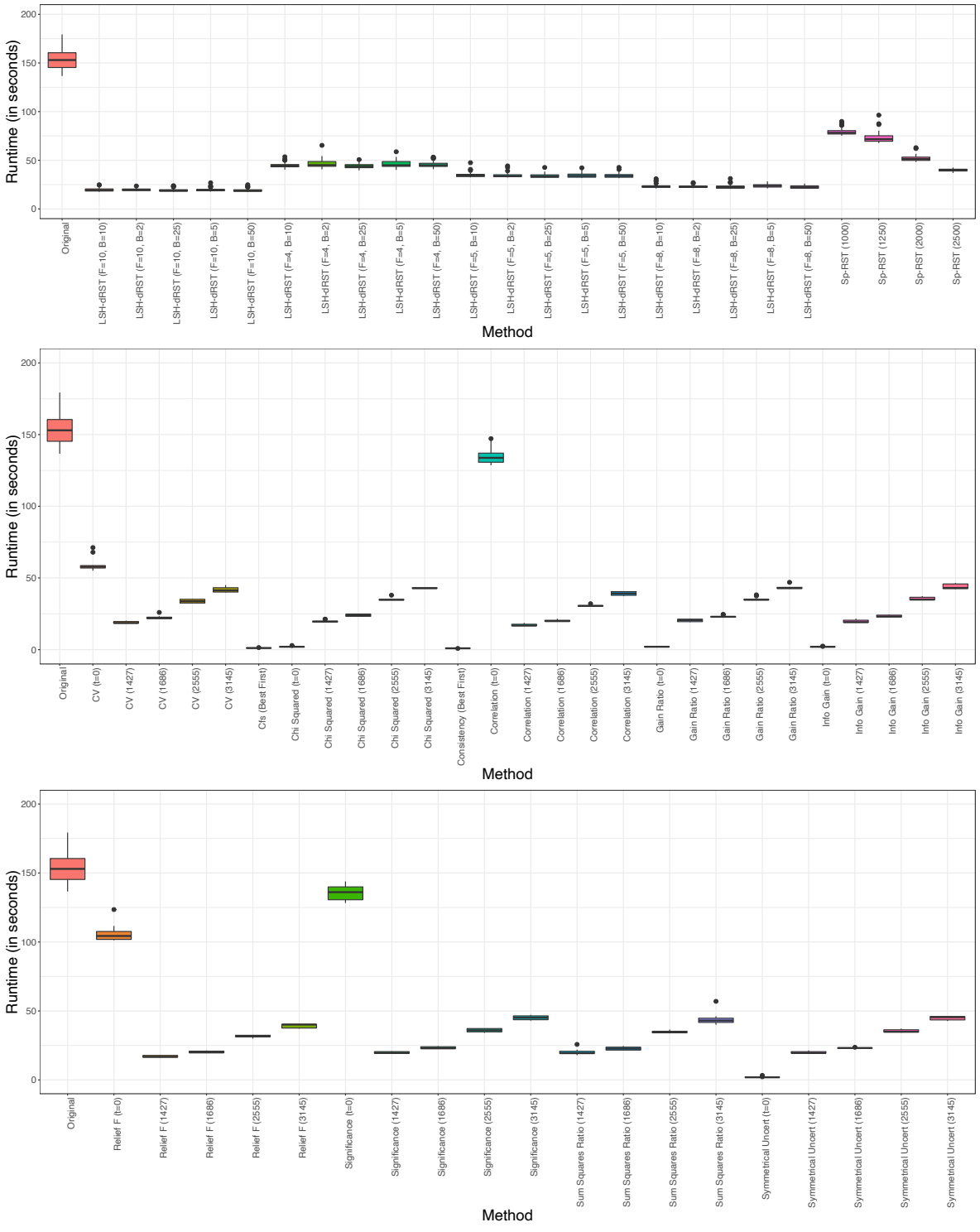


Figure 23. Naive Bayes: Runtimes for different parameters of LSH-dRST, Sp-RST and other feature selection techniques.

formed by many of the other feature selection techniques.[28] However, for some of these methods, the performance very much depends on the parameterization while the performance for LSH-dRST is again quite stable. Choosing the right parameters can be difficult in practice, thus it is promising to see the robustness of our method. Investigating the concrete reasons for this behavior and improving the overall performance when used with other classifiers such as Naive Bayes is subject to future research.

In terms of runtime for the classifier[29], we observe that all runtimes for Naive Bayes are much smaller than the ones for Random Forest, ranging from 0.924 seconds for ConsistencySubsetEval and 135.5330 seconds for SignificanceEval with threshold 0. The fastest parameter setting for LSH-dRST is $F = 10$ and $B = 50$ (19.1884 seconds). Again, consistency is the fastest in this respect, but as discussed previously it requires a long time for the feature selection process.

In terms of classification, the best results for LSH-dRST are achieved for $F = 4$ and $B = 25$ (0.5103 accuracy, 0.5486 precision, 0.5103 recall, 0.5149 F1 score). While this is slightly different than for Random Forest, where $B = 50$ performed best, we again observe that smaller values for F are generally better and outperform all other parameter settings. Thus, very similar conclusions as for the Random Forest classifier can be drawn with an overall recommended parameter setting $F = 4$ and $B = 25$ in this case.

The overall best classification result is again achieved by SumSquaresRatio. However, here, fewer features seem to be better with the best result achieved for 1 427 features: 0.6680 accuracy, 0.6862 precision, 0.6679 recall, 0.6689 F1 score. This parameter setting requires a runtime for the classifier of 88.5070, a medium value.

Finally, in comparison to our previous method (Sp-RST), again very similar conclusions can be drawn: Runtimes for the classification after using LSH-dRST appear to be smaller with slightly worse classification results. And while Sp-RST benefits from investing more computational power this is not necessarily the case for LSH-dRST.

8. Conclusion and Emerging Trends

With the emergence of big data concept, various disciplines of granular computing have recently received more attention by several researchers. Among these disciplines, which are involved in the foundations of granular computing and within the big data context, specifically when it comes to information and knowledge processing, we mention the theory of fuzzy sets and the theory of rough sets. The adaptation of these disciplines to the context of big data, by revising the different granular notions and concepts, presents a big challenge in granular computing.

In this paper, we have investigated the performance of the distributed rough set theory approach for feature selection. The proposed method, named LSH-dRST, is based on an Apache Spark distributed architecture, and integrates a hashing component, which is the Locality Sensitive Hashing algorithm (LSH). In this version, the granular concepts of rough set theory have been revised and adapted to the big data context and, accordingly, implemented in a distributed way. The main idea behind this version is to ensure a more intelligent way on how to partition the feature search space to guarantee data dependency, which is also considered as a big challenge in a distributed environment.

[28] Again all observations hold for all evaluation metrics used, i.e., accuracy, recall, precision, and F1 score.

[29] The runtime for the actual feature selection is discussed in Section 7.3.1.

We have investigated the number of features selected for different parameters as well as the scalability of the approach. Finally, we have performed model evaluation using two classifiers, Random Forest and Naive Bayes.

From the conducted deep analysis of LSH-dRST and the obtained results, we can highlight the different benefits and the impact of using the locality sensitive hashing in our proposed solution. Being incorporated into LSH-dRST, LSH could partition the high dimensional feature search space in a more reliable and intelligent way, and hence it can better preserve data dependency in the distributed environment. Another important benefit is that such an application can achieve a lower computational cost.

As future work, we aim to study the performance of the proposed distributed rough set theory versions for feature selection when these are applied to more complex real-world problems with more features and data items than the Amazon data set. Possible sources for such data sets include Kaggle data-science platforms.

It is also interesting to mention that up to now all of the proposed rough set theory versions dedicated to deal with the big data context focus on a single dimension of the input data set, i.e., the feature search space, to create the partitions. Therefore, it would be interesting to investigate other ways of partitioning the input data matrix where both dimensions of the input data set can be considered.

References

[1] Pedrycz W, Skowron A, Kreinovich V. Handbook of granular computing. John Wiley & Sons, 2008.

[2] Yao Y, et al. Granular computing: basic issues and possible solutions. In: Proceedings of the 5th joint conference on information sciences, volume 1. 2000 pp. 186–189.

[3] Lin TY, Yao YY, Zadeh LA. Data mining, rough sets and granular computing, volume 95. Physica, 2013.

[4] Tsumoto S, Hirano S, Inuiguchi M. Workshop on Rough Set Theory and Granular ComputingâĂŤSummary. In: Annual Conference of the Japanese Society for Artificial Intelligence. Springer, 2001 pp. 239–239.

[5] Zadeh LA. Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. *Fuzzy sets and systems*, 1997. **90**(2):111–127.

[6] Wang H, Xu Z, Pedrycz W. An overview on the roles of fuzzy set techniques in big data processing: Trends, challenges and opportunities. *Knowledge-Based Systems*, 2017. **118**:15–30.

