Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full length article

Rotation Forest for Big Data

Mario Juez-Gil [*], Álvar Arnaiz-González, Juan J. Rodríguez, Carlos López-Nozal, César García-Osorio

Escuela Politécnica Superior, Universidad de Burgos, 09006 Burgos, Spain

A R T I C L E   I N F O

Keywords:
Rotation Forest
Random Forest
Ensemble learning
Machine learning
Big Data
Spark

A B S T R A C T

The Rotation Forest classifier is a successful ensemble method for a wide variety of data mining applications. However, the way in which Rotation Forest transforms the feature space through PCA, although powerful, penalizes training and prediction times, making it unfeasible for Big Data. In this paper, a MapReduce Rotation Forest and its implementation under the Spark framework are presented. The proposed MapReduce Rotation Forest behaves in the same way as the standard Rotation Forest, training the base classifiers on a rotated space, but using a functional implementation of the rotation that enables its execution in Big Data frameworks. Experimental results are obtained using different cloud-based cluster configurations. Bayesian tests are used to validate the method against two ensembles for Big Data: Random Forest and PCARDE classifiers. Our proposal incorporates the parallelization of both the PCA calculation and the tree training, providing a scalable solution that retains the performance of the original Rotation Forest and achieves a competitive execution time (in average, at training, more than 3 times faster than other PCA-based alternatives). In addition, extensive experimentation shows that by setting some parameters of the classifier (i.e., bootstrap sample size, number of trees, and number of rotations), the execution time is reduced with no significant loss of performance using a small ensemble.

1. Introduction

We are living in the era of Big Data where the growing sizes of some data sets have never been seen before [1,2]. The way our world works is drastically changed by the influence of Big Data. Science, engineering, business, finances, sports, and healthcare are some fields where an important role is now played by Big Data processing techniques. Hence, the consolidation of Big Data as a research topic, that has been gaining special attention in academia, and as an essential process both for government and for industry.

From a theoretical point of view, we understand Big Data to mean high volumes of information that are processed at high velocity and in a wide variety of formats, most widely known as the three V's of Big Data [3]. Thus, designers of computer systems and data mining algorithms have therefore had to respond to the challenge [4] through innovative high-performance processing solutions, such as computer clusters.

While proprietary cluster environments could be built, in order to arrive at the large-scale and complex computing requirements that are needed for processing these data sets, it is now claimed that the most powerful architecture to address Big Data is cloud computing. Cloud computing provides flexible and scalable resources on-demand that have intensive processing capacities to conduct data analysis tasks [5]. The flexibility and versatility of cloud computing produce economic solutions that rival other cluster architectures, which may even become obsolete within a short period of time.

In the past few years, many frameworks, programming models, and algorithms have emerged to deal with large data sets. In most cases, the main focus of these solutions is the exploitation of the parallelization opportunities provided by multi-core processors and cluster architectures. Unfortunately, the use of these solutions makes it necessary to redesign the algorithms to allow their execution in Big Data frameworks. During the last decade two main Big Data frameworks have gained notoriety: Hadoop and Spark; both base their performance in the MapReduce programming model [6].

Likewise, ensemble methods have demonstrated their remarkable performance over the past few decades [7]. Their simplicity and flexibility make them useful in several domains. Moreover, their modularity (i.e., they are based on individual base classifiers or regressors) makes

* Corresponding author.
E-mail addresses: mariojg@ubu.es (M. Juez-Gil), alvarag@ubu.es (Á. Arnaiz-González), jjrodriguez@ubu.es (J.J. Rodríguez), clopezno@ubu.es (C. López-Nozal), cgosorio@ubu.es (C. García-Osorio).

https://doi.org/10.1016/j.inffus.2021.03.007
Received 10 June 2020; Received in revised form 28 January 2021; Accepted 21 March 2021
Available online 27 March 2021

their parallelization feasible. Unfortunately, the number of ensemble algorithms available on Big Data frameworks is still limited. The aim of this paper is to present the MapReduce design and implementation of the well-known Rotation Forest ensemble [8,9] and its evaluation. To assess whether the Rotation Forest ensemble maintains its good performance in Big Data, it was thoroughly compared against the few ensemble algorithms available for Big Data within a Spark cluster by using several large data sets with a number of instances ranging between 400 000 and 11 000 000, and a number of attributes ranging between 11 and 2 000.

Despite the remarkable performance of Rotation Forest, its main drawback is that it is a time-consuming algorithm. A Rotation Forest classifier needs to perform multiple PCA calculations as one of its steps, and rotate both the training and the testing data, making it slower than other ensemble methods. Nonetheless, an efficient design and implementation in Big Data frameworks can make it suitable for the new challenges posed by the need to process large data sets.

In the study presented in this paper, the original Rotation Forest algorithm was adapted to Big Data using the parallelization approach provided by Spark (i.e., using MapReduce). The parallelization was performed at critical points of the code (i.e., those with the largest execution times). Spark's parallel implementation of PCA and Random Forest were used, and the parallelization was also applied to the reordering of rotation matrices and to matrix multiplications, among other steps within the algorithm, making the proposal highly scalable.

The rest of the paper will be organized as follows. Works related to Big Data will be summarized in Section 2. In Section 3, the background to Random Forest and Rotation Forest ensembles, and MapReduce will be presented. In Section 4, the MapReduce implementation of the Rotation Forest algorithm will be explained. The experimentation will be presented in Section 5, and the conclusions and future research lines, in Section 6.

2. Related works

Since Google laid the foundations of MapReduce programming model [6], several frameworks have emerged for Big Data, such as Hadoop [10] and Spark [11]. As it is well known, Spark makes intensive use of memory rather than disk, which means it is faster and more convenient for data processing [12]; an advantage that has made it increasingly popular for parallel computation. This section presents a brief review of algorithms and libraries for both Hadoop and Spark.

Spark MLlib is the Spark's Machine Learning library [13]. It offers several algorithms for a broad variety of tasks, including classification, regression, and clustering, among others [14]. Nevertheless, the number of algorithms is still scarce in comparison with other Machine Learning tools such as Weka [15] and Scikit-learn [16].

Lazy learners, such as k-NN (k-Nearest Neighbors) [17,18], are successful methods that have demonstrated their value in several domains. Unfortunately, their high memory requirements, consequence of storing the entire data set, rather than calculating a model, makes their MapReduce implementation difficult. Despite this, there are recent Spark implementation proposals for both batch learning [19] and online learning [20].

Ensemble methods rely on the fact that a bunch of base learners (regressors or classifiers) are more likely to make proper predictions than the base methods alone [7]. One of the benefits of ensembles is that their construction can be easily parallelized. At the moment, one active research line within the area of Big Data focuses on adapting ensemble algorithms to the MapReduce paradigm. Albeit scarce, some implementations have already been proposed in recent years: In 2011, Tyree et al. proposed the pGBRT algorithm [21], a MapReduce version of the Gradient Boosting Regression Trees ensemble. Later, in 2016 Chen et al. published a parallel Random Forest for the Apache Spark framework [22]. In 2017 a set of ensemble implementations for multi-label learning on Apache Spark was proposed by Gonzalez-Lopez et al. [23]. PCARDE, presented by García-Gil et al. [24], is another Big Data adaptation of an ensemble algorithm for Apache Spark; it is based on Principal Components Analysis and Random Discretization.

Another popular family of supervised classification models is Bayesian Networks [25]. The most expensive task of these methods is the computation of multidimensional contingency tables, which requires the estimation of probability distributions [26]. A MapReduce version, implemented in Spark, for discrete Bayesian network classifiers was recently presented by Arias et al. [26].

It is not only classifiers and regressors that are used in data mining, but also data preprocessing methods, which constitute a very important stage in the knowledge discovery process [27,28]. This is what explains why many preprocessing techniques have also recently been adapted to Big Data frameworks [3]. There are several tasks related to data preprocessing, such as discretizers [28], noise filtering [29], feature selection [30], and instance selection [31–33], among others.

3. Background

This section presents the background of the paper, focusing on Random and Rotation Forest ensemble algorithms and the MapReduce runtime environment.

3.1. Random forest

Ensemble learning relies on the idea of generating several models (called base classifiers, in classification problems) instead of only one, looking for a combination of models that outperforms the individual performance of the models that are combined. The key to the success of this process is to achieve ensembles that, without damaging the average performance of the models, make them diverse (in the sense that their predictions are different from each other).

The Random Forest algorithm, proposed by Breiman [34] in 2001, is still considered a state-of-the-art classifier [35]. Its simply, yet effective, underlying idea is to produce a bootstrap sample (as bagging [36] does) and it then randomly selects features at each node of the tree being constructed in order to increase the diversity of the ensemble. That is, instead of searching for the best feature (and threshold value for dividing the data), it only considers a random subset of all the features. Its simplicity and effectiveness have made it an extremely popular ensemble method.

