Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 246139, 11 pages
http://dx.doi.org/10.1155/2015/246139

Research Article

Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach

Daniel Peralta,[1] Sara del Río,[1] Sergio Ramírez-Gallego,[1] Isaac Triguero,[2,3] Jose M. Benitez,[1] and Francisco Herrera[1]

[1]*Department of Computer Science and Artificial Intelligence, CITIC-UGR, University of Granada, 18071 Granada, Spain*
[2]*Department of Respiratory Medicine, Ghent University, 9000 Ghent, Belgium*
[3]*VIB Inflammation Research Center, 9052 Zwijnaarde, Belgium*

Correspondence should be addressed to Daniel Peralta; dperalta@decsai.ugr.es

Received 27 February 2015; Accepted 14 June 2015

Academic Editor: Sangmin Lee

Nowadays, many disciplines have to deal with big datasets that additionally involve a high number of features. Feature selection methods aim at eliminating noisy, redundant, or irrelevant features that may deteriorate the classification performance. However, traditional methods lack enough scalability to cope with datasets of millions of instances and extract successful results in a delimited time. This paper presents a feature selection algorithm based on evolutionary computation that uses the MapReduce paradigm to obtain subsets of features from big datasets. The algorithm decomposes the original dataset in blocks of instances to learn from them in the map phase; then, the reduce phase merges the obtained partial results into a final vector of feature weights, which allows a flexible application of the feature selection procedure using a threshold to determine the selected subset of features. The feature selection method is evaluated by using three well-known classifiers (SVM, Logistic Regression, and Naive Bayes) implemented within the Spark framework to address big data problems. In the experiments, datasets up to 67 millions of instances and up to 2000 attributes have been managed, showing that this is a suitable framework to perform evolutionary feature selection, improving both the classification accuracy and its runtime when dealing with big data problems.

1. Introduction

Learning from very large databases is a major issue for most of the current data mining and machine learning algorithms [1]. This problem is commonly named with the term "big data," which refers to the difficulties and disadvantages of processing and analyzing huge amounts of data [2–4]. It has attracted much attention in a great number of areas such as bioinformatics, medicine, marketing, or financial businesses [5], because of the enormous collections of raw data that are stored. Recent advances on Cloud Computing technologies allow for adapting standard data mining techniques in order to apply them successfully over massive amounts of data [4, 6, 7].

The adaptation of data mining tools for big data problems may require the redesigning of the algorithms and their inclusion in parallel environments. Among the different alternatives, the MapReduce paradigm [8, 9] and its distributed file system [10], originally introduced by Google, offer an effective and robust framework to address the analysis of big datasets. This approach is currently taken into consideration in data mining, rather than other parallelization schemes such as MPI (Message Passing Interface) [11], because of its fault-tolerant mechanism and its simplicity. Many recent works have been focused on the parallelization of machine learning tools using the MapReduce approach [12, 13].

Recently, new and more flexible workflows have appeared to extend the standard MapReduce approach, such as Apache Spark [14], which has been successfully applied over various data mining and machine learning problems [15–17].

Data preprocessing methods, and more concretely data reduction models, are intended to clean and simplify input data [18]. Thus, they attempt to accelerate data mining

algorithms and also to improve their accuracy by eliminating noisy and redundant data. The specialized literature describes two main types of data reduction models. On the one hand, instance selection [19, 20] and instance generation [21] processes are focused on the instance level. On the other hand, feature selection [22–25] and feature extraction [26] models work at the level of characteristics.

Among the existing techniques, evolutionary approaches have been successfully used for feature selection techniques [27]. Nevertheless, an excessive increment of the individual size can limit their applicability, being unable to provide a preprocessed dataset in a reasonable time when dealing with very large problems. In the current literature, there are no approaches to tackle the feature space with evolutionary big data models.

The main objective of this paper is to enable Evolutionary Feature Selection (EFS) models to be applied on big data. To do this, a MapReduce algorithm has been developed, which splits the data and performs a bunch of EFS processes in parallel in the map phase and then combines the solutions in the reduce phase to get the most interesting features. This algorithm will be denoted "MapReduce for Evolutionary Feature Selection" (MR-EFS).

More specifically, the purposes of this paper are

(i) to design an EFS technique over the MapReduce paradigm for big data,

(ii) to analyze and illustrate the scalability of the proposed scheme in terms of classification accuracy and time necessary to build the classifiers.

To analyze the proposed approach, experiments on two big data classification datasets with up to 67 millions instances and up to 2000 features will be carried out, focusing on the CHC algorithm [28] as EFS method. With the characteristics selected by this model, its influence on the classification performance of the Spark implementation of three different algorithms (Support Vector Machine, Logistic Regression, and Naive Bayes), available in MLlib [29], will be analyzed.

The rest of the paper is organized as follows. Section 2 provides some background information about EFS and MapReduce. Section 3 describes the MapReduce algorithm proposed for EFS. The empirical results are discussed and analyzed in Section 4. Finally, Section 5 summarizes the conclusions of the paper.

2. Background

This section describes the topics used in this paper. Section 2.1 presents some preliminaries about EFS and its main drawbacks to deal with big data classification problems. Section 2.2 introduces the MapReduce paradigm, as well as two of the main frameworks for big data: Hadoop and Spark.

2.1. Feature Selection: Problems with Big Datasets. Feature selection models attempt to reduce a dataset by removing irrelevant or redundant features. The feature selection process seeks to obtain a minimum set of attributes, such that the results of the data mining techniques that are applied over the reduced dataset are as close as possible (or even better) to the results obtained using all attributes [25]. This reduction facilitates the understanding of the patterns extracted and increases the speed of posterior learning stages.

Feature selection methods can be classified into three categories:

(i) *Wrapper methods*: The selection criterion is part of the fitness function and therefore depends on the learning algorithm [30].

(ii) *Filtering methods*: The selection is based on data-related measures, such as separability or crowding [22].

(iii) *Embedded methods*: The optimal subset of features is built within the classifier construction [24].