[7] Mukkamala RR, Hussain A, Vatrapu R. Fuzzy-set based sentiment analysis of big social data. In: 2014 IEEE 18th International Enterprise Distributed Object Computing Conference. IEEE, 2014 pp. 71–80.

[8] Zhang J, Li T, Pan Y. Parallel rough set based knowledge acquisition using MapReduce from big data. In: Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications. ACM, 2012 pp. 20–27.

[9] Dagdia ZC, Zarges C, Schannes B, Micalef M, Galiana L, Rolland B, de Fresnoye O, Benchoufi M. Rough set theory as a data mining technique: A case study in epidemiology and cancer incidence prediction. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 2018 pp. 440–455.

[10] Hamidinekoo A, Dagdia ZC, Suhail Z, Zwiggelaar R. Distributed Rough Set Based Feature Selection Approach to Analyse Deep and Hand-crafted Features for Mammography Mass Classification. In: 2018 IEEE International Conference on Big Data (Big Data). IEEE, 2018 pp. 2423–2432.

[11] Dagdia ZC, Zarges C, Beck G, Lebbah M. A distributed rough set theory based algorithm for an efficient big data pre-processing under the spark framework. In: 2017 IEEE International Conference on Big Data (Big Data). IEEE, 2017 pp. 911–916.

[12] Dagdia ZC, Zarges C, Beck G, Lebbah M. Nouveau Modèle de Sélection de Caractéristiques basé sur la Théorie des Ensembles Approximatifs pour les Données Massives: Méthode de sélection de caractéristiques pour les données massives. In: Conférence Internationalle sur l'Extraction et la Gestion des Connaissances. 2018 pp. 377–378.

[13] Düntsch I, Gediga G. Rough set data analysis. *Encyclopedia of Computer Science and Technology*, 2000. **43**(28):281–301.

[14] Dagdia ZC, Zarges C, Beck G, Lebbah M. A Scalable and Effective Rough Set Theory based Approach for Big Data Pre-processing. *Knowledge and Information Systems*, 2020. **2020**.

[15] Grzegorowski M, Ślęzak D. On resilient feature selection: Computational foundations of rC-reducts. *Information Sciences*, 2019. **499**:25–44.

[16] Thangavel K, Pethalakshmi A. Dimensionality reduction based on rough set theory: A review. *Applied Soft Computing*, 2009. **9**(1):1–12.

[17] Gionis A, Indyk P, Motwani R. Similarity search in high dimensions via hashing. In: VLDB. 1999 pp. 518–529.

[18] Liu W, Wang J, Ji R, Jiang YG, Chang SF. Supervised hashing with kernels. In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. IEEE, 2012 pp. 2074–2081.

[19] Weiss Y, Torralba A, Fergus R. Spectral hashing. In: Advances in neural information processing systems. 2009 pp. 1753–1760.

[20] Cai D. A revisit of hashing algorithms for approximate nearest neighbor search. *arXiv preprint arXiv:1612.07545*, 2016.

[21] Dagdia ZC, Zarges C, Beck G, Azzag H, Lebbah M. A Distributed Rough Set Theory Algorithm based on Locality Sensitive Hashing for an Efficient Big Data Pre-processing. In: 2018 IEEE International Conference on Big Data (Big Data). IEEE, 2018 pp. 2597–2606.

[22] Liu H, Motoda H, Setiono R, Zhao Z. Feature selection: An ever evolving frontier in data mining. In: Feature Selection in Data Mining. 2010 pp. 4–13.

[23] Liu H, Zhao Z. Manipulating data and dimension reduction methods: feature selection. In: Computational Complexity, pp. 1790–1800. Springer, 2012.

[24] Bolón-Canedo V, Rego-Fernández D, Peteiro-Barral D, Alonso-Betanzos A, Guijarro-Berdiñas B, Sánchez-Maroño N. On the scalability of feature selection methods on high-dimensional data. *Knowledge and Information Systems*, 2018. pp. 1–48.

[25] Dean J, Ghemawat S. MapReduce: a flexible data processing tool. *Communications of the ACM*, 2010. **53**(1):72–77.

[26] Vinh NX, Chan J, Romano S, Bailey J, Leckie C, Ramamohanarao K, Pei J. Discovering outlying aspects in large datasets. *Data Mining and Knowledge Discovery*, 2016. **30**(6):1520–1555.

[27] Zhang J, Wang S, Chen L, Gallinari P. Multiple Bayesian discriminant functions for high-dimensional massive data classification. *Data Mining and Knowledge Discovery*, 2017. **31**(2):465–501.

[28] Zhai T, Gao Y, Wang H, Cao L. Classification of high-dimensional evolving data streams via a resource-efficient online ensemble. *Data Mining and Knowledge Discovery*, 2017. pp. 1–24.

[29] Shanahan JG, Dai L. 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.

[30] Peralta D, del Río S, Ramírez-Gallego S, Triguero I, Benitez JM, Herrera F. Evolutionary feature selection for big data classification: A mapreduce approach. *Mathematical Problems in Engineering*, 2015. **2015**.

[31] Peng H, Long F, Ding C. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 2005. **27**(8):1226–1238.

[32] Datar M, Immorlica N, Indyk P, Mirrokni VS. Locality-sensitive Hashing Scheme Based on P-stable Distributions. In: Proceedings of the Twentieth Annual Symposium on Computational Geometry, SCG '04. ACM, New York, NY, USA. ISBN 1-58113-885-7, 2004 pp. 253–262. doi:10.1145/997817.997857.

[33] Charikar MS. Similarity Estimation Techniques from Rounding Algorithms. In: Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing, STOC '02. ACM, New York, NY, USA. ISBN 1-58113-495-9, 2002 pp. 380–388. doi:10.1145/509907.509965.

[34] Indyk P, Motwani R. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC '98. ACM, New York, NY, USA. ISBN 0-89791-962-9, 1998 pp. 604–613. doi:10.1145/276698.276876.

[35] Broder A. On the Resemblance and Containment of Documents. In: Proceedings of the Compression and Complexity of Sequences 1997, SEQUENCES '97. IEEE Computer Society, Washington, DC, USA. ISBN 0-8186-8132-2, 1997 pp. 21–.

[36] Weiss Y, Torralba A, Fergus R. Spectral Hashing. In: Proceedings of the 21st International Conference on Neural Information Processing Systems, NIPS'08. Curran Associates Inc., USA. ISBN 978-1-6056-0-949-2, 2008 pp. 1753–1760.

[37] He J, Liu W, Chang SF. Scalable Similarity Search with Optimized Kernel Hashing. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '10. ACM, New York, NY, USA. ISBN 978-1-4503-0055-1, 2010 pp. 1129–1138. doi:10.1145/1835804.1835946.

[38] He J, Radhakrishnan R, Chang SF, Bauer C. Compact Hashing with Joint Optimization of Search Accuracy and Time. In: Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition, CVPR '11. IEEE Computer Society, Washington, DC, USA. ISBN 978-1-4577-0394-2, 2011 pp. 753–760. doi:10.1109/CVPR.2011.5995518.

[39] Wang J, Shen HT, Song J, Ji J. Hashing for Similarity Search: A Survey. *CoRR*, 2014. **abs/1408.2927**. 1408.2927.

[40] Pawlak Z, Skowron A. Rudiments of rough sets. *Information sciences*, 2007. **177**(1):3–27.

[41] Sakr S, Liu A, Batista DM, Alomari M. A survey of large scale data management approaches in cloud environments. *IEEE Communications Surveys & Tutorials*, 2011. **13**(3):311–336.

[42] Snir M. MPI–the Complete Reference: the MPI core, volume 1. MIT press, 1998.

[43] Fernández A, del Río S, López V, Bawakid A, del Jesus MJ, Benítez JM, Herrera F. Big Data with Cloud Computing: an insight on the computing environment, MapReduce, and programming frameworks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2014. **4**(5):380–409.

[44] https://sparkapacheorg/mllib/. MLlib Website.

[45] Xu X, Jäger J, Kriegel HP. A fast parallel clustering algorithm for large spatial databases. In: High Performance Data Mining, pp. 263–290. Springer, 1999.

[46] Asuncion A, Newman D. UCI machine learning repository, 2007.

[47] Labrinidis A, Jagadish HV. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 2012. **5**(12):2032–2033.

[48] Fan W, Bifet A. Mining big data: current status, and forecast to the future. *ACM sIGKDD Explorations Newsletter*, 2013. **14**(2):1–5.

[49] Afendi FM, Ono N, Nakamura Y, Nakamura K, Darusman LK, Kibinge N, Morita AH, Tanaka K, Horai H, Altaf-Ul-Amin M, et al. Data mining methods for omics and knowledge of crude medicinal plants toward big data biology. *Computational and Structural Biotechnology Journal*, 2013. **4**(5):1–14.