3.2. Rotation forest

The Rotation Forest ensemble algorithm was presented for classification in [8]. It relies on the inherent instability of the tree construction algorithms when the input space is rotated [7]. Data rotation, which is in fact the cornerstone of Rotation Forest, is performed internally, prior to training the base classifiers. Rather than an arbitrary rotation, the rotation is operated using Principal Component Analysis (PCA). Base classifiers (commonly trees) can therefore divide the decision space both parallel to the feature axes and in other directions after the rotation. This feature makes it much more powerful than other traditional ensemble techniques, especially when all the attributes have real values [37].

Since the standard Rotation Forest was presented, several variants have been proposed: Rotation Forests for regression [38], and Boosting of Rotation Forests [39], among others.

3.3. Mapreduce

The MapReduce runtime environment [6] has become the most widely used paradigm in Big Data scenarios nowadays [40,41]. Since the release of Hadoop in 2006, the first open source implementation of MapReduce, some drawbacks have been identified. The main shortcoming of Hadoop is its disk usage to ensure cluster fault tolerance, making it slow, especially for iterative processes. Spark was therefore developed in 2010 in an attempt to use memory intensively and to minimize disk access.

Both, Hadoop and Spark use the MapReduce programming model. To take advantage of MapReduce, an algorithm must be redesigned so that it consists of two stages: Map and Reduce. The map phase divides the data into several parts and applies a first processing step to each of them, while the reduction phase is responsible for applying a second processing step in which the data from the first phase is collected and integrated to obtain the final result. A typical example is the task of counting words in a document. The sequential version processes the document word by word while storing the words and their number of occurrences in a dictionary or hash table. On the contrary, in the MapReduce version of the algorithm, after dividing the document into several parts, the map step transforms every single word into a tuple of `(word, 1)`; then, the reduce step groups all the tuples with the same key (i.e., the word) and adds the values (the "ones"); and, finally, the process is repeated in a final reduction step that calculates the global word count for the entire document. The result is a collection of tuples with the word as the key and its number of occurrences as the value.

Therefore, the implementation of a sequential algorithm into a parallel (MapReduce) environment requires the redesign of the algorithm itself, dividing its internal processing into mapping and reducing phases. This task is not always trivial and can in fact be a paramount challenge in some cases [27].

4. Rotation forest for Big Data

This section presents the parallel implementation of the Rotation Forest classifier [8] for Big Data. Our proposal maintains the main structure of the standard Rotation Forest, but adds some changes to face the problems that Big Data induces. The main difference is the paradigm shift (from sequential to functional) and the use of Random Forest as the base classifier instead of a single decision tree. This last idea was presented in [42] to show that rotation of Random Forests outperform most widely used ensembles. Here, we have found that this is also a very promising approach to optimize training times when using large data sets. An advantage of using Random Forest is the versatility that it offers: it can be parameterized to behave as a single decision tree, as a Bagging of decision trees, or as Bagging of random trees (i.e., Random Forest) among others.

Training a Rotation Forest is a very time-consuming task, which is its main drawback, at least in the context of Big Data. As noted, this is mainly because the PCA calculation requires more computing resources, which is greater as the number of instances and features increase. In Rotation Forest, data are transformed through a sparse rotation matrix computed by arranging K PCA rotation matrices (each of them calculated for K random feature subsets). Additionally, since Rotation Forest is an ensemble, as many sparse rotation matrices as the size of the ensemble will be computed (i.e., for an ensemble of size L, PCA is computed $L \times K$ times).

Nonetheless, this can be done in an efficient parallel way, as PCA can be solved using matrix algebra (i.e., singular value decomposition or covariance matrix calculations followed by an eigenvalue decomposition). In the Spark framework, PCA is already implemented using a parallel singular value decomposition (SVD) algorithm. Decision tree-based algorithms, such as Random Forest, can also be parallelized in many ways. Spark provides its own parallel Random Forest implementation, whose optimizations are based on the PLANET project [43].

Both PCA and Random Forest parallel implementations provided by Spark are used, in order to take advantage of Rotation Forest for Big Data processing.

The training stage of Rotation Forest is presented in Algorithm 1. The algorithm rotates the input data $\mathbf{X}$ and then trains a Random Forest of size T using the rotated data. The process is performed L times, in order to build an ensemble that is composed of L rotation matrices and L Random Forests. Hence, the total number of trees in the ensemble is $L \times T$.

Algorithm 1: Rotation Forest for Big Data (Training stage).

```
Input: A training set $(\mathbf{X}, \mathbf{Y})$ where $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ defined in a
       feature set $\mathbf{F}$, $\mathbf{Y} = \{y_1, \ldots, y_n\}$ with labels
       $y_i \in \Omega = \{\omega_1, \ldots, \omega_c\}$ representing $c$ classes, number of
       rotations $L$, number of trees $T$, number of feature
       subsets $K$, bootstrap size $B$.
Output: Trained ensemble $E$ (tuples: rotation matrix $\mathbf{R}^a$, base
        classifier $D$).

1  $E \leftarrow$ map $i \in \{1, \ldots, L\}$
2      $\mathbf{Q} \leftarrow$ random partition of $\mathbf{F}$ into $K$ subsets of features
3      $\mathbf{M} \leftarrow$ map $\mathbf{S} \in \mathbf{Q}$
4          $\mathbf{W} \leftarrow$ submatrix of $\mathbf{X}$ with the columns corresponding to
           the features in $\mathbf{S}$
5          $\mathbf{Y}' \leftarrow$ random selection of classes in $\Omega$
6          $\mathbf{W}' \leftarrow$ submatrix of $\mathbf{W}$ with rows corresponding to
           instances of classes in $\mathbf{Y}'$
7          $\mathbf{W}'' \leftarrow$ bootstrap sample of size $B$% of the number of
           instances in $\mathbf{W}'$
8          $\mathbf{C} \leftarrow$ rotation matrix from parallel-PCA($\mathbf{W}''$)
9          emit $\langle \mathbf{C} \rangle$
10     $\mathbf{R} \leftarrow$ reduce ($\mathbf{M}$)          // block diagonal matrix
11     $\mathbf{P} \leftarrow$ permutation matrix, matching the order of the features
       in $\mathbf{F}$
12     $\mathbf{R}^a \leftarrow \mathbf{P}\mathbf{R}$      // rearrangement of $\mathbf{R}$ (in parallel)
13     $D \leftarrow$ train-parallel-random-forest($\mathbf{X}\mathbf{R}^a, \mathbf{Y}, T$)
14     emit $\langle \mathbf{R}^a, D \rangle$
15 return $E$
```

The construction of the rotation matrix $\mathbf{R}^a$ is performed in lines 3 to 12. Initially, the feature set, $\mathbf{F}$, of the input data, $\mathbf{X}$, is randomly split into a partition, $\mathbf{Q}$, of K subsets (K is a parameter of the algorithm). For each feature subset, $\mathbf{S}$, of $\mathbf{Q}$, a submatrix, $\mathbf{W}$, is extracted from $\mathbf{X}$ that only contains the features in $\mathbf{S}$ (that is, $\mathbf{W}$ only contains a subset of columns of $\mathbf{X}$). The matrix, $\mathbf{W}$, is now further reduced by removing some of its rows, more specifically a random selection of classes is made and all instances not belonging to the selected classes are removed. The result is a new matrix, $\mathbf{W}'$. An additional reduction step generates the matrix, $\mathbf{W}''$, by retaining a bootstrap sample of a percentage, B, of the instances in $\mathbf{W}'$. PCA is applied to this last matrix, $\mathbf{W}''$, to obtain a rotation matrix, $\mathbf{C}$.

In the reduction phase, all the resulting PCA rotation matrices, K, are arranged into a block diagonal matrix, $\mathbf{R}$, although it cannot yet be used to rotate the original input data, $\mathbf{X}$, because the order of its columns does not match the order of the corresponding features in the original data. In consequence, the columns of $\mathbf{R}$ have to be rearranged to match the original order of features using a permutation matrix, $\mathbf{P}$, and a MapReduce implementation of matrix multiplication.

Finally, a Random Forest classifier is trained using the data obtained by rotating the input data, $\mathbf{X}$, using the reorganized matrix, $\mathbf{R}^a$ ($\mathbf{X}\mathbf{R}^a$). Fig. 1 illustrates the process of calculating the rotation matrix $\mathbf{R}^a$ using as an example a data set $\mathbf{X}$ with 12 instances, 6 features, and 3 classes $\{0, 1, 2\}$; and algorithm parameters, K and B, equal to 3 and to 50%, respectively.

