A scalable and effective rough set theory-based approach for big data pre-processing

Zaineb Chelly Dagdia, Christine Zarges, Gaël Beck, Mustapha Lebbah

Knowledge and Information Systems
https://doi.org/10.1007/s10115-020-01467-y

REGULAR PAPER

A scalable and effective rough set theory-based approach for big data pre-processing

Zaineb Chelly Dagdia[1,2,3] · Christine Zarges[2] · Gaël Beck[4] · Mustapha Lebbah[4]

Received: 13 September 2019 / Revised: 16 March 2020 / Accepted: 21 March 2020

Abstract
A big challenge in the knowledge discovery process is to perform data pre-processing, specifically feature selection, on a large amount of data and high dimensional attribute set. A variety of techniques have been proposed in the literature to deal with this challenge with different degrees of success as most of these techniques need further information about the given input data for thresholding, need to specify noise levels or use some feature ranking procedures. To overcome these limitations, rough set theory (RST) can be used to discover the dependency within the data and reduce the number of attributes enclosed in an input data set while using the data alone and requiring no supplementary information. However, when it comes to massive data sets, RST reaches its limits as it is highly computationally expensive. In this paper, we propose a scalable and effective rough set theory-based approach for large-scale data pre-processing, specifically for feature selection, under the Spark framework. In our detailed experiments, data sets with up to 10,000 attributes have been considered, revealing that our proposed solution achieves a good speedup and performs its feature selection task well without sacrificing performance. Thus, making it relevant to big data.

Keywords Big data · Data pre-processing · Rough set theory · Distributed processing · Scalability · High-performance computing

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.

✉ Zaineb Chelly Dagdia
chelly.zaineb@gmail.com; zaineb.chelly-dagdia@inria.fr

Christine Zarges
c.zarges@aber.ac.uk

Gaël Beck
beck@lipn.univ-paris13.fr

Mustapha Lebbah
mustapha.lebbah@lipn.univ-paris13.fr

1 CNRS, Inria, LORIA, Université de Lorraine, 54000 Nancy, France

2 Department of Computer Science, Aberystwyth University, Aberystwyth, UK

3 LARODEC, Institut Supérieur de Gestion de Tunis, Tunis, Tunisia

4 Computer Science Laboratory (LIPN), University Paris-North - 13, Villetaneuse, France

Published online: 02 May 2020

1 Introduction

In a broad variety of domains, data are being gathered and stored at an intense pace due to the Internet and the widespread use of databases [7]. This ongoing rapid growth of data that led to a new terminology 'big data' has created an immense need for a novel generation of computational techniques, theories and approaches to extract useful information, i.e., knowledge, from these voluminous gathered data. These theories and approaches are the key elements of the emerging domain of knowledge discovery in databases (KDD) [21]. More precisely, big data arise with many challenges such as in clustering [35], in classification [34], in mining [38] but mainly in dimensionality reduction and more precisely in feature selection as this is usually a source of potential data loss [13]. This motivated researchers to build an efficient and an automated knowledge discovery process with a special focus on its third step, namely data reduction.

Data reduction is an important point of interest as many real-world applications may have a very large number of features (attributes) [41]. For instance, among the most practically relevant and high-impact applications are biochemistry, genetics and molecular biology. In these biological sciences, the collected data, e.g., gene expression data, may easily have a number of attributes which is more than 10,000 [1]. However, not all of these attributes are crucial and needed since many of them can be redundant in the context of some other features or even completely irrelevant and insignificant to the task being handled. Therefore, several important issues arise when learning in such a situation among these are the problems of over-fitting to insignificant aspects of the given data, as well as the computational burden due to the process of several similar attributes that give some redundant information [17]. These problems may decrease the performance of any learning technique, e.g., a classification algorithm. Hence, to solve these problems, it is an important and significant research direction to automatically look for and only select a small subset of relevant attributes from the initial large set of attributes; that is, to perform feature selection. In fact, by removing the irrelevant and redundant attributes, feature selection is capable of reducing the dimensionality of the input data while speeding up the learning process, simplifying the learned model as well as increasing the performance [9,17].

At an abstract level, to reduce the high dimensionality of data sets, suitable techniques can be applied with respect to the requirements of the future KDD process. The taxonomy of these techniques falls into two main groups namely *feature selection techniques* and *feature extraction techniques* [17]. The main difference between the two approaches is that techniques for feature selection select a subset from the initial features while techniques for feature extraction create new attributes from the initial feature set. More precisely, feature extraction techniques transform the underlying semantic (meaning) of the attributes while feature selection techniques preserve the data set semantics in the process of reduction. In knowledge discovery, feature selection techniques are notably desirable as these ease the interpretability of the output knowledge. In this paper, we mainly focus on the feature selection category for big data pre-processing.

Technically, feature selection is a challenging process due to the very large search space that reflects the combinatorially large number of all possible feature combinations to select from. This task is becoming more difficult as the total number of attributes is increasing in many big data application domains combined with the increased complexity of those problems. Therefore, to cope with the vast amount of the given data, most of the state-of-the-art techniques employ some degree of reduction, and thus, an effective feature reduction technique is needed.

As one of data analysis techniques, rough set theory (RST) [27]-based approaches have been successfully and widely applied in data mining and knowledge discovery [16], and particularly for feature selection [31]. Nonetheless, in spite of being powerful rough set-based feature selection techniques, most of the classical algorithms are sequential ones, computationally expensive and can only handle non-large data sets. The fact that the RST-based algorithms are computationally expensive and the reason behind the methods' incapacity to handle high dimensional data is explained by the need to first generate all the possible combinations of attributes at once, then process these in turn to finally select the most pertinent and relevant set of attributes.

Nevertheless, as previously mentioned, since the number of attributes is becoming very large this task becomes more critical and challenging, and at this point the RST-based approaches reach their limits. More precisely, it is unfeasible to generate all the possible attribute combinations at once because of both hardware and memory constraints.

This leads us to advance in this disjointed field and broaden the application of the theory of rough sets in the domain of data mining and knowledge discovery for big data. This paper proposes a scalable and effective algorithm based on rough sets for large-scale data pre-processing, and specifically for big data feature selection. Based on a distributed implementation design using both Scala and the Apache Spark framework [36], our proposed distributed algorithm copes with the RST computational inefficiencies and its restriction to be only applied to non-large data sets. To deeply analyze the proposed distributed approach, experiments on big data sets with up to 10,000 features will be carried out for feature selection and classification. Results demonstrate that our proposed solution achieves a good speedup and performs its feature selection task well without sacrificing performance, making it relevant to big data.

The rest of this paper is structured as follows. Section 2 presents preliminary information and related work. Section 3 reviews the fundamentals of rough set theory for feature selection. Section 4 formalizes the motivation of this work and introduces our novel distributed algorithm based on rough sets for large-scale data pre-processing. The experimental setup is introduced in Sect. 5. The results of the performance analysis are given in Sect. 6, and the conclusion is given in Sect. 7.

2 Literature review

Feature selection is defined as the process that selects a subset of the most relevant and pertinent attributes from a large input set of original attributes. For example, feature selection is the task of finding key genes (i.e., biomarkers) from the very huge number of candidate genes in biological and biomedical problems [3]. It is also the task of discovering core indicators (i.e., attributes) to describe the dynamic business environment [25], or to select key terms (e.g., words or phrases) in text mining [2] or to construct essential visual contents (e.g., pixel, color, texture, or shape) in image analysis [15].

In many data mining and machine learning real-world problems, feature selection became a crucial and highly important data pre-processing step due to the abundance of noisy, irrelevant and/or misleading features that are in big data. To cope with this, the usefulness of a feature can be measured by its relevancy as well as its redundancy. In fact, a feature is considered to be relevant if it can predict the decision feature(s); otherwise, it is said to be irrelevant as it provides no useful information with reference to any context. On the other hand, a feature is considered to be redundant if it provides the same piece of information for the currently

selected features; this means that it is highly correlated with them. Hence, feature selection must provide beneficial results from big data as it should detect those attributes that present a high correlation with the decision feature(s), but at the same time are uncorrelated with each other.

In the literature, feature selection techniques can be broadly grouped into two main approaches which are *filter approaches* and *wrapper approaches* [9,17]. The key difference between the two approaches is that wrapper approaches involve a specific learning algorithm, e.g., classification algorithm, when it comes to evaluating the attribute subset. The applied learning algorithm is mainly used as a *black box* by the wrapper approach to evaluate the quality (i.e., the classification performance) of the selected attribute set. Technically, when an algorithm performs feature selection in an independent way of any learning algorithm, the approach is defined as a filter where the set of the irrelevant features are filtered out before the induction process. Filter approaches tend to be applicable to most real-world domains since they are independent from any specific induction algorithm. On the other side, if the evaluation task is linked or dependent to the task of the learning algorithm then the feature selection approach is a wrapper technique. This approach searches through the attribute subsets space using the training (or validation) accuracy value of a specific induction algorithm as the measure of utility for a candidate subset. Therefore, these approaches may generate subsets that are overly explicit and specific to the used learning algorithm, and hence, any modification in the learning model might render the attribute set suboptimal.

Each of these two feature selection categories has its advantages and shortcomings where the main distinguishing aspects are the computational speed and the possibility of over-fitting. Overall, in terms of speed of computation, filter algorithms are usually computationally less expensive and more general than the wrapper techniques. Wrappers are computationally expensive and can easily break down when dealing with a very large number of attributes. This is due to the adoption of a learning algorithm in the evaluation process of subsets [24,26]. In terms of over-fitting, the wrapper techniques have a higher learning capability so are more likely to overfit than filter techniques. It is important to mention that in the literature, some researchers classified feature selection techniques into three separate categories, namely the wrapper techniques, the embedded techniques and the filter techniques [24]. The embedded approaches tend to fuse feature selection and the learning approach into a single process. For large-scaled data sets having large number of features, the filter methods are usually a good option. Focusing on this category is the main scope of this paper.

Meanwhile, in the context of big data, it is worth mentioning that a detailed study was conducted in [6] where authors performed a deep analysis of the scalability of the state-of-the-art feature selection techniques that belong to the filter, the embedded and the wrapper techniques. In [6], it was demonstrated that the state-of-the-art feature selection techniques will obviously have scalability issues when dealing with big data. Authors have proved that the existent techniques will be inadequate for handling a high number of attributes in terms of training time and/or effectiveness in selecting the relevant set of features. Thus, 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 [10] which was introduced by Google and which 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 [40,43,44]. Recently, a set of new and more flexible paradigms have been proposed aiming at extending the standard

MapReduce approach, mainly Apache Spark[1] [36] which has been applied with success over a number of data mining and machine learning real-world problems [36]. Further details and descriptions of such distributed processing frameworks will be given in Sect. 4.1.

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 such as nn-gram, principal component analysis, discrete cosine transform, tokenizer, PolynomialExpansion, ElementwiseProduct, etc., and very few feature selection techniques which 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 [29], and an improved implementation of the classical minimum Redundancy and Maximum Relevance feature selection method [29]. This implementation includes several optimizations such as cache marginal probabilities, accumulation of redundancy (greedy approach) and a data-access by columns.[4] Nevertheless, most of these techniques suffer from some shortcomings. For instance, they usually require the user or expert to deal with the algorithms' parameterisation, noise levels specification, where some other techniques simply order the attributes set and let the user choose his/her own subset. There are some other feature selection techniques that require the user to indicate how many attributes should be selected, or they must give a threshold that determines when the algorithm should end, which are all counted as significant drawbacks. All of these require users to make a decision based on their own (possibly subjective) perception. To overcome the shortcomings of the state-of-the-art techniques, it seemed 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 [39].

The use of rough set theory in data mining and knowledge discovery, specifically for feature selection, has proved to be very successful in many application domains such as in classification [22], clustering [23] and in supply chain [5]. This success is explained by the several aspects of the theory in dealing with data. For example, the theory is able to analyze the facts hidden in data, does not need any supplementary information about the given data such as thresholds or expert knowledge on a particular domain and is also capable to find a minimal knowledge representation [11]. This is achieved by making use of the granularity structure of the provided data only.

Although algorithms based on rough sets have been widely used as efficient filter feature selectors, most of the classical rough set algorithms are sequential ones, computationally expensive and can only deal with non-large data sets. 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 unmanageable to build the set of all possible combinations of features.

[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.

In order to overcome these weaknesses, a set of parallel and distributed rough set methods has been proposed in the literature to ensure feature selection but in different contexts. For example, some of these distributed methods adopt some evolutionary algorithms, such as the work proposed in [12], where authors defined a hierarchical MapReduce implementation of a parallel genetic algorithm for determining the minimum rough set reduct, i.e., the set of the selected features. Within another context, the context of limited labeled big data, in [32], authors introduced a theoretic framework called local rough set and developed a series of corresponding concept approximation and attribute reduction algorithms with linear time complexity, which can efficiently and effectively work in limited labeled big data. In the context of distributed decision information systems, i.e., several separate data sets dealing with different contents/topics but concerning the same data items, in [19], authors proposed a distributed definition of rough sets to deal with the reduction of these information systems.

In this paper, and in contrast to the state-of-the-art methods, we mainly focus on the formalization of rough set theory in a distributed manner by using its granular concepts only, and without making use of any heuristics, e.g., evolutionary algorithms. We also focus on a single information system, i.e., a single big data set, which covers a single content/topic and which is characterized by a full and complete labeled data. Within this focus, and in the literature, a first attempt presenting a parallel rough set model was given in [8]. The main idea in [8] is to split the given big data set into several partitions, each with a smaller number of 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. However, it is important to mention that the scalability of [8] was only validated in terms of sizeup and scaleup with a change in the standard metrics definitions (the standard definitions are given in Sect. 6.2). Actually, the used definition of these two metrics was based on the number of features per partition instead of the standard definition where the evaluation has to be based on the total number of features in the database used.

In this paper, we propose a redesign of rough set theory for feature selection by giving a better definition of the work presented in [8], specifically when it comes to the validation of the method (Sect. 6). Our work, which is an extension of [8], 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. For the validation of our method, and in contrast to [8], we believe that using the overall number of attributes is a much more natural setup as it will give insights into the performance depending on the input data set rather than the partitions.

3 Rough sets for feature selection

Rough set theory (RST) [27,28] is a formal approximation of the conventional set theory that supports approximations in decision making. This approach can extract knowledge from a problem domain in a concise way and retain the information content while reducing the involved amount of data [39]. This section focuses mainly on highlighting the fundamentals of RST for feature selection.

3.1 Preliminaries

In rough set theory, the training data set is called an *information table* or an *information system*. It is represented by a table where rows represent objects or instances and columns represent attributes or features. The information table can be defined as a tuple $S = (U, A)$, where

$U = \{u_1, u_2, \ldots, u_N\}$ is a non-empty finite set of N instances (or objects), called *universe*, and A is a non-empty set of $(n + k)$ attributes. The feature set $A = C \cup D$ can be partitioned into two subsets, namely the *conditional* feature set $C = \{a_1, a_2, \ldots, a_n\}$ consisting of n conditional attributes or predictors and the *decision* attribute $D = \{d_1, d_2, \ldots, d_k\}$ consisting of k decision attributes or output variables. Each feature $a \in A$ is described with a set of possible values V_a named the *domain* of a.

For each non-empty subset of attributes $P \subset C$, a binary relation called *P-indiscernibility* relation, which is the central concept of rough set theory, is defined as follows:

$$IND(P) = \{(u_1, u_2) \in U \times U : \forall a \in P, a(u_1) = a(u_2)\}. \tag{1}$$

where $a(u_i)$ refers to the value of attribute a for the instance u_i. This means if $(u_1, u_2) \in IND(P)$, then u_1 is indistinguishable (indiscernible) from u_2 by the attributes P. This relation is reflexive, symmetric and transitive.

The induced set of equivalence classes is denoted as $[u]_P$ where $u \in U$, and it partitions U into different blocks denoted as U/P.

The rough set approximates a concept or a target set of objects $X \subseteq U$ using the equivalence classes induced using P as follows:

$$\underline{P}(X) = \{u : [u]_P \subseteq X\}. \tag{2}$$

$$\overline{P}(X) = \{u : [u]_P \cap X \neq \emptyset\}. \tag{3}$$

where $\underline{P}(X)$ and $\overline{P}(X)$ denote the *P-lower* (certainly classified as members of X) and *P-upper* (possibly classified as members of X) approximations of X, respectively. The notation $\cap$ denotes the intersection operation.

The concept that defines the set of instances that are not certainly, but can possibly be classified in a specific way is named the *boundary region* and is defined as the difference between the two approximations. X is a *crisp set* if the boundary region is an empty set, i.e., accurate approximation, $\overline{P}(X) = \underline{P}(X)$; otherwise, it is a *rough set*.

To compare subsets of attributes, a *dependency measure* is defined. For instance, the dependency measure of an attribute subset Q on another attribute subset P is given as:

$$\gamma_P(Q) = \frac{|POS_P(Q)|}{|U|}. \tag{4}$$

where $0 \leq \gamma_P(Q) \leq 1$, $\cup$ denotes the union operation, $||$ denotes the set cardinality, and $POS_P(Q)$ is defined as:

$$POS_P(Q) = \bigcup_{X \in [u]_Q} \underline{P}(X). \tag{5}$$

$POS_P(Q)$ is the *positive region* of Q with respect to P and is the set of all elements of U that can be uniquely classified to blocks of the partition $[u]_Q$, by means of P. The closer $\gamma_P(Q)$ is to 1, the more Q depends on P.

Based on these basics, RST defines two important concepts for feature selection which are the *Core* and the *Reduct*.

3.2 Reduction process

The theory of rough sets aims at finding the smallest subset of the conditional attribute set in a way that the resulting reduced database remains consistent with respect to the decision attribute. A database is considered to be consistent in case where for every set of objects,

having identical feature values, the corresponding decision features are the same. To achieve this, the theory defines the *Reduct* concept and the *Core* concept.

Formally, in an information table, the unnecessary attributes can be categorized into either irrelevant features or into redundant features. The point is to define an heuristic that defines a measure to evaluate the necessity of a feature. Nevertheless, it is not easy to define an heuristic based on these qualitative definitions of *irrelevance* and *redundancy*. Therefore, authors in [20] defined *strong* relevance and *weak* relevance of an attribute based on the probability of the target concept occurrence given this attribute. The set of the strong relevant attributes presents the *indispensable* features in the sense that they cannot be removed from the information table without causing a loss of the prediction accuracy. On the other hand, the set of the weak relevant features can in some cases contribute to the prediction accuracy. Based on these definitions, both of the strong and the weak relevance concepts can provide good basics upon which the description of the importance of each feature can be defined. In the rough set terminology, the set of strong relevant attributes can be mapped to the *Core* concept while the *Reduct* concept defines a mixture of all strong relevant attributes and some weak relevant attributes.

To define these key concept, RST sets the following formalizations: A subset $R \subseteq C$ is said to be a *reduct* of C in the case where

$$\gamma_R(D) = \gamma_C(D) \tag{6}$$

and there is no $R' \subset R$ such that $\gamma_{R'}(D) = \gamma_R(D)$. Based on this formula, the *Reduct* can be defined as the minimal set of selected features that preserve the same dependency degree as the whole set of features.

In practice, from the given information table, it is possible that the theory generates a set of reducts: $RED_C^F(D)$. In this situation, any reduct in $RED_C^F(D)$ can be selected to describe the original information table.

The theory also defines the *Core* concept which is the set of features that are enclosed in all reducts. The *Core* concept is defined as

$$CORE_C(D) = \bigcap RED_C^F(D). \tag{7}$$

More precisely, the *Core* is defined as the set of features that cannot be omitted from the information table without inducing a collapse of the equivalence class structure. Thus, the *Core* is the most important subset of attributes, since none of its elements can be removed without affecting the classification power of attributes. This means that all the features which are in the *Core* are indispensable.

4 Parallel computing frameworks and the MapReduce programming model

In this section, we highlight the main solutions for big data processing. We, also, give a description of the MapReduce paradigm.

4.1 Parallel computing frameworks

With the dramatic increase of the amount of data, it has become crucial to implement a new set of technologies and tools that permit improved decision making and insight discovery. In this context, different techniques [33] 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 [37].

The encountered challenges in this concern 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 [14]. 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 process of computations is 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. The programming details of MapReduce as well as its basic concepts will be given in Sect. 4.2.

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 conducted research, we focus on Apache Spark. The distributed open source framework was initially developed in the UC Berkeley AMPLab for big data processing. Apache Spark is characterized by its capability of improving the system's effectiveness—which is

[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/.

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 Hadoop, in Hadoop MapReduce multiple jobs would be adjusted together to build a data pipeline. In this process, and in every level of that built 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 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 [36]. Spark affords 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 stream processing, machine learning and graph processing, e.g., MLlib [18]. The choice of this specific framework to design our proposed algorithm based on rough sets for big data feature selection is essentially based on several reasons which are as follows: (1) to offer a general solution based on a hybrid parallel framework, (2) Apache Spark provides high-speed benefits with a trade-off in the usage of high memory, (3) Spark is one of the well-known and certified distributed frameworks and also a mature hybrid system specifically when comparing it to some other frameworks in the market. These are considered as more niche in terms of their usage but more importantly they are still in their initial periods of adoption.[10]

4.2 The MapReduce paradigm

MapReduce [10] is one of the most popular processing techniques and program models for distributed computing to deal with big data. It was proposed by Google in 2004 and designed to easily scale data processing over multiple computing nodes. The MapReduce paradigm is composed of two main tasks/phases, namely the *map* phase and the *reduce* phase. At an 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 representation of the MapReduce framework is given in Fig. 1.

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:

$$map(key, value) \rightarrow \{(key', value'), \ldots\}. \tag{8}$$

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:

$$reduce(key', \{value', \ldots\}) \rightarrow (key', value''). \tag{9}$$

[10] https://www.digitalocean.com/community/tutorials/hadoop-storm-samza-spark-and-flink-big-data-frameworks-compared.

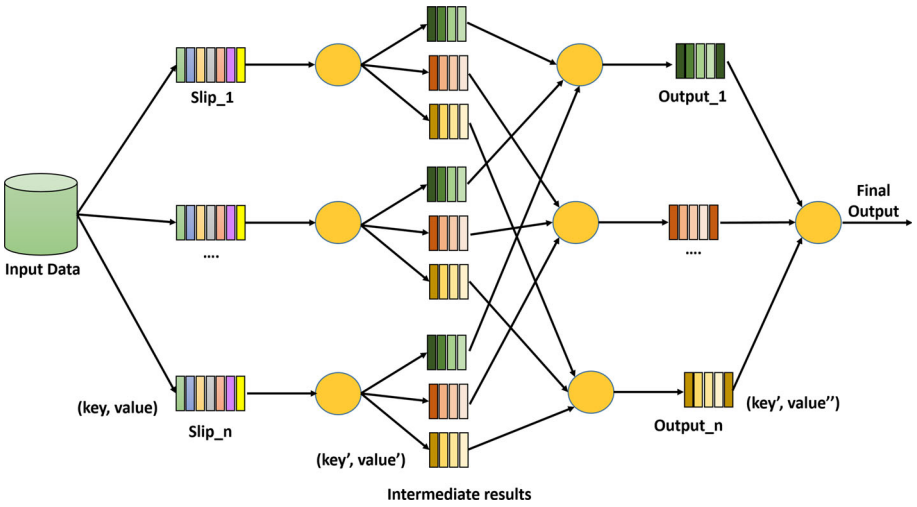


Fig. 1 The process of the MapReduce framework

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 focused on the use of Apache Spark.