For more information about specific feature selection methods, the reader can refer to the published surveys on the topic [22–24].

A recent, interesting proposal for applying feature selection to big datasets is presented in [31]. In that paper, the authors describe an algorithm that is able to efficiently cope with ultrahigh-dimensional datasets and select a small subset of interesting features from them. However, the number of selected features is assumed to be several orders of magnitude lower than the total of features, and the algorithm is designed to be executed in a single machine. Therefore, this approach is not scalable to arbitrarily large datasets.

A particular way of tackling feature selection is by using evolutionary algorithms [27]. Usually, the set of features is encoded as a binary vector, where each position determines if a feature is selected or not. This allows to perform feature selection with the exploration capabilities of evolutionary algorithms. However, they lack the scalability necessary to address big datasets (from millions of instances onwards). The main problems found when dealing with big data are as follows:

(i) *Runtime*: The complexity of EFS models is at least $O(n^2D)$, where n is the number of instances and D the number of features. When either of these variables becomes too large, the application of EFS may be too time-consuming for real situations.

(ii) *Memory consumption*: Most EFS methods need to store the entire training dataset in memory, along with additional computation data and results. When these data are too big, their size could easily exceed the available RAM memory.

In order to overcome these weaknesses, distributed partitioning procedures are used, within a MapReduce paradigm, that divide the dataset into disjoint subsets that are manageable by EFS methods.

2.2. Big Data: MapReduce, Hadoop, and Spark. This section describes the main solutions for big data processing. Section 2.2.1 focuses on the MapReduce programming model, whilst Section 2.2.2 introduces two of the main frameworks to deal with big data.

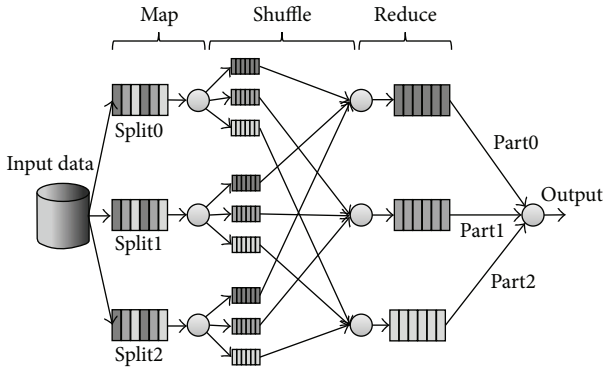

Figure 1: Flowchart of the MapReduce framework.

2.2.1. MapReduce. MapReduce [8, 9] is one of the most popular programming models to deal with big data. It was proposed by Google in 2004 and designed for processing huge amounts of data using a cluster of machines. The MapReduce paradigm is composed of two phases: map and reduce. In general terms, in the map phase, the input dataset is processed producing some intermediate results. Then, the reduce phase combines them in some way to form the final output.

The MapReduce model is based on a basic data structure known as the ⟨key, value⟩ pair. In the map phase, each application of the map function receives a single ⟨key, value⟩ pair as input and generates a list of intermediate ⟨key, value⟩ pairs as output. This is represented by the following form:

$$\text{map}\left(\text{key1}, \text{value1}\right) \longrightarrow \left\{\left(\text{key2}, \text{value2}\right), \ldots\right\}. \quad (1)$$

Then, the MapReduce library groups all intermediate ⟨key, value⟩ pairs by key. Finally, the reduce function takes the aggregated pairs and generates a new ⟨key, value⟩ pair as output. This is depicted by the following form:

$$\text{reduce}\left(\text{key2}, \{\text{value2}, \ldots\}\right) \longrightarrow \left(\text{key2}, \text{value3}\right). \quad (2)$$

A flowchart of the MapReduce framework is presented in Figure 1.

2.2.2. Hadoop and Spark. Different implementations of the MapReduce programming model have appeared in the last years. The most popular one is Apache Hadoop [32], an open-source framework written in Java that allows the processing and management of large datasets in a distributed computing environment. In addition, Hadoop works on top of the Hadoop Distributed File System (HDFS), which replicates the data files in many storage nodes, facilitating rapid data transfer rates among nodes and allowing the system to continue operating without interruption when one or several nodes fail.

In this paper, Apache Hadoop is used to implement the proposal, MR-EFS, as described in Section 3.2.

Another Apache project that is tightly related to Hadoop is Spark [14]. It is a cluster computing framework originally developed in the UC Berkeley AMP Lab for large-scale data processing that improves the efficiency by the use of intensive memory. Spark uses HDFS and has high-level libraries for stream processing and for machine learning and graph processing, such as MLlib [29].

For this work, several classifiers included in MLlib are used to test the MR-EFS algorithm: SVM, Naive Bayes, and Logistic Regression. Their parameters are specified in Section 4.1.

3. MR-EFS: MapReduce for Evolutionary Feature Selection

This section describes the proposed MapReduce approach for EFS, as well as its integration in a generic classification process. In particular, the MR-EFS algorithm is based on the CHC algorithm to perform feature selection, as described in Section 3.1.

First, MR-EFS is applied over the original dataset to obtain a vector of weights that indicates the relevance of each attribute (Section 3.2). Then, this vector is used within another MapReduce process to produce the resulting reduced dataset (Section 3.3). Finally, the reduced dataset is used by a classification algorithm.

3.1. CHC Algorithm for Feature Selection. The CHC algorithm [28] is a binary-coded genetic algorithm that combines a very high selective pressure with an elitist selection strategy, along with several components that introduce diversity. The main parts of CHC are the following:

(i) *Half Uniform Crossover (HUX)*: This crossover operator aims at enforcing a high diversity and reducing the risk of premature convergence. It selects at random half of the bits that are different between both parents. Then, it obtains two offspring that are at the maximum Hamming distance from their parents.

(ii) *Elitist selection*: In each generation, the new population is composed of the best individuals (those with the best values of the fitness function) among both the current and the offspring populations. In case of draw between a parent and an offspring, the parent is selected.