Algorithm 2 shows the steps for the Rotation Forest prediction stage. A novelty with regard to the standard Rotation Forest, is that the prediction is not performed for a single instance at a time but for a set of instances in parallel, which is more convenient in Big Data. An ensemble of size L has L rotation matrices and L trained base

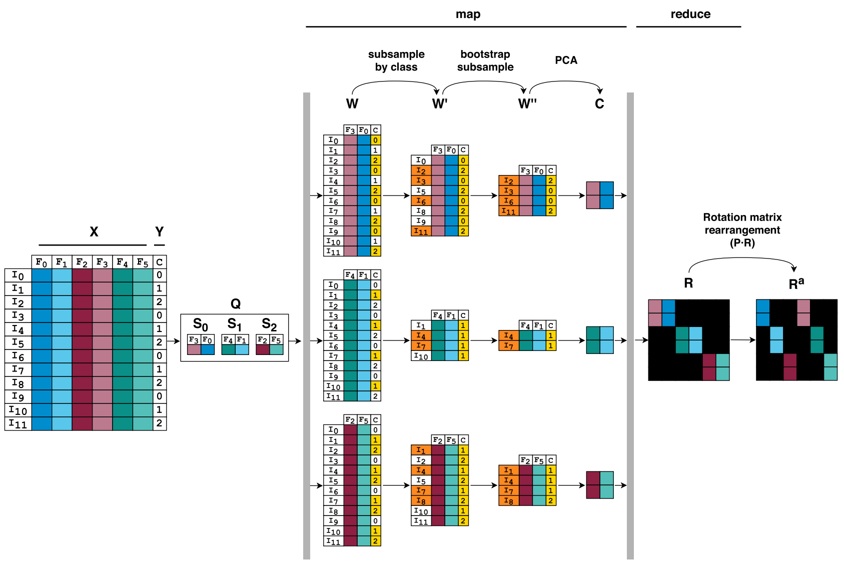


Fig. 1. MapReduce implementation of the process to generate the rotation matrix ($\mathbf{R}^a$).

classifiers (i.e., L Random Forest). Firstly, each rotation matrix, $\mathbf{R}^a$, and its corresponding base classifier, D, are used in the map phase (see Line 1), where the set of instances, $\mathbf{X}$, is rotated through a parallel matrix multiplication ($\mathbf{X}\,\mathbf{R}^a$), and then used as input to a base classifier D. Its output, $\mathbf{P}$, will consist of the predicted probabilities of each instance in the set for each class (i.e., the dimension of the matrix is $n{\times}c$, where c is the number of classes and n the number of instances for which the prediction is being made). All predictions (i.e., base classifier probabilities) are arranged in a three-dimensional matrix, $\mathbf{S}$, of size $L{\times}n{\times}c$. Then, the probabilities of each base classifier for each class for the instances are added up and averaged in the reduce phase, thus the $\mathbf{T}$ matrix is of size $n{\times}c$. Finally, the last map (see Line 6) computes the final prediction of the whole Rotation Forest ensemble. The predicted class for each instance (ω_j) will be the class with the highest probability in $\mathbf{u}$, and an array, $\mathbf{y}$, containing the predicted classes of the instances, will be the final output of the ensemble.

Algorithm 2: Rotation Forest for Big Data (Prediction stage).

```
Input: A set $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ defined in a feature set $\mathbf{F}$, the
       ensemble $E$ (tuples: rotation matrix $\mathbf{R}^a$ and base classifier
       $D$).
Output: Predicted classes $\mathbf{y} = \{y_1, \dots, y_n\}$ with labels
        $y_i \in \Omega = \{\omega_1, \dots, \omega_c\}$ representing $c$ classes
1 S ← map $(\mathbf{R}^a, D) \in E$
2     $\mathbf{X}' \leftarrow \mathbf{X}\,\mathbf{R}^a$          // parallel rotation of input set
3     $\mathbf{P} \leftarrow D(\mathbf{X}')$                   // parallel prediction
4     emit $\langle \mathbf{P} \rangle$
5 T ← reduce (S)              // average the probabilities
6 y ← map $\mathbf{u} \in \mathbf{T}$
7     $j \leftarrow \operatorname{argmax}_{i\in\{1,\dots,c\}} u_i$
8     emit $\langle \omega_j \rangle$
9 return y
```

5. Experimental results

The aim in this section is to assess whether the MapReduce version of Rotation Forest, presented in this paper, is suitable for Big Data processing. A thorough experimentation was performed taking into account both accuracy and execution time, using several representative

Table 1
Experimental data sets.

Dataset	Instances	Attributes	Classes	Size (GB)
poker-hand	1 025 010	11	10	0.02
covtype	581 012	54	7	0.07
susy	5 000 000	18	2	2.33
higgs	11 000 000	28	2	7.84
hepmass	10 500 000	28	2	7.58
epsilon	400 000	2000	2	11.87

data sets. Since the proposed Rotation Forest[1] is a tree-based ensemble classifier, it was compared to the official Spark implementation of Random Forest [34]. It was also compared to the Principal Components Analysis Random Discretization Ensemble (PCARDE) [24], because it shares the idea of using PCA with Rotation Forest as part of its data transformation step.

5.1. Experimental framework

Six popular classification data sets for Big Data were used for conducting the experiments.[2] Table 1 summarizes the data sets. All features of the data sets were normalized. As the features of Rotation Forest need to be numeric, nominal features were binarized using one-hot encoding.

The experimental setup of Rotation Forest was as follows. The number of trees, T, was set to 1, so the ensemble size was the same as the number of rotations, L.[3] Following [8], we used three different ensemble sizes, to represent small (10 trees), medium (50 trees), and large ensembles (100 trees). The bootstrap value was set at 25% and 4 was selected as the number of feature subsets K. As stated before, the Rotation Forest classifier proposed in this study uses Random Forest

[1] The Rotation Forest classifier implementation for Spark is publicly available at https://github.com/mjuez/rotation-forest-spark.

[2] The poker-hand, covtype, susy, higgs, and hepmass data sets are available at the UCI Machine Learning repository [44] https://archive.ics.uci.edu/ml/index.php. The epsilon data set is available at the LIBSVM data repository https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#epsilon.

[3] This is what the standard Rotation Forest does, in which there is a different rotation for each tree, later we will experiment with other approach where the same rotation is shared among several trees.

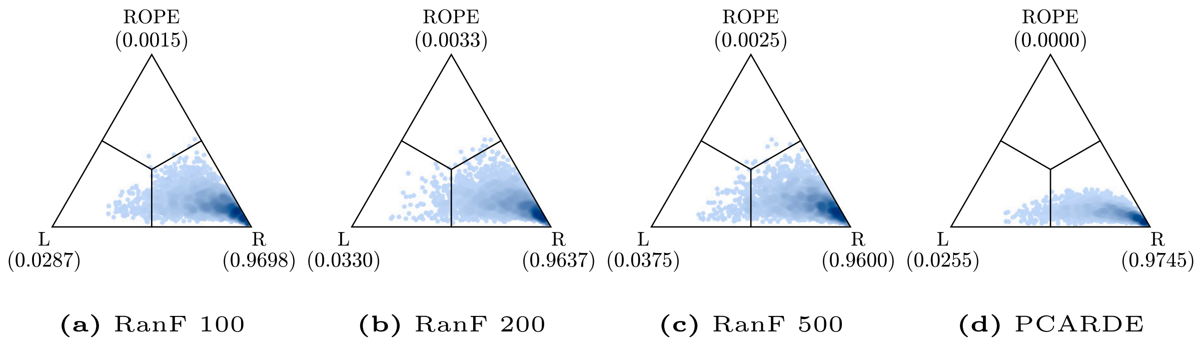


Fig. 2. Hierarchical Bayesian test heatmap for Random Forest and PCARDE against Rotation Forest with 10 trees.

as its base classifier. The Spark Random Forest was parameterized in such a way that it behaves as a decision tree (as the standard Rotation Forest does). It was achieved by setting featureSubsetStrategy to "all" (i.e., all features are used for training). For the rest of the parameters, their default values were used.

Regarding the experimental setup of Random Forest and PCARDE, the default values were used for all the parameters except for the number of trees, which was set to 100 for small, to 200 for medium, and to 500 for large ensembles. In the case of PCARDE, only a single ensemble size with 10 trees was used because this was the setting recommended by the authors [24] (they reported equivalent results for small, medium, and large ensembles).

Given that the values of the parameters were not changed for each individual data set, the results obtained were not entirely optimal. If the parameters had been individually tuned for each data set, then the results could have been improved. It would however have made the experimentation unnecessarily time-consuming. Nevertheless, the same restriction was used with all the methods, so the comparison was fair.

The experiments were carried out using 5-fold cross-validation. Ensemble learning requires randomization to train different and diverse base classifiers. For this reason, with the aim of enabling experimental repeatability, a random seed value was fixed at 46.