5 The rough set distributed algorithm for big data feature selection

In this section, we will introduce our developed parallel rough set-based algorithm, that we name 'Sp-RST,' for big data pre-processing and specifically for feature selection. Sp-RST has a distributed architecture based on Apache Spark for a distributed and in-memory computation task. First, we will highlight the main motivation for developing the distributed Sp-RST algorithm by identifying the computational inefficiencies of the classical rough set theory which limit its application to small data sets only. Secondly, we will elucidate our Sp-RST solution as an efficient approach capable of performing big data feature selection without sacrificing performance.

5.1 Motivation and problem statement

Rough set theory for feature selection is an exhaustive search as the theory needs to compute every possible combination of attributes. The number of possible attribute subsets with m attributes from a set of N total attributes is $\binom{N}{m} = \frac{N!}{m!(N-m)!}$ [17]. Thus, the total number of feature subsets to generate is $\sum_{i=1}^{N} \binom{N}{i} = 2^N - 1$. For example, for $N = 30$ we have roughly 1 billion combinations. This constraint prevents us to use high-dimensional data sets as the number of feature subsets is growing exponentially in the total number of features N. Moreover, hardware constraints, specifically memory consumption, do not allow us to

store a high number of entries. This is because the system has to store the entire training data set in memory, together with all the supplementary data computations as well as the generated results. All of this data can be so big that its size can easily exceed the available RAM memory. These are the main motivations for our proposed Sp-RST solution, which makes use of parallelization.

5.2 The proposed solution

To overcome the standard RST inadequacy to perform feature selection in the context of big data, we propose our distributed Sp-RST solution. Technically, to handle a large set of data it is crucial to store all the given data set in a parallel framework and perform computations in a distributed way. Based on these requirements, we first partition the overall rough set feature selection process into a set of smaller and basic tasks that each can be processed independently. After that, we combine the generated intermediate outputs to finally build the sought result, i.e., the reduct set.

5.2.1 General model formalization

For feature selection, our learning problem aims to select a set of highly discriminating attributes from the initial large-scale input data set. The input base refers to the data stored in the distributed file system (DFS). To perform distributed tasks on the given DFS, a resilient distributed data set (RDD) is built. The latter can be formalized as a given information table that we name T_{RDD}. T_{RDD} is defined via a universe $U = \{x_1, x_2, \ldots, x_N\}$, which refers to the set of data instances (items), a large conditional feature set $C = \{c_1, c_2, \ldots, c_V\}$ that includes all the features of the T_{RDD} information table and finally via a decision feature D of the given learning problem. D refers to the label (also called class) of each T_{RDD} data item and is defined as follows: $D = \{d_1, d_2, \ldots, d_W\}$. C presents the conditional attribute pool from where the most significant attributes will be selected.

As explained in Sect. 5.1, the classical RST cannot deal with a very large number of features, which is defined as C in the T_{RDD} information table. Thus, to ensure the scalability of our proposed algorithm when dealing with a large number of attributes, Sp-RST first partitions the input T_{RDD} information table (the big data set) into a set of m data blocks based on splits from the conditional feature set C, i.e., m smaller data sets with a fewer number of features instead of using a single data block (T_{RDD}) with an unmanageable C number of features that we note as $T_{RDD}(C)$. The key idea is to generate m smaller data sets that we name $T_{RDD_{(i)}}$, where $i \in \{1, \ldots, m\}$, from the big T_{RDD} data set, where each $T_{RDD_{(i)}}$ is defined via a manageable number of features r, where $r \lll C = \{c_1, c_2, \ldots, c_V\}$ and $r \in \{1, \ldots, V\}$. The definition of the parameter r will be further explained in what follows. We note the resulting data block as $T_{RDD_{(i)}}(C_r)$. This leads to the following formalization: $T_{RDD} = \bigcup_{i=1}^{m} T_{RDD_{(i)}}(C_r)$, where $r \in \{1, \ldots, V\}$. As mentioned above, r defines the number of attributes that will be considered to build every $T_{RDD_{(i)}}$ data block. Based on this, every $T_{RDD_{(i)}}$ is built using r random attributes which are selected from C. Each $T_{RDD_{(i)}}$ is constructed based on r distinct features as there are no common attributes between all the built $T_{RDD_{(i)}}$. This leads to the following formalization: $\forall T_{RDD_{(i)}} : \nexists \{c_r\} = \bigcap_{i=1}^{m} T_{RDD_{(i)}}$. Figure 2 presents this data partitioning phase.

With respect to the parallel implementation design, the distributed Sp-RST algorithm will be applied to every $T_{RDD_{(i)}}(C_r)$ while gathering all the intermediate results from the distinct m created partitions; rather than being applied to the complete T_{RDD} that encloses the whole set

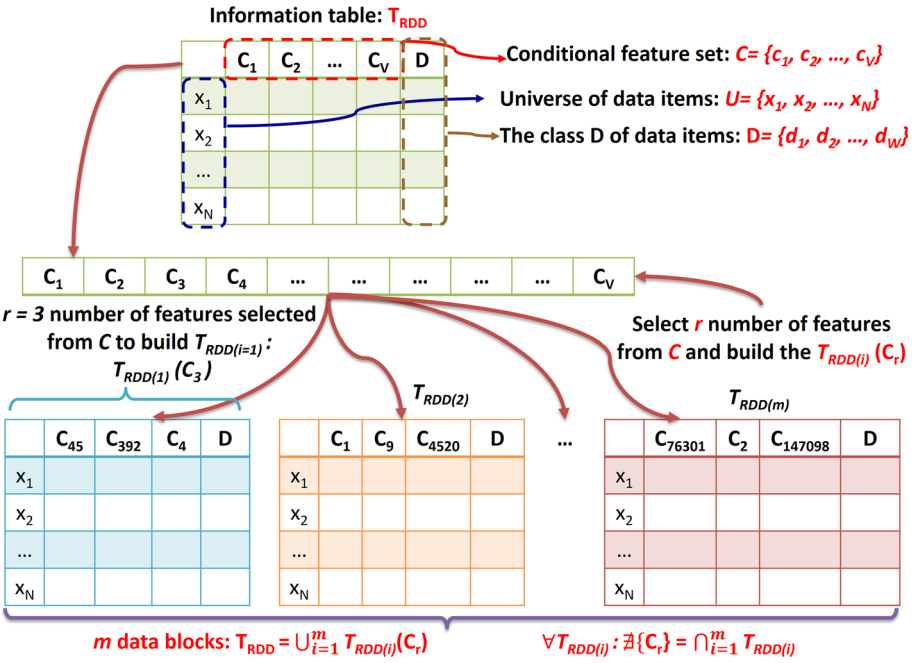


Fig. 2 The process of data partitioning

C of conditional features. Based on this design, we can ensure that the algorithm can perform its feature selection task on a computable number of attributes and therefore overcome the standard rough set computational inefficiencies. The pseudocode of our proposed distributed Sp-RST solution is highlighted in Algorithm 1.

To further guarantee the Sp-RST feature selection performance while avoiding any critical information loss, to evolve the algorithm and to refine it, Sp-RST runs over N iterations on the T_{RDD} m data blocks, i.e., N iterations on all the m built $T_{RDD_{(i)}}(C_r)$. Through all these N iterations, Sp-RST will first randomly build the m distinct $T_{RDD_{(i)}}(C_r)$ as explained above. Once this is achieved and for each partition, the algorithm's distributed tasks defined in Algorithm 1 (lines 5–10) will be performed. As noticed, line 1 in Algorithm 1 that defines the initial Sp-RST parallel job is performed outside the loop iteration. This process calculates the indiscernibility relation $IND(D)$ of the decision class D. The main reason for this implementation is that this process is totally separated from the m created partitions. This is because the output is tied to the label of the data instances and not on the attribute set.

Out from the iteration loop (line 12), the outcome of each created partition can be either only one reduct $RED_{i_{(D)}}(C_r)$ or a set (a family) of reducts $RED^{F}_{i_{(D)}}(C_r)$. As previously highlighted in Sect. 3, any reduct among the $RED^{F}_{i_{(D)}}(C_r)$ reducts can be selected to describe the $T_{RDD_{(i)}}(C_r)$ information table. Therefore, in case where Sp-RST generates a single reduct for a specific $T_{RDD_{(i)}}(C_r)$ partition, the final output of this attribute selection phase is the set of features defined in $RED_{i_{(D)}}(C_r)$. These attributes represent the most informative features among the C_r features and generate a new reduced $T_{RDD_{(i)}}$ defined as: $T_{RDD_{(i)}}(RED)$. The latter reduced base guarantees nearly the same data quality as its corresponding $T_{RDD_{(i)}}(C_r)$ which is based on the full attribute set C_r. In the other case where Sp-RST generates multiple reducts, the algorithm performs a random selection of a single reduct among the generated

Algorithm 1 The Sp-RST Algorithm

Inputs: T_{RDD}: information table, m: number of partitions, N: number of iterations
Output: *Reduct*

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

family of reducts $RED^{F}_{i_{(D)}}(C_r)$ to describe the corresponding $T_{RDD_{(i)}}(C_r)$. This random selection is supported by the RST fundamentals and is explained by the same level of importance of all the reducts defined in $RED^{F}_{i_{(D)}}(C_r)$. More precisely, any reduct included in the family of reducts $RED^{F}_{i_{(D)}}(C_r)$ can be selected to replace the $T_{RDD_{(i)}}(C_r)$ attributes.

At this level, the output of every i data block is $RED_{i_{(D)}}(C_r)$ which refers to the selected set of features. Nevertheless, since every $T_{RDD_{(i)}}$ is described using r distinct attributes and with respect to $T_{RDD} = \bigcup_{i=1}^{m} T_{RDD_{(i)}}(C_r)$, a union operator on the generated selected attributes is needed to represent the original T_{RDD}. This is defined as $Reduct_m = \bigcup_{i=1}^{m} RED_{i_{(D)}}(C_r)$ (Algorithm 1, lines 12–14). As previously highlighted, Sp-RST will perform its distributed tasks over the N iterations generating N $Reduct_m$. Therefore, finally, an intersection operator applied on all the obtained $Reduct_m$ is required. This is defined as $Reduct = \bigcap_{n=1}^{N} Reduct_m$. Sp-RST could diminish the dimensionality of the original data set from $T_{RDD}(C)$ to $T_{RDD}(Reduct)$ by removing irrelevant and redundant features at each computation level. Sp-RST could also simplify the learned model, speed up the overall learning process, and increase the performance of an algorithm, e.g., a classification algorithm, as will be discussed in the experimental setup section (Sect. 6). Figure 3 illustrates the global functioning of Sp-RST. In what follows, we will elucidate the different Sp-RST elementary distributed tasks.

5.2.2 Algorithmic details

As previously highlighted, the elementary Sp-RST distributed tasks will be executed on every $T_{RDD_{(i)}}$ partition defined by its C_r features ($T_{RDD_{(i)}}(C_r)$), except for the first step, Algorithm 1—line 1, which deals with the calculation of the indiscernibility relation for the decision class D: $IND(D)$. Sp-RST performs seven main distributed jobs to generate the final output, i.e., *Reduct*.

Sp-RST stars first of all by computing the indiscernibility relation for the decision class $D = \{d_1, d_2, \ldots, d_W\}$. We define the indiscernibility relation as $IND(D)$: $IND(d_i)$, where $i \in \{1, 2, \ldots, W\}$. Sp-RST will calculate $IND(D)$ for each decision class d_i by associating

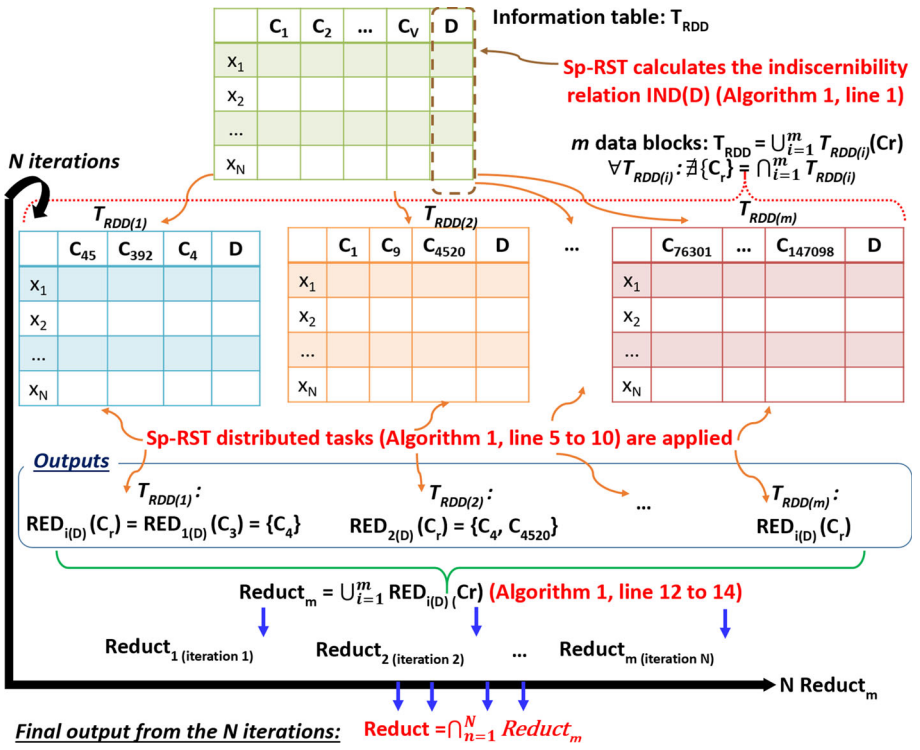


Fig. 3 The global functioning of Sp-RST

the same T_{RDD} data items (instances) that are expressed in the universe $U = \{x_1, \ldots, x_N\}$ and that belong to the same decision class d_i.

To achieve this task, Sp-RST 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 the T_{RDD} define the values. After that, the $foldByKey()$[11] transformation operation is applied to merge all values of each key in the transformed RDD output. This is to represent the sought $IND(D)$: $IND(d_i)$. The pseudo-code related to this distributed job is highlighted in Algorithm 2.

Algorithm 2 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 the values of each generated key using the $foldByKey()$ operation
3: Return $IND(D)$

[11] https://spark.apache.org/docs/0.7.3/api/core/spark/PairRDDFunctions.html.

After that and within a specific partition i, where $i \in \{1, 2, \ldots, m\}$ and m is the number of partitions, the algorithm generates the $AllComb_{(C_r)}$ RDD which reflects all the possible combinations of the C_r set of attributes. This is based on transforming the C_r RDD to the $AllComb_{(C_r)}$ RDD using the $flatmap()$[12] transformation operation and by using the $combinations()$ operation. This is shown in Algorithm 3.

Algorithm 3 Generate $AllComb_{(C_r)}$

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

In its third distributed job, Sp-RST calculates the indiscernibility relation $IND(AllComb_{(C_r)})$ for every created combination, i.e., the indiscernibility relation of every element in the output of Algorithm 3, and that we name $AllComb_{(C_r)_i}$. In this task and as described in Algorithm 4, 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 $AllComb_{(C_r)}$. 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: $\langle (AllComb_{(C_r)_i}$, 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 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 ($AllComb_{(C_r)_i}$, List of the features of x_i). This is to represent the sought $IND(AllComb_{(C_r)})$. At its third step, Sp-RST prepares the set of features that will be selected in the coming steps.

Algorithm 4 Calculate $IND(AllComb_{(C_r)})$

Inputs: T_{RDD_i}, $AllComb_{(C_r)}$
Output: $IND(AllComb_{(C_r)})$
1: Map the T_{RDD_i} 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 $\langle (AllComb_{(C_r)_i}$, 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 $AllComb_{(C_r)} : IND(AllComb_{(C_r)})$

In a next stage, Sp-RST computes the dependency degrees $\gamma(AllComb_{(C_r)})$ of each attribute combination as described in Algorithm 5. For this task, the distributed job requires three input parameters which are the calculated indiscernibility relations $IND(D)$, the $IND(AllComb_{(C_r)})$ and the set of all attribute combinations $AllComb_{(C_r)}$.

For every element $AllComb_{(C_r)_i}$ in $AllComb_{(C_r)}$, and using the $intersection()$ transformation, the job tests first if the intersection of every $IND(d_i)$ of $IND(d)$ with each element $IND(AllComb_{(C_r)})_i$ in $IND(AllComb_{(C_r)})$ holds all the elements in the latter parameter. This process refers to the calculation of the lower approximation as detailed in Sect. 3. We name the length of the resulting intersection as *LengthIntersect*. If the condition is satisfied then a

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

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 $IND(AllComb_{(C_r)})$ elements together with a $sum()$ function applied on the previous calculated results which are indeed based on the elements having the same $AllComb_{(C_r)_i}$.

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

Algorithm 5 Generate $DEP(AllComb_{(C_r)})$

Inputs: $AllComb_{(C_r)}$, $IND(D)$, $IND(AllComb_{(C_r)})$
Outputs: $\gamma(AllComb_{(C_r)})$, $Size_{(AllComb_{(C_r)})}$

1: **for** each element $AllComb_{(C_r)_i}$ in $AllComb_{(C_r)}$ **do**
2: **for** each element $IND(d_i)$ in $IND(D)$ **do**
3: **for** each element $IND(AllComb_{(C_r)})_i$ in $IND(AllComb_{(C_r)})$ **do**
4: Apply the $intersection()$ transformation over $IND(d_i)$ and $IND(AllComb_{(C_r)})_i$
5: Get the length of the resulting intersection that we name as $LengthIntersect$
6: **if** $LengthIntersect$ = length of $IND(AllComb_{(C_r)})_i$ **then** Score = $LengthIntersect$
7: **else** Score = 0
8: **End if**
9: **End for**
10: Apply a $reduce$ function over $IND(D)$ based on a $sum()$ function on the calculated scores which are indeed based on the elements having the same $IND(d_i)$
11: **End for**
12: Apply a $reduce$ function over $AllComb_{(C_r)}$ based on a $sum()$ function on the calculated results of Step (10) which are indeed based on the elements having the same $AllComb_{(C_r)_i}$
13: **End for**
14: Return $AllComb_{(C_r)} : \gamma(AllComb_{(C_r)}), Size_{(AllComb_{(C_r)})}$

Once all the dependencies are calculated, in Algorithm 6, Sp-RST looks for the maximum value of the dependency among all the computed $\gamma(AllComb_{(C_r)})$ using the $max()$ function operated on the given RDD input and which is referred to as RDD[$AllComb_{(C_r)}$, $Size_{(AllComb_{(C_r)})}$, $\gamma(AllComb_{(C_r)})$]. Specifically, the $max()$ function will be applied on the third argument of the given RDD, i.e., $\gamma(AllComb_{(C_r)})$.

Algorithm 6 Select $DEP_{max}(AllComb_{(C_r)})$

Input: RDD[$AllComb_{(C_r)}$, $Size_{(AllComb_{(C_r)})}$, $\gamma(AllComb_{(C_r)})$]
Output: $MaxDependency$

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

Let us recall that based on the RST preliminaries (seen in Sect. 3), the maximum dependency refers to not only the dependency of the whole attribute set (C_r) describing the $T_{RDD_i}(C_r)$ but also to the dependency of all the possible attribute combinations sat-

isfying the following constraint: $\gamma(AllComb_{(C_r)}) = \gamma(C_r)$. The maximum dependency $MaxDependency$ reflects the baseline value for the feature selection task.

In a next step, Sp-RST performs a filtering process using the $filter()$ function to only keep the set of all combinations which have the same dependency degrees, as the already selected dependency baseline value ($MaxDependency$), i.e., $\gamma(AllComb_{(C_r)}) = MaxDependency$. This is described in Algorithm 7. In fact, through these computations, the algorithm removes in each level the unnecessary attributes that may negatively influence the performance of any learning algorithm.

Algorithm 7 Filter $DEP_{max}(AllComb_{(C_r)})$

Inputs: RDD[$AllComb_{(C_r)}$, $Size_{(AllComb_{(C_r)})}$, $\gamma(AllComb_{(C_r)})$], $MaxDependency$

Outputs: Filtered-RDD[$AllComb_{(C_r)}$, $Size_{(AllComb_{(C_r)})}$, $\gamma(AllComb_{(C_r)})$]

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

At a final stage, and using the results generated from the previous step, which is the input of Algorithm 8, Sp-RST applies first the $min()$ operator to look for the minimum number of features among all the $Size_{(AllComb_{(C_r)})}$; 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 Sect. 3: $\gamma(AllComb_{(C_r)}) = \gamma(C_r)$ while there is no $AllComb'_{(C_r)} \subset AllComb_{(C_r)}$ such that $\gamma(AllComb'_{(C_r)}) = \gamma(AllComb_{(C_r)})$. Every combination that satisfies this constraint is evaluated as a possible minimum reduct set. The features defining the reduct set describe all concepts in the initial $T_{RDD_i}(C_r)$ training data set.

Algorithm 8 Filter $NbF_{min}(DEP_{max}(AllComb_{(C_r)}))$

Input: Filtered-RDD[$AllComb_{(C_r)}$, $Size_{(AllComb_{(C_r)})}$, $\gamma(AllComb_{(C_r)})$]

Output: Reduct

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

5.3 Sp-RST: a working example

We apply Sp-RST 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 $U = \{x_1, x_2, \ldots, x_5\}$ which refers to the set of data instances (items), a large conditional feature set C = {Headache, Muscle-pain, Temperature} that includes all the features of the $T_{RDD}(C)$ information table and finally via a decision feature Flu of the given learning problem. Flu refers to the label (or class) of each $T_{RDD}(C)$ data item and is defined as follows: $Flu = \{yes, no\}$. C presents the conditional attribute pool from where the most significant attributes will be selected.