(iii) *Incest prevention*: Two individuals are not allowed to mate if the Hamming similarity between them exceeds a threshold d (usually initialized to $d = L/2$, where L is the chromosome length). The threshold is decremented by one when no offspring is obtained in one generation, which indicates that the algorithm is converging.

(iv) *Restarting process*: When $d = 0$ (which happens after several generations without any new offspring), the population is considered to be stagnated. In such a case, a new population is generated: the best individual is kept, and the remaining individuals have a certain percentage of their bits flipped.

The basic execution scheme of CHC is shown in Figure 2. This algorithm naturally adapts to a feature selection

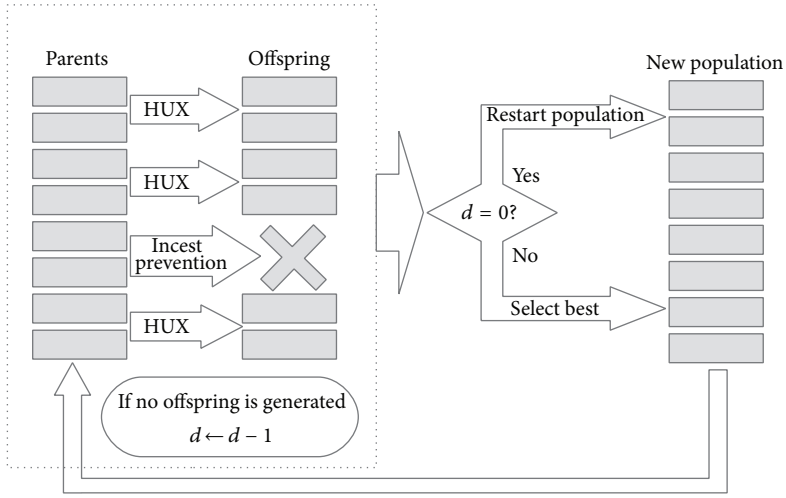

Figure 2: Flowchart of the CHC algorithm.

problem, as each feature can be represented as a bit in the solution vector. Thus, each position of the vector indicates if the corresponding feature is selected or not. Therefore, this approach falls within the wrapper method category, according to the classification established in Section 2.1. The fitness function used to evaluate new individuals applies a k-Nearest Neighbors classifier (k-NN) [33] over the dataset that would be obtained after removing the corresponding features. The fitness value is the weighted sum of the k-NN accuracy and the feature reduction rate.

3.2. MR-EFS Algorithm. This section describes the parallelization of the CHC algorithm, by using a MapReduce procedure to obtain a vector of weights.

Let T be a training set, stored in HFDS and randomized as described in [34]. Let m be the number of map tasks. The splitting procedure of MapReduce divides T in m disjoint subsets of instances. Then, each T_i subset ($i \in \{1, 2, \ldots, m\}$) is processed by the corresponding Map_i task. As this partitioning is performed sequentially, all subsets will have approximately the same number of instances, and the randomization of the T file ensures an adequate balance of the classes.

The map phase over each T_i consists of the EFS algorithm (in this case, based on CHC) as described in Section 3.1. Therefore, the output of each map task is a binary vector $\mathbf{f_i} = \{f_{i1}, \ldots, f_{iD}\}$, where D is the number of features, that indicates which features were selected by the CHC algorithm. The reduce phase averages all the binary vectors, obtaining a vector $\mathbf{x}$ as defined in (3), where x_j is the proportion of EFS applications that include the feature j in their result. This vector is the result of the overall EFS process and is used to build the reduced dataset that will be used for further machine learning purposes:

$$\mathbf{x} = \{x_1, \ldots, x_D\},$$
$$x_j = \frac{1}{m}\sum_{i=1}^{m} f_{ij}, \quad j \in \{1, 2, \ldots, D\}. \tag{3}$$

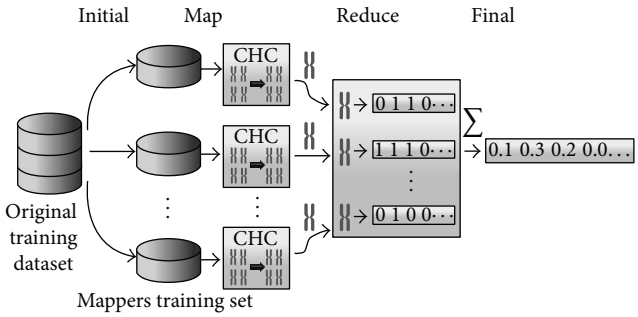

Figure 3: Flowchart of MR-EFS algorithm.

In the implementation used for the experiments, the reduce phase is carried out by a single task, which reduces the runtime by decreasing the MapReduce overhead [35]. The whole MapReduce process for EFS is depicted in Figure 3. It is noteworthy that the whole procedure is performed within a single iteration of the MapReduce workflow, avoiding additional disk accesses.

3.3. Dataset Reduction with MapReduce. Once vector $\mathbf{x}$ is calculated, the objective is to remove the less promising features from the original dataset. To do so in a scalable manner, an additional MapReduce process was designed. First, vector $\mathbf{x}$ is binarized using a threshold θ:

$$\mathbf{b} = \{b_1, \ldots, b_D\},$$
$$b_j = \begin{cases} 1, & \text{if } x_j \geq \theta, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Vector $\mathbf{b}$ indicates which features will be selected for the reduced dataset. The number of selected features ($D' = \sum_{j=1}^{D} b_j$) can be controlled with θ: with a high threshold, only a few features will be selected, while a lower threshold allows more features to be picked.