Accuracy measure (defined in Eq. (1)) was used for evaluating classification performance.

$$accuracy = \frac{\text{correctly classified instances}}{\text{total number of instances}} \tag{1}$$

Bayesian analysis [45] was used to compare the ensemble classifiers (the library to perform the analysis was *baycomp*[4]). The number of samples for all Bayesian comparisons was set to 50 000. The value for the region of practical equivalence (ROPE) was set to 0.01, which means that two algorithms with a difference in accuracy of less than 1% will be considered equivalent. Bayesian comparison results were represented through ternary plots (e.g., Fig. 2) where the space is divided into three areas of interest: L (opponent wins), ROPE (both tie), and R (Rotation Forest wins).

The experimentation was performed in cloud-based clusters provided by the Google Cloud Platform. A cluster was composed of one master node and seven computing/worker nodes. All nodes were of the n1-standard-16 type, which had 16 virtual CPUs, and 60 GB of RAM. Hence, the cluster size was 128 vCPUs and 480 GB of RAM. At that point in time, the vCPUs of n1-standard nodes could be of one of the following types: Intel Xeon (Skylake), Intel Xeon E5 (Sandy Bridge), Intel Xeon E5 v2 (Ivy Bridge), Intel Xeon E5 v3 (Haswell), or Intel Xeon E5 v4 (Broadwell E5). We used Google Dataproc software, version 1.4, running on Debian 9, with Apache Hadoop 2.9.2, and Apache Spark 2.4.5. Google Cloud Storage was used as a distributed file system for storing data sets and experimental results.

[4] The baycomp library is publicly available at https://baycomp.readthedocs.io/en/latest/.

5.2. Accuracy performance

Table 2 gathers the accuracy results of the three ensemble methods: Random Forest (RanF), PCARDE, and Rotation Forest (RotF). The best results are highlighted in bold. As can be seen, for all data sets, the best results were achieved by Rotation Forest. The table also shows that the size of the ensemble for Random Forest has very little influence on the results. On the other hand, this influence appears to be somewhat greater in the case of the Rotation Forest. Although at first glance medium-size Rotation Forests appear to be a better option than small-size Rotation Forests, Section 5.4 will show that, from the point of view of Bayesian statistical analysis, Rotation Forest performs equivalently for all sizes.

Fig. 2 shows the hierarchical Bayesian test in heatmap representations for Rotation Forest with 10 trees against Random Forest (Fig. 2.a–c) and PCARDE (Fig. 2.d). The R area in the triangles corresponds to Rotation Forest while the L area corresponds to the opponent (i.e., PCARDE or Random Forest). The high density of points (each point corresponds to one Bayesian simulation) concentrated in the right corners means that Rotation Forest clearly outperformed the opponents (i.e., R probability close to 1).

In the previous comparison, cross-validation accuracy results for all data sets were used to get a general overview of Rotation Forest performance compared to Random Forest and PCARDE. Cross-validation accuracy results for a single data set could also be analyzed through the Bayesian correlated *t* test [46]. Fig. 3 shows the density plots comparing Rotation Forest with 10 trees against PCARDE and Random Forest for each data set. As in ternary plots, there are three areas of interest: the region to the left of the leftmost line, which corresponds to the statistical primacy of the opponent; the region between the two lines, which corresponds to the ROPE (i.e., statistical equivalence between two classifiers); and the region to the right of the rightmost line, which corresponds to the statistical primacy of Rotation Forest. This test shows that, for five out of six data sets, Rotation Forest performed better than PCARDE. Only for epsilon was PCARDE better (blue density curve). In the comparison with Random Forest (gray, red, and green density curves), Rotation Forest performed better for four data sets, while Rotation and Random Forest performed equivalently for susy and higgs.

Table 3 gives another perspective on this information, showing the detailed probabilities for the three areas of interest of the Bayesian correlated *t* test explained earlier. The region (Left, Right, or ROPE) with the highest probability, is highlighted in bold. Most of the time, the highest probability was over 98%, showing that the best classifier far outperformed its contender. In most cases, this clear winner was Rotation Forest.

5.3. Execution time analysis

The evaluation in the previous section was conducted in terms of accuracy. Although the strengths of Rotation Forest were demonstrated

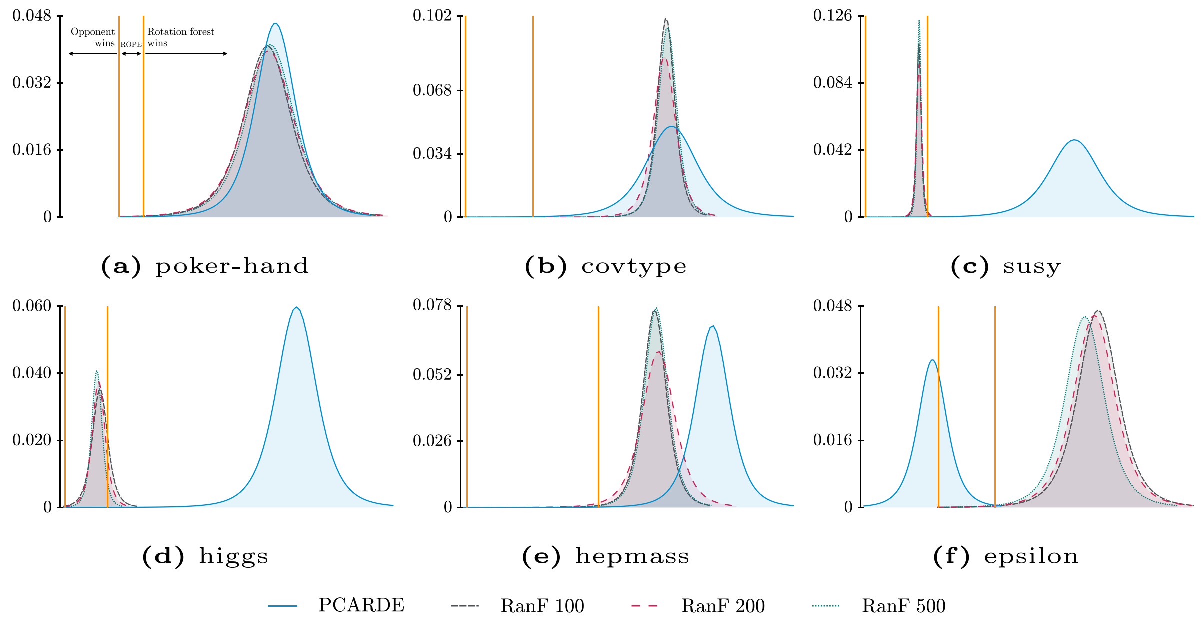


Fig. 3. Bayesian correlated *t* test density plots. It compares Random Forest and PCARDE against Rotation Forest with 10 trees.

Table 2
Experimental results in terms of classification accuracy for Random Forest, PCARDE, and Rotation Forest. Best results are highlighted in bold. The value at the right of each accuracy corresponds to the standard deviation.

Dataset	Random forest			PCARDE	Rotation forest		
	100	200	500	10	10	50	100
poker-hand	51.42 ±0.40	51.32 ±0.30	51.18 ±0.31	50.77 ±1.36	**62.46** ±3.20	58.22 ±1.12	57.26 ±1.16
covtype	67.17 ±0.26	67.22 ±0.29	67.13 ±0.22	67.01 ±0.85	72.08 ±0.38	**72.33** ±0.16	72.28 ±0.14
susy	77.67 ±0.05	77.67 ±0.04	77.66 ±0.01	72.64 ±1.34	78.40 ±0.09	**78.59** ±0.06	78.59 ±0.06
higgs	67.52 ±0.31	67.58 ±0.28	67.67 ±0.11	58.33 ±1.66	68.16 ±0.33	68.70 ±0.10	**68.80** ±0.10
hepmass	82.21 ±0.08	82.15 ±0.12	82.19 ±0.07	81.33 ±0.33	84.06 ±0.32	**84.44** ±0.12	84.42 ±0.11
epsilon	72.56 ±0.31	72.69 ±0.33	73.02 ±0.35	78.40 ±0.13	77.19 ±0.91	80.12 ±0.44	**80.33** ±0.29

Table 3
Bayesian correlated *t* test probabilities for Random Forest and PCARDE against Rotation Forest with 10 trees. The right probability is the probability of Rotation Forest performing better than the opponent. The region (Left, Right, or ROPE) with the highest probability is highlighted in bold.