Table 1 Toy data set

Patient ID	Headache	Muscle-pain	Temperature	Flu
x_0	Yes	Yes	Very-high	Yes
x_1	Yes	No	High	Yes
x_2	Yes	No	High	No
x_3	No	Yes	Normal	No
x_4	No	Yes	High	Yes
x_5	No	Yes	Very-high	Yes

Table 2 Toy data set—split 1

Patient ID	Headache	Muscle-pain	Temperature	Flu
x_0	Yes	Yes	Very-high	Yes
x_1	Yes	No	High	Yes
x_2	Yes	No	High	No

Table 3 Toy data set—split 2

Patient ID	Headache	Muscle-pain	Temperature	Flu
x_3	No	Yes	Normal	No
x_4	No	Yes	High	Yes
x_5	No	Yes	Very-high	Yes

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

- *From Split 1:*
  - $\langle$yes, $x_0\rangle$
  - $\langle$yes, $x_1\rangle$
  - $\langle$no, $x_2\rangle$
- *From Split 2:*
  - $\langle$no, $x_3\rangle$
  - $\langle$yes, $x_4\rangle$
  - $\langle$yes, $x_5\rangle$

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

- $yes, \{x_0, x_1, x_4, x_5\}$
- $no, \{x_2, x_3\}$

In this example, we assume that we have two partitions $m = 2$. For the first partition, $m = 1$, a random number $r = 2$ is selected to build the first $T_{RDD_{i=1}}(C_r)$. For the second

Table 4 Partition $m = 1$—split 1

Patient ID	Muscle-pain	Temperature	Flu
x_0	Yes	Very-high	Yes
x_1	No	High	Yes
x_2	No	High	No

Table 5 Partition $m = 1$—split 2

Patient ID	Muscle-pain	Temperature	Flu
x_3	Yes	Normal	No
x_4	Yes	High	Yes
x_5	Yes	Very-high	Yes

Table 6 Partition $m = 2$—split 1

Patient ID	Headache	Flu
x_0	Yes	Yes
x_1	Yes	Yes
x_2	Yes	No

Table 7 Partition $m = 2$—split 2

Patient ID	Headache	Flu
x_3	No	No
x_4	No	Yes
x_5	No	Yes

partition, $m = 2$, a random number $r = 1$ is selected to build the first $T_{RDD_{i=2}}(C_r)$. Based on these assumptions, the following partitions and splits based on Apache Spark are obtained (Tables 4, 5, 6, 7).

Based on the first partition $m = 1$, and by applying Algorithm 3, which aims to generate all the $AllComb_{(C_r)}$ possible combinations of the C_r set of attributes, the output from both Apache Spark splits is the following:

- Muscle-pain
- Temperature
- Muscle-pain, temperature

In its third distributed job, Sp-RST calculates the indiscernibility relation $IND(AllComb_{(C_r)})$ for every created combination, i.e., the indiscernibility relation of every element in the output of the previous step (Algorithm 3). By applying Algorithm 4 and based on both Apache Spark splits, the output is the following:

- *From* $m = 1$—*Split 1:*
  - Muscle-pain, $\{x_0\}, \{x_1, x_2\}$
  - Temperature, $\{x_0\}, \{x_1, x_2\}$
  - Muscle-pain, Temperature, $\{x_0\}, \{x_1, x_2\}$
- *From* $m = 1$—*Split 2:*
  - Muscle-pain, $\{x_3, x_4, x_5\}$

– Temperature, $\{x_3\}$, $\{x_4\}$, $\{x_5\}$
– Muscle-pain, Temperature, $\{x_3\}$, $\{x_4\}$, $\{x_5\}$

In a next stage, and by using the previous output as well as $IND(Flu)$, Sp-RST computes the dependency degrees $\gamma(AllComb_{(C_r)})$ of each attribute combination as described in Algorithm 5. This distributed job generates two outputs namely the set of dependency degrees $\gamma(AllComb_{(C_r)})$ of the attribute combinations $AllComb_{(C_r)}$ as well as their associated sizes $Size_{(AllComb_{(C_r)})}$. The output from both splits for $m = 1$ is the following:

– Muscle-pain, 1, 1
– Temperature, 4, 1
– Muscle-pain, temperature, 4, 2

Once all the dependencies are calculated, in Algorithm 6, Sp-RST looks for the maximum value of the dependency among all the computed $\gamma(AllComb_{(C_r)})$. The maximum dependency reflects the baseline value for the feature selection task. The output is the following:

– 4

In a next step, Sp-RST 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 ($MaxDependency = 4$), i.e., $\gamma(AllComb_{(C_r)}) = MaxDependency = 4$. By applying Algorithm 7, the following output is obtained:

– Temperature, 4, 1
– Headache, Temperature, 4, 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, and using the results generated from the previous step and by applying Algorithm 8, Sp-RST looks for the minimum number of features among all the $Size_{(AllComb_{(C_r)})}$. 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 $T_{RDD_i}(C_r)$ training data set. The output of Algorithm 8 and which presents the *Reduct* for $m = 1$ is the following:

– Temperature

Based on these calculations, for $m = 1$, Sp-RST reduced the $TDD_{i=1}(C_{r=2})$ to $Reduct_{m=1} = \{Temperature\}$.

The same calculations will be applied to $m = 2$, and the output is $Reduct_{m=2} = \{Headache\}$ (as the data is composed of a single feature).

At this stage, different reducts are generated from the different m partitions. With respect to Algorithm 1, lines 12–14, 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 $Reduct = \{Headache, Temperature\}$.

In this example, we presented a single iteration of Sp-RST, i.e., $N = 1$. Therefore, line 16 on Algorithm 1 will not be covered in this example.

Sp-RST could reduce the big information table presented in Table 1 from $T_{RDD}(C)$ to $T_{RDD}(Reduct)$. The output is presented in Table 8.

Table 8 Reduct

Patient ID	Headache	Temperature	Flu
x_0	Yes	Very-high	Yes
x_1	Yes	High	Yes
x_2	Yes	High	No
x_3	No	Normal	No
x_4	No	High	Yes
x_5	No	Very-high	Yes

6 Experimental setup

6.1 Benchmark

To validate the effectiveness of Sp-RST, we require a data set with a large number of attributes that is also defined by a large number of data instances. The Amazon Commerce reviews data set from the UCI machine learning repository [4] fulfills this requirement. The Amazon data set was initially build from several customer reviews on the Amazon commerce Web site. The base was constructed based on the identification of the most active users, aiming at performing authorship identification. The database enclosed a total of 1500 data instances which are described using 10,000 features (linguistic style such punctuation, length of words, sentences, etc.) and 50 distinct classes (referring to authors). The Amazon data items are identically distributed across the data set classes, i.e., for each class there are 30 items.

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 1000, 2000, 4000, 6000, and 8000 out of the original 10,000 attributes. We use these sets to evaluate our proposed method as discussed in Sect. 6.2 and refer to them as Amazon1000, Amazon2000, …, Amazon10,000 in the following.

6.2 Evaluation metrics

To evaluate the scalability of the parallel Sp-RST, we consider the standard metrics which are the *speedup*, the *scaleup*, and the *sizeup* from literature [42]. 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 [42] 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 Sect. 6.1. We use 1000 features as a baseline and consider 2000, 4000, 6000, 8000, 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 1000 features as a baseline. Here, a scaleup of 1 implies 'linear' scaleup, which similarly to linear speedup is difficult to achieve.

As previously highlighted in Sect. 2, a preliminary version of our proposed solution was introduced in [8] as an attempt to deal with feature selection in the big data context. Yet, let us recall that in [8], both of the sizeup and scaleup were measured based on the number of features per partition, and hence, they are based on a modified definition of the standard metrics which are detailed above. However, we think that using the overall number of attributes is a much more natural setup as it will give insights into the performance depending on the input data set rather than the partitions, i.e., the proper definitions of the metrics are adopted in this paper.

To demonstrate that our distributed Sp-RST solution performs its feature selection task well without sacrificing performance, we perform model evaluation using a Naive Bayes and a random forest classifier. For the evaluation, we use the standard measures which are the precision, the recall, the accuracy and F1 score as well as the runtime (measured in seconds), to compare the quality of the feature set selected by Sp-RST with other feature selection methods as described in Sect. 6.3. The metrics definitions 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:

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

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

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

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

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

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

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

We remark that Sp-RST is a stochastic algorithm. For Sp-RST, the process of randomization is applied twice. Once when partitioning the data set into m data blocks and second during the selection of one reduct among the generated set (family) of reducts. To diminish

the effect of the first randomization process, we perform several iterations of the main part of the algorithm (Algorithm 1) and we only keep the set of attributes that are selected in all iterations. The second randomization process is already justified and supported by the fundamentals of the theory of rough sets as presented in Sect. 3. We, therefore, conduct a deep analysis of the stability of the attribute sets which were selected by performing several runs of Algorithm 1 and then report averages and standard deviations whenever appropriate.

To investigate the significance of any noticed variation in the classification performance when random forest and Naive Bayes are applied to the initial data set and to the reduced set generated by Sp-RST and other feature selection techniques, we perform Wilcoxon signed rank tests with Bonferroni correction.

6.3 Experimental environment

In the following, we conduct a detailed study of various parameters of Sp-RST with the aim to analyze how these can affect the system's runtime as well as the stability of the attribute selection task. We then apply a Naive Bayes and a random forest classifier on the original data set and the reduced data sets produced by Sp-RST and other feature selection techniques. We use the scikit-learn random forest implementation[13] with the following parameters: n_estimators $= 1000$, n_jobs $= -1$, and oob_score $=$ True. A Stratified 10-Folds cross-validator[14] is used for all our conducted experiments. Moreover, we use the Naive Bayes implementation from Weka 3.8.2.[15]

The Sp-RST algorithm is implemented in Scala 2.11 within the Spark 2.1.1 framework. Our experiments for Sp-RST are performed on Grid5000,[16] 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 two classifiers, these experiments are run on a standard laptop configuration with Intel(R) Core(TM) i7-7500U CPU, 16 GB RAM, 64-bit, Windows-10.

Preliminary results revealed that a maximum of 10 features per partition is the limit that can be processed by Sp-RST. We therefore perform experiments using 4, 5, 8, and 10 features per partition in Algorithm 1. We run all settings on 1, 2, 4, 8, 16, and 32 nodes on Grid5000. When considering scalability, we set the number of iterations in Algorithm 1 to 10 (based on preliminary experiments). However, we perform an additional analysis of the feature selection process across different iterations in Sect. 7.1.

To ensure a fair comparison, we restrict our comparison with other feature selection methods to filter techniques. 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 Sp-RST, i.e., the average number of features selected for each parameter setting considered and additionally use 0 as a threshold. We determine the sets of features selected by these methods and then perform model evaluation with a Naive Bayes and a random forest classifier as discussed previously. Subset selection:

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

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

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

[16] http://www.grid5000.fr.

– *CfsSubsetEval* considers the individual predictive ability of each feature along with the degree of redundancy between them
– *ConsistencySubsetEval* considers the level of consistency in the class values when the training instances are projected onto the subset of attributes

Attribute selection:

– *Sum squares ratio*, which measures the ratio of between-groups to within-groups sum of squares.
– *Chi squared*, which computes the value of the chi-squared statistic with respect to the class.
– *Gain ratio*, which measures the gain ratio with respect to the class.
– *Information gain*, which measures the information gain with respect to the class.
– *Correlation*, which measures the correlation (Pearson's) between it and the class.
– *CV*, which 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.
– *ReliefF*, which repeatedly samples an instance and considers the value of the given attribute for the nearest instance of the same and different class.
– *Significance*, which computes the probabilistic significance as a two-way function (attribute-classes and classes-attribute association).
– *Symmetrical uncertainty*, which evaluates the symmetrical uncertainty with respect to the class.

For sum squares ratio, we have used the version implemented in Smile;[17] while for the other three techniques, we have used the implementation provided in Weka 3.8.2.[18]

7 Discussion of results

We first examine the feature selection process over several iterations, which is a crucial parameter of Sp-RST (Sect. 7.1). Afterwards, we analyze the stability (Sect. 7.2) and scalability (Sect. 7.3) of our proposed feature selection approach. Finally, we compare its performance with other state-of-the-art feature selection techniques (Sect. 7.4).

7.1 Number of iterations in Sp-RST

We first have a closer look at one of the parameters of Sp-RST, i.e., the number of iterations (N in Algorithm 1). We perform four independent runs (or repetitions) of Sp-RST with 1, 2, …, 20 iterations, and for each iteration record the elected features as well as the elapsed time. We plot the average and standard deviation of the number of remaining features after each iteration over these four runs and for different parameter settings in Fig. 4.

We also perform runs of Sp-RST with 1, 2, …, 20 iterations on different numbers of nodes, i.e., 1, 4, 8, and 16 nodes. The corresponding runtimes split by the number of the nodes are shown in Fig. 5. Here, the average and standard deviation are taken across all runs with the same number of iterations.

From Fig. 4, we observe that independently of the number of iterations, there is a clear ordering with respect to the number of selected features: The smaller the number of features

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

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

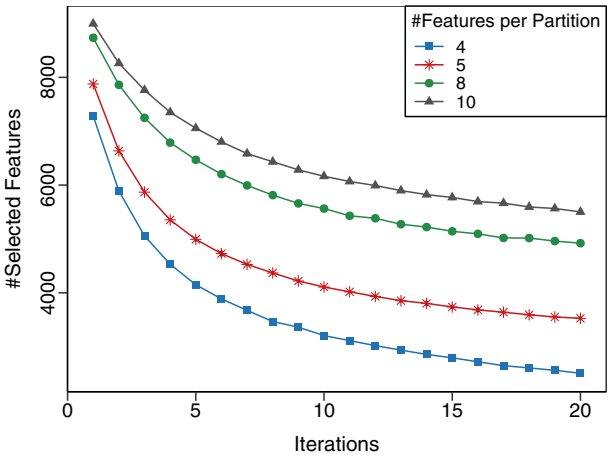


Fig. 4 Number of features selected depending on the number of iterations executed in a run of Sp-RST (average and standard deviation over 4 runs)

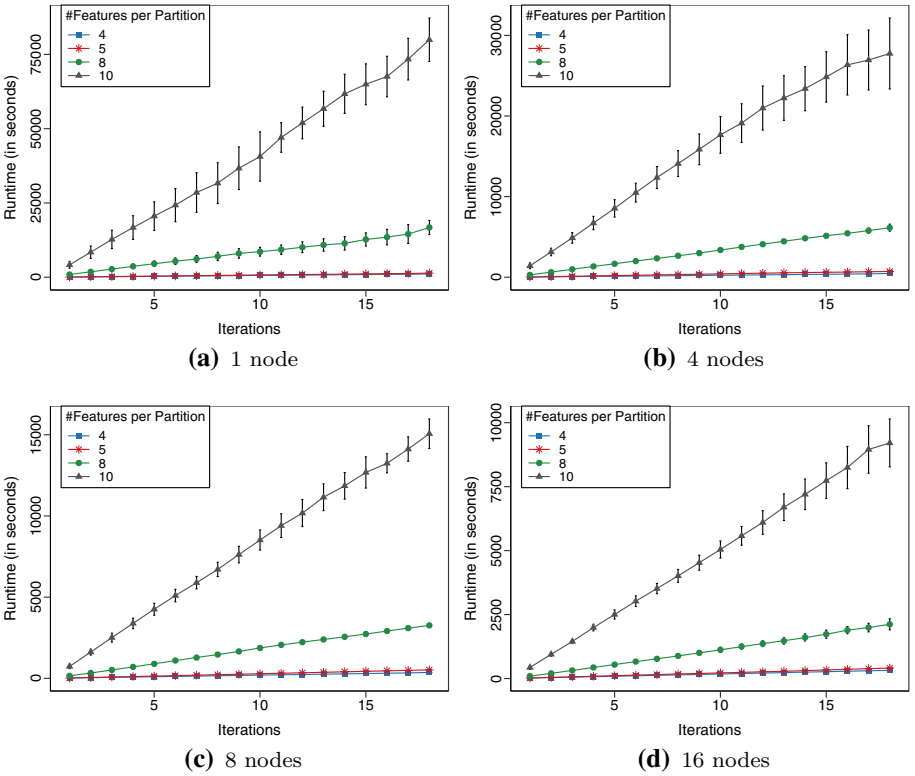


Fig. 5 Runtime of Sp-RST depending on the number of iterations

per partition, the fewer features are selected by Sp-RST. The very small standard deviation (< 40) is hardly visible in the graphs and clearly demonstrates the stability of Sp-RST with respect to the number of features selected in an iteration. Recall that Sp-RST returns the intersection of the reducts from all iterations performed. Thus, the number of selected features strictly decreases with the number of iterations.

As discussed before, the runtime for the rough set component of our methods grows exponentially with respect to the number of features per partition. Thus, the runtime behavior in Fig. 5 is not surprising and clearly demonstrates that the number of features per partition should not grow too large. We also see that the runtime per iteration is quite stable so that the overall runtime grows linearly with the number of iterations. Based on these experiments, we have decided to use a medium number of iterations for the remainder of our analysis, namely 10.

7.2 Stability of feature selection

To validate the stability of the feature selection of Sp-RST, we have a closer look at the concrete features selected. We perform two sets of experiments. First, we look at the features selected by Sp-RST with a single iteration ($N = 1$). Second, we consider our standard parameter setting of 10 iterations ($N = 10$) and perform several independent runs of Algorithm 1. We particularly look at two extreme cases: the number of features that are always selected and the number of features that are never selected over a given number of these runs or iterations.

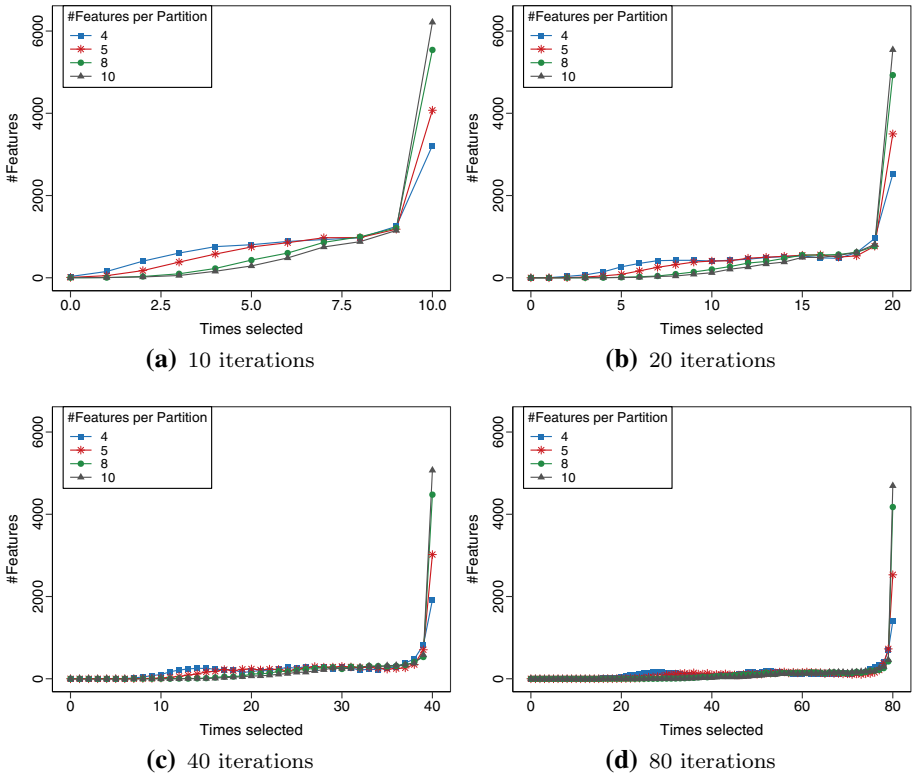
7.2.1 Iterations

We first consider features selected during a single iteration ($N = 1$). We run 10, 20, 40, and 80 independent runs of Sp-RST with 1 iteration and for each run record the set of selected features. For each feature, we count the number of times it has been selected and depict the results in Fig. 6. Table 9 additionally shows the mean, standard deviation (SD), min and max of the number of features selected in a single iteration (over 80 runs performed independently).

From Fig. 6 and Table 9, we see that the number of features selected in a single iteration is very stable. However, depending on the number of features per partition, only about 45% (4, 5 features), 55% (8 features), and 70% (10 features) of these features are selected in all 10 iterations and already for 20 iterations all features will have been selected at least once. The latter also holds for 40 and 80 iterations. This demonstrates that the set of 'core' features (defined is Sect. 3.2) that are reliably selected is much smaller than the set of features selected in a single iteration. This observation is the main motivation for using several iterations and only returning features that are always selected as the result of Sp-RST.

7.2.2 Complete algorithm

To confirm that using the intersection of several iterations improves the stability of the feature selection, we now consider the complete algorithm with 10 iterations. We run 6 independent repetitions of Sp-RST with 10 iterations and plot the number of features selected 0, 1, …, 6 times in these 6 runs in Fig. 7.