[50] Wu X, Zhu X, Wu GQ, Ding W. Data mining with big data. *IEEE transactions on knowledge and data engineering*, 2014. **26**(1):97–107.

[51] Chen M, Mao S, Liu Y. Big data: A survey. *Mobile Networks and Applications*, 2014. **19**(2):171–209.

[52] Larose DT. Discovering knowledge in data: an introduction to data mining. John Wiley & Sons, 2014.

[53] Grzymala-Busse JW, Ziarko W. Data Mining and Rough Set Theory. *Commun. ACM*, 2000. **43**(4):108–109.

[54] Liu H, Yu L. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on knowledge and data engineering*, 2005. **17**(4):491–502.

[55] Talukder N, Zaki MJ. A distributed approach for graph mining in massive networks. *Data Mining and Knowledge Discovery*, 2016. **30**(5):1024–1052.

[56] Schäfer P. Scalable time series classification. *Data Mining and Knowledge Discovery*, 2016. **30**(5):1273–1298.

[57] Qian Y, Liang J, Pedrycz W, Dang C. Positive approximation: An accelerator for attribute reduction in rough set theory. *Artificial Intelligence*, 2010. **174**(9):597 – 618.

[58] Schneider J, Vlachos M. Scalable density-based clustering with quality guarantees using random projections. *Data Mining and Knowledge Discovery*, 2017. pp. 1–34.

[59] White T. Hadoop: The definitive guide. " O'Reilly Media, Inc.", 2012.

[60] Tan M, Tsang IW, Wang L. Towards ultrahigh dimensional feature selection for big data. *Journal of Machine Learning Research*, 2014. **15**:1371–1429.

[61] Guyon I, Elisseeff A. An introduction to variable and feature selection. *Journal of machine learning research*, 2003. **3**(Mar):1157–1182.

[62] Dash M, Liu H. Feature selection for classification. *Intelligent data analysis*, 1997. **1**(1-4):131–156.

[63] Keogh E, Mueen A. Curse of dimensionality. In: Encyclopedia of Machine Learning, pp. 257–258. Springer, 2011.

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

[65] Wang X, Yang J, Teng X, Xia W, Jensen R. Feature selection based on rough sets and particle swarm optimization. *Pattern recognition letters*, 2007. **28**(4):459–471.

[66] Swiniarski RW, Skowron A. Rough set methods in feature selection and recognition. *Pattern recognition letters*, 2003. **24**(6):833–849.

[67] Guyon I, Elisseeff A. An introduction to feature extraction. *Feature extraction*, 2006. pp. 1–25.

[68] John GH, Kohavi R, Pfleger K, et al. Irrelevant features and the subset selection problem. In: Machine learning: proceedings of the eleventh international conference. 1994 pp. 121–129.

[69] Prinzie A, Van den Poel D. Random forests for multiclass classification: Random multinomial logit. *Expert systems with Applications*, 2008. **34**(3):1721–1732.

[70] Ahmed S, Zhang M, Peng L. Enhanced feature selection for biomarker discovery in LC-MS data using GP. In: Evolutionary Computation (CEC), 2013 IEEE Congress on. IEEE, 2013 pp. 584–591.

[71] Aghdam MH, Ghasem-Aghaee N, Basiri ME. Text feature selection using ant colony optimization. *Expert systems with applications*, 2009. **36**(3):6843–6853.

[72] Ghosh A, Datta A, Ghosh S. Self-adaptive differential evolution for feature selection in hyperspectral image data. *Applied Soft Computing*, 2013. **13**(4):1969–1977.

[73] Lingras P. Unsupervised rough set classification using GAs. *Journal of Intelligent Information Systems*, 2001. **16**(3):215–228.

[74] Lingras P. Rough set clustering for web mining. In: Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on, volume 2. IEEE, 2002 pp. 1039–1044.

[75] Bai C, Sarkis J. Integrating sustainability into supplier selection with grey system and rough set methodologies. *International Journal of Production Economics*, 2010. **124**(1):252–264.

[76] Guller M. Big Data Analytics with Spark: A Practitioner's Guide to Using Spark for Large Scale Data Analysis. Springer, 2015.

[77] Indyk P, Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. ACM, 1998 pp. 604–613.

[78] Charikar MS. Similarity estimation techniques from rounding algorithms. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. ACM, 2002 pp. 380–388.

[79] Lv Q, Josephson W, Wang Z, Charikar M, Li K. Ferret: a toolkit for content-based similarity search of feature-rich data. *ACM SIGOPS Operating Systems Review*, 2006. **40**(4):317–330.

[80] Wang J, Shen HT, Song J, Ji J. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014.

9. Appendix

We provide more detailed experimental results in the appendix for the sake of completeness. We particularly include tables with numeric runtime and classification results. In the tables, we highlight the best results (smallest runtime, highest classification metric) in bold. We do so separately for the best parameter setting of LSH-dRST and other feature selection techniques. Whenever applicable, tables display averages and standard deviation (in brackets) over the runs performed in our experiments.

9.1. Runtime Results (Overview)

Method	#Features	Selection	Random Forest	Naive Bayes
LSH (F=4,B=2)	3162.6 (20.93562)	1 node: 293.9634 2 nodes: 330.9884 4 nodes: 273.6784 8 nodes: 267.9994 16 nodes: 264.1068	163.0903 (24.4049)	46.1034 (4.2355)
LSH (F=4,B=5)	3153.4 (28.18333)	1 node: 94.33579 2 nodes: 121.487 4 nodes: 105.7729 8 nodes: 83.20773 16 nodes: 80.88177	137.6660 (6.0934)	46.0554 (3.9327)
LSH (F=4,B=10)	3136.6 (20.59854)	1 node: 68.99441 2 nodes: 95.21888 4 nodes: 56.27094 8 nodes: 52.24907 16 nodes: 54.99504	143.4032 (8.8300)	44.7714 (2.7593)
LSH (F=4,B=25)	3129.6 (30.31171)	1 node: 52.77961 2 nodes: 81.43544 4 nodes: 48.86025 8 nodes: 43.4144 16 nodes: 39.97918	196.8976 (132.1669)	44.1836 (2.5568)
LSH (F=4,B=50)	3143.8 (27.93206)	**1 node: 49.00843** **2 nodes: 74.64357** **4 nodes: 46.26422** **8 nodes: 41.5312** **16 nodes: 38.68473**	156.7633 (20.5751)	45.5628 (3.0586)
LSH (F=5,B=2)	2562.6 (6.94982)	1 node: 356.9446 2 nodes: 292.1586 4 nodes: 253.7213 8 nodes: 258.2594 16 nodes: 218.4629	136.3867 (6.5513)	34.5402 (2.3956)
LSH (F=5,B=5)	2549.2 (27.28919)	1 node: 116.4721 2 nodes: 170.3086 4 nodes: 98.40898 8 nodes: 87.09421 16 nodes: 77.23412	142.6852 (8.2735)	34.4690 (2.2838)
LSH (F=5,B=10)	2579.0 (13.83835)	1 node: 81.31365 2 nodes: 137.0798 4 nodes: 70.01903 8 nodes: 51.65553 16 nodes: 54.28747	163.2165 (35.9467)	34.8486 (2.5035)

Table 7. Runtime results for different parameters of LSH-dRST (Part 1).