Dataset	Method	Ensemble size	Left prob.	ROPE	Right prob.
poker-hand	RanF	100	0.18%	0.17%	**99.65%**
		200	0.22%	0.20%	**99.58%**
		500	0.17%	0.16%	**99.67%**
	PCARDE	10	0.08%	0.08%	**99.84%**
covtype	RanF	100	0.00%	0.01%	**99.99%**
		200	0.00%	0.02%	**99.98%**
		500	0.00%	0.01%	**99.99%**
	PCARDE	10	0.07%	0.25%	**99.68%**
susy	RanF	100	0.00%	**98.94%**	1.06%
		200	0.00%	**98.18%**	1.82%
		500	0.00%	**99.26%**	0.74%
	PCARDE	10	0.07%	0.17%	**99.76%**
higgs	RanF	100	0.59%	**79.76%**	19.65%
		200	0.37%	**87.13%**	12.50%
		500	0.21%	**93.82%**	5.97%
	PCARDE	10	0.02%	0.02%	**99.96%**
hepmass	RanF	100	0.00%	0.49%	**99.51%**
		200	0.02%	1.21%	**98.77%**
		500	0.00%	0.44%	**99.56%**
	PCARDE	10	0.01%	0.14%	**99.85%**
epsilon	RanF	100	0.08%	0.31%	**99.61%**
		200	0.09%	0.38%	**99.53%**
		500	0.10%	0.46%	**99.44%**
	PCARDE	10	**64.37%**	34.95%	0.68%

in [8] for small and medium-sized data sets, this paper reinforces that conclusion by working with large data sets and using modern Bayesian statistical tests. Nevertheless, the main contribution of this research is the adaptation of Rotation Forest to work in Big Data environments where execution time is crucial. In this regard, an evaluation and comparison is presented in terms of both execution time and speedup.

Fig. 4 shows a comparison of execution times for Rotation Forest with 10 trees (blue bar), PCARDE (green bar), and Random Forest (burgundy bars). The figures on the top row refer to training times (in seconds), while those on the bottom row refer to prediction times (in milliseconds).

Regarding training time, Random Forest with 100 trees was the fastest on six data sets, as expected. If we compare the two methods that use PCA as part of their data transformation step (Rotation Forest and PCARDE, both with 10 trees), the fastest method was clearly Rotation Forest.

Looking close at prediction times, it is somewhat surprising that the Rotation Forest classifier with 10 trees was the fastest on four out of six data sets. However, this is because the size of Rotation Forest was ten times smaller than Random Forest, and PCA was not computed at prediction (i.e., only a matrix multiplication for rotating data was performed). Regarding the comparison of Rotation Forest against PCARDE, as we might expect, prediction was also faster with Rotation Forest.

For the epsilon data set, the difference between Random Forest against the other two is dramatic. The explanation of this may be found in the number of features. While for the other five data sets, the numbers of features range between 11 and 54, epsilon data set has 2 000. PCA calculation and data rotation require much more time as the number of features (i.e., data set dimensionality) grows, and thus,

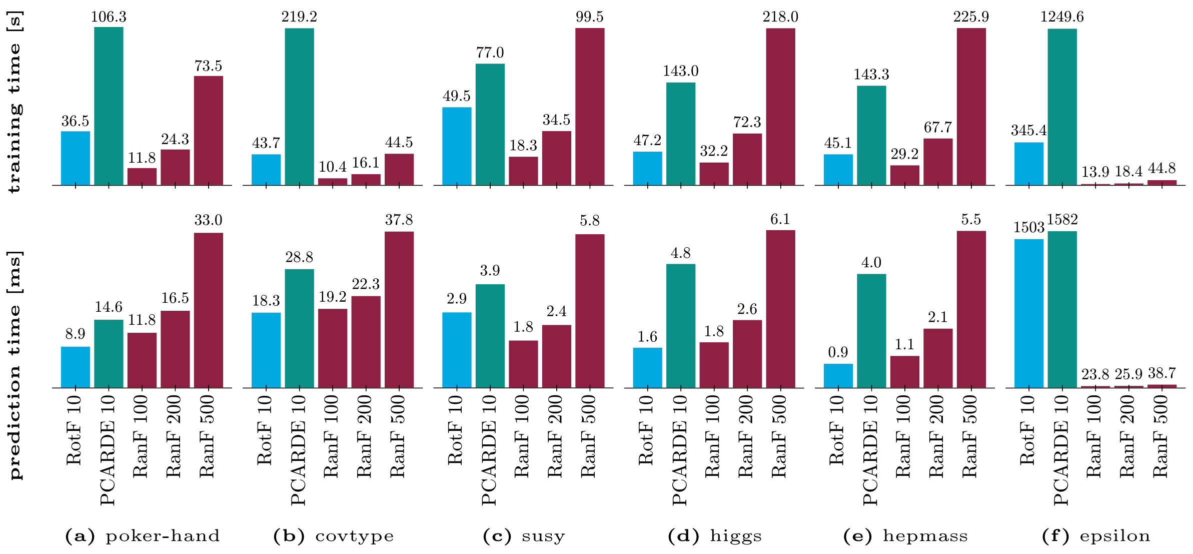


Fig. 4. Training times (top row) and prediction times (bottom row) for each data set. Blue bar corresponds to Rotation Forest, green bar corresponds to PCARDE, and burgundy bars correspond to Random Forest. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Rotation Forest and PCARDE execution times are highly penalized. Nevertheless, we consider that the execution time for Rotation Forest with the epsilon data set was acceptable for Big Data scenarios.

Having shown the competitiveness of Rotation Forest execution times, it is of great importance to determine how well it scales (i.e., how runtime decreases as the number of machines increases). In order to evaluate this, a cluster with one master node and a number of worker nodes ranging from 2 to 24 was used. Each node was of the n1-highmem-2 type, which had 2 virtual cores of the same type as the nodes described in Section 5.1. The following comparison of execution time and speedup, was performed on the medium-size susy data set.

Fig. 5 shows the evolution of training time (left) and prediction time (right). As was expected, the execution time decreased as the number of workers increased. The improvement begins to be less remarkable around the fourteenth or the sixteenth worker, moreover, with the addition of the eighteenth worker, the time actually increases. This behavior is expected because every algorithm has a certain execution time that cannot be optimized, our experiments with the susy data set are a good example of that. This is because the data set is not extremely large, thus, when the number of workers increases above 20, the time required for data communication and transfer was greater than the benefit of increasing the computational power (i.e., additional workers).

Speedup (see Eq. (2)) is conventionally used as a metric for measuring algorithm scalability in a cluster.

$$speedup = \frac{\text{sequential execution time}}{\text{parallel execution time}} \tag{2}$$

For computing the sequential (not parallel) execution time, a machine with one n1-highmem-2 node, and one Spark partition was used.

Fig. 6 shows the speedup evolution of training and prediction. As seen in Fig. 5, the improvement of using more nodes both in training and testing is clear up to 16 nodes. Thereafter, the improvement was minimal, and using more than 20 workers decreased the speedup. It is worth noting that the speedup is higher in prediction than in training, because PCA computation for the K feature subgroups was not fully parallel in training. However, in the prediction phase, the input data rotation was a fully parallel MapReduce matrix multiplication.

5.4. Study of ensemble size

When Rotation Forest classifiers are trained, PCA is the most time-consuming step. Therefore, the main optimization will be to reduce the number of PCAs that have to be calculated. For this purpose, three parameters of the algorithm can be adjusted: L, the number of rotations; T, the number of trees; and K, the number of feature subsets. In this section, we will focus only on the first two, because they determine the ensemble size.

Up until this point, all the experiments had been performed with $T = 1$, which meant that the ensemble size was equal to L (i.e. $L\times 1$ as in the standard Rotation Forest). Fig. 7 shows a hierarchical Bayesian test for statistical comparison of the three ensemble sizes that have so far been used. It can be roughly concluded that small (10×1), medium (50×1), and large (100×1) ensembles achieved equivalent levels of accuracy. Specifically, Fig. 7.a shows the similar performance levels of small and medium ensembles in 90.5% of cases; in 7.b, small and large ensembles are shown to perform similarly in 69.2% of cases; and, no statistical difference between medium and large ensembles is shown in 7.c. The use of additional trees in the ensemble offered no significant advantage, and therefore, a small Rotation Forest with 10 trees was both sufficiently acceptable (taking into account accuracy) and computationally cheaper.

Commonly, the calculation of a data rotation matrix requires much more time than the training of a single tree. Thus, a strategy that could be followed to accelerate the training of Rotation Forest of a certain size is to decrease the number of rotations, but at the same time, in order not to reduce the number of base classifiers in the ensemble, more than one tree will be trained with the same rotation. This approach, already proposed in [42], acquires special relevance in the context of Big Data, since it would allow the reduction of the computing workload. Of course, as long as this simplification of the process does not significantly affect the final performance. This is precisely what will be evaluated in this section. Specifically, a small Rotation Forest ensemble of 10 trees could be built by rotating the data ten times and training one tree with each rotation ($L\times T = 10\times 1$), or by rotating the data five times and training two trees with each rotation ($L\times T = 5\times 2$). Training the 5×2 ensemble will take about half the time of training the 10×1 ensemble. With this in mind, different experiments were launched varying both the number of rotations and the ensemble size.