(a) 10 iterations **(b)** 20 iterations

(c) 40 iterations **(d)** 80 iterations

Fig. 6 Number of features selected i times in i iterations

Table 9 Number of features selected in a single iteration

#features per partition	4	5	8	10
Mean (80 iterations)	7278.15	7875.2	8731.725	8994.25
SD (80 iterations)	13.5629	12.5329	5.1484	2.7212
Min (80 iterations)	7247	7841	8721	8985
max (80 iterations)	7313	7909	8744	9000
Always (10 iterations)	3216	4074	5541	6214
Never (10 iterations)	31	9	2	0
Always (20 iterations)	2521	3498	4928	5545
Never (20 iterations)	0	0	0	0
Always (40 iterations)	1914	3020	4477	5070
Never (40 iterations)	0	0	0	0
Always (80 iterations)	1396	2528	4176	4693
Never (80 iterations)	0	0	0	0

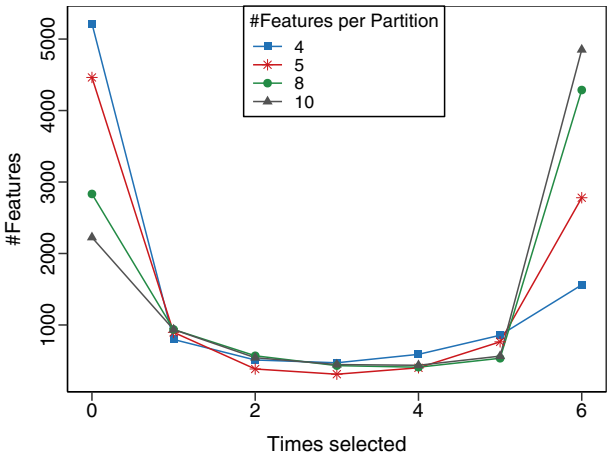


Fig. 7 Number of features selected 0, 1, …, 6 times in six independent runs of Sp-RST

Table 10 Number of features selected in a run of Sp-RST

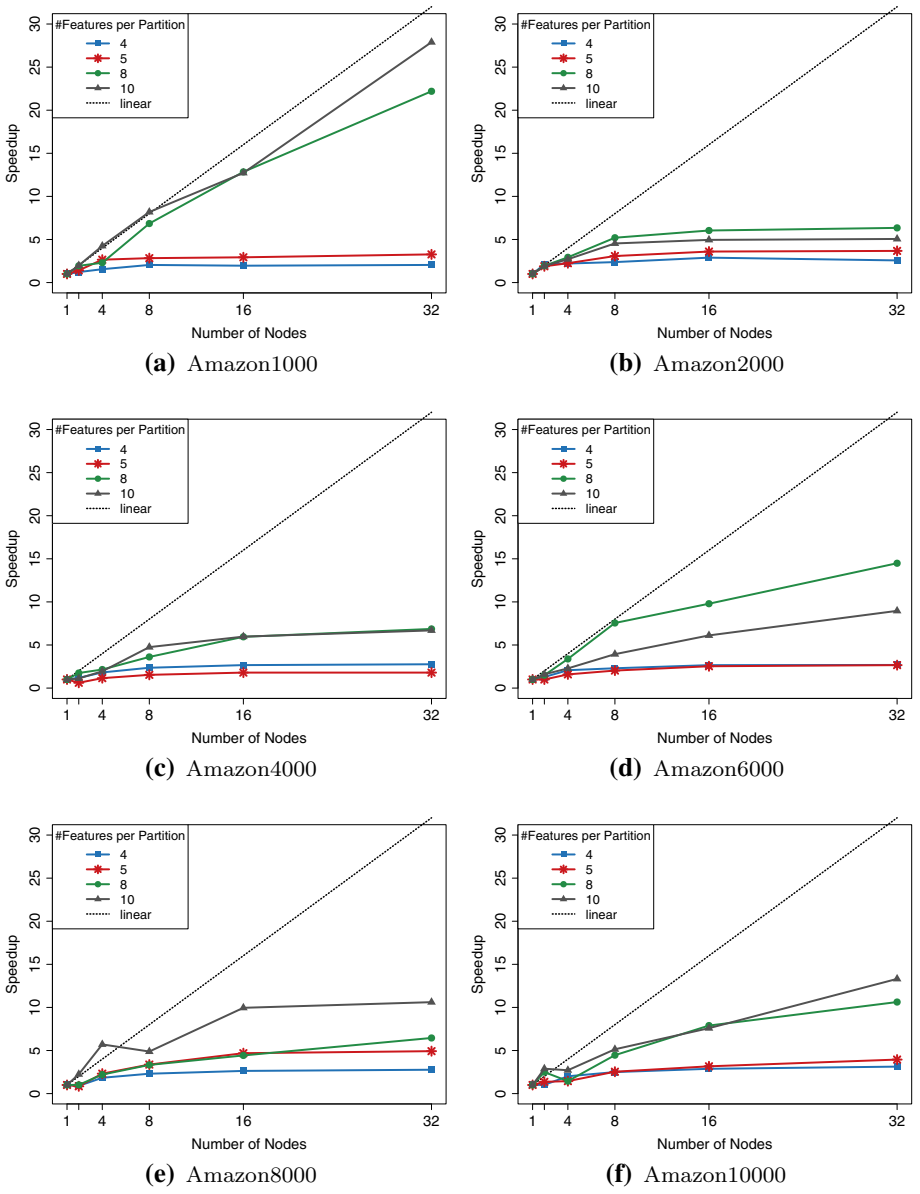
#features per partition	4	5	8	10
Mean	3205.333	4117	5566	6171.333
SD	24.3530	18.9526	23.8914	26.0128
Min	3178	4093	5542	6131
Max	3245	4138	5598	6197
Always	1561	2780	4287	4851
Never	5216	4463	2833	2225

From Fig. 7, it is obvious that most features are either always or never selected. This demonstrates that Sp-RST reliably selects the same features, or in other words is able to identify the most relevant (always selected) and least relevant (never selected) features. As before, we provide the number of features selected in a run of Sp-RST in Table 10 and observe that the number of features selected is again very stable.

We remark that it is not a weakness of our proposed method that the actual features selected from one run to another differ. As discussed earlier in Sect. 3.2, there can be more than one reduct, a family of reducts, and selecting an arbitrary one among these is appropriate. Only core features appear in all the generated reducts set. The results presented in Fig. 7 support that our method is able to identify core features as well as features that are not in any of the resulting reducts.

7.3 Scalability

We measure the scalability of Sp-RST based on its speedup (Fig. 8), sizeup (Fig. 10) and scaleup (Fig. 12) as discussed in Sect. 6.2 and additionally plot the measured runtimes by the number of nodes and the different data sets (Figs. 9, 11). We see that the runtime increases considerably with the number of features per partition and decreases with the number of nodes used (Fig. 9). However, in the latter case, increasing the number of nodes from 1 to 2



(a) Amazon1000 **(b)** Amazon2000

(c) Amazon4000 **(d)** Amazon6000

(e) Amazon8000 **(f)** Amazon10000

Fig. 8 Speedup for the six data sets discussed in Sect. 6.1

or 4 has a much larger effect than increasing it further (Fig. 9). Moreover, it increases with the size of the database (Fig. 11), where size is measured in terms of the number of features.

In terms of speedup (Fig. 8), we see that the speedup for our smallest data set (Amazon1000) with 8 or 10 features per partition is approximately linear. However, in general and in particular for larger databases, the speedup is sub-linear. It is better for settings with more features per partition (with 8 and 10 features always being the best parameter settings).

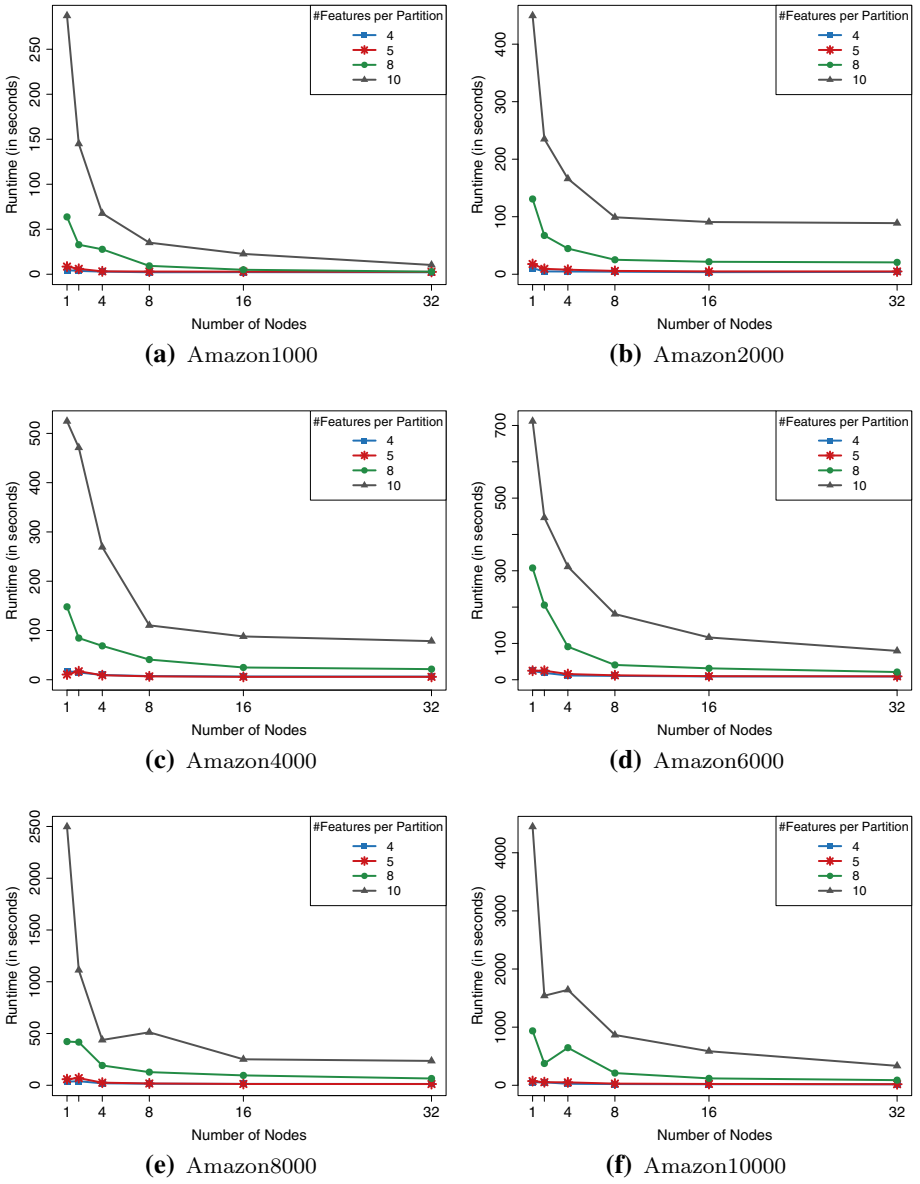


Fig. 9 Runtimes (in seconds) for the six data sets discussed in Sect. 6.1

This implies that having more partitions is not beneficial with respect to the parallel running time—even though the runtime for a single partition grows exponentially in the number of its features. This indicates that settings with 4 and 5 features per partition generate too small sub-databases that generate unnecessary large overhead due to high communication cost.

Figure 10 shows that Sp-RST has a very good sizeup performance; however, the more nodes are used, the more important the parameterization becomes. We see that 4 or 5 features per partition yield generally better sizeup than 8 or 10 features. While 4, 5 as well as 8 and 10 features show very similar sizeup performance. For the latter, the sizeup deteriorates for $m \geq 6$. This is in stark contrast to the speedup results discussed earlier where we have observed better speedups for more features per partition. Thus, depending on the size of the database in terms of features and the number of nodes available different parameter settings will be more appropriate.

Figure 12 shows the scaleup for 1, 2, 4, and 8 nodes and the corresponding data sets Amazon1000, Amazon2000, Amazon4000, and Amazon8000. For up to 4 nodes, the scaleup is close to 1 for most parameter settings; however, it drops below 1 for 8 nodes. Therefore, we can conclude that using a very large number of nodes does not necessarily yield much better runtimes while we obtain large improvements with moderate parallelization. This is in line with the speedup results in Fig. 8 where the speedup was close to linear for most parameter settings for up to 4 or 8 nodes and only deteriorates quickly with more nodes.

7.4 Comparison with other feature selection techniques

To demonstrate that our method is suitable with respect to classification, i.e., to demonstrate that Sp-RST performs its feature selection task well without sacrificing performance, we perform model evaluation and investigate the influence of Sp-RST on the classification of a Naive Bayes (Sect. 7.4.1) and a random forest (Sect. 7.4.2) classifier and compare the results with the original data set with 10,000 features and other feature selection techniques (see Sect. 6.3).

To account for the stochasticity of Sp-RST, we obtain six different feature sets for each parameter setting in independent runs of Sp-RST. For all other methods, we obtain a unique data set. We run the two classifiers on all these data sets and combine results for data sets based on the same parameter setting in Sp-RST. We present average and standard deviation of 10 independent runs for each considered data set and visualize the results as boxplots.

7.4.1 Naive Bayes

From Fig. 13 and Table 11 in "Appendix A", we can see that the classification results strictly improve when the number of features per partition in Sp-RST is increased. Moreover, they only slightly worsen in comparison with the original data set. For example, the accuracy for 4 features is 0.5634, for 10 features 0.5934 and for the original data set 0.6099.

Results differ widely depending on parameterisation for the other feature selection techniques as shown in Figs. 14 and 15 and Table 11 ("Appendix A"). This demonstrates that the other techniques tested are much more sensitive to suitable parameter settings than Sp-RST. As noted before, parameterisation can be a very difficult task to perform, thus demonstrating a clear strength of our method.

Recall that the parameter to determine the number of features to be selected was set in a way that mirrors the number of features selected by Sp-RST, namely the average number of features selected in the four different parameter settings (see Table 10). Additionally, we used a standard threshold value of 0. Looking more closely into the accuracy results for the best parameter setting of Sp-RST, i.e., 10 features, we see that it outperforms all parameterisations of CV and Consistency, but is outperformed by all parameterisations of Cfs, Correlation, ReliefF and Sum Squares Ratio. For the remaining methods, only the parameterisation with

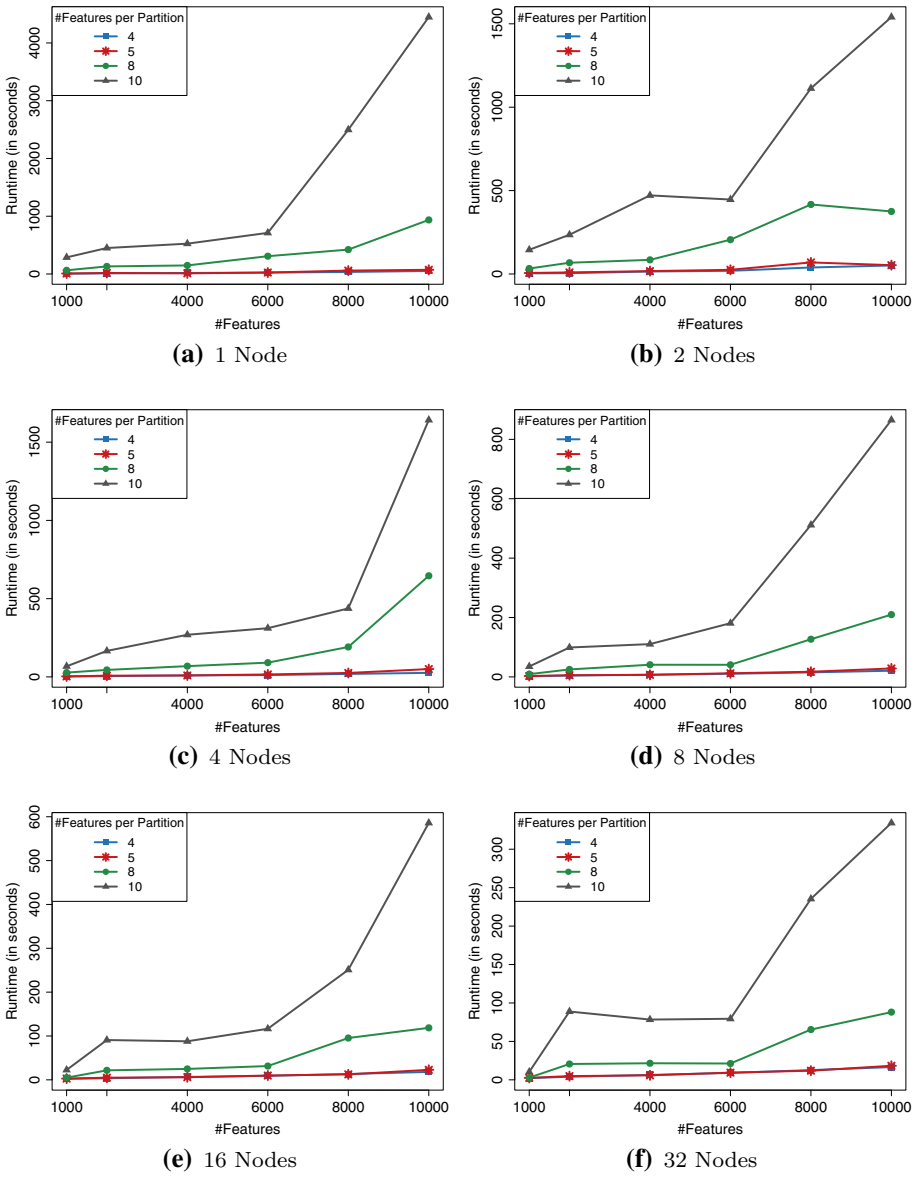


Fig. 10 Sizeup for different numbers of nodes

the largest number of features (6171) outperforms Sp-RST (with the exception of significance which also obtains good results for a threshold of 0). It should be noted that the overall best accuracy is obtained by Sum Squares Ratio with 4117 features (0.6556) while the worst accuracy is obtained CV with 3205 features (0.4877). Recall that the accuracy of Sp-RST ranges from 0.5634 to 0.5934.

To further validate these conclusions, pairwise Wilcoxon rank sum tests with Bonferroni correction were executed (see Tables 14, 15, 16, 17 in "Appendix B"). We observe that the

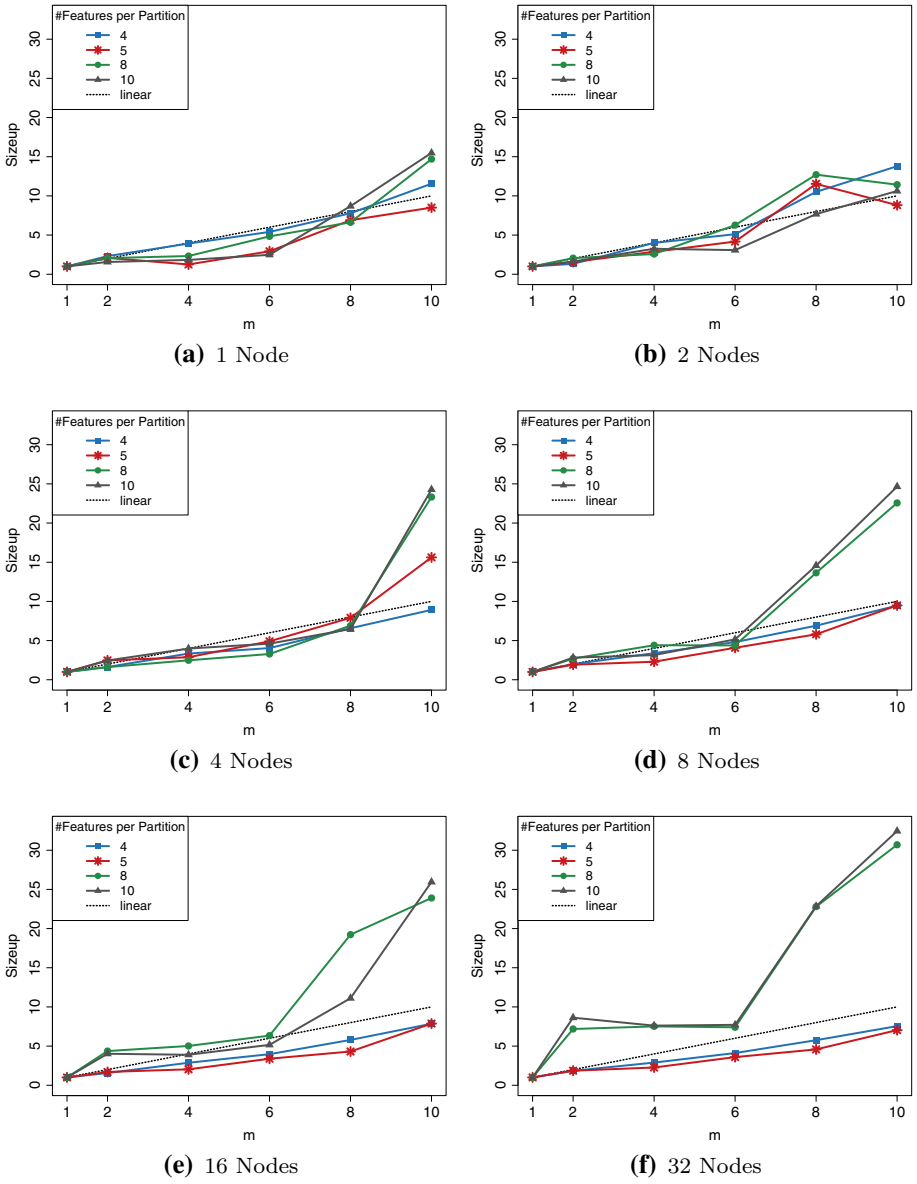


Fig. 11 Runtimes for different numbers of nodes

majority of observed differences are statistically significant at a confidence level of 0.05, but there are some notable exceptions where statistical significance could not be confirmed by the test. These include Sp-RST with 8 and 10 features Sp-RST with 10 features in comparison with Chi-Squared, Gain Ratio, Info Gain, Significance and Symmetrical Uncertainty with parameters 5566 and 6171. We remark that these are exactly the methods that only outperformed Sp-RST (10) for the parameter setting of 6171, but were worse for the other settings of the number of parameters to be selected.