Method	#Features	Selection	Random Forest	Naive Bayes
LSH (F=5,B=25)	2548.2 (17.93600)	1 node: 66.17459 2 nodes: 141.0207 4 nodes: 57.69587 8 nodes: 44.21889 16 nodes: 47.42742	156.7915 (31.7525)	34.0446 (2.0342)
LSH (F=5,B=50)	2535.4 (30.44339)	1 node: 62.64974 2 nodes: 122.0032 4 nodes: 57.4125 8 nodes: 47.37777 16 nodes: 45.36554	178.9157 (35.8738)	34.2858 (2.2643)
LSH (F=8,B=2)	1700.8 (10.32957)	1 node: 1130.755 2 nodes: 1158.858 4 nodes: 468.436 8 nodes: 346.6934 16 nodes: 257.8431	118.2328 (7.4608)	22.9064 (1.1685)
LSH (F=8,B=5)	1709.8 (18.28114)	1 node: 1032.926 2 nodes: 1123.54 4 nodes: 540.2123 8 nodes: 235.8572 16 nodes: 196.4199	111.6114 (4.5041)	23.7212 (1.6567)
LSH (F=8,B=10)	1684.8 (14.09610)	1 node: 1009.47 2 nodes: 1279.771 4 nodes: 394.2123 8 nodes: 227.9812 16 nodes: 200.1706	112.3749 (4.0889)	23.3394 (1.9714)
LSH (F=8,B=25)	1673.0 (34.72751)	1 node: 993.3145 2 nodes: 1452.427 4 nodes: 395.2873 8 nodes: 242.5078 16 nodes: 219.3121	115.1896 (7.9205)	22.8154 (1.9989)
LSH (F=8,B=50)	1659.8 (12.11198)	1 node: 987.3648 2 nodes: 1513.367 4 nodes: 333.6505 8 nodes: 237.296 16 nodes: 380.8814	**109.4952** (3.7340)	22.5440 (1.3949)
LSH (F=10,B=2)	1452.2 (14.04279)	1 node: 1428.017 2 nodes: 6220.443 4 nodes: 1561.927 8 nodes: 1057.972 16 nodes: 1097.766	112.4193 (9.1338)	19.7990 (1.2075)
LSH (F=10,B=5)	1438.8 (27.55358)	1 node: 1329.887 2 nodes: 6610.68 4 nodes: 1602.414 8 nodes: 947.0734 16 nodes: 982.1677	109.9828 (13.6445)	19.7598 (1.6335)
LSH (F=10,B=10)	1423.6 (21.19670)	1 node: 1330.115 2 nodes: 6868.82 4 nodes: 1661.798 8 nodes: 1052.526 16 nodes: 930.264	148.4756 (42.4020)	19.7192 (1.7981)
LSH (F=10,B=25)	1407.0 (16.00000)	1 node: 1302.886 2 nodes: 5493.28 4 nodes: 1737.555 8 nodes: 1024.793 16 nodes: 1086.141	138.7421 (30.9709)	19.2248 (1.5202)
LSH (F=10,B=50)	1414.2 (21.62637)	1 node: 1300.365 2 nodes: 6891.06 4 nodes: 1803.13 8 nodes: 1317.976 16 nodes: 964.0759	112.2374 (11.9036)	**19.1884** (1.6507)

Table 8. Runtime results for different parameters of LSH-dRST (Part 2).

Method	#Features	Selection	Random Forest	Naive Bayes
Original	10000	0	445.7113	154.3220
	(-)	(-)	(8.9163)	(13.4778)
CV (t=0)	4149	49	210.7067	59.7090
	(-)	(-)	(2.9873)	(5.3565)
CV (1427)	1427	47	98.2336	18.9600
	(-)	(-)	(0.9268)	(0.7974)
CV (1686)	1686	46	108.8291	22.4600
	(-)	(-)	(0.9020)	(1.3662)
CV (2555)	2555	47	144.7988	33.8160
	(-)	(-)	(1.4936)	(1.3971)
CV (3145)	3145	46	169.9300	41.7430
	(-)	(-)	(1.9708)	(1.8759)
Cfs (Greedy)	41	296	40.7580	1.1660
	(-)	(-)	(0.1532)	(0.1060)
Chi Squared (t=0)	113	2	43.2237	2.0520
	(-)	(-)	(0.1800)	(0.2946)
Chi Squared (1427)	1427	3	93.9593	19.8070
	(-)	(-)	(0.6953)	(0.7972)
Chi Squared (1686)	1686	2	104.1886	23.9980
	(-)	(-)	(0.8936)	(0.9149)
Chi Squared (2555)	2555	2	140.0667	35.0810
	(-)	(-)	(1.2154)	(1.0366)
Chi Squared (3145)	3145	3	166.6786	42.9120
	(-)	(-)	(1.6763)	(0.3186)
Consistency (Greedy)	30	608	**40.6593**	**0.9240**
	(-)	(-)	(0.1933)	(0.0534)
Correlation (t=0)	10000	10	442.0538	135.5330
	(-)	(-)	(4.2028)	(6.5574)
Correlation (1427)	1427	8	103.3509	17.1700
	(-)	(-)	(0.6582)	(0.7420)
Correlation (1686)	1686	7	114.5108	20.2100
	(-)	(-)	(1.1578)	(0.6911)
Correlation (2555)	2555	7	149.8167	30.6060
	(-)	(-)	(1.4952)	(0.5625)
Correlation (3145)	3145	8	174.2044	39.1150
	(-)	(-)	(1.9106)	(1.4067)
Gain Ratio (t=0)	113	7	43.2204	1.9850
	(-)	(-)	(0.1460)	(0.1276)
Gain Ratio (1427)	1427	7	93.6727	20.3990
	(-)	(-)	(0.9373)	(0.9114)
Gain Ratio (1686)	1686	7	104.5484	23.1170
	(-)	(-)	(2.1300)	(0.7720)
Gain Ratio (2555)	2555	8	140.4722	35.3750
	(-)	(-)	(1.3630)	(1.3214)
Gain Ratio (3145)	3145	8	166.8755	43.3280
	(-)	(-)	(2.6151)	(1.3403)
Info Gain (t=0)	113	7	43.2661	2.0050
	(-)	(-)	(0.1277)	(0.1910)
Info Gain (1427)	1427	7	93.8349	19.7290
	(-)	(-)	(1.2257)	(1.0071)
Info Gain (1686)	1686	7	104.4151	23.3660
	(-)	(-)	(0.5484)	(0.7313)
Info Gain (2555)	2555	8	139.8821	35.4830
	(-)	(-)	(1.2951)	(1.1791)
Info Gain (3145)	3145	8	166.2922	44.0000
	(-)	(-)	(1.3921)	(1.7727)

Table 9. Runtime results for other feature selection techniques (Part 1).

Method	#Features	Selection	Random Forest	Naive Bayes
ReliefF (t=0)	7957 (-)	**1** (-)	363.0645 (2.4074)	106.4700 (6.8651)
ReliefF (1427)	1427 (-)	8 (-)	104.0523 (0.6797)	17.0170 (0.7430)
ReliefF (1686)	1686 (-)	8 (-)	115.5863 (0.8737)	20.2280 (0.7300)
ReliefF (2555)	2555 (-)	8 (-)	150.6049 (1.1557)	31.5970 (0.9336)
ReliefF (3145)	3145 (-)	8 (-)	175.4633 (2.7690)	39.1670 (1.4395)
Significance (t=0)	10000 (-)	3 (-)	443.0512 (7.1975)	135.7860 (5.4555)
Significance (1427)	1427 (-)	3 (-)	93.4174 (0.7614)	19.8110 (0.7376)
Significance (1686)	1686 (-)	3 (-)	104.0673 (0.8551)	23.3070 (0.8137)
Significance (2555)	2555 (-)	3 (-)	141.2948 (1.0859)	35.9880 (1.3315)
Significance (3145)	3145 (-)	3 (-)	165.8484 (0.8897)	45.0320 (1.5426)
Sp-RST (1000)	6171.333 (26.01282)	1 node: 4448.072 2 nodes: 1540.4 4 nodes: 1641.565 8 nodes: 865.2048 16 nodes: 585.8696 32 nodes: 334.1958	294.0260 (5.0313)	79.3787 (3.6052)
Sp-RST (1250)	5566.000 (23.89142)	1 node: 935.9299 2 nodes: 374.9 4 nodes: 646.0979 8 nodes: 209.8693 16 nodes: 118.5509 32 nodes: 88.09231	268.5728 (3.2815)	72.9547 (5.2135)
Sp-RST (2000)	4117.000 (18.95257)	1 node: 72.27206 2 nodes: 52.9 4 nodes: 49.97752 8 nodes: 28.46254 16 nodes: 22.83241 32 nodes: 18.30543	212.4548 (2.3847)	51.9187 (2.8680)
Sp-RST (2500)	3205.333 (24.35296)	1 node: 51.98853 2 nodes: 51 4 nodes: 25.86191 8 nodes: 20.81937 16 nodes: 18.0587 32 nodes: 16.60878	177.0313 (2.6335)	39.8933 (1.2833)
Sum Squares Ratio (1427)	1427 (-)	2 (-)	88.5070 (1.4030)	20.1640 (2.3277)
Sum Squares Ratio (1686)	1686 (-)	2 (-)	96.4577 (0.5372)	22.7440 (1.2330)
Sum Squares Ratio (2555)	2555 (-)	2 (-)	146.0271 (2.3981)	34.8390 (0.9321)
Sum Squares Ratio (3145)	3145 (-)	2 (-)	155.0862 (1.8683)	44.3340 (4.8147)
Symmetrical Uncert (t=0)	113 (-)	7 (-)	43.1611 (0.1603)	2.0700 (0.4490)
Symmetrical Uncert (1427)	1427 (-)	8 (-)	93.4457 (0.8489)	19.9050 (0.8217)
Symmetrical Uncert (1686)	1686 (-)	8 (-)	104.1815 (0.9198)	23.1590 (0.2291)
Symmetrical Uncert (2555)	2555 (-)	8 (-)	140.9627 (1.1890)	35.4490 (1.1330)
Symmetrical Uncert (3145)	3145 (-)	8 (-)	168.7001 (2.3883)	44.8310 (1.4125)

Table 10. Runtime results for other feature selection techniques (Part 2).