Table 4 shows the results of the hierarchical Bayesian tests comparing different configurations of number of rotations (L) and number of trees per rotation (T). In all cases, for a given ensemble size, the left region corresponds to the method with the highest number of rotations and the right region to the fastest alternative, with a reduced number of rotations. The results of using the proposed strategy to accelerate

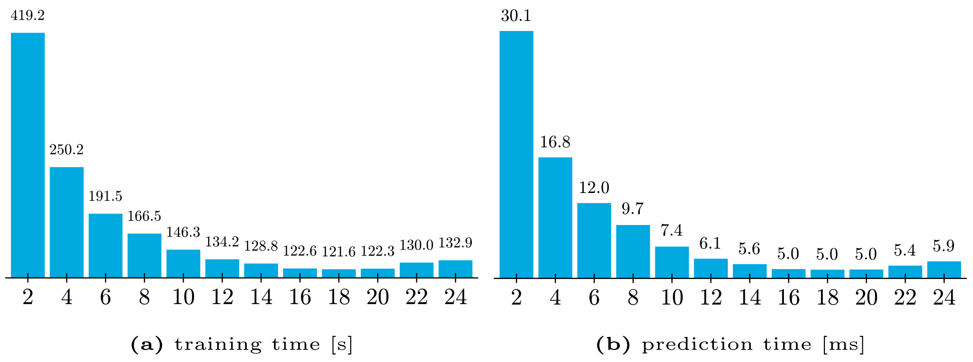


Fig. 5. Execution time evolution for susy data set. The left bar plot refers to the training time (in seconds) while the right plot shows prediction time (in milliseconds). The x-axis indicates the number of worker nodes.

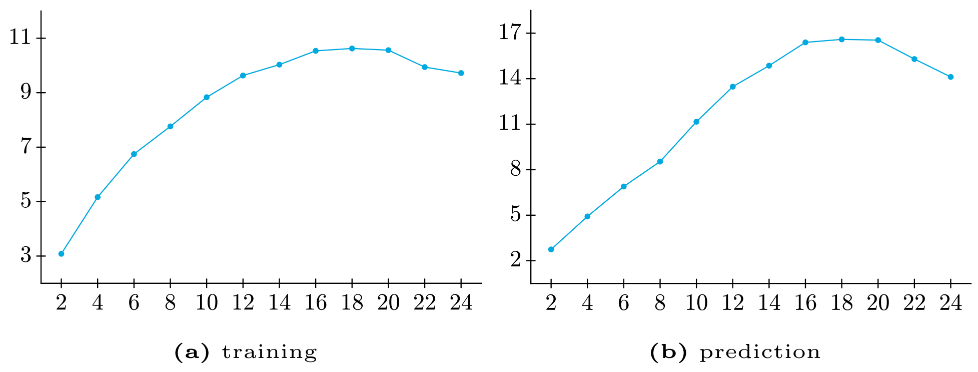


Fig. 6. Speedup (y-axis) for susy data set. The left plot refers to training and the right plot to prediction. The x-axis indicates the number of worker nodes.

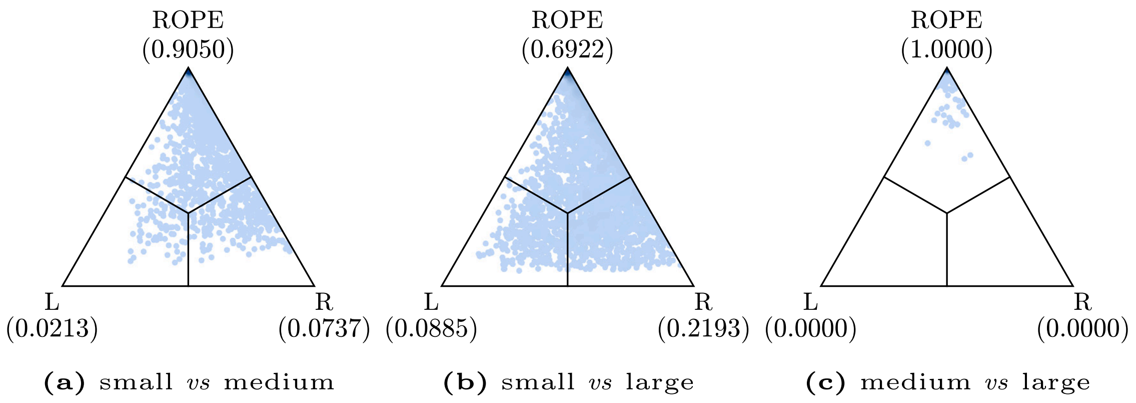


Fig. 7. Hierarchical Bayesian test heatmap for comparing the influence of ensemble size in Rotation Forest. Three sizes were compared: (a) 10 × 1 vs 50 × 1, (b) 10 × 1 vs 100 × 1, and (c) 50 × 1 vs 100 × 1.

the training phase of Rotation Forest are quite interesting. For small Rotation Forest (size 10), in 94.43% of cases, 10 × 1 and 5 × 2, were equivalent. For medium Rotation Forest (size 50), in 88.82% of cases, 50 × 1 and 10 × 5, performed in a similar way. In 54.5% of cases, 50 × 1 resulted better than 5 × 10 , which means that for this specific scenario, reducing the number of rotations tenfold would be counterproductive for accuracy (it appears that here the additional diversity provided by the 50 rotations is relevant to obtain good results). Ending with the last of the medium Rotation Forests comparisons (10 × 5 vs. 5 × 10), in 89.18% of occasions they turned out to be equivalent. Regarding the large Rotation Forest (size 100), in 98.35% of cases, 100 × 1 and 10 × 10, were equivalent, as well as 100 × 1 and 5 × 20, which showed to be equivalent in 83.65% of cases. Finally, equivalent performance is also shown between 10 × 10 and 5 × 20, which occurred in 94.15% of cases.

We trained small, medium, and large Rotation Forest classifiers with $L = 5$ (5 × 2, 5 × 10, and 5 × 20, respectively), and with $L = 10$ (10 × 1, 10 × 5, and 10 × 10, respectively), in what follows, the value of L is fixed (to 5 or to 10) and the influence of the parameter T is analyzed. Fig. 8 shows the hierarchical Bayesian test heatmap for $L = 5$ and $L = 10$ on the top and bottom row, respectively. For both values of L, all the tests shown in Fig. 8.a–f depict statistical equivalence with independence of the value of T.

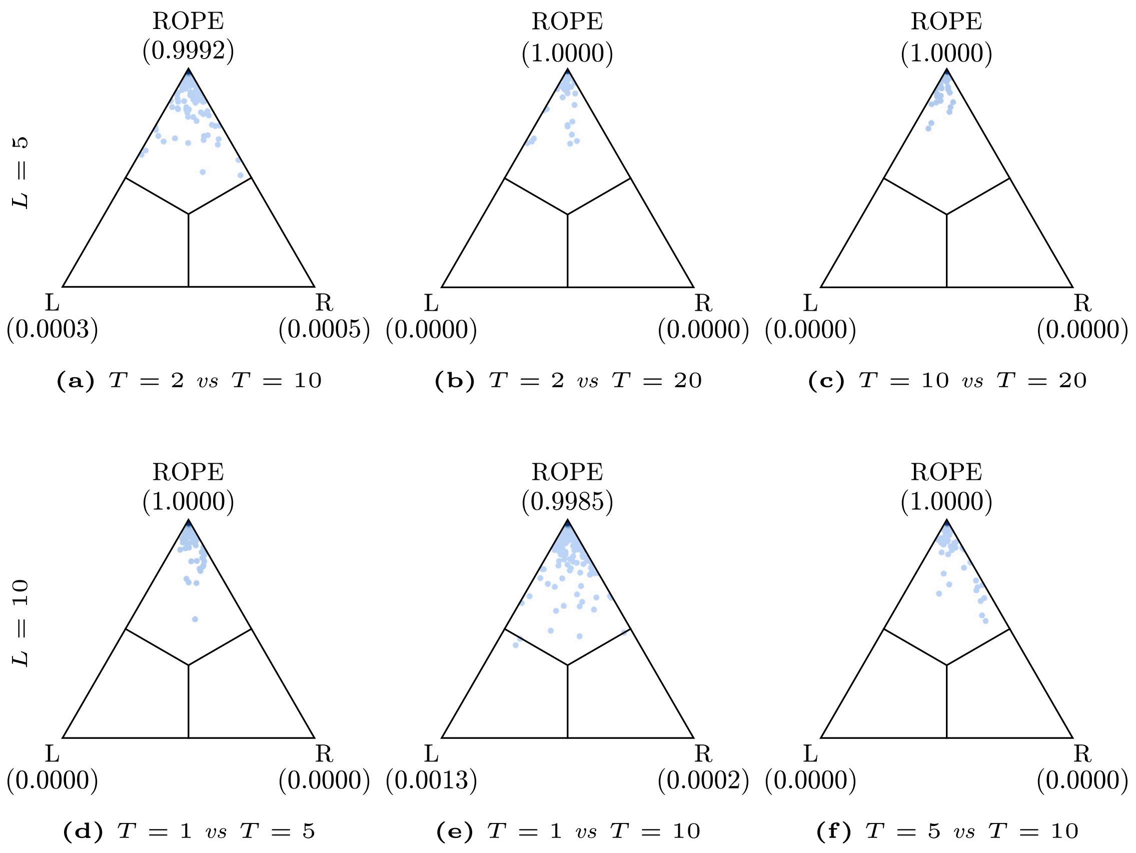


Fig. 8. Hierarchical Bayesian test heatmap for comparing the influence of increasing the number of trees for each rotation: on the top row, 5 rotations with comparisons of 2, 10, and 20 trees. On the bottom row, 10 rotations with comparisons of 1, 5, and 10 trees.