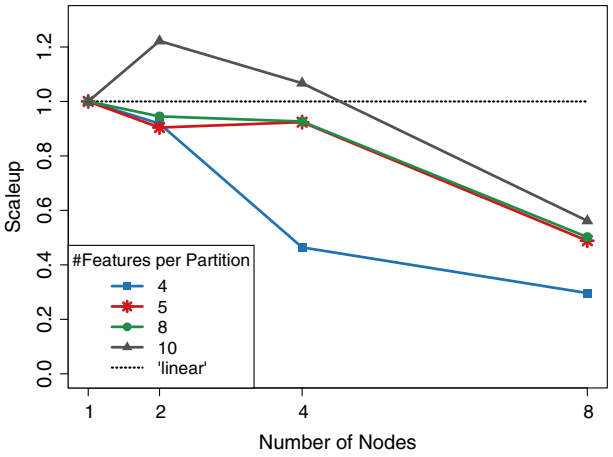


Fig. 12 Scaleup

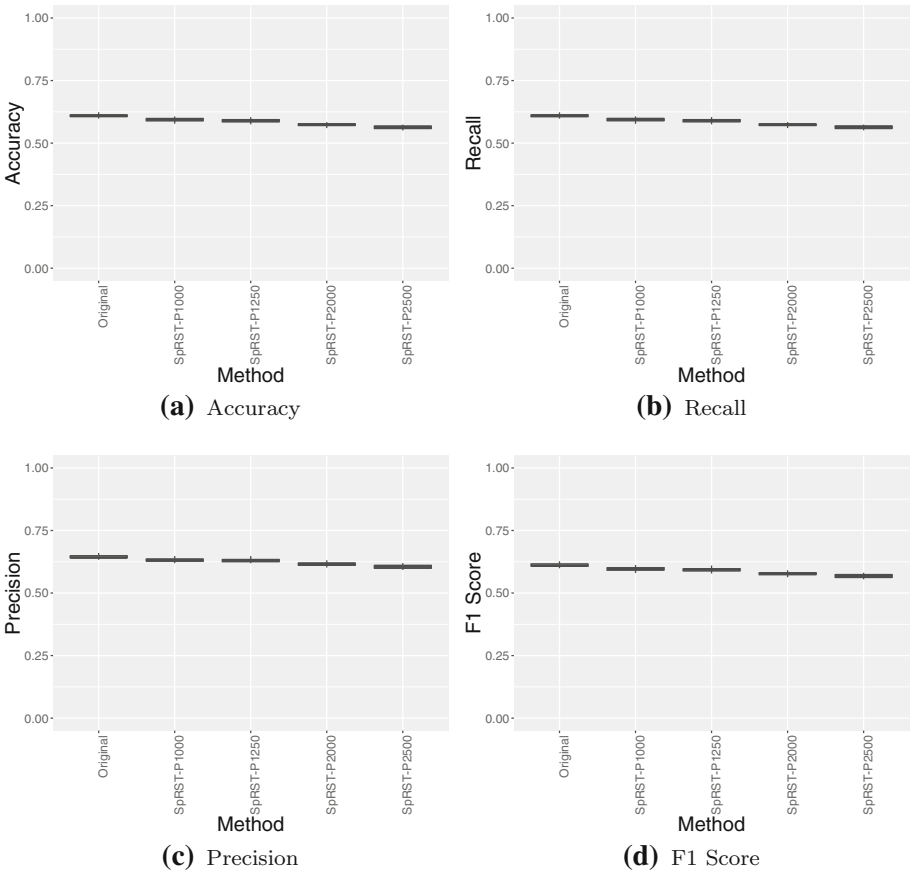
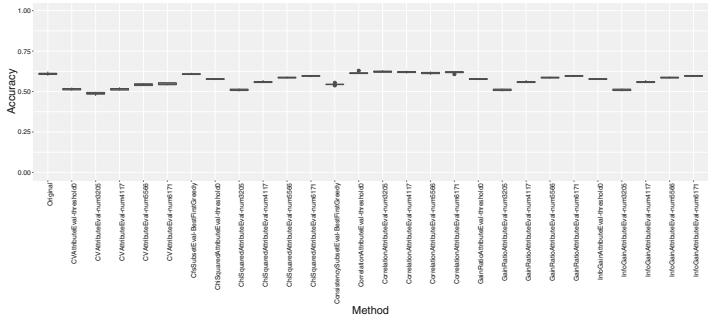
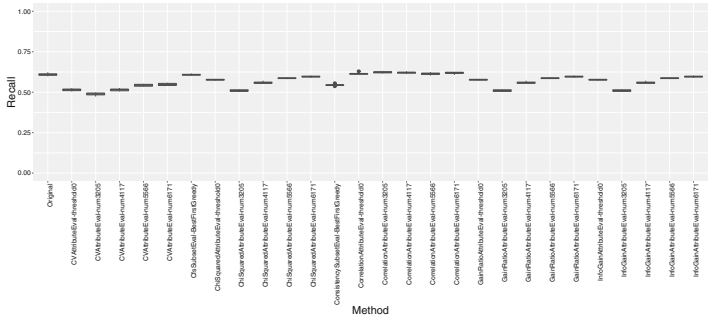


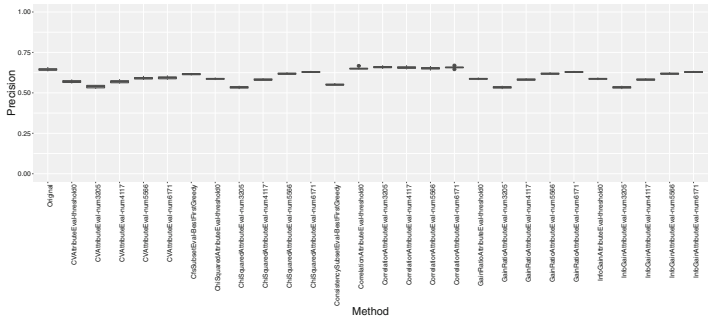
Fig. 13 Naive Bayes classification results for different feature selection techniques



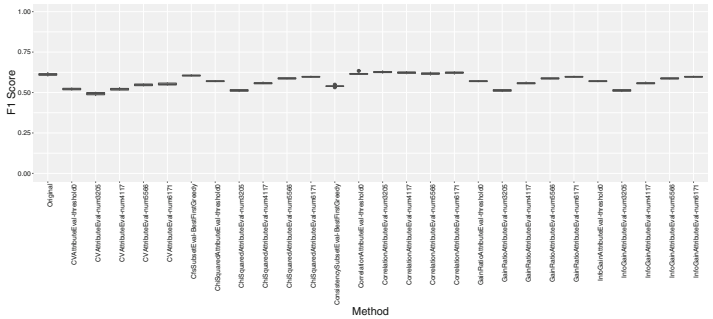
(a) Accuracy



(b) Recall

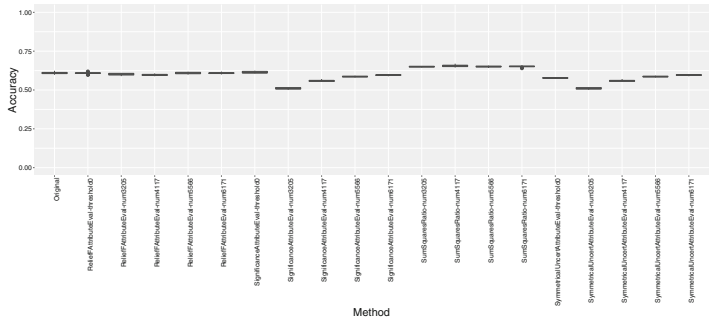


(c) Precision

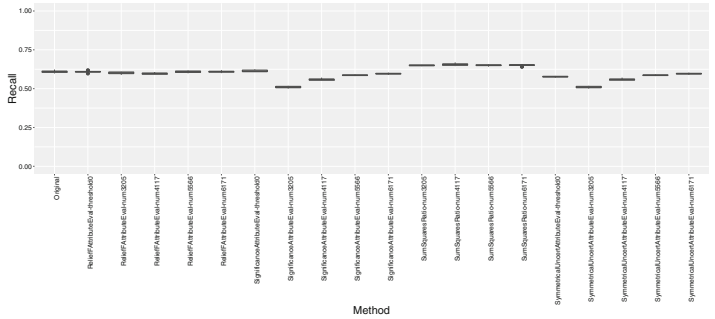


(d) F1 Score

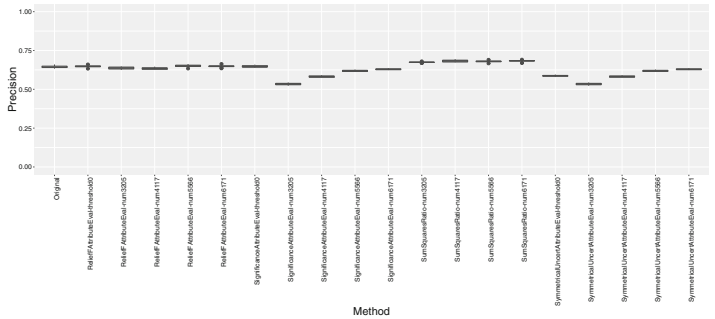
Fig. 14 Naive Bayes classification results for other feature selection techniques (Part 1)



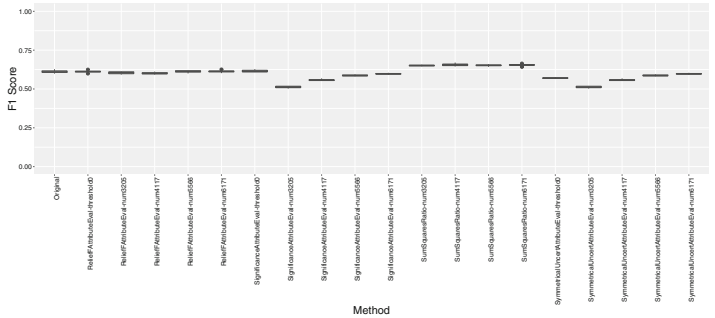
(a) Accuracy



(b) Recall



(c) Precision



(d) F1 Score

Fig. 15 Naive Bayes classification results for other feature selection techniques (Part 2)

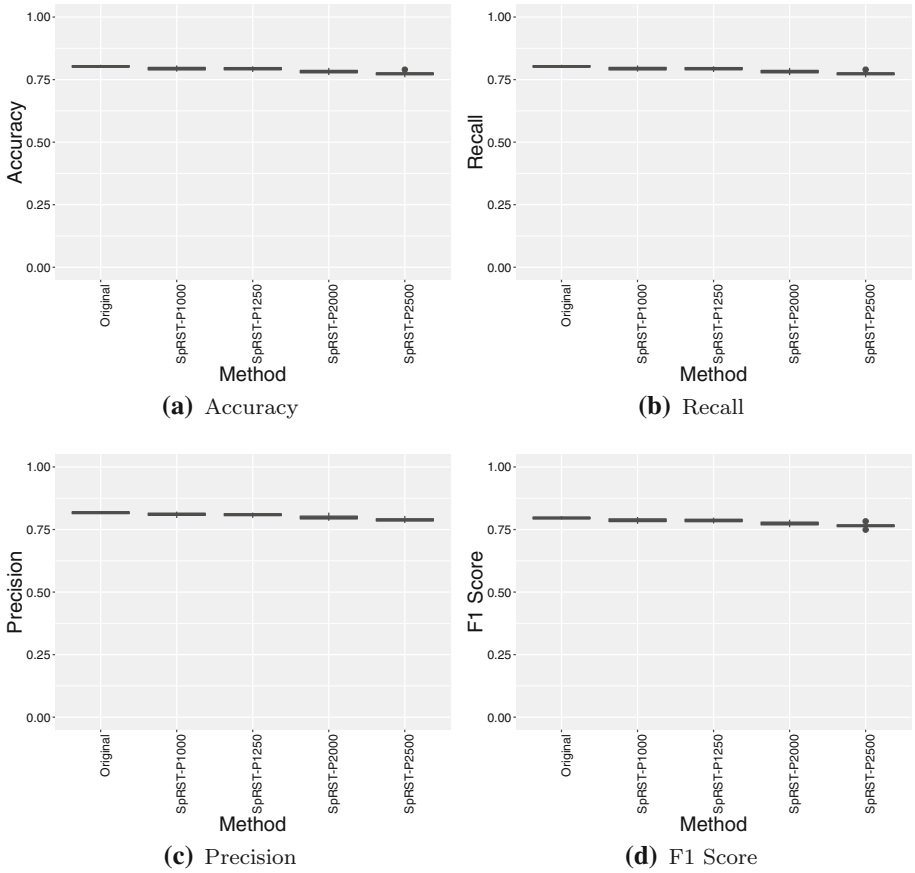
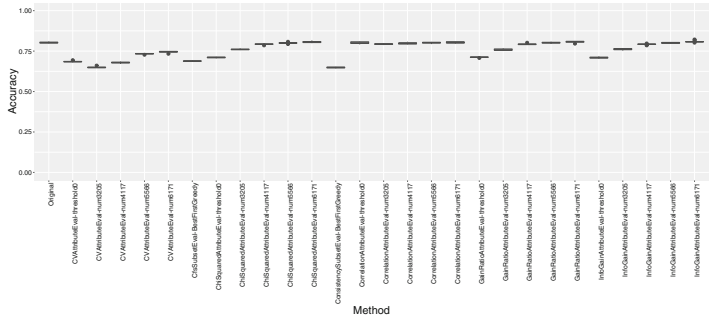


Fig. 16 Random forest classification results for different feature selection techniques

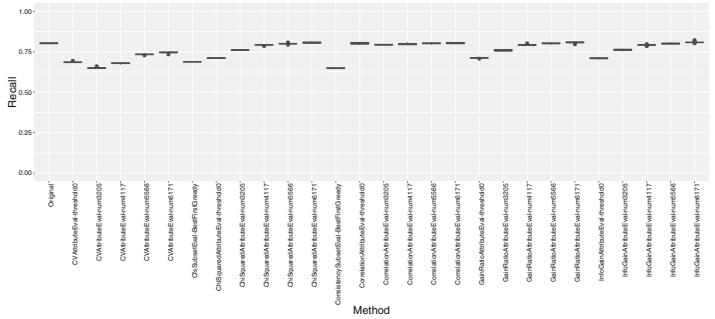
While we have only provided a detailed for the accuracy metric, identical observations can be made for the other three metrics (precision, recall, and F1).

7.4.2 Random forest

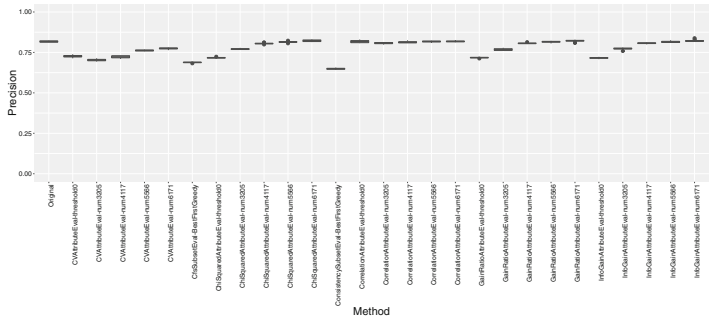
The overall classification performance of the random forest classifier is much better than the performance of the Naive Bayes classifier. To be more precise, the accuracy ranges from 0.8121 (Significance with 6171 features) to 0.6483 (Consistency). The accuracy of Sp-RST is between 0.7733 and 0.7938 and again strictly increasing with the number of features per partition, thus comparable to the best performing methods. Detailed results can be found in Figs. 16, 17 and 18 and Table 12 in "Appendix A". We see that the overall ranking of the methods tested is very similar to the Naive Bayes results and that Sp-RST outperforms/is outperformed by roughly the same methods and parameterisations. The main exception is that for methods where only parameter 6171 performed better in the case of Naive Bayes, now also parameter 5566 produces slightly better results. However, the overall differences are much smaller than in the case of the Naive Bayes classifier.



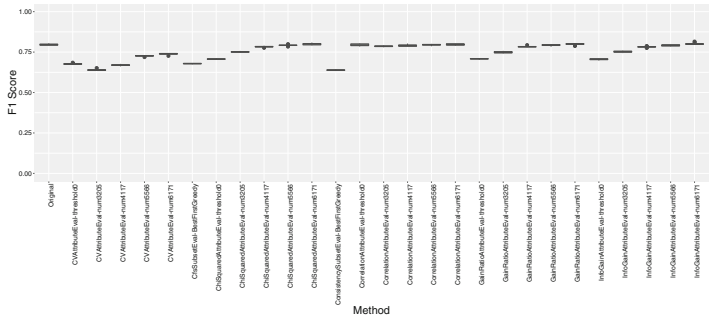
(a) Accuracy



(b) Recall

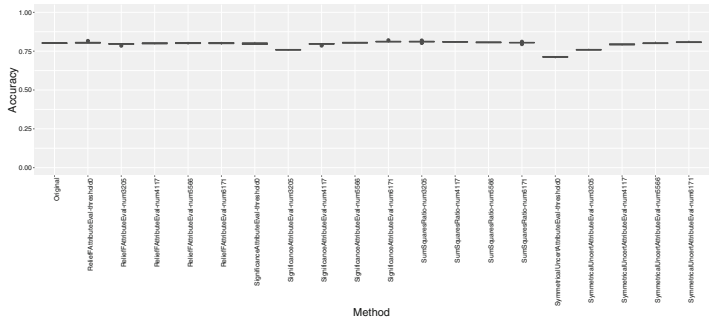


(c) Precision

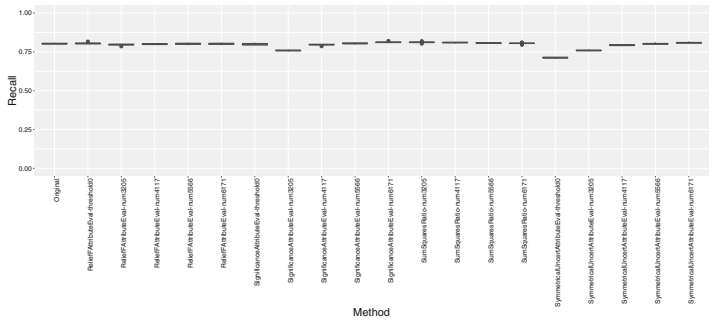


(d) F1 Score

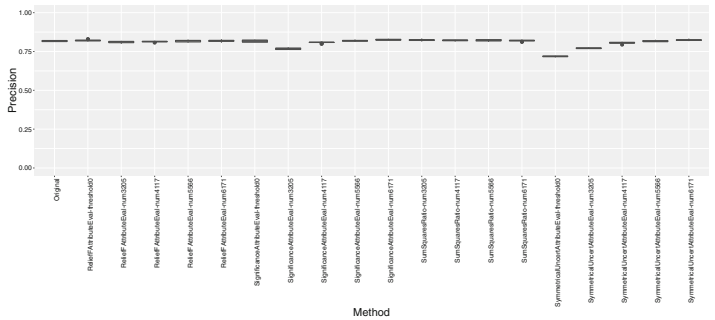
Fig. 17 Random forest classification results for other feature selection techniques (Part 1)



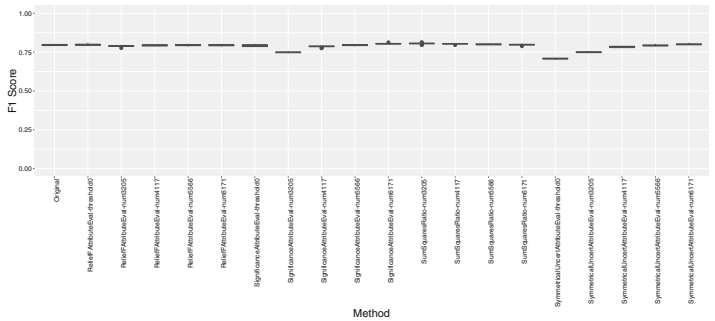
(a) Accuracy



(b) Recall



(c) Precision



(d) F1 Score

Fig. 18 Random forest classification results for other feature selection techniques (Part 2)

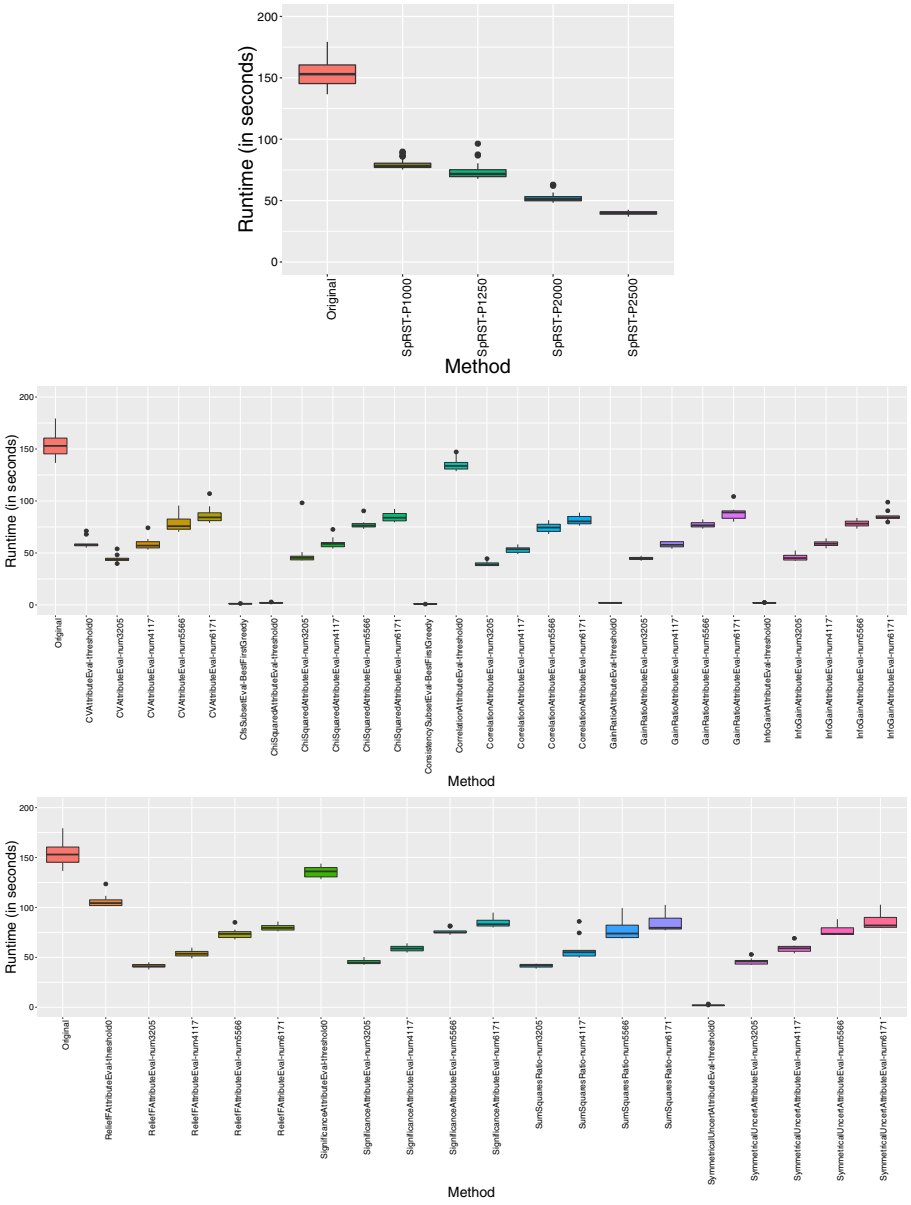


Fig. 19 Runtime of the Naive Bayes classifier for the results of different feature selection techniques

Again, we have executed pairwise Wilcoxon rank sum tests with Bonferroni correction to validate our results (see Tables 18, 19, 20, 21 in "Appendix B"). While most results for 4, 5 and 8 features per partition are statistically significant at confidence level 0.05, for quite a few comparisons with SP-RST (10) no statistical significance could be observed. This particularly holds for medium parameters such as 4117 and 5566 features, while in the case of the random forest classifier the parameter setting 6171 produces statistically different results in terms of accuracy. Recall that this was different for the Naive Bayes classifier.

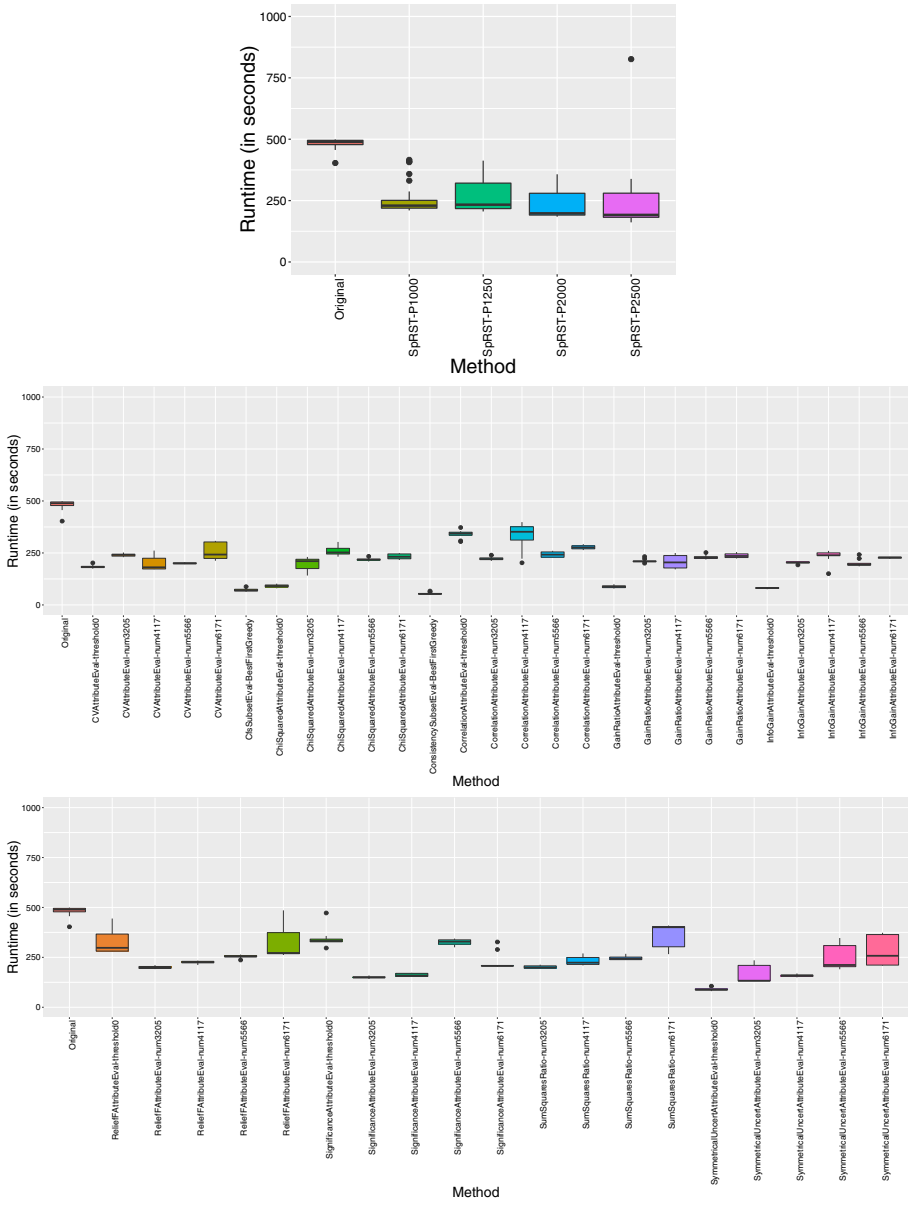


Fig. 20 Runtime of the random forest classifier for the results of different feature selection techniques

While we have only discussed the accuracy results in detail, identical observations can be made for the other three metrics shown in the graphs and tables.