9.2. Runtime Results (Detailed)

Method	LSH Part	RST Part	Total
LSH (F=4,B=2)	1 node: 249.1861	1 node: 44.77727	1 node: 293.9634
	2 nodes: 281.8208	2 nodes: 49.16752	2 nodes: 330.9884
	4 nodes: 252.6248	4 nodes: 21.05362	4 nodes: 273.6784
	8 nodes: 250.6381	8 nodes: 17.36132	8 nodes: 267.9994
	16 nodes: 253.5395	16 nodes: 10.56732	16 nodes: 264.1068
LSH (F=4,B=5)	1 node: 70.68899	1 node: 23.6468	1 node: 94.33579
	2 nodes: 76.22245	2 nodes: 45.26454	2 nodes: 121.487
	4 nodes: 86.15117	4 nodes: 19.62175	4 nodes: 105.7729
	8 nodes: 67.20983	8 nodes: 15.99789	8 nodes: 83.20773
	16 nodes: 64.00956	16 nodes: 16.87221	16 nodes: 80.88177
LSH (F=4,B=10)	1 node: 43.64644	1 node: 25.34797	1 node: 68.99441
	2 nodes: 43.62331	2 nodes: 51.59557	2 nodes: 95.21888
	4 nodes: 34.44596	4 nodes: 21.82497	4 nodes: 56.27094
	8 nodes: 36.65276	8 nodes: 15.59631	8 nodes: 52.24907
	16 nodes: 38.99837	16 nodes: 15.99668	16 nodes: 54.99504
LSH (F=4,B=25)	1 node: 28.85205	1 node: 23.92756	1 node: 52.77961
	2 nodes: 31.73694	2 nodes: 49.69851	2 nodes: 81.43544
	4 nodes: 24.98696	4 nodes: 23.8733	4 nodes: 48.86025
	8 nodes: 29.10556	8 nodes: 14.30883	8 nodes: 43.4144
	16 nodes: 23.40279	16 nodes: 16.5764	16 nodes: 39.97918
LSH (F=4,B=50)	1 node: 25.68635	1 node: 23.32208	1 node: 49.00843
	2 nodes: 29.4082	2 nodes: 45.23537	2 nodes: 74.64357
	4 nodes: 23.8163	4 nodes: 22.44792	4 nodes: 46.26422
	8 nodes: 27.05567	8 nodes: 14.47553	8 nodes: 41.5312
	16 nodes: 22.43465	16 nodes: 16.25009	16 nodes: 38.68473
LSH (F=5,B=2)	1 node: 263.9243	1 node: 93.02023	1 node: 356.9446
	2 nodes: 190.1182	2 nodes: 102.0403	2 nodes: 292.1586
	4 nodes: 221.3475	4 nodes: 32.37374	4 nodes: 253.7213
	8 nodes: 238.0008	8 nodes: 20.25855	8 nodes: 258.2594
	16 nodes: 198.6282	16 nodes: 19.83463	16 nodes: 218.4629
LSH (F=5,B=5)	1 node: 72.9804	1 node: 43.49173	1 node: 116.4721
	2 nodes: 60.77011	2 nodes: 109.5385	2 nodes: 170.3086
	4 nodes: 63.23359	4 nodes: 35.17539	4 nodes: 98.40898
	8 nodes: 66.70733	8 nodes: 20.38688	8 nodes: 87.09421
	16 nodes: 56.6734	16 nodes: 20.56072	16 nodes: 77.23412
LSH (F=5,B=10)	1 node: 39.88054	1 node: 41.43311	1 node: 81.31365
	2 nodes: 43.79706	2 nodes: 93.2827	2 nodes: 137.0798
	4 nodes: 32.63216	4 nodes: 37.38687	4 nodes: 70.01903
	8 nodes: 31.49336	8 nodes: 20.16216	8 nodes: 51.65553
	16 nodes: 31.91303	16 nodes: 22.37444	16 nodes: 54.28747
LSH (F=5,B=25)	1 node: 28.36333	1 node: 37.81126	1 node: 66.17459
	2 nodes: 29.15167	2 nodes: 111.8691	2 nodes: 141.0207
	4 nodes: 22.67827	4 nodes: 35.0176	4 nodes: 57.69587
	8 nodes: 30.128	8 nodes: 14.09089	8 nodes: 44.21889
	16 nodes: 26.06495	16 nodes: 21.36248	16 nodes: 47.42742
LSH (F=5,B=50)	1 node: 25.76798	1 node: 36.88176	1 node: 62.64974
	2 nodes: 26.97555	2 nodes: 95.02764	2 nodes: 122.0032
	4 nodes: 21.72356	4 nodes: 35.68894	4 nodes: 57.4125
	8 nodes: 23.83919	8 nodes: 23.53858	8 nodes: 47.37777
	16 nodes: 23.56475	16 nodes: 21.80079	16 nodes: 45.36554

Table 11. Detailed runtime (split by LSH and RST part) for different parameters of LSH-dRST (Part 1).

Method	LSH Part	RST Part	Total
LSH (F=8,B=2)	1 node: 153.141	1 node: 977.6144	1 node: 1130.755
	2 nodes: 161.741	2 nodes: 997.1169	2 nodes: 1158.858
	4 nodes: 145.5205	4 nodes: 322.9155	4 nodes: 468.436
	8 nodes: 163.9673	8 nodes: 182.7261	8 nodes: 346.6934
	16 nodes: 137.1214	16 nodes: 120.7217	16 nodes: 257.8431
LSH (F=8,B=5)	1 node: 56.87063	1 node: 976.0553	1 node: 1032.926
	2 nodes: 49.36937	2 nodes: 1074.171	2 nodes: 1123.54
	4 nodes: 44.11492	4 nodes: 496.0974	4 nodes: 540.2123
	8 nodes: 45.80784	8 nodes: 190.0493	8 nodes: 235.8572
	16 nodes: 42.46138	16 nodes: 153.9585	16 nodes: 196.4199
LSH (F=8,B=10)	1 node: 34.52373	1 node: 974.9466	1 node: 1009.47
	2 nodes: 38.80107	2 nodes: 1240.97	2 nodes: 1279.771
	4 nodes: 27.2223	4 nodes: 366.99	4 nodes: 394.2123
	8 nodes: 26.39205	8 nodes: 201.5891	8 nodes: 227.9812
	16 nodes: 26.84447	16 nodes: 173.3261	16 nodes: 200.1706
LSH (F=8,B=25)	1 node: 25.40424	1 node: 967.9102	1 node: 993.3145
	2 nodes: 27.50088	2 nodes: 1424.926	2 nodes: 1452.427
	4 nodes: 22.44209	4 nodes: 372.8452	4 nodes: 395.2873
	8 nodes: 22.03004	8 nodes: 220.4777	8 nodes: 242.5078
	16 nodes: 23.92952	16 nodes: 195.3825	16 nodes: 219.3121
LSH (F=8,B=50)	1 node: 24.64281	1 node: 962.722	1 node: 987.3648
	2 nodes: 28.00279	2 nodes: 1485.364	2 nodes: 1513.367
	4 nodes: 28.49751	4 nodes: 305.153	4 nodes: 333.6505
	8 nodes: 28.49436	8 nodes: 208.8016	8 nodes: 237.296
	16 nodes: 21.51559	16 nodes: 359.3658	16 nodes: 380.8814
LSH (F=10,B=2)	1 node: 131.8026	1 node: 1296.215	1 node: 1428.017
	2 nodes: 127.2752	2 nodes: 6093.168	2 nodes: 6220.443
	4 nodes: 120.0283	4 nodes: 1441.898	4 nodes: 1561.927
	8 nodes: 134.1363	8 nodes: 923.8355	8 nodes: 1057.972
	16 nodes: 114.2739	16 nodes: 983.4926	16 nodes: 1097.766
LSH (F=10,B=5)	1 node: 42.83346	1 node: 1287.053	1 node: 1329.887
	2 nodes: 46.27018	2 nodes: 6564.41	2 nodes: 6610.68
	4 nodes: 36.7243	4 nodes: 1565.689	4 nodes: 1602.414
	8 nodes: 39.44878	8 nodes: 907.6247	8 nodes: 947.0734
	16 nodes: 39.67369	16 nodes: 942.494	16 nodes: 982.1677
LSH (F=10,B=10)	1 node: 30.9031	1 node: 1299.212	1 node: 1330.115
	2 nodes: 31.69981	2 nodes: 6837.121	2 nodes: 6868.82
	4 nodes: 26.93947	4 nodes: 1634.859	4 nodes: 1661.798
	8 nodes: 25.8574	8 nodes: 1026.669	8 nodes: 1052.526
	16 nodes: 24.99526	16 nodes: 905.2687	16 nodes: 930.264
LSH (F=10,B=25)	1 node: 25.36443	1 node: 1277.522	1 node: 1302.886
	2 nodes: 26.52125	2 nodes: 5466.758	2 nodes: 5493.28
	4 nodes: 23.93727	4 nodes: 1713.618	4 nodes: 1737.555
	8 nodes: 21.49535	8 nodes: 1003.298	8 nodes: 1024.793
	16 nodes: 21.51204	16 nodes: 1064.629	16 nodes: 1086.141
LSH (F=10,B=50)	1 node: 24.48343	1 node: 1275.881	1 node: 1300.365
	2 nodes: 26.19428	2 nodes: 6864.866	2 nodes: 6891.06
	4 nodes: 22.40849	4 nodes: 1780.721	4 nodes: 1803.13
	8 nodes: 20.85069	8 nodes: 1297.125	8 nodes: 1317.976
	16 nodes: 21.04216	16 nodes: 943.0337	16 nodes: 964.0759

Table 12. Detailed runtime (split by LSH and RST part) for different parameters of LSH-dRST (Part 2).