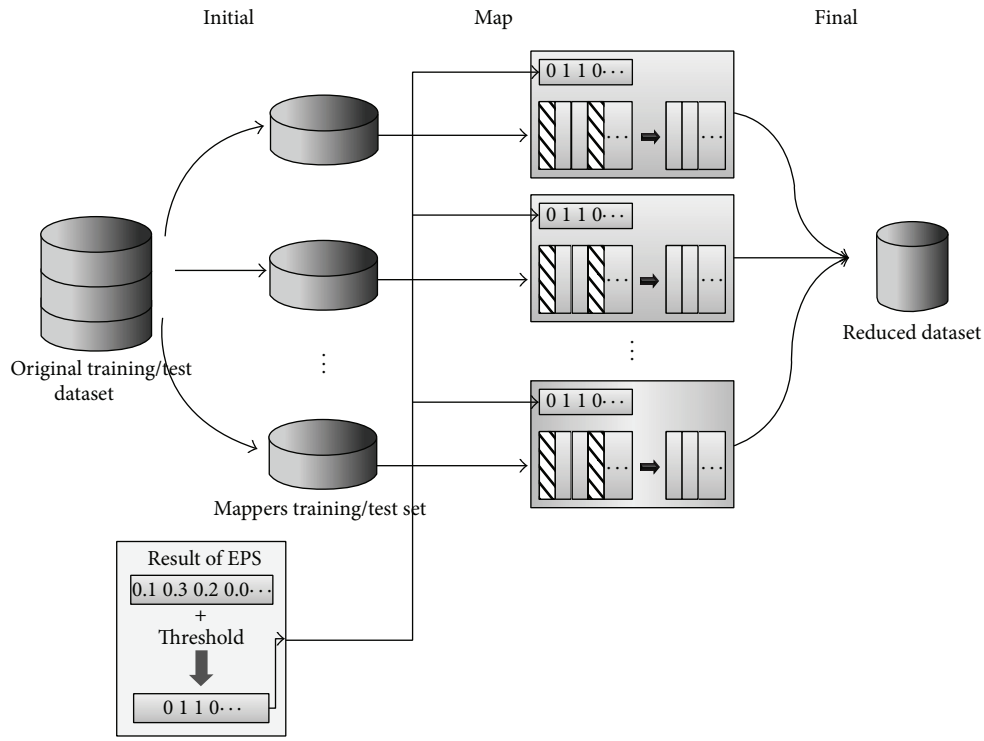

Figure 4: Flowchart of the dataset reduction process.

The MapReduce process for the dataset reduction works as follows. Each map processes an instance and generates a new one that only contains the features selected in **b**. Finally, the instances generated are concatenated to form the final reduced dataset, without the need of a reduce step. The dataset reduction process, using the result of MR-EFS and an arbitrary threshold, is depicted in Figure 4.

4. Experimental Framework and Analysis

This section describes the performed experiments and their results. First, Section 4.1 describes the datasets and the methods used for the experiments. Section 4.2 details the underlying hardware and software support. Finally, Sections 4.3 and 4.4 present the results obtained using two different datasets.

4.1. Datasets and Methods. This experimental study uses two large binary classification datasets in order to analyze the quality of the solutions provided by the MR-EFS algorithm.

First, the epsilon dataset was used, which is composed of 500 000 instances with 2000 numerical features. This dataset was artificially created for the Pascal Large Scale Learning Challenge [36] in 2008. The version provided by LIBSVM [37] was used.

Additionally, this study includes the dataset used at the data mining competition of the Evolutionary Computation for Big Data and Big Learning held on July 14, 2014, in Vancouver (Canada), under the international conference GECCO-2014 (from now on, it is referred to as ECBDL14) [38]. This dataset has 631 features (including both numerical and categorical attributes), and it is composed of approximately 32 million instances. Moreover, the class distribution is not balanced: 98% of the instances belong to the negative class.

In order to deal with the imbalance problem, the MapReduce approach of the Random Oversampling (ROS) algorithm presented in [39] was applied over the original training set for ECBDL14. The aim of ROS is to replicate the minority class instances from the original dataset until the number of instances from both classes is the same.

Despite the inconvenience of increasing the size of the dataset, this technique was proven in [39] to yield better performance than other common approaches to deal with imbalance problems, such as undersampling and cost-sensitive methods. These two approaches suffer from the small sample size problem for the minority class when they are used within a MapReduce model.

The main characteristics of these datasets are summarized in Table 1. For each dataset, the number of instances for both training and test sets and the number of attributes are shown, along with the number of splits in which MR-EFS divided each dataset. Note that the imbalanced version of ECBDL14 is not used in the experiments, as only the balanced ECBDL14-ROS version is considered.

The parameters for the CHC algorithm are presented in Table 2.

After applying MR-EFS over the described datasets, the behavior of the obtained reduced datasets was tested using three different classifiers implemented in Spark, available in MLlib: SVM [40], Logistic Regression [41], and Naive

Table 1: Summary of the used big data classification datasets.

Dataset	Training instances	Test instances	Features	Splits	Instances per split
Epsilon	400 000	100 000	2000	512	~780
ECBDL14	31 992 921	2 897 917	631	—	—
ECBDL14-ROS	65 003 913	2 897 917	631	32 768	~1984

Table 2: Parameter specification for all the methods involved in the experimentation.

Algorithm	Parameters
CHC	k value for k-NN: 1
	Trade-off between reduction and accuracy: 0.5
	Proportion of flipped bits in restarting process: 0.05
	Population size: 40
	Number of evaluations: 1000
Naive Bayes	Lambda: 1.0 (default)
Logistic Regression	Iterations: 100 (default)
	StepSize: 1.0 (default)
	miniBatchFraction: 1.0
SVM	Regularization parameter: 0.0; 0.5
	Iterations: 100 (default)
	StepSize: 1.0 (default)
	miniBatchFraction: 1.0

Bayes [42]. The reader may refer to the provided references or to the MLlib guide [43] for further details about their internal functioning. The parameters used for these classifiers are listed in Table 2. Particularly, two different variants of SVM were used, modifying the regularization parameter, which allows the algorithm to calculate simpler models by penalizing complex models in the objective function.