7.4.3 Runtime comparison

We have provided a detailed analysis of the runtime for Sp-RST in Sect. 7.3. It should be noted that all other methods have negligible runtime for the feature selection part, and thus, there

is a clear trade-off between runtime and feature selection quality. Based on the comparable classification results, we argue that the additional runtime is worthwhile, particularly if the reduced feature set has the potential to be used repeatedly in different applications. We summarize the runtime for the classification in Table 13 and depict boxplots in Figs. 19 and 20.

Besides these results, it is important to recall that as proved in [6], these existent methods are inadequate to cope with a higher number of features as they will have scalability problems. Hence, based on our detailed analysis and based on the results obtained from [6], we can conclude that our proposed Sp-RST solution is a scalable and effective method for large-scale data pre-processing. Thus, it is relevant to big data.

8 Conclusion and future work

In this paper, we have introduced a parallelized filter feature selection technique based on rough set theory, called Sp-RST. We have presented a comprehensive experimental study to investigate parameter settings and to demonstrate the stability and scalability of our proposed method. Moreover, we have compared Sp-RST with other commonly used filter techniques. We have used model evaluation using a Naive Bayes and a random forest classifier to evaluate the quality of the feature set selected by Sp-RST and the other methods considered.

Our experiments show that the proposed method effectively performs feature selection in a founded way without sacrificing performance. In terms of scalability, using a moderate number of nodes yields a considerably improvement of the runtime and good speedup performance. However, the improvement quickly stagnates if 8 or more nodes are used. Thus, improving the speedup, sizeup and scaleup performance for more nodes is subject to future work.

Our results show that Sp-RST is competitive or better against other methods on the Amazon data set and only induces very small information loss: For the best Sp-RST parameter setting, the classification accuracy for Naive Bayes is 6.22% smaller than for the best comparator (Sum Squares), while for random forest it is only 1.83% (Signficance). In comparison with the original data set, we lose only 1.65% for Naive Bayes and 0.88% for random forest. Moreover, we have demonstrated that Sp-RST is able to reliably identify the most and least important features in the data set, which can be an important aspect when interpreting the feature selection results from an application perspective. Improving the overall classification ratio of Sp-RST is subject to future work. Particularly, we plan to investigate the performance and behavior of Sp-RST on other larger data sets to further our understanding of its working principles.

Appendix A: Detailed classification and runtime results

See Tables 11, 12 and 13.

Table 11 Naive Bayes classification results for different feature selection techniques

Method	#Features	Accuracy	Precision	Recall	F1 Score
Original	10,000	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 (3205)	3205	0.4877	0.5387	0.4878	0.4925
	(–)	(0.0090)	(0.0094)	(0.0089)	(0.0094)
CV (4117)	4117	0.5153	0.5701	0.5153	0.5213
	(–)	(0.0074)	(0.0093)	(0.0075)	(0.0074)
CV (5566)	5566	0.5431	0.5906	0.5432	0.5475
	(–)	(0.0069)	(0.0075)	(0.0070)	(0.0067)
CV (6171)	6171	0.5477	0.5933	0.5478	0.5522
	(–)	(0.0072)	(0.0082)	(0.0072)	(0.0072)
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 (3205)	3205	0.5103	0.5340	0.5102	0.5123
	(–)	(0.0057)	(0.0071)	(0.0058)	(0.0059)
Chi squared (4117)	4117	0.5595	0.5823	0.5596	0.5580
	(–)	(0.0057)	(0.0051)	(0.0058)	(0.0053)
Chi squared (5566)	5566	0.5863	0.6190	0.5865	0.5872
	(–)	(0.0038)	(0.0047)	(0.0037)	(0.0041)
Chi squared (6171)	6171	0.5965	0.6295	0.5966	0.5976
	(–)	(0.0042)	(0.0041)	(0.0042)	(0.0042)
Consistency (Greedy)	30	0.5443	0.5514	0.5443	0.5391
	(–)	(0.0044)	(0.0050)	(0.0045)	(0.0044)
Correlation ($t = 0$)	10,000	0.6147	0.6512	0.6147	0.6170
	(–)	(0.0059)	(0.0063)	(0.0059)	(0.0069)
Correlation (3205)	3205	0.6232	0.6588	0.6231	0.6266
	(–)	(0.0048)	(0.0065)	(0.0048)	(0.0056)
Correlation (4117)	4117	0.6207	0.6570	0.6208	0.6234
	(–)	(0.0049)	(0.0077)	(0.0049)	(0.0058)
Correlation (5566)	5566	0.6145	0.6519	0.6143	0.6176
	(–)	(0.0061)	(0.0079)	(0.0061)	(0.0070)
Correlation (6171)	6171	0.6187	0.6574	0.6189	0.6225
	(–)	(0.0063)	(0.0065)	(0.0063)	(0.0064)
gain ratio ($t = 0$)	113	0.5771	0.5870	0.5771	0.5705
	(–)	(0.0039)	(0.0055)	(0.0038)	(0.0043)

Table 11 continued

Method	#Features	Accuracy	Precision	Recall	F1 Score
gain ratio (3205)	3205	0.5103	0.5340	0.5102	0.5123
	(–)	(0.0057)	(0.0071)	(0.0058)	(0.0059)
gain ratio (4117)	4117	0.5595	0.5823	0.5596	0.5580
	(–)	(0.0057)	(0.0051)	(0.0058)	(0.0053)
gain ratio (5566)	5566	0.5863	0.6190	0.5865	0.5872
	(–)	(0.0038)	(0.0047)	(0.0037)	(0.0041)
gain ratio (6171)	6171	0.5965	0.6295	0.5966	0.5976
	(–)	(0.0042)	(0.0041)	(0.0042)	(0.0042)
Info gain ($t = 0$)	113	0.5771	0.5870	0.5771	0.5705
	(–)	(0.0039)	(0.0055)	(0.0038)	(0.0043)
Info gain (3205)	3205	0.5103	0.5340	0.5102	0.5123
	(–)	(0.0057)	(0.0071)	(0.0058)	(0.0059)
Info gain (4117)	4117	0.5595	0.5823	0.5596	0.5580
	(–)	(0.0057)	(0.0051)	(0.0058)	(0.0053)
Info gain (5566)	5566	0.5863	0.6190	0.5865	0.5872
	(–)	(0.0038)	(0.0047)	(0.0037)	(0.0041)
Info gain (6171)	6171	0.5965	0.6294	0.5966	0.5976
	(–)	(0.0042)	(0.0041)	(0.0042)	(0.0042)
ReliefF ($t = 0$)	7957	0.6087	0.6476	0.6087	0.6120
	(–)	(0.0061)	(0.0069)	(0.0060)	(0.0067)
ReliefF (3205)	3205	0.6008	0.6367	0.6007	0.6044
	(–)	(0.0076)	(0.0074)	(0.0076)	(0.0075)
ReliefF (4117)	4117	0.5972	0.6353	0.5973	0.6010
	(–)	(0.0060)	(0.0068)	(0.0061)	(0.0065)
ReliefF (5566)	5566	0.6087	0.6497	0.6086	0.6121
	(–)	(0.0062)	(0.0088)	(0.0062)	(0.0069)
ReliefF (6171)	6171	0.6093	0.6479	0.6093	0.6132
	(–)	(0.0053)	(0.0076)	(0.0055)	(0.0065)
Significance ($t = 0$)	10,000	0.6139	0.6475	0.6138	0.6151
	(–)	(0.0065)	(0.0065)	(0.0066)	(0.0068)
Significance (3205)	3205	0.5103	0.5340	0.5102	0.5123
	(–)	(0.0057)	(0.0071)	(0.0058)	(0.0059)
Significance (4117)	4117	0.5595	0.5823	0.5596	0.5580
	(–)	(0.0057)	(0.0051)	(0.0058)	(0.0053)
Significance (5566)	5566	0.5863	0.6190	0.5865	0.5872
	(–)	(0.0038)	(0.0047)	(0.0037)	(0.0041)
Significance (6171)	6171	0.5965	0.6295	0.5966	0.5976
	(–)	(0.0042)	(0.0041)	(0.0042)	(0.0042)

Table 11 continued

Method	#Features	Accuracy	Precision	Recall	F1 Score
SpRST (4)	3205.333	0.5634	0.6050	0.5635	0.5681
	(24.35296)	(0.0061)	(0.0073)	(0.0061)	(0.0068)
SpRST (5)	4117.000	0.5736	0.6156	0.5736	0.5772
	(18.95257)	(0.0058)	(0.0071)	(0.0058)	(0.0061)
SpRST (8)	5566.000	0.5893	0.6304	0.5893	0.5929
	(23.89142)	(0.0066)	(0.0067)	(0.0066)	(0.0067)
SpRST (10)	6171.333	0.5934	0.6321	0.5934	0.5966
	(26.01282)	(0.0067)	(0.0065)	(0.0067)	(0.0069)
SumSquaresRatio (3205)	3205	0.6498	0.6740	0.6497	0.6510
	(–)	(0.0031)	(0.0033)	(0.0029)	(0.0035)
SumSquaresRatio (4117)	4117	0.6556	0.6824	0.6557	0.6558
	(–)	(0.0074)	(0.0070)	(0.0073)	(0.0077)
SumSquaresRatio (5566)	5566	0.6507	0.6794	0.6507	0.6522
	(–)	(0.0045)	(0.0061)	(0.0045)	(0.0049)
SumSquaresRatio (6171)	6171	0.6511	0.6825	0.6510	0.6536
	(–)	(0.0056)	(0.0059)	(0.0055)	(0.0059)
Symmetrical uncert ($t = 0$)	113	0.5771	0.5870	0.5771	0.5705
	(–)	(0.0039)	(0.0055)	(0.0038)	(0.0043)
Symmetrical uncert (3205)	3205	0.5103	0.5340	0.5102	0.5123
	(–)	(0.0057)	(0.0071)	(0.0058)	(0.0059)
Symmetrical uncert (4117)	4117	0.5595	0.5823	0.5596	0.5580
	(–)	(0.0057)	(0.0051)	(0.0058)	(0.0053)
Symmetrical uncert (5566)	5566	0.5863	0.6190	0.5865	0.5872
	(–)	(0.0038)	(0.0047)	(0.0037)	(0.0041)
Symmetrical uncert (6171)	6171	0.5965	0.6294	0.5966	0.5976
	(–)	(0.0042)	(0.0041)	(0.0042)	(0.0042)

Averages and standard deviation in brackets

Table 12 Random forest classification results for different feature selection techniques

Method	#Features	Accuracy	Precision	Recall	F1 Score
Original	10,000	0.8026	0.8174	0.8026	0.7963
	(–)	(0.0038)	(0.0042)	(0.0038)	(0.0044)
CV ($t = 0$)	4149	0.6855	0.7264	0.6855	0.6766
	(–)	(0.0036)	(0.0070)	(0.0036)	(0.0040)
CV (3205)	3205	0.6503	0.7029	0.6503	0.6402
	(–)	(0.0045)	(0.0060)	(0.0045)	(0.0052)
CV (4117)	4117	0.6789	0.7224	0.6789	0.6698
	(–)	(0.0042)	(0.0079)	(0.0042)	(0.0043)
CV (5566)	5566	0.7337	0.7621	0.7337	0.7265
	(–)	(0.0031)	(0.0033)	(0.0031)	(0.0038)
CV (6171)	6171	0.7451	0.7740	0.7451	0.7381
	(–)	(0.0050)	(0.0051)	(0.0050)	(0.0050)
Cfs (Greedy)	41	0.6881	0.6881	0.6881	0.6786
	(–)	(0.0019)	(0.0027)	(0.0019)	(0.0022)
Chi squared ($t = 0$)	113	0.7107	0.7171	0.7107	0.7070
	(–)	(0.0029)	(0.0033)	(0.0029)	(0.0029)
Chi squared (3205)	3205	0.7607	0.7717	0.7607	0.7513
	(–)	(0.0029)	(0.0042)	(0.0029)	(0.0029)
Chi squared (4117)	4117	0.7927	0.8054	0.7927	0.7834
	(–)	(0.0036)	(0.0038)	(0.0036)	(0.0039)
Chi squared (5566)	5566	0.8002	0.8145	0.8002	0.7919
	(–)	(0.0038)	(0.0044)	(0.0038)	(0.0041)
Chi squared (6171)	6171	0.8073	0.8225	0.8073	0.7998
	(–)	(0.0058)	(0.0059)	(0.0058)	(0.0062)
Consistency (Greedy)	30	0.6483	0.6489	0.6483	0.6383
	(–)	(0.0033)	(0.0047)	(0.0033)	(0.0035)
Correlation ($t = 0$)	10,000	0.8016	0.8178	0.8016	0.7949
	(–)	(0.0064)	(0.0081)	(0.0064)	(0.0068)
Correlation (3205)	3205	0.7930	0.8073	0.7930	0.7860
	(–)	(0.0032)	(0.0049)	(0.0032)	(0.0036)
Correlation (4117)	4117	0.7979	0.8134	0.7979	0.7909
	(–)	(0.0054)	(0.0054)	(0.0054)	(0.0060)
Correlation (5566)	5566	0.8017	0.8169	0.8017	0.7949
	(–)	(0.0040)	(0.0041)	(0.0040)	(0.0041)
Correlation (6171)	6171	0.8029	0.8186	0.8029	0.7966
	(–)	(0.0049)	(0.0041)	(0.0049)	(0.0054)
Gain ratio ($t = 0$)	113	0.7122	0.7178	0.7122	0.7083
	(–)	(0.0024)	(0.0029)	(0.0024)	(0.0027)
Gain ratio (3205)	3205	0.7586	0.7681	0.7586	0.7487
	(–)	(0.0054)	(0.0064)	(0.0054)	(0.0055)

Table 12 continued

Method	#Features	Accuracy	Precision	Recall	F1 Score
Gain ratio (4117)	4117	0.7927	0.8065	0.7927	0.7837
	(–)	(0.0038)	(0.0034)	(0.0038)	(0.0040)
Gain ratio (5566)	5566	0.8021	0.8158	0.8021	0.7929
	(–)	(0.0042)	(0.0048)	(0.0042)	(0.0047)
Gain ratio (6171)	6171	0.8066	0.8211	0.8066	0.7985
	(–)	(0.0057)	(0.0063)	(0.0057)	(0.0058)
Info gain ($t = 0$)	113	0.7097	0.7159	0.7097	0.7058
	(–)	(0.0044)	(0.0042)	(0.0044)	(0.0046)
Info gain (3205)	3205	0.7621	0.7725	0.7621	0.7524
	(–)	(0.0048)	(0.0067)	(0.0048)	(0.0048)
Info gain (4117)	4117	0.7921	0.8067	0.7921	0.7823
	(–)	(0.0033)	(0.0040)	(0.0033)	(0.0037)
Info gain (5566)	5566	0.8007	0.8151	0.8007	0.7913
	(–)	(0.0039)	(0.0054)	(0.0039)	(0.0044)
Info gain (6171)	6171	0.8093	0.8231	0.8093	0.8017
	(–)	(0.0058)	(0.0064)	(0.0058)	(0.0063)
ReliefF ($t = 0$)	7957	0.8051	0.8210	0.8051	0.7989
	(–)	(0.0051)	(0.0047)	(0.0051)	(0.0060)
ReliefF (3205)	3205	0.7959	0.8106	0.7959	0.7891
	(–)	(0.0051)	(0.0064)	(0.0051)	(0.0059)
ReliefF (4117)	4117	0.8000	0.8133	0.8000	0.7935
	(–)	(0.0039)	(0.0037)	(0.0039)	(0.0040)
ReliefF (5566)	5566	0.8017	0.8172	0.8017	0.7949
	(–)	(0.0056)	(0.0071)	(0.0056)	(0.0058)
ReliefF (6171)	6171	0.8015	0.8186	0.8015	0.7951
	(–)	(0.0061)	(0.0066)	(0.0061)	(0.0066)
Significance ($t = 0$)	10,000	0.7998	0.8170	0.7998	0.7933
	(–)	(0.0070)	(0.0079)	(0.0070)	(0.0078)
Significance (3205)	3205	0.7587	0.7682	0.7587	0.7490
	(–)	(0.0035)	(0.0060)	(0.0035)	(0.0039)
Significance (4117)	4117	0.7952	0.8076	0.7952	0.7860
	(–)	(0.0052)	(0.0040)	(0.0052)	(0.0053)
Significance (5566)	5566	0.8043	0.8187	0.8043	0.7960
	(–)	(0.0040)	(0.0052)	(0.0040)	(0.0049)
Significance (6171)	6171	0.8121	0.8258	0.8121	0.8047
	(–)	(0.0042)	(0.0037)	(0.0042)	(0.0042)
SpRST (4)	3205.333	0.7733	0.7878	0.7733	0.7652
	(24.35296)	(0.0058)	(0.0062)	(0.0058)	(0.0059)
SpRST (5)	4117.000	0.7823	0.7980	0.7823	0.7743
	(18.95257)	(0.0065)	(0.0072)	(0.0065)	(0.0070)
SpRST (8)	5566.000	0.7931	0.8092	0.7931	0.7861
	(23.89142)	(0.0060)	(0.0053)	(0.0060)	(0.0063)

Table 12 continued

Method	#Features	Accuracy	Precision	Recall	F1 Score
SpRST (10)	6171.333	0.7938	0.8107	0.7938	0.7870
	(26.01282)	(0.0063)	(0.0063)	(0.0063)	(0.0069)
SumSquaresRatio (3205)	3205	0.8111	0.8239	0.8111	0.8051
	(–)	(0.0052)	(0.0055)	(0.0052)	(0.0055)
SumSquaresRatio (4117)	4117	0.8091	0.8212	0.8091	0.8034
	(–)	(0.0035)	(0.0037)	(0.0035)	(0.0038)
SumSquaresRatio (5566)	5566	0.8066	0.8213	0.8066	0.8002
	(–)	(0.0032)	(0.0059)	(0.0032)	(0.0033)
SumSquaresRatio (6171)	6171	0.8045	0.8203	0.8045	0.7982
	(–)	(0.0044)	(0.0048)	(0.0044)	(0.0048)
Symmetricaluncert ($t = 0$)	113	0.7117	0.7182	0.7117	0.7079
	(–)	(0.0044)	(0.0038)	(0.0044)	(0.0044)
Symmetricaluncert (3205)	3205	0.7593	0.7713	0.7593	0.7498
	(–)	(0.0030)	(0.0033)	(0.0030)	(0.0037)
Symmetricaluncert (4117)	4117	0.7930	0.8055	0.7930	0.7839
	(–)	(0.0042)	(0.0054)	(0.0042)	(0.0047)
Symmetricaluncert (5566)	5566	0.8021	0.8170	0.8021	0.7936
	(–)	(0.0048)	(0.0047)	(0.0048)	(0.0052)
Symmetricaluncert (6171)	6171	0.8089	0.8242	0.8089	0.8011
	(–)	(0.0042)	(0.0054)	(0.0042)	(0.0048)

Averages and standard deviation in brackets

Table 13 Summary of runtimes (in seconds)

Method	#Features	Selection	Random forest	Naive Bayes
Original	10,000	0	478.1466	154.3220
	(–)	(–)	(29.3932)	(13.4778)
CV ($t = 0$)	4149	49	184.2093	59.7090
	(–)	(–)	(8.0784)	(5.3565)
CV (3205)	3205	46	240.1020	44.5060
	(–)	(–)	(6.9240)	(3.9978)
CV (4117)	4117	47	199.5176	58.9020
	(–)	(–)	(36.3792)	(6.2692)
CV (5566)	5566	48	200.4283	78.9990
	(–)	(–)	(3.1720)	(9.0758)
CV (6171)	6171	48	258.3198	86.9790
	(–)	(–)	(40.7532)	(8.5235)
Cfs (Greedy)	41	296	72.0946	1.1660
	(–)	(–)	(7.3634)	(0.1060)
Chi squared ($t = 0$)	113	2	91.0451	2.0520
	(–)	(–)	(6.9113)	(0.2946)
Chi squared (3205)	3205	3	197.3493	50.5860
	(–)	(–)	(33.7688)	(16.9048)
Chi squared (4117)	4117	2	258.7080	59.6300
	(–)	(–)	(21.4711)	(5.5790)
Chi squared (5566)	5566	2	218.7398	77.6480
	(–)	(–)	(7.0253)	(4.9175)
Chi squared (6171)	6171	3	233.1519	84.5110
	(–)	(–)	(13.3455)	(4.3186)
Consistency (Greedy)	30	608	55.0971	0.9240
	(–)	(–)	(5.3676)	(0.0534)
Correlation ($t = 0$)	10,000	10	339.7830	135.5330
	(–)	(–)	(20.4477)	(6.5574)
Correlation (3205)	3205	8	222.5547	39.6600
	(–)	(–)	(8.3334)	(2.3442)
Correlation (4117)	4117	8	409.2935	53.0690
	(–)	(–)	(266.9731)	(2.9146)
Correlation (5566)	5566	9	242.3133	74.3500
	(–)	(–)	(13.5045)	(4.3644)
Correlation (6171)	6171	8	277.2288	81.8020
	(–)	(–)	(9.0772)	(4.5509)
Gain ratio ($t = 0$)	113	7	88.3953	1.9850
	(–)	(–)	(7.3338)	(0.1276)
Gain ratio (3205)	3205	8	211.9993	44.6560
	(–)	(–)	(8.9143)	(1.6366)
Gain ratio (4117)	4117	9	207.8572	58.0990
	(–)	(–)	(33.3359)	(2.7974)