9.3. Classification Results (Random Forest)

Method	#Features	Accuracy	Precision	Recall	F1 Score
LSH-dRST (F=4,B=2)	3162.6 (20.93562)	0.7259 (0.0170)	0.7473 (0.0173)	0.7259 (0.0170)	0.7165 (0.0172)
LSH-dRST (F=4,B=5)	3153.4 (28.18333)	0.7299 (0.0108)	0.7544 (0.0090)	0.7299 (0.0108)	0.7209 (0.0105)
LSH-dRST (F=4,B=10)	3136.6 (20.59854)	0.7342 (0.0080)	0.7564 (0.0081)	0.7342 (0.0080)	0.7253 (0.0082)
LSH-dRST (F=4,B=25)	3129.6 (30.31171)	0.7317 (0.0173)	0.7539 (0.0162)	0.7317 (0.0173)	0.7230 (0.0168)
LSH-dRST (F=4,B=50)	3143.8 (27.93206)	**0.7402** (0.0205)	**0.7626** (0.0189)	**0.7402** (0.0205)	**0.7326** (0.0212)
LSH-dRST (F=5,B=2)	2562.6 (6.94982)	0.7185 (0.0307)	0.7404 (0.0307)	0.7185 (0.0307)	0.7090 (0.0319)
LSH-dRST (F=5,B=5)	2549.2 (27.28919)	0.7163 (0.0117)	0.7408 (0.0119)	0.7163 (0.0117)	0.7077 (0.0118)
LSH-dRST (F=5,B=10)	2579.0 (13.83835)	0.7189 (0.0110)	0.7418 (0.0114)	0.7189 (0.0110)	0.7101 (0.0108)
LSH-dRST (F=5,B=25)	2548.2 (17.93600)	0.7096 (0.0138)	0.7344 (0.0124)	0.7096 (0.0138)	0.7002 (0.0139)
LSH-dRST (F=5,B=50)	2535.4 (30.44339)	0.7058 (0.0233)	0.7276 (0.0240)	0.7058 (0.0233)	0.6967 (0.0250)
LSH-dRST (F=8,B=2)	1700.8 (10.32957)	0.6542 (0.0282)	0.6839 (0.0270)	0.6542 (0.0282)	0.6435 (0.0286)
LSH-dRST (F=8,B=5)	1709.8 (18.28114)	0.6459 (0.0250)	0.6734 (0.0233)	0.6459 (0.0250)	0.6355 (0.0255)
LSH-dRST (F=8,B=10)	1684.8 (14.09610)	0.6751 (0.0339)	0.6987 (0.0311)	0.6751 (0.0339)	0.6649 (0.0347)
LSH-dRST (F=8,B=25)	1673.0 (34.72751)	0.6423 (0.0241)	0.6695 (0.0227)	0.6423 (0.0241)	0.6314 (0.0234)
LSH-dRST (F=8,B=50)	1659.8 (12.11198)	0.6597 (0.0409)	0.6844 (0.0399)	0.6597 (0.0409)	0.6490 (0.0426)
LSH-dRST (F=10,B=2)	1452.2 (14.04279)	0.6521 (0.0195)	0.6758 (0.0181)	0.6521 (0.0195)	0.6404 (0.0213)
LSH-dRST (F=10,B=5)	1438.8 (27.55358)	0.6444 (0.0246)	0.6664 (0.0265)	0.6444 (0.0246)	0.6322 (0.0263)
LSH-dRST (F=10,B=10)	1423.6 (21.19670)	0.6409 (0.0146)	0.6662 (0.0152)	0.6409 (0.0146)	0.6294 (0.0148)
LSH-dRST (F=10,B=25)	1407.0 (16.00000)	0.6518 (0.0330)	0.6747 (0.0330)	0.6518 (0.0330)	0.6394 (0.0337)
LSH-dRST (F=10,B=50)	1414.2 (21.62637)	0.6266 (0.0173)	0.6535 (0.0210)	0.6266 (0.0173)	0.6147 (0.0174)

Table 13. Random Forest: Classification results for different parameters of LSH-dRST.

Method	#Features	Accuracy	Precision	Recall	F1 Score
Original	10000	0.7997	0.8151	0.7997	0.7925
	(-)	(0.0068)	(0.0068)	(0.0068)	(0.0072)
CV (t=0)	4149	0.6792	0.7245	0.6792	0.6705
	(-)	(0.0055)	(0.0054)	(0.0055)	(0.0051)
CV (1427)	1427	0.5325	0.6084	0.5325	0.5266
	(-)	(0.0074)	(0.0119)	(0.0074)	(0.0082)
CV (1686)	1686	0.5526	0.6153	0.5526	0.5425
	(-)	(0.0060)	(0.0100)	(0.0060)	(0.0073)
CV (2555)	2555	0.6090	0.6714	0.6090	0.6023
	(-)	(0.0061)	(0.0073)	(0.0061)	(0.0063)
CV (3145)	3145	0.6386	0.6886	0.6386	0.6288
	(-)	(0.0043)	(0.0065)	(0.0043)	(0.0042)
Cfs (Greedy)	41	0.6857	0.6864	0.6857	0.6772
	(-)	(0.0049)	(0.0058)	(0.0049)	(0.0048)
Chi Squared (t=0)	113	0.7115	0.7187	0.7115	0.7074
	(-)	(0.0034)	(0.0037)	(0.0034)	(0.0034)
Chi Squared (1427)	1427	0.7312	0.7386	0.7312	0.7214
	(-)	(0.0043)	(0.0063)	(0.0043)	(0.0049)
Chi Squared (1686)	1686	0.7322	0.7407	0.7322	0.7224
	(-)	(0.0058)	(0.0064)	(0.0058)	(0.0063)
Chi Squared (2555)	2555	0.7399	0.7508	0.7399	0.7293
	(-)	(0.0030)	(0.0037)	(0.0030)	(0.0032)
Chi Squared (3145)	3145	0.7529	0.7668	0.7529	0.7432
	(-)	(0.0044)	(0.0066)	(0.0044)	(0.0049)
Consistency (Greedy)	30	0.6493	0.6500	0.6493	0.6395
	(-)	(0.0060)	(0.0069)	(0.0060)	(0.0060)
Correlation (t=0)	10000	0.7988	0.8167	0.7988	0.7921
	(-)	(0.0050)	(0.0043)	(0.0050)	(0.0053)
Correlation (1427)	1427	0.7818	0.7940	0.7818	0.7744
	(-)	(0.0033)	(0.0038)	(0.0033)	(0.0037)
Correlation (1686)	1686	0.7860	0.7977	0.7860	0.7789
	(-)	(0.0032)	(0.0027)	(0.0032)	(0.0033)
Correlation (2555)	2555	0.7872	0.8026	0.7872	0.7802
	(-)	(0.0040)	(0.0056)	(0.0040)	(0.0046)
Correlation (3145)	3145	0.7936	0.8097	0.7936	0.7861
	(-)	(0.0067)	(0.0065)	(0.0067)	(0.0071)
Gain Ratio (t=0)	113	0.7061	0.7141	0.7061	0.7020
	(-)	(0.0045)	(0.0037)	(0.0045)	(0.0043)
Gain Ratio (1427)	1427	0.7337	0.7402	0.7337	0.7243
	(-)	(0.0057)	(0.0062)	(0.0057)	(0.0064)
Gain Ratio (1686)	1686	0.7343	0.7418	0.7343	0.7243
	(-)	(0.0045)	(0.0051)	(0.0045)	(0.0049)
Gain Ratio (2555)	2555	0.7405	0.7502	0.7405	0.7305
	(-)	(0.0040)	(0.0043)	(0.0040)	(0.0044)
Gain Ratio (3145)	3145	0.7543	0.7675	0.7543	0.7450
	(-)	(0.0037)	(0.0040)	(0.0037)	(0.0038)
Info Gain (t=0)	113	0.7089	0.7174	0.7089	0.7053
	(-)	(0.0034)	(0.0033)	(0.0034)	(0.0035)
Info Gain (1427)	1427	0.7335	0.7404	0.7335	0.7240
	(-)	(0.0046)	(0.0041)	(0.0046)	(0.0042)
Info Gain (1686)	1686	0.7305	0.7415	0.7305	0.7213
	(-)	(0.0056)	(0.0060)	(0.0056)	(0.0061)
Info Gain (2555)	2555	0.7408	0.7502	0.7408	0.7308
	(-)	(0.0048)	(0.0063)	(0.0048)	(0.0057)
Info Gain (3145)	3145	0.7559	0.7692	0.7559	0.7461
	(-)	(0.0051)	(0.0057)	(0.0051)	(0.0055)

Table 14. Random Forest: Classification results for other feature selection techniques (Part 1).