5.5. *Influence of bootstrap in Big Data*

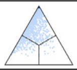
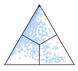
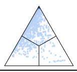
Bootstrapping is used in Rotation Forest as a part of the PCA calculation step, to avoid repeatedly obtaining the same principal components. The reason for doing so is to attempt to maximize the diversity of the ensemble. In [8], a bootstrap size of 75% was recommended. Nevertheless, this value could be lower, because the number of instances in Big Data is very high. Hence, a bootstrap sample containing a low percentage of instances should be sufficiently representative of the entire classification problem. The representativeness of the bootstrap sample is therefore not as important as the fact that it can generate different principal components, which is why lower bootstrap values could be beneficial or, at least, not harmful. Another thing to bear in mind is that in our case, the smaller the bootstrap sample, the faster the computation of the PCA. So, fixing a parameter that is as low as possible will shorten the training time.

Small Rotation Forests with $L = 10$ and $T = 1$ were trained with different bootstrap sizes – 10%, 25%, and 50% – to assess whether the bootstrap size might have an effect on the accuracy rates. Fig. 9 shows the hierarchical Bayesian test heatmaps, so that the results for the three bootstrap size cases may be compared. The Bayesian test shows that the bootstrap size did not have a significant impact on the accuracy of Rotation Forest, because the ROPE values in Figs. 9.a, 9.b, and 9.c were close to 100%. Hence, for Big Data environments, faster training times could be achieved by using small bootstrap sample sizes, e.g., 10%.

6. Conclusions and future work

This paper has presented a MapReduce design of a variant of Rotation Forest and its implementation in Spark (the implementation is publicly available.[5]). A thorough experimental campaign with Big

Table 4
Hierarchical Bayesian test results comparing the influence of different variants of ensemble size in Rotation Forest. The variants are obtained by setting a combination of L and T parameters for constructing small, medium, and large ensembles. The region (Left, Right, or ROPE) that gather the highest probability is highlighted in bold.

Ensemble size	Left			Right			ROPE	Heatmap
	L	T	Prob.	L	T	Prob.		
small (10)	10	1	0.0505	5	2	0.0052	**0.9443**	
	50	1	0.0873	10	5	0.0245	**0.8882**	
medium (50)	50	1	**0.5450**	5	10	0.1695	0.2855	
	10	5	0.0757	5	10	0.0325	**0.8918**	
	100	1	0.0100	10	10	0.0065	**0.9835**	
large (100)	100	1	0.1340	5	20	0.0295	**0.8365**	
	10	10	0.0490	5	20	0.0095	**0.9415**	

[5] The implementation for Spark is publicly available at https://github.com/mjuez/rotation-forest-spark.

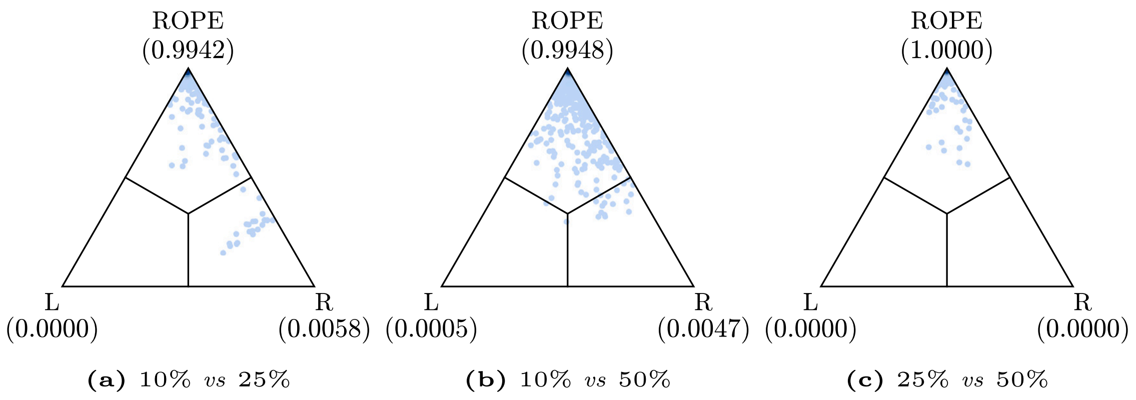


Fig. 9. Hierarchical Bayesian test heatmap for comparing the influence of bootstrap size in Rotation Forest. Three sizes were compared: 10%, 25%, and 50%.

Data sets has been performed, to assess the viability of the Rotation Forest algorithm for Big Data processing. The proposal presented in this paper has demonstrated its high performance and fast execution time, as well as its good scalability performance in cloud-based clusters. All of it with six large data sets having a number of instances ranging between 400 000 and 11 000 000, and a number of attributes ranging between 11 and 2 000.

Modern Bayesian tests (hierarchical and correlated *t* test) were used for evaluating the statistical differences between the different algorithms that were tested. Our experiments have consolidated the results of [8], in which the Rotation Forest algorithm was reported to be a better option than Random Forest, and the same conclusion has now been corroborated with massive data sets. Furthermore, the superiority of Rotation Forest in relation to PCARDE, a very recent ensemble algorithm for Big Data, has been demonstrated.

An experimental exploration of some algorithm parameters and their optimal values has been performed. Through that study, the approach for training Rotation Forest with Random Forest rather than of a single decision tree as the base classifier has proved that accurate models with fewer data rotations, and therefore models that train and predict faster, are indeed feasible. The analysis also reported that small ensembles, consisting of 10 trees, are accurate enough for the Big Data sets used in the study.

Additionally an evaluation of the influence of the bootstrap sample size with large data sets has been conducted. The conclusion of that evaluation was that sampling 10% of the data provided classification models with an equivalent performance to those that sampled 25% or 50% of the data. The use of low percentages therefore means simpler PCA calculations and faster training of the Rotation Forest.

Rotation Forest code has been carefully developed following the Spark ML API guidelines, aiming towards its incorporation in the API within the near future. The number of ensemble algorithms available for Big Data is still scarce, specially for tasks such as online learning and unbalanced data sets, among others. Ensembles have also been used for data stream analysis [47], the adaptation of Rotation Forest for streaming Big Data frameworks is therefore a challenging task that might mean that it could be applied to several real world problems.

In this paper, the presentation of Rotation Forest only covered classification problems and its adaptation to Big Data regression problems is still a future research line. Likewise, the adaptation of Rotation Forest for unbalanced learning [48] in Big Data scenarios, represents an area of potential interest for the scientific community.

CRediT authorship contribution statement

Mario Juez-Gil: Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization, Funding acquisition. **Álvar Arnaiz-González:** Conceptualization, Software, Validation, Resources, Writing - original draft, Writing - review & editing, Supervision, Project administration. **Juan J. Rodríguez:** Conceptualization, Writing - review & editing, Supervision, Project administration, Funding acquisition. **Carlos López-Nozal:** Conceptualization, Writing - review & editing, Supervision. **César García-Osorio:** Conceptualization, Writing - review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported through project TIN2015-67534-P (MINECO/FEDER, UE) of the *Ministerio de Economía y Competitividad* of the Spanish Government, projects BU085P17 and BU055P20 (JCyL/FEDER, UE) of the *Junta de Castilla y León, Spain* (both projects co-financed through European Union FEDER funds), and by the *Consejería de Educación* of the *Junta de Castilla y León*, Spain and the European Social Fund through a pre-doctoral grant (EDU/1100/2017). The project leading to these results has received also funding from "la Caixa" Foundation, Spain, under agreement LCF/PR/PR18/51130007. This material is based upon work supported by Google Cloud.