Table 13 continued

Method	#Features	Selection	Random forest	Naive Bayes
Gain ratio (5566)	5566	8	229.9701	77.2070
	(–)	(–)	(10.6783)	(3.2169)
Gain ratio (6171)	6171	9	236.6279	88.3710
	(–)	(–)	(10.8751)	(7.0199)
Info gain ($t = 0$)	113	7	81.1204	2.0050
	(–)	(–)	(2.7119)	(0.1910)
Info gain (3205)	3205	8	203.8654	45.8720
	(–)	(–)	(5.2012)	(3.2261)
Info gain (4117)	4117	8	234.0230	58.9890
	(–)	(–)	(31.3471)	(2.7353)
Info gain (5566)	5566	9	200.8629	78.1260
	(–)	(–)	(17.7479)	(3.4878)
Info gain (6171)	6171	8	227.2942	85.6750
	(–)	(–)	(3.2503)	(5.4645)
ReliefF ($t = 0$)	7957	1	327.6763	106.4700
	(–)	(–)	(63.6214)	(6.8651)
ReliefF (3205)	3205	8	199.1945	41.4520
	(–)	(–)	(6.3894)	(2.0377)
ReliefF (4117)	4117	8	225.2873	53.7170
	(–)	(–)	(6.5349)	(3.5077)
ReliefF (5566)	5566	9	253.8634	73.7800
	(–)	(–)	(8.2094)	(5.1618)
ReliefF (6171)	6171	9	321.8576	80.0170
	(–)	(–)	(87.1975)	(3.2470)
Significance ($t = 0$)	10,000	3	345.5262	135.7860
	(–)	(–)	(47.0985)	(5.4555)
Significance (3205)	3205	3	150.2230	45.2610
	(–)	(–)	(4.9146)	(2.3983)
Significance (4117)	4117	2	161.6598	58.9500
	(–)	(–)	(8.0198)	(3.0718)
Significance (5566)	5566	3	326.4685	76.0120
	(–)	(–)	(14.4817)	(3.0610)
Significance (6171)	6171	3	227.0810	84.6230
	(–)	(–)	(43.6380)	(4.6004)
SpRST (4)		1 node: 51.98853		
		2 nodes: 51		
	3205.333	4 nodes: 25.86191	239.5910	39.8933
	(24.35296)	8 nodes: 20.81937	(97.0897)	(1.2833)
		16 nodes: 18.0587		
		32 nodes: 16.60878		

Table 13 continued

Method	#Features	Selection	Random forest	Naive Bayes
SpRST (5)		1 node: 72.27206		
		2 nodes: 52.9		
	4117.000	4 nodes: 49.97752	227.8856	51.9187
	(18.95257)	8 nodes: 28.46254	(49.6911)	(2.8680)
		16 nodes: 22.83241		
		32 nodes: 18.30543		
SpRST (8)		1 node: 935.9299		
		2 nodes: 374.9		
	5566.000	4 nodes: 646.0979	257.9724	72.9547
	(23.89142)	8 nodes: 209.8693	(53.8414)	(5.2135)
		16 nodes: 118.5509		
		32 nodes: 88.09231		
SpRST (10)		1 node: 4448.072		
		2 nodes: 1540.4		
	6171.333	4 nodes: 1641.565	245.9802	79.3787
	(26.01282)	8 nodes: 865.2048	(46.8001)	(3.6052)
		16 nodes: 585.8696		
		32 nodes: 334.1958		
Sum squares ratio (3205)	3205	2	201.7381	41.4670
	(–)	(–)	(7.0755)	(1.7756)
Sum squares ratio (4117)	4117	2	231.8041	58.8280
	(–)	(–)	(21.1133)	(11.8832)
Sum squares ratio (5566)	5566	2	246.9675	77.6570
	(–)	(–)	(10.4719)	(10.3124)
Sum squares ratio (6171)	6171	2	363.0283	84.5040
	(–)	(–)	(65.0364)	(9.1140)
Symmetricaluncert ($t = 0$)	113	7	90.2343	2.0700
	(–)	(–)	(7.6894)	(0.4490)
Symmetricaluncert (3205)	3205	8	163.9254	45.9670
	(–)	(–)	(47.5122)	(3.1956)
Symmetricaluncert (4117)	4117	9	158.4968	59.0650
	(–)	(–)	(5.7051)	(4.4659)
Symmetricaluncert (5566)	5566	8	249.9402	76.5380
	(–)	(–)	(64.8124)	(5.3429)
Symmetricaluncert (6171)	6171	9	282.1294	917.6640
	(–)	(–)	(74.9137)	(2630.7532)

Averages and standard deviation in brackets

Appendix B: Results of Wilcoxon rank sum tests

See Tables 14, 15, 16, 17, 18, 19, 20 and 21

Table 14 Results of Wilcoxon rank sum tests with Bonferroni correction for Naive Bayes and the accuracy metric

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
SpRST (5)	1.1e−09	–	–	–
SpRST (8)	< 2e−16	6.0e−15	–	–
SpRST (10)	< 2e−16	< 2e−16	1.0000	–
Original (–)	0.0018	0.0018	0.0032	0.0170
Cfs (Greedy)	0.0018	0.0018	0.0020	0.0032
Chi squared (3205)	0.0018	0.0018	0.0018	0.0018
Chi squared (4117)	1.0000	0.0177	0.0018	0.0018
Chi squared (5566)	0.0018	0.0051	1.0000	1.0000
Chi squared (6171)	0.0018	0.0018	1.0000	1.0000
Chi squared ($t = 0$)	0.0041	1.0000	0.0403	0.0038
Consistency (Greedy)	0.0026	0.0018	0.0018	0.0018
Correlation (3205)	0.0018	0.0018	0.0018	0.0018
Correlation (4117)	0.0018	0.0018	0.0018	0.0018
Correlation (5566)	0.0018	0.0018	0.0020	0.0022
Correlation (6171)	0.0018	0.0018	0.0018	0.0019
Correlation ($t = 0$)	0.0018	0.0018	0.0018	0.0018
CV (3205)	0.0018	0.0018	0.0018	0.0018
CV (4117)	0.0018	0.0018	0.0018	0.0018
CV (5566)	0.0026	0.0018	0.0018	0.0018
CV (6171)	0.0176	0.0018	0.0018	0.0018
CV ($t = 0$)	0.0018	0.0018	0.0018	0.0018
Gain ratio (3205)	0.0018	0.0018	0.0018	0.0018
Gain ratio (4117)	1.0000	0.0177	0.0018	0.0018
Gain ratio (5566)	0.0018	0.0051	1.0000	1.0000
Gain ratio (6171)	0.0018	0.0018	1.0000	1.0000
Gain ratio ($t = 0$)	0.0041	1.0000	0.0403	0.0038
Info gain (3205)	0.0018	0.0018	0.0018	0.0018
Info gain (4117)	1.0000	0.0177	0.0018	0.0018
Info gain (5566)	0.0018	0.0051	1.0000	1.0000
Info gain (6171)	0.0018	0.0018	1.0000	1.0000
Info gain ($t = 0$)	0.0041	1.0000	0.0403	0.0038
ReliefF (3205)	0.0018	0.0018	1.0000	1.0000
ReliefF (4117)	0.0018	0.0018	1.0000	1.0000
ReliefF (5566)	0.0018	0.0018	0.0031	0.0139
ReliefF (6171)	0.0018	0.0018	0.0024	0.0061
ReliefF ($t = 0$)	0.0018	0.0018	0.0032	0.0157

Table 14 continued

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
Significance (3205)	0.0018	0.0018	0.0018	0.0018
Significance (4117)	1.0000	0.0177	0.0018	0.0018
Significance (5566)	0.0018	0.0051	1.0000	1.0000
Significance (6171)	0.0018	0.0018	1.0000	1.0000
Significance ($t = 0$)	0.0018	0.0018	0.0018	0.0018
Sum squares ratio (3205)	0.0018	0.0018	0.0018	0.0018
Sum squares ratio (4117)	0.0018	0.0018	0.0018	0.0018
Sum squares ratio (5566)	0.0018	0.0018	0.0018	0.0018
Sum squares ratio (6171)	0.0018	0.0018	0.0018	0.0018
Symmetricaluncert (3205)	0.0018	0.0018	0.0018	0.0018
Symmetricaluncert (4117)	1.0000	0.0177	0.0018	0.0018
Symmetricaluncert (5566)	0.0018	0.0051	1.0000	1.0000
Symmetricaluncert (6171)	0.0018	0.0018	1.0000	1.0000
Symmetricaluncert ($t = 0$)	0.0041	1.0000	0.0403	0.0038

Table 15 Results of Wilcoxon rank sum tests with Bonferroni correction for Naive Bayes and the recall metric

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
SpRST (5)	1.7e−09	–	–	–
SpRST (8)	$<$ 2e−16	5.9e−15	–	–
SpRST (10)	$<$ 2e−16	$<$ 2e−16	1.0000	–
Original (–)	0.0018	0.0018	0.0032	0.0181
Cfs (Greedy)	0.0018	0.0018	0.0020	0.0035
Chi squared (3205)	0.0018	0.0018	0.0018	0.0018
Chi squared (4117)	1.0000	0.0204	0.0018	0.0018
Chi squared (5566)	0.0018	0.0044	1.0000	1.0000
Chi squared (6171)	0.0018	0.0018	1.0000	1.0000
Chi squared ($t = 0$)	0.0042	1.0000	0.0383	0.0038
Consistency (Greedy)	0.0025	0.0018	0.0018	0.0018
Correlation (3205)	0.0018	0.0018	0.0018	0.0018
Correlation (4117)	0.0018	0.0018	0.0018	0.0018
Correlation (5566)	0.0018	0.0018	0.0020	0.0022
Correlation (6171)	0.0018	0.0018	0.0018	0.0019
Correlation ($t = 0$)	0.0018	0.0018	0.0018	0.0018
CV (3205)	0.0018	0.0018	0.0018	0.0018
CV (4117)	0.0018	0.0018	0.0018	0.0018
CV (5566)	0.0025	0.0018	0.0018	0.0018
CV (6171)	0.0178	0.0018	0.0018	0.0018
CV ($t = 0$)	0.0018	0.0018	0.0018	0.0018

Table 15 continued

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
Gain ratio (3205)	0.0018	0.0018	0.0018	0.0018
Gain ratio (4117)	1.0000	0.0204	0.0018	0.0018
Gain ratio (5566)	0.0018	0.0044	1.0000	1.0000
Gain ratio (6171)	0.0018	0.0018	1.0000	1.0000
Gain ratio ($t = 0$)	0.0042	1.0000	0.0383	0.0038
Info gain (3205)	0.0018	0.0018	0.0018	0.0018
Info gain (4117)	1.0000	0.0204	0.0018	0.0018
Info gain (5566)	0.0018	0.0044	1.0000	1.0000
Info gain (6171)	0.0018	0.0018	1.0000	1.0000
Info gain ($t = 0$)	0.0042	1.0000	0.0383	0.0038
ReliefF (3205)	0.0018	0.0018	1.0000	1.0000
ReliefF (4117)	0.0018	0.0018	1.0000	1.0000
ReliefF (5566)	0.0018	0.0018	0.0032	0.0167
ReliefF (6171)	0.0018	0.0018	0.0024	0.0060
ReliefF ($t = 0$)	0.0018	0.0018	0.0032	0.0154
Significance (3205)	0.0018	0.0018	0.0018	0.0018
Significance (4117)	1.0000	0.0204	0.0018	0.0018
Significance (5566)	0.0018	0.0044	1.0000	1.0000
Significance (6171)	0.0018	0.0018	1.0000	1.0000
Significance ($t = 0$)	0.0018	0.0018	0.0018	0.0020
Sum squares ratio (3205)	0.0018	0.0018	0.0018	0.0018
Sum squares ratio (4117)	0.0018	0.0018	0.0018	0.0018
Sum squares ratio (5566)	0.0018	0.0018	0.0018	0.0018
Sum squares ratio (6171)	0.0018	0.0018	0.0018	0.0018
Symmetricaluncert (3205)	0.0018	0.0018	0.0018	0.0018
Symmetricaluncert (4117)	1.0000	0.0204	0.0018	0.0018
Symmetricaluncert (5566)	0.0018	0.0044	1.0000	1.0000
Symmetricaluncert (6171)	0.0018	0.0018	1.0000	1.0000
Symmetricaluncert ($t = 0$)	0.0042	1.0000	0.0383	0.0038

Table 16 Results of Wilcoxon rank sum tests with Bonferroni correction for Naive Bayes and the precision metric

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
SpRST (5)	3.3e−07	–	–	–
SpRST (8)	$<$ 2e−16	2.2e−13	–	–
SpRST (10)	$<$ 2e−16	1.1e−14	1.0000	–
Original (–)	0.0018	0.0018	0.0540	0.1821
Cfs (Greedy)	0.6941	1.0000	0.0060	0.0031
Chi squared (3205)	0.0018	0.0018	0.0018	0.0018
Chi squared (4117)	0.0018	0.0018	0.0018	0.0018
Chi squared (5566)	0.0122	1.0000	0.0787	0.0194

Table 16 continued

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
Chi squared (6171)	0.0018	0.0132	1.0000	1.0000
Chi squared ($t = 0$)	0.0040	0.0018	0.0018	0.0018
Consistency (Greedy)	0.0018	0.0018	0.0018	0.0018
Correlation (3205)	0.0018	0.0018	0.0019	0.0019
Correlation (4117)	0.0018	0.0018	0.0020	0.0019
Correlation (5566)	0.0018	0.0018	0.0044	0.0065
Correlation (6171)	0.0018	0.0018	0.0020	0.0019
Correlation ($t = 0$)	0.0018	0.0018	0.0024	0.0024
CV (3205)	0.0018	0.0018	0.0018	0.0018
CV (4117)	0.0018	0.0018	0.0018	0.0018
CV (5566)	0.0715	0.0029	0.0018	0.0018
CV (6171)	1.0000	0.0041	0.0018	0.0018
CV ($t = 0$)	0.0018	0.0018	0.0018	0.0018
Gain ratio (3205)	0.0018	0.0018	0.0018	0.0018
Gain ratio (4117)	0.0018	0.0018	0.0018	0.0018
Gain ratio (5566)	0.0122	1.0000	0.0787	0.0194
Gain ratio (6171)	0.0018	0.0132	1.0000	1.0000
Gain ratio ($t = 0$)	0.0040	0.0018	0.0018	0.0018
Info gain (3205)	0.0018	0.0018	0.0018	0.0018
Info gain (4117)	0.0018	0.0018	0.0018	0.0018
Info gain (5566)	0.0122	1.0000	0.0787	0.0194
Info gain (6171)	0.0018	0.0138	1.0000	1.0000
Info gain ($t = 0$)	0.0040	0.0018	0.0018	0.0018
ReliefF (3205)	0.0018	0.0040	1.0000	1.0000
ReliefF (4117)	0.0018	0.0040	1.0000	1.0000
ReliefF (5566)	0.0018	0.0018	0.0167	0.0325
ReliefF (6171)	0.0018	0.0018	0.0142	0.0247
ReliefF ($t = 0$)	0.0018	0.0018	0.0126	0.0246
Significance (3205)	0.0018	0.0018	0.0018	0.0018
Significance (4117)	0.0018	0.0018	0.0018	0.0018
Significance (5566)	0.0122	1.0000	0.0787	0.0194
Significance (6171)	0.0018	0.0132	1.0000	1.0000
Significance ($t = 0$)	0.0018	0.0018	0.0060	0.0101
Sum squares ratio (3205)	0.0018	0.0018	0.0018	0.0018
Sum squares ratio (4117)	0.0018	0.0018	0.0018	0.0018
Sum squares ratio (5566)	0.0018	0.0018	0.0018	0.0018
Sum squares ratio (6171)	0.0018	0.0018	0.0018	0.0018
Symmetricaluncert (3205)	0.0018	0.0018	0.0018	0.0018
Symmetricaluncert (4117)	0.0018	0.0018	0.0018	0.0018
Symmetricaluncert (5566)	0.0122	1.0000	0.0787	0.0194
Symmetricaluncert (6171)	0.0018	0.0138	1.0000	1.0000
Symmetricaluncert ($t = 0$)	0.0040	0.0018	0.0018	0.0018

Table 17 Results of Wilcoxon rank sum tests with Bonferroni correction for Naive Bayes and the F1 metric

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
SpRST (5)	6e−07	–	–	–
SpRST (8)	< 2e−16	1e−14	–	–
SpRST (10)	< 2e−16	< 2e−16	1.0000	–
Original (–)	0.0018	0.0018	0.0074	0.0562
Cfs (Greedy)	0.0018	0.0018	0.0270	1.0000
Chi squared (3205)	0.0018	0.0018	0.0018	0.0018
Chi squared (4117)	0.5275	0.0032	0.0018	0.0018
Chi squared (5566)	0.0018	0.1024	1.0000	0.5425
Chi squared (6171)	0.0018	0.0019	1.0000	1.0000
Chi squared ($t = 0$)	1.0000	1.0000	0.0018	0.0018
Consistency (Greedy)	0.0018	0.0018	0.0018	0.0018
Correlation (3205)	0.0018	0.0018	0.0018	0.0018
Correlation (4117)	0.0018	0.0018	0.0018	0.0018
Correlation (5566)	0.0018	0.0018	0.0021	0.0033
Correlation (6171)	0.0018	0.0018	0.0018	0.0020
Correlation ($t = 0$)	0.0018	0.0018	0.0019	0.0022
CV (3205)	0.0018	0.0018	0.0018	0.0018
CV (4117)	0.0018	0.0018	0.0018	0.0018
CV (5566)	0.0024	0.0018	0.0018	0.0018
CV (6171)	0.0399	0.0019	0.0018	0.0018
CV ($t = 0$)	0.0018	0.0018	0.0018	0.0018
Gain ratio (3205)	0.0018	0.0018	0.0018	0.0018
Gain ratio (4117)	0.5275	0.0032	0.0018	0.0018
Gain ratio (5566)	0.0018	0.1024	1.0000	0.5425
Gain ratio (6171)	0.0018	0.0019	1.0000	1.0000
Gain ratio ($t = 0$)	1.0000	1.0000	0.0018	0.0018
Info gain (3205)	0.0018	0.0018	0.0018	0.0018
Info gain (4117)	0.5275	0.0032	0.0018	0.0018
Info gain (5566)	0.0018	0.1024	1.0000	0.5425
Info gain (6171)	0.0018	0.0019	1.0000	1.0000
Info gain ($t = 0$)	1.0000	1.0000	0.0018	0.0018
ReliefF (3205)	0.0018	0.0018	0.8417	1.0000
ReliefF (4117)	0.0018	0.0018	1.0000	1.0000
ReliefF (5566)	0.0018	0.0018	0.0058	0.0326
ReliefF (6171)	0.0018	0.0018	0.0033	0.0094
ReliefF ($t = 0$)	0.0018	0.0018	0.0058	0.0230
Significance (3205)	0.0018	0.0018	0.0018	0.0018
Significance (4117)	0.5275	0.0032	0.0018	0.0018
Significance (5566)	0.0018	0.1024	1.0000	0.5425
Significance (6171)	0.0018	0.0019	1.0000	1.0000
Significance ($t = 0$)	0.0018	0.0018	0.0023	0.0033

Table 17 continued

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
Sum squares ratio (3205)	0.0018	0.0018	0.0018	0.0018
Sum squares ratio (4117)	0.0018	0.0018	0.0018	0.0018
Sum squares ratio (5566)	0.0018	0.0018	0.0018	0.0018
Sum squares ratio (6171)	0.0018	0.0018	0.0018	0.0018
Symmetricaluncert (3205)	0.0018	0.0018	0.0018	0.0018
Symmetricaluncert (4117)	0.5275	0.0032	0.0018	0.0018
Symmetricaluncert (5566)	0.0018	0.1024	1.0000	0.5425
Symmetricaluncert (6171)	0.0018	0.0019	1.0000	1.0000
Symmetricaluncert ($t = 0$)	1.0000	1.0000	0.0018	0.0018

Table 18 Results of Wilcoxon rank sum tests with Bonferroni correction for random forest and the accuracy metric

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
SpRST (05)	2.2e−07	–	–	–
SpRST (08)	$<$2e−16	2.5e−09	–	–
SpRST (10)	$<$2e−16	4.1e−10	1.0000	–
Original (–)	0.0018	0.0018	0.1252	0.5138
Cfs (Greedy)	0.0018	0.0018	0.0018	0.0018
Chi squared (3205)	0.0061	0.0018	0.0018	0.0018
Chi squared (4117)	0.0020	0.0897	1.0000	1.0000
Chi squared (5566)	0.0018	0.0024	1.0000	1.0000
Chi squared (6171)	0.0018	0.0018	0.0113	0.0216
Chi squared ($t = 0$)	0.0018	0.0018	0.0018	0.0018
Consistency (Greedy)	0.0018	0.0018	0.0018	0.0018
Correlation (3205)	0.0022	0.0616	1.0000	1.0000
Correlation (4117)	0.0018	0.0058	1.0000	1.0000
Correlation (5566)	0.0018	0.0024	0.4050	1.0000
Correlation (6171)	0.0018	0.0020	0.2470	0.7423
Correlation ($t = 0$)	0.0018	0.0031	1.0000	1.0000
CV (3205)	0.0018	0.0018	0.0018	0.0018
CV (4117)	0.0018	0.0018	0.0018	0.0018
CV (5566)	0.0018	0.0018	0.0018	0.0018
CV (6171)	0.0018	0.0018	0.0018	0.0018
CV ($t = 0$)	0.0018	0.0018	0.0018	0.0018
Gain ratio (3205)	0.0100	0.0018	0.0018	0.0018
Gain ratio (4117)	0.0024	0.1245	1.0000	1.0000
Gain ratio (5566)	0.0018	0.0022	0.3406	1.0000
Gain ratio (6171)	0.0018	0.0020	0.0361	0.0803
Gain ratio ($t = 0$)	0.0018	0.0018	0.0018	0.0018