Method	#Features	Accuracy	Precision	Recall	F1 Score
Relief F (t=0)	7957	0.8005	0.8174	0.8005	0.7941
	(-)	(0.0042)	(0.0046)	(0.0042)	(0.0045)
Relief F (1427)	1427	0.7813	0.7955	0.7813	0.7735
	(-)	(0.0028)	(0.0033)	(0.0028)	(0.0029)
Relief F (1686)	1686	0.7889	0.8005	0.7889	0.7816
	(-)	(0.0050)	(0.0064)	(0.0050)	(0.0061)
Relief F (2555)	2555	0.7909	0.8065	0.7909	0.7844
	(-)	(0.0058)	(0.0058)	(0.0058)	(0.0063)
Relief F (3145)	3145	0.7944	0.8103	0.7944	0.7878
	(-)	(0.0045)	(0.0049)	(0.0045)	(0.0049)
Significance (t=0)	10000	0.7988	0.8159	0.7988	0.7914
	(-)	(0.0067)	(0.0073)	(0.0067)	(0.0079)
Significance (1427)	1427	0.7314	0.7392	0.7314	0.7224
	(-)	(0.0052)	(0.0059)	(0.0052)	(0.0056)
Significance (1686)	1686	0.7330	0.7421	0.7330	0.7238
	(-)	(0.0038)	(0.0046)	(0.0038)	(0.0039)
Significance (2555)	2555	0.7426	0.7524	0.7426	0.7329
	(-)	(0.0038)	(0.0055)	(0.0038)	(0.0046)
Significance (3145)	3145	0.7538	0.7675	0.7538	0.7442
	(-)	(0.0046)	(0.0047)	(0.0046)	(0.0052)
Sp-RST (1000)	6171.333	0.7912	0.8095	0.7912	0.7838
	(26.01282)	(0.0063)	(0.0064)	(0.0063)	(0.0069)
Sp-RST (1250)	5566.000	0.7902	0.8059	0.7902	0.7826
	(23.89142)	(0.0077)	(0.0079)	(0.0077)	(0.0083)
Sp-RST (2000)	4117.000	0.7784	0.7951	0.7784	0.7697
	(18.95257)	(0.0067)	(0.0070)	(0.0067)	(0.0071)
Sp-RST (2500)	3205.333	0.7710	0.7869	0.7710	0.7627
	(24.35296)	(0.0065)	(0.0070)	(0.0065)	(0.0070)
Sum Squares Ratio (1427)	1427	0.7139	0.7310	0.7251	0.6582
	(-)	(0.0022)	(0.0028)	(0.0023)	(0.0031)
Sum Squares Ratio (1686)	1686	0.8719	0.8865	0.8866	0.8338
	(-)	(0.0008)	(0.0016)	(0.0098)	(0.0089)
Sum Squares Ratio (2555)	2555	0.8585	0.8219	0.8642	0.8025
	(-)	(0.0004)	(0.0249)	(0.0005)	(0.0010)
Sum Squares Ratio (3145)	3145	**0.9513**	**0.9534**	**0.9577**	**0.9345**
	(-)	(0.0000)	(0.0000)	(0.0000)	(0.0000)
Symmetrical Uncert (t=0)	113	0.7099	0.7168	0.7099	0.7055
	(-)	(0.0036)	(0.0032)	(0.0036)	(0.0033)
Symmetrical Uncert (1427)	1427	0.7309	0.7383	0.7309	0.7215
	(-)	(0.0041)	(0.0051)	(0.0041)	(0.0041)
Symmetrical Uncert (1686)	1686	0.7345	0.7418	0.7345	0.7250
	(-)	(0.0060)	(0.0062)	(0.0060)	(0.0066)
Symmetrical Uncert (2555)	2555	0.7425	0.7521	0.7425	0.7325
	(-)	(0.0039)	(0.0035)	(0.0039)	(0.0043)
Symmetrical Uncert (3145)	3145	0.7533	0.7657	0.7533	0.7432
	(-)	(0.0049)	(0.0068)	(0.0049)	(0.0049)

Table 15. Random Forest: Classification results for other feature selection techniques (Part 2).

9.4. Classification Results (Naive Bayes)

Method	#Features	Accuracy	Precision	Recall	F1 Score
LSH (F=4,B=2)	3162.6 (20.93562)	0.5025 (0.0085)	0.5385 (0.0102)	0.5025 (0.0085)	0.5069 (0.0084)
LSH (F=4,B=5)	3153.4 (28.18333)	0.5037 (0.0099)	0.5413 (0.0114)	0.5037 (0.0099)	0.5087 (0.0104)
LSH (F=4,B=10)	3136.6 (20.59854)	0.5065 (0.0103)	0.5410 (0.0136)	0.5065 (0.0103)	0.5098 (0.0120)
LSH (F=4,B=25)	3129.6 (30.31171)	**0.5103** (0.0070)	**0.5486** (0.0100)	**0.5103** (0.0070)	**0.5149** (0.0078)
LSH (F=4,B=50)	3143.8 (27.93206)	0.5004 (0.0140)	0.5354 (0.0180)	0.5004 (0.0141)	0.5047 (0.0152)
LSH (F=5,B=2)	2562.6 (6.94982)	0.4770 (0.0108)	0.5132 (0.0096)	0.4769 (0.0108)	0.4810 (0.0105)
LSH (F=5,B=5)	2549.2 (27.28919)	0.4837 (0.0074)	0.5174 (0.0075)	0.4837 (0.0073)	0.4873 (0.0068)
LSH (F=5,B=10)	2579.0 (13.83835)	0.4804 (0.0070)	0.5156 (0.0085)	0.4804 (0.0070)	0.4845 (0.0071)
LSH (F=5,B=25)	2548.2 (17.93600)	0.4854 (0.0164)	0.5142 (0.0194)	0.4854 (0.0164)	0.4876 (0.0171)
LSH (F=5,B=50)	2535.4 (30.44339)	0.4807 (0.0132)	0.5134 (0.0144)	0.4808 (0.0132)	0.4839 (0.0132)
LSH (F=8,B=2)	1700.8 (10.32957)	0.4550 (0.0138)	0.4848 (0.0172)	0.4550 (0.0138)	0.4592 (0.0148)
LSH (F=8,B=5)	1709.8 (18.28114)	0.4496 (0.0078)	0.4794 (0.0092)	0.4496 (0.0077)	0.4535 (0.0077)
LSH (F=8,B=10)	1684.8 (14.09610)	0.4513 (0.0153)	0.4776 (0.0179)	0.4512 (0.0153)	0.4537 (0.0159)
LSH (F=8,B=25)	1673.0 (34.72751)	0.4538 (0.0144)	0.4835 (0.0153)	0.4539 (0.0144)	0.4580 (0.0142)
LSH (F=8,B=50)	1659.8 (12.11198)	0.4550 (0.0169)	0.4846 (0.0179)	0.4551 (0.0169)	0.4586 (0.0167)
LSH (F=10,B=2)	1452.2 (14.04279)	0.4538 (0.0080)	0.4797 (0.0113)	0.4537 (0.0080)	0.4564 (0.0086)
LSH (F=10,B=5)	1438.8 (27.55358)	0.4391 (0.0111)	0.4657 (0.0136)	0.4391 (0.0111)	0.4433 (0.0123)
LSH (F=10,B=10)	1423.6 (21.19670)	0.4427 (0.0112)	0.4678 (0.0105)	0.4426 (0.0111)	0.4449 (0.0105)
LSH (F=10,B=25)	1407.0 (16.00000)	0.4543 (0.0139)	0.4829 (0.0156)	0.4543 (0.0140)	0.4576 (0.0138)
LSH (F=10,B=50)	1414.2 (21.62637)	0.4443 (0.0120)	0.4660 (0.0116)	0.4442 (0.0120)	0.4467 (0.0120)

Table 16. Naive Bayes: Classification results for different parameters of LSH-dRST.