References

[1] G. Bello-Orgaz, J.J. Jung, D. Camacho, Social big data: Recent achievements and new challenges, Inf. Fusion 28 (2016) 45–59.
[2] M. Chen, S. Mao, Y. Liu, Big data: A survey, Mobile Netw. Appl. 19 (2) (2014) 171–209.
[3] J. Luengo, D. García-Gil, S. Ramírez-Gallego, S. García, F. Herrera, Big Data Preprocessing: Enabling Smart Data, Springer International Publishing, Cham, 2020.
[4] C. Moretti, K. Steinhaeuser, D. Thain, N.V. Chawla, Scaling up classifiers to cloud computers, in: 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 472–481.
[5] I.A.T. Hashem, I. Yaqoob, N.B. Anuar, S. Mokhtar, A. Gani, S. Ullah Khan, The rise of big data on cloud computing: Review and open research issues, Inf. Syst. 47 (2015) 98–115.
[6] J. Dean, S. Ghemawat, Mapreduce: Simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[7] L.I. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, John Wiley & Sons, 2014.
[8] J.J. Rodriguez, L.I. Kuncheva, C.J. Alonso, Rotation forest: A new classifier ensemble method, IEEE Trans. Pattern Anal. Mach. Intell. 28 (10) (2006) 1619–1630.
[9] L.I. Kuncheva, J.J. Rodríguez, An experimental study on rotation forest ensembles, in: M. Haindl, J. Kittler, F. Roli (Eds.), Multiple Classifier Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 459–468.
[10] S. Ghemawat, H. Gobioff, S.-T. Leung, The google file system, in: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP '03, Association for Computing Machinery, New York, NY, USA, 2003, pp. 29–43.

[11] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Presented As Part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), USENIX, San Jose, CA, 2012, pp. 15–28.

[12] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, et al., Spark: Cluster computing with working sets, HotCloud 10 (10–10) (2010) 95.

[13] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al., MLlib: Machine learning in Apache Spark, J. Mach. Learn. Res. 17 (1) (2016) 1235–1241.

[14] M. Assefi, E. Behravesh, G. Liu, A.P. Tafti, Big data machine learning using Apache Spark MLlib, in: 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 3492–3498.

[15] I.H. Witten, E. Frank, M.A. Hall (Eds.), Data Mining: Practical Machine Learning Tools and Techniques, third ed., in: The Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann, Boston, 2011.

[16] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, G. Varoquaux, API design for machine learning software: experiences from the scikit-learn project, in: ECML PKDD Workshop: Languages for Data Mining and Machine Learning, 2013, pp. 108–122.

[17] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Trans. Inform. Theory 13 (1) (1967) 21–27.

[18] E. Fix, J.L. Hodges Jr, Discriminatory analysis-nonparametric discrimination: consistency properties, Technical Report, California Univ Berkeley., 1951.

[19] J. Maillo, S. Ramírez, I. Triguero, F. Herrera, kNN-IS: An iterative spark-based design of the k-nearest neighbors classifier for big data, in: Variety and Velocity in Data Science, Knowl.-Based Syst. 117 (2017) 3–15.

[20] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, J.M. Benítez, F. Herrera, Nearest neighbor classification for high-speed big data streams using spark, IEEE Trans. Syst. Man Cybern. Syst. 47 (10) (2017) 2727–2739.

[21] S. Tyree, K.Q. Weinberger, K. Agrawal, J. Paykin, Parallel boosted regression trees for web search ranking, in: Proceedings of the 20th International Conference on World Wide Web, Association for Computing Machinery, New York, NY, USA, 2011, pp. 387–396.

[22] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, K. Li, A parallel random forest algorithm for big data in a spark cloud computing environment, IEEE Trans. Parallel Distrib. Syst. 28 (4) (2017) 919–933.

[23] J. Gonzalez-Lopez, A. Cano, S. Ventura, Large-scale multi-label ensemble learning on Spark, in: 2017 IEEE Trustcom/BigDataSE/ICESS, 2017, pp. 893–900.

[24] D. García-Gil, S. Ramírez-Gallego, S. García, F. Herrera, Principal components analysis random discretization ensemble for big data, Knowl.-Based Syst. 150 (2018) 166–174.

[25] N. Friedman, D. Geiger, M. Goldszmidt, Bayesian network classifiers, Mach. Learn. 29 (2–3) (1997) 131–163.

[26] J. Arias, J.A. Gamez, J.M. Puerta, Learning distributed discrete bayesian network classifiers under MapReduce with Apache Spark, in: Variety and Velocity in Data Science, Knowl.-Based Syst. 117 (2017) 16–26.

[27] S. García, S. Ramírez-Gallego, J. Luengo, J.M. Benítez, F. Herrera, Big data preprocessing: methods and prospects, Big Data Anal. 1 (1) (2016) 9.

[28] S. Ramírez-Gallego, S. García, J. Benítez, F. Herrera, A distributed evolutionary multivariate discretizer for big data processing on Apache Spark, Swarm Evol. Comput. 38 (2018b) 240–250.

[29] D. García-Gil, J. Luengo, S. García, F. Herrera, Enabling smart data: Noise filtering in big data classification, Inform. Sci. 479 (2019) 135–152.

[30] S. Ramírez-Gallego, S. García, N. Xiong, F. Herrera, BELIEF: A Distance-Based Redundancy-Proof Feature Selection Method for Big Data, 2018.

[31] Á. Arnaiz-González, J.-F. Díez-Pastor, J.J. Rodríguez, C. García-Osorio, Instance selection of linear complexity for big data, Knowl.-Based Syst. 107 (2016) 83–95.

[32] Á. Arnaiz-González, A. González-Rogel, J.-F. Díez-Pastor, C. López-Nozal, MR-DIS: democratic instance selection for big data by MapReduce, Progress in Artificial Intelligence 6 (3) (2017) 211–219.

[33] C. García-Osorio, A.de. Haro-García, N. García-Pedrajas, Democratic instance selection: A linear complexity instance selection algorithm based on classifier ensemble concepts, Artificial Intelligence 174 (5–6) (2010) 410–441.

[34] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.

[35] M. Fernández-Delgado, E. Cernadas, S. Barro, D. Amorim, Do we need hundreds of classifiers to solve real world classification problems? J. Mach. Learn. Res. 15 (1) (2014) 3133–3181.

[36] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.

[37] A. Bagnall, M. Flynn, J. Large, J. Line, A. Bostrom, G. Cawley, Is rotation forest the best classifier for problems with continuous features? 2018.

[38] C. Pardo, J.F. Diez-Pastor, C. García-Osorio, J.J. Rodríguez, Rotation forests for regression, Appl. Math. Comput. 219 (19) (2013) 9914–9924.

[39] C.-X. Zhang, J.-S. Zhang, RotBoost: A technique for combining Rotation Forest and AdaBoost, Pattern Recognit. Lett. 29 (10) (2008) 1524–1536.

[40] A. Fernández, S. del Río, N.V. Chawla, F. Herrera, An insight into imbalanced big data classification: outcomes and challenges, Complex Intell. Syst. 3 (2) (2017) 105–120.

[41] S. Ramírez-Gallego, A. Fernández, S. García, M. Chen, F. Herrera, Big data: Tutorial and guidelines on information and process fusion for analytics algorithms with mapreduce, Inf. Fusion 42 (2018a) 51–61.

[42] G. Stiglic, J.J. Rodriguez, P. Kokol, Rotation of Random Forests for Genomic and Proteomic Classification Problems, Springer New York, New York, NY, 2011, pp. 211–221.

[43] B. Panda, J.S. Herbach, S. Basu, R.J. Bayardo, PLANET: Massively parallel learning of tree ensembles with MapReduce, in: Proceedings of the 35th International Conference on Very Large Data Bases (VLDB-2009), 2009.

[44] D. Dua, C. Graff, UCI Machine learning repository, 2017.

[45] A. Benavoli, G. Corani, J. Demšar, M. Zaffalon, Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis, J. Mach. Learn. Res. 18 (1) (2017) 2653–2688.

[46] G. Corani, A. Benavoli, A Bayesian approach for comparing cross-validated algorithms on multiple data sets, Mach. Learn. 100 (2–3) (2015) 285–304.

[47] B. Krawczyk, L.L. Minku, J. Gama, J. Stefanowski, M. Woźniak, Ensemble learning for data stream analysis: A survey, Inf. Fusion 37 (2017) 132–156.

[48] J.F. Díez-Pastor, J.J. Rodríguez, C.I. García-Osorio, L.I. Kuncheva, Diversity techniques improve the performance of the best imbalance learning ensembles, Inform. Sci. 325 (2015) 98–117.