Table 18 continued

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
Info gain (3205)	0.0453	0.0020	0.0018	0.0018
Info gain (4117)	0.0022	0.1735	1.0000	1.0000
Info gain (5566)	0.0018	0.0022	1.0000	1.0000
Info gain (6171)	0.0018	0.0018	0.0024	0.0032
Info gain ($t = 0$)	0.0018	0.0018	0.0018	0.0018
ReliefF (3205)	0.0020	0.0285	1.0000	1.0000
ReliefF (4117)	0.0018	0.0026	1.0000	1.0000
ReliefF (5566)	0.0018	0.0028	1.0000	1.0000
ReliefF (6171)	0.0018	0.0035	1.0000	1.0000
ReliefF ($t = 0$)	0.0018	0.0018	0.0170	0.0359
Significance (3205)	0.0030	0.0018	0.0018	0.0018
Significance (4117)	0.0022	0.0509	1.0000	1.0000
Significance (5566)	0.0018	0.0018	0.0333	0.0831
Significance (6171)	0.0018	0.0018	0.0018	0.0020
Significance ($t = 0$)	0.0018	0.0056	1.0000	1.0000
Sum squares ratio (3205)	0.0018	0.0018	0.0024	0.0032
Sum squares ratio (4117)	0.0018	0.0018	0.0022	0.0028
Sum squares ratio (5566)	0.0018	0.0018	0.0028	0.0051
Sum squares ratio (6171)	0.0018	0.0020	0.0263	0.0640
Symmetricaluncert (3205)	0.0035	0.0018	0.0018	0.0018
Symmetricaluncert (4117)	0.0024	0.1251	1.0000	1.0000
Symmetricaluncert (5566)	0.0018	0.0019	0.6292	1.0000
Symmetricaluncert (6171)	0.0018	0.0018	0.0018	0.0032
Symmetricaluncert ($t = 0$)	0.0018	0.0018	0.0018	0.0018

Table 19 Results of Wilcoxon rank sum tests with Bonferroni correction for Random forest and the recall metric

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
SpRST (05)	2.2e−07	–	–	–
SpRST (08)	$<$ 2e−16	2.5e−09	–	–
SpRST (10)	$<$ 2e−16	4.1e−10	1.0000	–
Original (–)	0.0018	0.0018	0.1252	0.5138
Cfs (Greedy)	0.0018	0.0018	0.0018	0.0018
Chi squared (3205)	0.0061	0.0018	0.0018	0.0018
Chi squared (4117)	0.0020	0.0897	1.0000	1.0000
Chi squared (5566)	0.0018	0.0024	1.0000	1.0000
Chi squared (6171)	0.0018	0.0018	0.0113	0.0216
Chi squared ($t = 0$)	0.0018	0.0018	0.0018	0.0018

Table 19 continued

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
Consistency (Greedy)	0.0018	0.0018	0.0018	0.0018
Correlation (3205)	0.0022	0.0616	1.0000	1.0000
Correlation (4117)	0.0018	0.0058	1.0000	1.0000
Correlation (5566)	0.0018	0.0024	0.4050	1.0000
Correlation (6171)	0.0018	0.0020	0.2470	0.7423
Correlation ($t = 0$)	0.0018	0.0031	1.0000	1.0000
CV (3205)	0.0018	0.0018	0.0018	0.0018
CV (4117)	0.0018	0.0018	0.0018	0.0018
CV (5566)	0.0018	0.0018	0.0018	0.0018
CV (6171)	0.0018	0.0018	0.0018	0.0018
CV ($t = 0$)	0.0018	0.0018	0.0018	0.0018
Gain ratio (3205)	0.0100	0.0018	0.0018	0.0018
Gain ratio (4117)	0.0024	0.1245	1.0000	1.0000
Gain ratio (5566)	0.0018	0.0022	0.3406	1.0000
Gain ratio (6171)	0.0018	0.0020	0.0361	0.0803
Gain ratio ($t = 0$)	0.0018	0.0018	0.0018	0.0018
Info gain (3205)	0.0453	0.0020	0.0018	0.0018
Info gain (4117)	0.0022	0.1735	1.0000	1.0000
Info gain (5566)	0.0018	0.0022	1.0000	1.0000
Info gain (6171)	0.0018	0.0018	0.0024	0.0032
Info gain ($t = 0$)	0.0018	0.0018	0.0018	0.0018
ReliefF (3205)	0.0020	0.0285	1.0000	1.0000
ReliefF (4117)	0.0018	0.0026	1.0000	1.0000
ReliefF (5566)	0.0018	0.0028	1.0000	1.0000
ReliefF (6171)	0.0018	0.0035	1.0000	1.0000
ReliefF ($t = 0$)	0.0018	0.0018	0.0170	0.0359
Significance (3205)	0.0030	0.0018	0.0018	0.0018
Significance (4117)	0.0022	0.0509	1.0000	1.0000
Significance (5566)	0.0018	0.0018	0.0333	0.0831
Significance (6171)	0.0018	0.0018	0.0018	0.0020
Significance ($t = 0$)	0.0018	0.0056	1.0000	1.0000
Sum squares ratio (3205)	0.0018	0.0018	0.0024	0.0032
Sum squares ratio (4117)	0.0018	0.0018	0.0022	0.0028
Sum squares ratio (5566)	0.0018	0.0018	0.0028	0.0051
Sum squares ratio (6171)	0.0018	0.0020	0.0263	0.0640
Symmetricaluncert (3205)	0.0035	0.0018	0.0018	0.0018
Symmetricaluncert (4117)	0.0024	0.1251	1.0000	1.0000
Symmetricaluncert (5566)	0.0018	0.0019	0.6292	1.0000
Symmetricaluncert (6171)	0.0018	0.0018	0.0018	0.0032
Symmetricaluncert ($t = 0$)	0.0018	0.0018	0.0018	0.0018

Table 20 Results of Wilcoxon rank sum tests with Bonferroni correction for random forest and the precision metric

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
SpRST (05)	1.1e−07	–	–	–
SpRST (08)	< 2e−16	1.1e−09	–	–
SpRST (10)	< 2e−16	8.5e−11	1.0000	–
Original (–)	0.0019	0.0034	0.3327	1.0000
Cfs (Greedy)	0.0019	0.0019	0.0019	0.0019
Chi squared (3205)	0.0031	0.0019	0.0019	0.0019
Chi squared (4117)	0.0029	1.0000	1.0000	1.0000
Chi squared (5566)	0.0019	0.0110	1.0000	1.0000
Chi squared (6171)	0.0019	0.0024	0.0289	0.1315
Chi squared ($t = 0$)	0.0019	0.0019	0.0019	0.0019
Consistency (Greedy)	0.0019	0.0019	0.0019	0.0019
Correlation (3205)	0.0024	1.0000	1.0000	1.0000
Correlation (4117)	0.0019	0.0094	1.0000	1.0000
Correlation (5566)	0.0019	0.0034	0.7525	1.0000
Correlation (6171)	0.0019	0.0029	0.0783	1.0000
Correlation ($t = 0$)	0.0019	0.0052	1.0000	1.0000
CV (3205)	0.0019	0.0019	0.0019	0.0019
CV (4117)	0.0019	0.0019	0.0019	0.0019
CV (5566)	0.0019	0.0019	0.0019	0.0019
CV (6171)	0.0079	0.0019	0.0019	0.0019
CV ($t = 0$)	0.0019	0.0019	0.0019	0.0019
Gain ratio (3205)	0.0034	0.0019	0.0019	0.0019
Gain ratio (4117)	0.0024	1.0000	1.0000	1.0000
Gain ratio (5566)	0.0019	0.0102	1.0000	1.0000
Gain ratio (6171)	0.0019	0.0031	0.1415	0.4696
Gain ratio ($t = 0$)	0.0019	0.0019	0.0019	0.0019
Info gain (3205)	0.0067	0.0019	0.0019	0.0019
Info gain (4117)	0.0024	1.0000	1.0000	1.0000
Info gain (5566)	0.0019	0.0067	1.0000	1.0000
Info gain (6171)	0.0019	0.0019	0.0019	0.0247
Info gain ($t = 0$)	0.0019	0.0019	0.0019	0.0019
ReliefF (3205)	0.0022	0.2696	1.0000	1.0000
ReliefF (4117)	0.0019	0.0120	1.0000	1.0000
ReliefF (5566)	0.0019	0.0086	1.0000	1.0000
ReliefF (6171)	0.0019	0.0057	0.5027	1.0000
ReliefF ($t = 0$)	0.0019	0.0022	0.0073	0.1223
Significance (3205)	0.0022	0.0019	0.0019	0.0019
Significance (4117)	0.0022	0.2696	1.0000	1.0000
Significance (5566)	0.0019	0.0031	0.1636	1.0000
Significance (6171)	0.0019	0.0019	0.0019	0.0019
Significance ($t = 0$)	0.0019	0.0153	1.0000	1.0000

Table 20 continued

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
Sum squares ratio (3205)	0.0019	0.0020	0.0062	0.0210
Sum squares ratio (4117)	0.0019	0.0022	0.0067	0.0727
Sum squares ratio (5566)	0.0019	0.0029	0.0338	0.6158
Sum squares ratio (6171)	0.0019	0.0026	0.0267	0.1759
Symmetricaluncert (3205)	0.0020	0.0019	0.0019	0.0019
Symmetricaluncert (4117)	0.0073	1.0000	1.0000	1.0000
Symmetricaluncert (5566)	0.0019	0.0041	1.0000	1.0000
Symmetricaluncert (6171)	0.0019	0.0022	0.0024	0.0130
Symmetricaluncert ($t = 0$)	0.0019	0.0019	0.0019	0.0019

Table 21 Results of Wilcoxon rank sum tests with Bonferroni correction for random forest and the F1 metric

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
SpRST (05)	9.7e−07	–	–	–
SpRST (08)	$<$ 2e−16	9.0e−10	–	–
SpRST (10)	$<$ 2e−16	2.6e−10	1.0000	–
Original (–)	0.0019	0.0019	0.1415	0.9173
Cfs (Greedy)	0.0019	0.0019	0.0019	0.0019
Chi squared (3205)	0.0048	0.0019	0.0019	0.0019
Chi squared (4117)	0.0031	0.8589	1.0000	1.0000
Chi squared (5566)	0.0019	0.0026	1.0000	1.0000
Chi squared (6171)	0.0019	0.0019	0.0228	0.0980
Chi squared ($t = 0$)	0.0019	0.0019	0.0019	0.0019
Consistency (Greedy)	0.0019	0.0019	0.0019	0.0019
Correlation (3205)	0.0022	0.0460	1.0000	1.0000
Correlation (4117)	0.0019	0.0052	1.0000	1.0000
Correlation (5566)	0.0019	0.0024	0.4095	1.0000
Correlation (6171)	0.0019	0.0019	0.2513	1.0000
Correlation ($t = 0$)	0.0019	0.0031	1.0000	1.0000
CV (3205)	0.0019	0.0019	0.0019	0.0019
CV (4117)	0.0019	0.0019	0.0019	0.0019
CV (5566)	0.0019	0.0019	0.0019	0.0019
CV (6171)	0.0019	0.0019	0.0019	0.0019
CV ($t = 0$)	0.0019	0.0019	0.0019	0.0019

Table 21 continued

	SpRST (4)	SpRST (5)	SpRST (8)	SpRST (10)
Gain ratio (3205)	0.0044	0.0019	0.0019	0.0019
Gain ratio (4117)	0.0029	0.6585	1.0000	1.0000
Gain ratio (5566)	0.0019	0.0026	1.0000	1.0000
Gain ratio (6171)	0.0019	0.0022	0.1055	0.2030
Gain ratio ($t = 0$)	0.0019	0.0019	0.0019	0.0019
Info gain (3205)	0.0141	0.0019	0.0019	0.0019
Info gain (4117)	0.0034	1.0000	1.0000	1.0000
Info gain (5566)	0.0019	0.0048	1.0000	1.0000
Info gain (6171)	0.0019	0.0019	0.0041	0.0079
Info gain ($t = 0$)	0.0019	0.0019	0.0019	0.0019
ReliefF (3205)	0.0020	0.0312	1.0000	1.0000
ReliefF (4117)	0.0019	0.0024	1.0000	1.0000
ReliefF (5566)	0.0019	0.0026	1.0000	1.0000
ReliefF (6171)	0.0019	0.0037	1.0000	1.0000
ReliefF ($t = 0$)	0.0019	0.0019	0.0289	0.1415
Significance (3205)	0.0026	0.0019	0.0019	0.0019
Significance (4117)	0.0026	0.1415	1.0000	1.0000
Significance (5566)	0.0019	0.0020	0.3822	1.0000
Significance (6171)	0.0019	0.0019	0.0019	0.0022
Significance ($t = 0$)	0.0019	0.0057	1.0000	1.0000
Sum squares ratio (3205)	0.0019	0.0019	0.0029	0.0041
Sum squares ratio (4117)	0.0019	0.0019	0.0022	0.0037
Sum squares ratio (5566)	0.0019	0.0019	0.0026	0.0073
Sum squares ratio (6171)	0.0019	0.0020	0.0267	0.0783
Symmetricaluncert (3205)	0.0031	0.0019	0.0019	0.0019
Symmetricaluncert (4117)	0.0026	0.9794	1.0000	1.0000
Symmetricaluncert (5566)	0.0019	0.0022	1.0000	1.0000
Symmetricaluncert (6171)	0.0019	0.0019	0.0026	0.0073
Symmetricaluncert ($t = 0$)	0.0019	0.0019	0.0019	0.0019

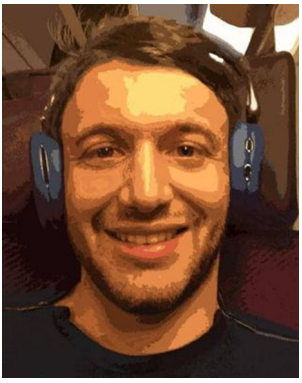
References

1. Afendi FM, Ono N, Nakamura Y, Nakamura K, Darusman LK, Kibinge N, Morita AH, Tanaka K, Horai H, Altaf-Ul-Amin M et al (2013) Data mining methods for omics and knowledge of crude medicinal plants toward big data biology. Comput Struct Biotechnol J 4(5):1–14
2. Aghdam MH, Ghasem-Aghaee N, Basiri ME (2009) Text feature selection using ant colony optimization. Expert Syst Appl 36(3):6843–6853
3. Ahmed S, Zhang M, Peng L (2013) Enhanced feature selection for biomarker discovery in LC-MS data using GP. In: Evolutionary computation (CEC), 2013 IEEE congress on. IEEE, pp 584–591
4. Asuncion A, Newman DJ (2007) UCI machine learning repository. http://www.ics.uci.edu/~mlearn/MLRepository.html
5. Bai C, Sarkis J (2010) Integrating sustainability into supplier selection with grey system and rough set methodologies. Int J Produ Econ 124(1):252–264
6. Bolón-Canedo V, Rego-Fernández D, Peteiro-Barral D, Alonso-Betanzos A, Guijarro-Berdiñas B, Sánchez-Maroño N (2018) On the scalability of feature selection methods on high-dimensional data. Knowl Inf Syst 56(2):395–442
7. Chen M, Mao S, Liu Y (2014) Big data: a survey. Mobile Netw Appl 19(2):171–209
8. Dagdia ZC, Zarges C, Beck G, Lebbah M (2017) A distributed rough set theory based algorithm for an efficient big data pre-processing under the spark framework. In: 2017 IEEE international conference on big data, BigData 2017, Boston, MA, USA, December 11–14, 2017, pp 911–916. https://doi.org/10.1109/BigData.2017.8258008
9. Dash M, Liu H (1997) Feature selection for classification. Intell Data Anal 1(1–4):131–156
10. Dean J, Ghemawat S (2010) MapReduce: a flexible data processing tool. Commun ACM 53(1):72–77
11. Düntsch I, Gediga G (2000) Rough set data analysis. Encycl Comput Sci Technol 43(28):281–301
12. El-Alfy ESM, Alshammari MA (2016) Towards scalable rough set based attribute subset selection for intrusion detection using parallel genetic algorithm in MapReduce. Simul Model Pract Theory 64:18–29
13. Fan W, Bifet A (2013) Mining big data: current status, and forecast to the future. ACM sIGKDD Explor Newsl 14(2):1–5
14. Fernández A, del Río S, López V, Bawakid A, del Jesus MJ, Benítez JM, Herrera F (2014) Big data with cloud computing: an insight on the computing environment, MapReduce, and programming frameworks. Wiley Interdiscip Rev Data Min Knowl Discov 4(5):380–409
15. Ghosh A, Datta A, Ghosh S (2013) Self-adaptive differential evolution for feature selection in hyperspectral image data. Appl Soft Comput 13(4):1969–1977
16. Grzymala-Busse JW, Ziarko W (2000) Data mining and rough set theory. Commun ACM 43(4):108–109
17. Guyon I, Elisseeff A (2003) An introduction to variable and feature selection. J Mach Learn Res 3(Mar):1157–1182
18. https://spark.apache.org/mllib/. Mllib website
19. Hu J, Pedrycz W, Wang G, Wang K (2016) Rough sets in distributed decision information systems. Knowl-Based Syst 94:13–22
20. John GH, Kohavi R, Pfleger K et al (1994) Irrelevant features and the subset selection problem. In: Machine learning: proceedings of the eleventh international conference, pp 121–129
21. Larose DT (2014) Discovering knowledge in data: an introduction to data mining. Wiley, New York
22. Lingras P (2001) Unsupervised rough set classification using GAs. J Intell Inf Syst 16(3):215–228
23. Lingras P (2002) Rough set clustering for web mining. In: Fuzzy systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE international conference on, vol 2. IEEE, pp 1039–1044
24. Liu H, Motoda H, Setiono R, Zhao Z (2010) Feature selection: an ever evolving frontier in data mining. In: Feature selection in data mining, pp 4–13
25. Liu H, Yu L (2005) Toward integrating feature selection algorithms for classification and clustering. IEEE Trans Knowl Data Eng 17(4):491–502
26. Liu H, Zhao Z (2012) Manipulating data and dimension reduction methods: feature selection. In: Computational complexity. Springer, Berlin, pp 1790–1800
27. Pawlak Z (2012) Rough sets: theoretical aspects of reasoning about data, vol 9. Springer, Berlin
28. Pawlak Z, Skowron A (2007) Rudiments of rough sets. Inf Sci 177(1):3–27
29. Peng H, Long F, Ding C (2005) Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Trans Pattern Anal Mach Intell 27(8):1226–1238
30. Peralta D, del Río S, Ramírez-Gallego S, Triguero I, Benitez JM, Herrera F (2015) Evolutionary feature selection for big data classification: a mapreduce approach. Math Probl Eng. https://doi.org/10.1155/2015/246139
31. Qian Y, Liang J, Pedrycz W, Dang C (2010) Positive approximation: an accelerator for attribute reduction in rough set theory. Artif Intell 174(9):597–618

32. Qian Y, Liang X, Wang Q, Liang J, Liu B, Skowron A, Yao Y, Ma J, Dang C (2018) Local rough set: a solution to rough data analysis in big data. Int J Approx Reason 97:38–63
33. Sakr S, Liu A, Batista DM, Alomari M (2011) A survey of large scale data management approaches in cloud environments. IEEE Commun Surv Tutor 13(3):311–336
34. Schäfer P (2016) Scalable time series classification. Data Min Knowl Discov 30(5):1273–1298
35. Schneider J, Vlachos M (2017) Scalable density-based clustering with quality guarantees using random projections. Data Mining Knowl Discov 31(4):972–1005
36. Shanahan JG, Dai L (2015) Large scale distributed data science using apache spark. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 2323–2324
37. Snir M (1998) MPI-the complete reference: the MPI core, vol 1. MIT Press, Cambridge
38. Talukder N, Zaki MJ (2016) A distributed approach for graph mining in massive networks. Data Mini Knowl Discov 30(5):1024–1052
39. Thangavel K, Pethalakshmi A (2009) Dimensionality reduction based on rough set theory: a review. Appl Soft Comput 9(1):1–12
40. Vinh NX, Chan J, Romano S, Bailey J, Leckie C, Ramamohanarao K, Pei J (2016) Discovering outlying aspects in large datasets. Data Min Knowl Discov 30(6):1520–1555
41. Wu X, Zhu X, Wu GQ, Ding W (2014) Data mining with big data. IEEE Trans Knowl Data Eng 26(1):97–107
42. Xu X, Jäger J, Kriegel HP (1999) A fast parallel clustering algorithm for large spatial databases. In: High performance data mining. Springer, pp 263–290
43. Zhai T, Gao Y, Wang H, Cao L (2017) Classification of high-dimensional evolving data streams via a resource-efficient online ensemble. Data Mining Knowl Discov 31(5):1242–1265
44. Zhang J, Wang S, Chen L, Gallinari P (2017) Multiple bayesian discriminant functions for high-dimensional massive data classification. Data Min Knowl Discov 31(2):465–501

Zaineb Chelly Dagdia is a Research and Development Technical Project Manager at National Institute for Research in Computer Science and Automation (Inria). She received her MSc and PhD in Computer Science at the University of Tunis (ISG-Campus), Tunisia, in 2010 and 2014, respectively. After that, she hold the position of a Marie Sklodowska Curie Research Fellow (MSCA-IF-2015-EF, Project No. 702527) at Aberystwyth University, UK (03/2017-02/2019). Her research interests include different aspects of artificial intelligence. She writes on machine learning, data mining and data analytics, evolutionary algorithms and artificial immune systems, big data, and uncertainty theories. She has a strong publication record. She was awarded the Young Researcher First Price (IEEE EHB'2013), the ACM-W Award, the Marie Sklodowska Curie Individual European Fellowship and the Best Reviewer Award (iCDEc 2018). She also acts as a Marie Sklodowska Curie Ambassador, selected as a Female Scientist Role Model, and selected to be among the 200 Heidelberg-Laureate-Forum (HLF) most qualified young researchers.

Christine Zarges is a Lecturer in the Department of Computer Science at Aberystwyth University, Wales, UK. She graduated with a PhD in Computer Science from TU Dortmund, Germany, in 2011. Her current main research interests include machine learning, optimization and heuristic search methods.



Gaël Beck is currently Data Scientist at Kameloon. He received his engineer degree in 2015 and his MS degree in Dauphine University, France, in 2015. After three years in Sorbonne Paris Nord University and LIPN Lab (UMR CNRS 7030), he received his PhD degree in Computer Science. His research interests are in the areas of scalable machine learning and data mining with emphasis on clustering and visualization. Find more information on https://github.com/beckgael.

Mustapha Lebbah is currently Associate Professor at the Sorbonne Paris Nord University and a member of Machine learning Team in LIPN Lab (UMR CNRS 7030). His main researches are centered on machine learning (unsupervised learning, mixture model, cluster analysis, scalable machine learning big data and data science). He graduated from USTO University where he received his engineer diploma in 1998. Thereafter, he gained an MSC (DEA) in Artificial Intelligence from the Sorbonne Paris Nord University in 1999. In 2003, after three years in RENAULT R&D, he received his PhD degree in Computer Science from the University of Versailles. He received the "Habilitation Diriger des Recherches" (accreditation to lead research) degree in Computer Science from Sorbonne Paris Nord University in 2012. Mustapha Lebbah is qualified by the French National Council of Universities in Computer Science (section CNU 27) and Statistic (Section CNU 26). Find more information on https://sites.google.com/site/lebbah/.