Method	#Features	Accuracy	Precision	Recall	F1 Score
Original	10000	0.6099	0.6451	0.6098	0.6118
	(-)	(0.0080)	(0.0082)	(0.0080)	(0.0087)
CV (t=0)	4149	0.5152	0.5703	0.5152	0.5213
	(-)	(0.0067)	(0.0082)	(0.0068)	(0.0067)
CV (1427)	1427	0.3777	0.4137	0.3776	0.3804
	(-)	(0.0053)	(0.0068)	(0.0054)	(0.0053)
CV (1686)	1686	0.3997	0.4382	0.3997	0.4022
	(-)	(0.0037)	(0.0065)	(0.0037)	(0.0040)
CV (2555)	2555	0.4528	0.5050	0.4528	0.4576
	(-)	(0.0068)	(0.0079)	(0.0066)	(0.0070)
CV (3145)	3145	0.4843	0.5342	0.4844	0.4883
	(-)	(0.0083)	(0.0092)	(0.0083)	(0.0087)
Cfs (Greedy)	41	0.6083	0.6156	0.6084	0.6053
	(-)	(0.0039)	(0.0049)	(0.0039)	(0.0043)
Chi Squared (t=0)	113	0.5771	0.5870	0.5771	0.5705
	(-)	(0.0039)	(0.0055)	(0.0038)	(0.0043)
Chi Squared (1427)	1427	0.5670	0.5880	0.5670	0.5703
	(-)	(0.0053)	(0.0059)	(0.0053)	(0.0055)
Chi Squared (1686)	1686	0.5609	0.5782	0.5610	0.5628
	(-)	(0.0065)	(0.0071)	(0.0066)	(0.0067)
Chi Squared (2555)	2555	0.5303	0.5484	0.5302	0.5317
	(-)	(0.0045)	(0.0040)	(0.0045)	(0.0045)
Chi Squared (3145)	3145	0.5027	0.5261	0.5027	0.5049
	(-)	(0.0043)	(0.0057)	(0.0041)	(0.0046)
Consistency (Greedy)	30	0.5443	0.5514	0.5443	0.5391
	(-)	(0.0044)	(0.0050)	(0.0045)	(0.0044)
Correlation (t=0)	10000	0.6147	0.6512	0.6147	0.6170
	(-)	(0.0059)	(0.0063)	(0.0059)	(0.0069)
Correlation (1427)	1427	0.6409	0.6614	0.6409	0.6420
	(-)	(0.0056)	(0.0056)	(0.0057)	(0.0057)
Correlation (1686)	1686	0.6363	0.6647	0.6363	0.6401
	(-)	(0.0058)	(0.0049)	(0.0058)	(0.0055)
Correlation (2555)	2555	0.6211	0.6553	0.6211	0.6254
	(-)	(0.0059)	(0.0069)	(0.0058)	(0.0061)
Correlation (3145)	3145	0.6239	0.6593	0.6240	0.6275
	(-)	(0.0061)	(0.0067)	(0.0061)	(0.0065)
Gain Ratio (t=0)	113	0.5771	0.5870	0.5771	0.5705
	(-)	(0.0039)	(0.0055)	(0.0038)	(0.0043)
Gain Ratio (1427)	1427	0.5670	0.5880	0.5670	0.5703
	(-)	(0.0053)	(0.0059)	(0.0053)	(0.0055)
Gain Ratio (1686)	1686	0.5609	0.5782	0.5610	0.5628
	(-)	(0.0065)	(0.0071)	(0.0066)	(0.0067)
Gain Ratio (2555)	2555	0.5303	0.5484	0.5302	0.5317
	(-)	(0.0045)	(0.0040)	(0.0045)	(0.0045)
Gain Ratio (3145)	3145	0.5027	0.5261	0.5027	0.5049
	(-)	(0.0043)	(0.0057)	(0.0041)	(0.0046)
Info Gain (t=0)	113	0.5771	0.5870	0.5771	0.5705
	(-)	(0.0039)	(0.0055)	(0.0038)	(0.0043)
Info Gain (1427)	1427	0.5670	0.5880	0.5670	0.5703
	(-)	(0.0053)	(0.0059)	(0.0053)	(0.0055)
Info Gain (1686)	1686	0.5609	0.5782	0.5610	0.5628
	(-)	(0.0065)	(0.0071)	(0.0066)	(0.0067)
Info Gain (2555)	2555	0.5303	0.5484	0.5302	0.5317
	(-)	(0.0045)	(0.0040)	(0.0045)	(0.0045)
Info Gain (3145)	3145	0.5027	0.5261	0.5027	0.5049
	(-)	(0.0043)	(0.0057)	(0.0041)	(0.0046)

Table 17. Naive Bayes: Classification results for other feature selection techniques (Part 1).

Method	#Features	Accuracy	Precision	Recall	F1 Score
ReliefF (t=0)	7957	0.6087	0.6476	0.6087	0.6120
	(-)	(0.0061)	(0.0069)	(0.0060)	(0.0067)
ReliefF (1427)	1427	0.6231	0.6475	0.6231	0.6253
	(-)	(0.0061)	(0.0055)	(0.0060)	(0.0060)
ReliefF (1686)	1686	0.6199	0.6506	0.6198	0.6234
	(-)	(0.0055)	(0.0057)	(0.0055)	(0.0060)
ReliefF (2555)	2555	0.6093	0.6409	0.6093	0.6118
	(-)	(0.0071)	(0.0077)	(0.0071)	(0.0074)
ReliefF (3145)	3145	0.6004	0.6367	0.6005	0.6046
	(-)	(0.0073)	(0.0069)	(0.0071)	(0.0074)
Significance (t=0)	10000	0.6139	0.6475	0.6138	0.6151
	(-)	(0.0065)	(0.0065)	(0.0066)	(0.0068)
Significance (1427)	1427	0.5670	0.5880	0.5670	0.5703
	(-)	(0.0053)	(0.0059)	(0.0053)	(0.0055)
Significance (1686)	1686	0.5609	0.5782	0.5610	0.5628
	(-)	(0.0065)	(0.0071)	(0.0066)	(0.0067)
Significance (2555)	2555	0.5303	0.5484	0.5302	0.5317
	(-)	(0.0045)	(0.0040)	(0.0045)	(0.0045)
Significance (3145)	3145	0.5027	0.5261	0.5027	0.5049
	(-)	(0.0043)	(0.0057)	(0.0041)	(0.0046)
Sp-RST (1000)	6171.333	0.5934	0.6321	0.5934	0.5966
	(26.01282)	(0.0067)	(0.0065)	(0.0067)	(0.0069)
Sp-RST (1250)	5566.000	0.5893	0.6304	0.5893	0.5929
	(23.89142)	(0.0066)	(0.0067)	(0.0066)	(0.0067)
Sp-RST (2000)	4117.000	0.5736	0.6156	0.5736	0.5772
	(18.95257)	(0.0058)	(0.0071)	(0.0058)	(0.0061)
Sp-RST (2500)	3205.333	0.5634	0.6050	0.5635	0.5681
	(24.35296)	(0.0061)	(0.0073)	(0.0061)	(0.0068)
SumSquaresRatio (1427)	1427	**0.6680**	**0.6862**	**0.6679**	**0.6689**
	(-)	(0.0018)	(0.0032)	(0.0017)	(0.0021)
SumSquaresRatio (1686)	1686	0.6667	0.6857	0.6666	0.6675
	(-)	(0.0032)	(0.0047)	(0.0032)	(0.0038)
SumSquaresRatio (2555)	2555	0.6420	0.6680	0.6420	0.6439
	(-)	(0.0051)	(0.0059)	(0.0051)	(0.0054)
SumSquaresRatio (3145)	3145	0.6480	0.6726	0.6480	0.6493
	(-)	(0.0037)	(0.0032)	(0.0038)	(0.0039)
Symmetrical Uncert (t=0)	113	0.5771	0.5870	0.5771	0.5705
	(-)	(0.0039)	(0.0055)	(0.0038)	(0.0043)
Symmetrical Uncert (1427)	1427	0.5670	0.5880	0.5670	0.5703
	(-)	(0.0053)	(0.0059)	(0.0053)	(0.0055)
Symmetrical Uncert (1686)	1686	0.5609	0.5782	0.5610	0.5628
	(-)	(0.0065)	(0.0071)	(0.0066)	(0.0067)
Symmetrical Uncert (2555)	2555	0.5303	0.5484	0.5302	0.5317
	(-)	(0.0045)	(0.0040)	(0.0045)	(0.0045)
Symmetrical Uncert (3145)	3145	0.5027	0.5261	0.5027	0.5049
	(-)	(0.0043)	(0.0057)	(0.0041)	(0.0046)

Table 18. Naive Bayes: Classification results for other feature selection techniques (Part 2).