In the remainder of this paper, two metrics are used to evaluate the performance of the three classifiers when applied over the obtained reduced datasets:

(i) *Area Under the Curve (AUC)*: This measure is defined as the area under the Receiver Operating Characteristic (ROC) curve. In this work, this value is approximated with the formula in (5), where TPR is the True Positive Rate and TNR is the True Negative Rate. These values can be directly obtained from the confusion matrix and are not affected by the imbalance of the dataset:

$$\text{AUC} = \frac{\text{TPR} + \text{TNR}}{2}. \quad (5)$$

(ii) *Training runtime*: It is the time (in seconds) used to train or build the classifier.

Note that for this study the test runtime is much less affected by the feature selection process, because at that point the classifier has already been built. For the sake of simplicity, only training runtimes are reported.

4.2. Hardware and Software Used. The experiments for this paper were carried out on a cluster of twenty computing nodes, plus a master node. Each one of these compute nodes has the following features:

(i) *Processors:* 2 x Intel Xeon CPU E5-2620.

(ii) *Cores:* 6 per processor (12 threads).

(iii) *Clock speed:* 2.00 GHz.

(iv) *Cache:* 15 MB.

(v) *Network:* QDR InfiniBand (40 Gbps).

(vi) *Hard drive:* 2 TB.

(vii) *RAM:* 64 GB.

Both Hadoop master processes—the NameNode and the JobTracker—are hosted in the master node. The former controls the HDFS, coordinating the slave machines by the means of their respective DataNode processes, while the latter is in charge of the TaskTrackers of each compute node, which execute the MapReduce framework. Spark follows a similar configuration, as the master process is located on the master node, and the worker processes are executed on the slave machines. Both frameworks share the underlying HDFS file system.

These are the details of the software used for the experiments:

(i) *MapReduce implementation:* Hadoop 2.0.0-cdh4.7.1. MapReduce 1 (Cloudera's open-source Apache Hadoop distribution).

(ii) *Spark version:* Apache Spark 1.0.0.

(iii) *Maximum maps tasks:* 320 (16 per node).

(iv) *Maximum reducer tasks:* 20 (1 per node).

(v) *Operating system:* CentOS 6.6.

Note that the total number of cores of the cluster is 240. However, a higher number of maps were kept to maximize the use of the cluster by allowing a higher parallelism and a better data locality, thereby reducing the network overload.

4.3. Experiments with the Epsilon Dataset. This section explains the results obtained for the epsilon dataset. First, Section 4.3.1 describes the performance of the feature selection procedure and compares it with a sequential approach. Then, Section 4.3.2 describes the results obtained in the classification.

Table 3: Execution times (in seconds) over the epsilon subsets.

Instances	Sequential CHC	MR-EFS	Splits
1000	391	419	1
2000	1352	409	2
5000	8667	413	5
10 000	39 576	431	10
15 000	91 272	445	15
20 000	159 315	455	20
400 000	—	6531	512

4.3.1. Feature Selection Performance. The complexity order of the CHC algorithm is approximately $O(n^2Dp)$, where p is the number of evaluations of the fitness function (a k-NN classifier in this case), n is the number of instances, and D is the number of features. Therefore, the algorithm is quadratic with respect to the number of instances.

When the dataset is divided into m splits within MR-EFS, each one of the m map tasks has complexity order $O((n^2/m^2)Dp)$, which is m^2 times faster than applying CHC over the whole dataset. If n_c cores are available for the map tasks, the complexity of the map phase within the MR-EFS procedure is approximately $O(\lceil m/n_c \rceil (n^2/m^2)Dp)$. This demonstrates the scalability of the approach presented in this paper: even if the maps are executed sequentially ($n_c = 1$), the procedure is still one order of magnitude faster than feeding a single CHC with all the instances at once.

In order to verify the performance of MR-EFS with respect to the sequential approach, a set of experiments were performed using subsets of the epsilon dataset. Both a sequential CHC algorithm and the parallel MR-EFS (with 1000 instances per split) were applied over those subsets. The obtained execution times are presented in Table 3 and Figure 5, along with the runtime of MR-EFS over the whole dataset.

The sequential runtimes described a quadratic shape, in concordance with the complexity order of CHC, which clearly states that the time necessary to tackle the whole dataset would be impractical. In opposite, the runtime of MR-EFS for the small datasets was nearly constant. The case with 1000 instances is particular, in the sense that MR-EFS only executed one map task; therefore, it executed a single CHC with 1000 instances. The time difference between CHC and MR-EFS in this case reflects the overhead introduced by the latter. Even though his overhead increased slightly as the number of map tasks grew, it represented a minor part of the overall runtime.

As for the full dataset, with 512 splits, the number of instances for each map task in MR-EFS is around 780. As the number of cores used for the experiments was 240, the map phase in MR-EFS should be roughly three times slower than the sequential CHC with 1000 instances, according to the complexity orders previously detailed. The times in Table 3 show a higher time gap, because MR-EFS includes as well the other phases of the MapReduce framework (namely, splitting, shuffle, and reduce), which are nonnegligible for a dataset of such size. Nevertheless, the overall MR-EFS execution time is highly scalable, as shown by the fact that MR-EFS was able to process the 400 000 instances faster than the time needed by CHC to process 5000 instances.

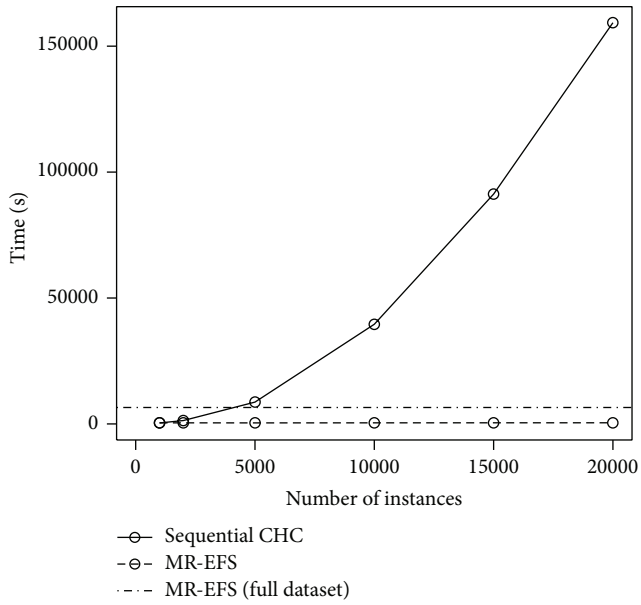

Figure 5: Execution times of the sequential CHC and MR-EFS.

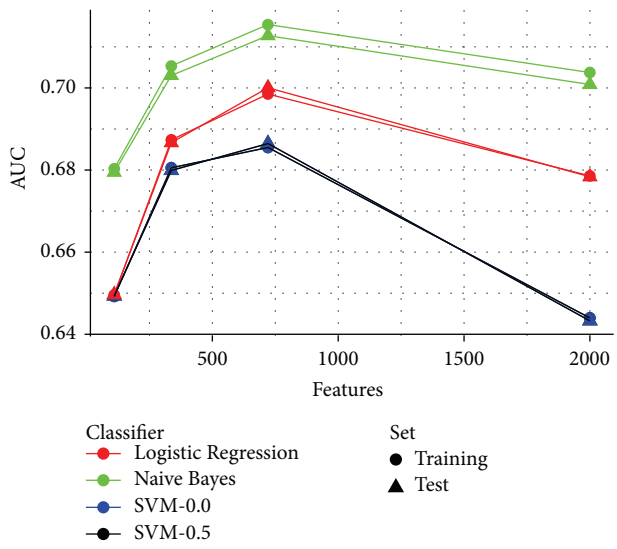

Figure 6: Accuracy with the epsilon dataset. Note that the number of features decreases as the threshold increases.

4.3.2. Classification Results. This section presents the results obtained by applying several classifiers over the full epsilon dataset with its features previously selected by using MR-EFS. The dataset was split among 512 map tasks, each of which computed around 780 instances.

The AUC values are shown in Table 4 and Figure 6, both for training and test sets, using three different thresholds for the dataset reduction step. Note that the zero-threshold corresponds to the original dataset, without performing any

Table 4: AUC results for the Spark classifiers using epsilon.

Threshold	Features	Logistic Regression		Naive Bayes		SVM ($\lambda = 0.0$)		SVM ($\lambda = 0.5$)	
		Training	Test	Training	Test	Training	Test	Training	Test
0.00	2000	0.6786	0.6784	0.7038	0.7008	0.6440	0.6433	0.6440	0.6433
0.55	721	**0.6985**	**0.7000**	**0.7154**	**0.7127**	**0.6855**	**0.6865**	**0.6855**	**0.6865**
0.60	337	0.6873	0.6867	0.7054	0.7030	0.6805	0.6799	0.6805	0.6799
0.65	110	0.6496	0.6497	0.6803	0.6794	0.6492	0.6493	0.6492	0.6493

Table 5: Training runtime (in seconds) for the Spark classifiers using epsilon.

Threshold	Features	Logistic Regression	Naive Bayes	SVM ($\lambda = 0.0$)	SVM ($\lambda = 0.5$)
0.00	2000	**367.29**	605.14	**334.18**	**331.69**
0.55	721	409.35	340.42	409.70	386.84
0.60	337	488.16	307.33	505.46	489.93
0.65	110	501.86	**264.26**	467.44	473.74

Table 6: Size of the epsilon dataset for each threshold.

Threshold	Set	MB	HDFS blocks
0.00	Training	8179.18	128
	Test	2044.80	32
0.55	Training	2946.52	47
	Test	736.63	12
0.60	Training	1377.21	22
	Test	344.30	6
0.65	Training	450.60	8
	Test	112.65	2

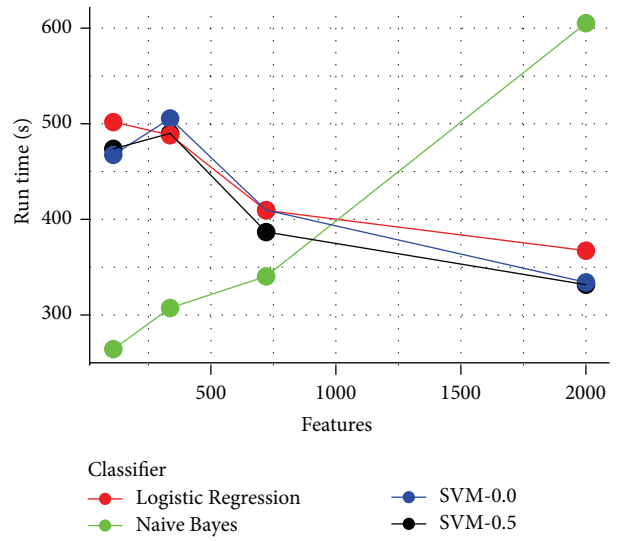

Figure 7: Training runtime with the epsilon dataset. Note that the number of features decreases as the threshold increases.

feature selection. The best results for each method are stressed in boldface. The table shows that the accuracy was improved by the removal of the adequate features, as the threshold 0.55 allowed for obtaining higher AUC values. The accuracy gain was especially large for SVM. Moreover, more than half of the features were removed, which reduces significantly the size of the dataset and therefore the complexity of the resulting classifier.

A threshold of value 0.60 also got to improve the accuracy results, while reducing even further the size of the dataset. Finally, for the 0.65 thresholds, only SVM saw its AUC improved.

It is also noteworthy that the two variants of SVM obtained the same results for all the tested thresholds. This fact indicates that the complexity of the obtained SVM model is relatively low.

The training runtime of the different algorithms and databases is shown in Table 5 and Figure 7. The obtained results were seemingly the opposite to the expectations: except for Naive Bayes, the classifiers needed more time to process the datasets as their number of features decreases.

However, this behavior can be explained: as the dataset gets smaller, it occupies less HDFS blocks, and therefore the full parallel capacity of the cluster is not exploited. The size of each version of the epsilon dataset and the number of HDFS blocks that are needed to store it are shown in Table 6. The computer cluster is composed of 20 machines; therefore, when the number of blocks is lower than 20, some of the machines remain idle because they have no HDFS block to process. Moreover, even if the number of blocks is slightly above 20, the affected HDFS blocks may not be evenly distributed among the computing nodes. This demonstrates the capacity of Spark to deal with big databases: as the size of the database (more concretely, the number of HDFS blocks) increases, the framework is able to distribute the processes more evenly, exploiting data locality, increasing the parallelism, and reducing the network overhead.

In order to deal with this problem, the same experiments were repeated over the epsilon dataset, after reorganizing the files with a smaller block size. For each of the eight sets, the block size S_b was calculated according to (6), where s is the size of the dataset in bytes and n_c is the number of cores in the cluster:

$$S_b = 2^K,$$
$$K = \left\lfloor \log_2 \frac{s}{n_c} \right\rfloor. \quad (6)$$

Table 7: Training runtime (in seconds) for the Spark classifiers using epsilon with customized block size.

Threshold	Features	Logistic Regression	Naive Bayes	SVM ($\lambda = 0.0$)	SVM ($\lambda = 0.5$)
0.00	2000	321.94	483.48	300.94	306.47
0.55	721	256.92	313.19	256.72	256.98
0.60	337	**236.16**	**248.59**	**231.23**	**228.03**
0.65	110	307.70	308.26	254.05	261.23

Table 8: AUC values for the Spark classifiers using ECBDL14-ROS.

Threshold	Features	Logistic Regression		Naive Bayes		SVM ($\lambda = 0.0$)		SVM ($\lambda = 0.5$)	
		Training	Test	Training	Test	Training	Test	Training	Test
0.00	631	0.5821	0.5808	0.6714	0.6506	0.5966	0.6046	0.5875	0.5897
0.55	234	**0.6416**	**0.6352**	0.6673	0.6489	**0.6369**	**0.6307**	**0.6228**	**0.6148**
0.60	119	0.6309	0.6235	**0.6732**	**0.6516**	0.5884	0.5841	0.6116	0.6054
0.65	46	0.5017	0.5022	0.6136	0.6093	0.5032	0.5039	0.5000	0.5000

Table 9: Training runtime (in seconds) for the Spark classifiers using ECBDL14-ROS.

Threshold	Features	Logistic Regression	Naive Bayes	SVM ($\lambda = 0.0$)	SVM ($\lambda = 0.5$)
0.00	631	4649.19	1581.52	5283.88	5065.87
0.55	234	2107.66	613.50	2321.22	2179.18
0.60	119	1162.98	322.06	1352.85	1226.72
0.65	46	**978.38**	**215.09**	**914.32**	**864.28**

The runtime for the dataset with the block size customized for each subset is displayed in Table 7 and Figure 8. It is observed that the runtime was smaller than that with the default block size. Furthermore, the curves show the expected behavior: as the number of features of the dataset was reduced, the runtime decreased. In the extreme case (for threshold 0.65), the runtime increased again, because with such a small dataset the synchronization times of Spark become bigger than the computing times, even with the customized block size.

In the next section, MR-EFS is tested over a very large dataset, validating these observations.

4.4. Experiments with the ECBDL14-ROS Dataset. This section presents the classification accuracy and runtime results obtained with the ECBDL14-ROS dataset. As described in Section 4.1, a random oversampling technique [39] was previously applied over the original ECBDL14 dataset to overcome the problems originated by its imbalance. The MR-EFS method was applied using 32 768 map tasks; therefore, each map task computed around 1984 instances.

The obtained results in terms of accuracy are depicted in Table 8 and Figure 9. MR-EFS improved the results in all cases, with different thresholds. The accuracy gain was especially important for the Logistic Regression and SVM algorithms. As expected, the SVM with $\lambda = 0.0$ obtained better results than the one with $\lambda = 0.5$, as the latter attempted to reduce the complexity of the obtained model. However, it is noteworthy that, for 119 features, SVM-0.5 was able to outperform SVM-0.0. This hints that after removing noisy features, the simpler obtained models represented better the true knowledge underlying the data. With even less features, both variants of SVM obtained roughly the same accuracy.

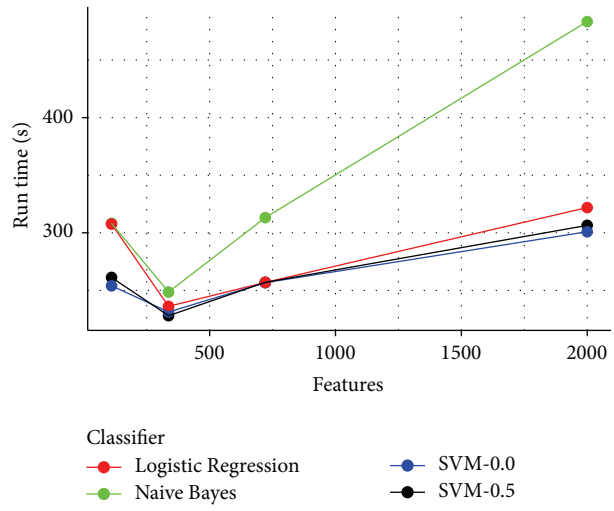

Figure 8: Training runtime with the epsilon dataset with customized block size. Note that the number of features decreases as the threshold increases.

To conclude this study, the runtime necessary to train the classifiers with all variants of ECBDL14-ROS is presented in Table 9 and Figure 10. In this case, the runtime behaved as expected: the time was roughly linear with respect to the number of features, for all tested models. This means that MR-EFS was able to improve both the runtime and the accuracy for all those classifiers.

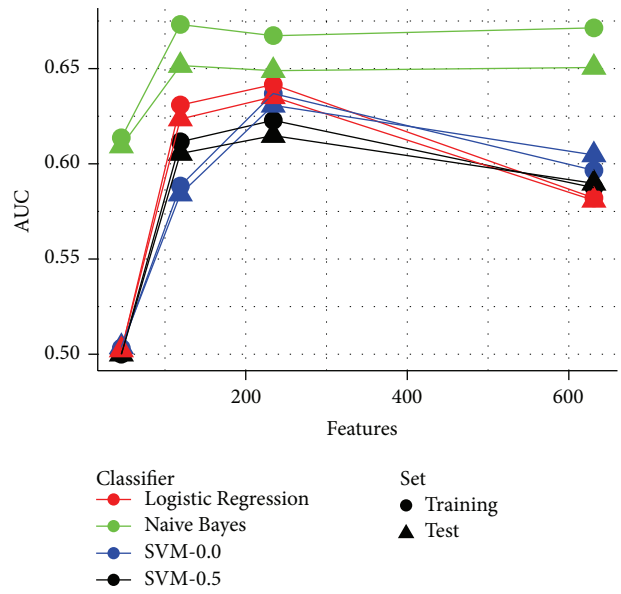

Figure 9: Accuracy with the ECBDL14-ROS dataset. Note that the number of features decreases as the threshold increases.

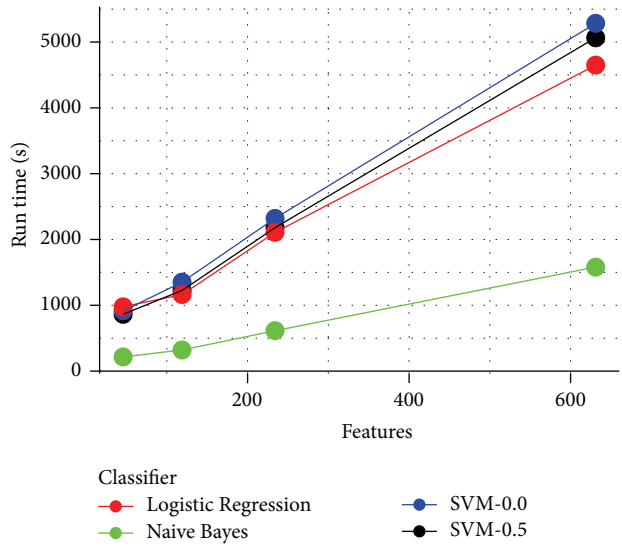

Figure 10: Training runtime with the ECBDL14-ROS dataset. Note that the number of features decreases as the threshold increases.

5. Concluding Remarks

This paper presents MR-EFS, an Evolutionary Feature Selection algorithm designed upon the MapReduce paradigm, intended to preprocess big datasets so that they become affordable for other machine learning techniques, such as classification techniques, that are currently not scalable enough to deal with such datasets. The algorithm has been implemented using Apache Hadoop, and it has been applied over two different large datasets. The resulting reduced datasets have been tested using three different classifiers, implemented in Apache Spark, over a cluster of 20 computers.

The theoretical evaluation of the model highlights the full scalability of MR-EFS with respect to the number of features in the dataset, in comparison with a sequential approach. This behavior has been further confirmed after the empirical procedures.

According to the obtained classification results, it can be claimed that MR-EFS is able to reduce adequately the number of features of large datasets, leading to reduced versions of them, that are at the same time smaller to store, faster to compute, and easier to classify. These facts have been observed with the two different datasets and for all tested classifiers.

For the epsilon dataset, the relation between the reduced datasets size and the number of nodes is forced to modify the HDFS block size, proving that the hardware resources can be optimally used by Hadoop and Spark, with the correct design. One of the obtained reduced ECDBL14-ROS datasets, with more than 67 million instances and several hundred features, could be processed by the classifiers in less than half of the time than that of the original dataset, and with an improvement of around 5% in terms of AUC.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is supported by the Research Projects TIN2014-57251-P, P10-TIC-6858, P11-TIC-7765, P12-TIC-2958, and TIN2013-47210-P. D. Peralta and S. Ramírez-Gallego hold two FPU scholarships from the Spanish Ministry of Education and Science (FPU12/04902, FPU13/00047). I. Triguero holds a BOF postdoctoral fellowship from the Ghent University.

References

[1] E. Alpaydin, *Introduction to Machine Learning*, MIT Press, Cambridge, Mass, USA, 2nd edition, 2010.

[2] M. Minelli, M. Chambers, and A. Dhiraj, *Big Data, Big Analytics: Emerging Business Intelligence and Analytic Trends for Today's Businesses (Wiley CIO)*, Wiley, 1st edition, 2013.

[3] V. Marx, "The big challenges of big data," *Nature*, vol. 498, no. 7453, pp. 255–260, 2013.

[4] A. Fernández, S. del Río, V. López et al., "Big data with cloud computing: an insight on the computing environment, MapReduce, and programming frameworks," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 5, pp. 380–409, 2014.

[5] E. Merelli, M. Pettini, and M. Rasetti, "Topology driven modeling: the IS metaphor," *Natural Computing*, 2014.

[6] S. Sakr, A. Liu, D. M. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *IEEE Communications Surveys and Tutorials*, vol. 13, no. 3, pp. 311–336, 2011.

[7] J. Bacardit and X. Llorà, "Large-scale data mining using genetics-based machine learning," *Wiley Interdisciplinary*

Reviews: Data Mining and Knowledge Discovery, vol. 3, no. 1, pp. 37–61, 2013.

[8] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[9] J. Dean and S. Ghemawat, "Map reduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[10] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pp. 29–43, October 2003.

[11] M. Snir and S. Otto, *MPI—The Complete Reference: The MPI Core*, MIT Press, Boston, Mass, USA, 1998.

[12] W. Zhao, H. Ma, and Q. He, "Parallel k-means clustering based on MapReduce," in *Cloud Computing*, M. Jaatun, G. Zhao, and C. Rong, Eds., vol. 5931 of *Lecture Notes in Computer Science*, pp. 674–679, Springer, Berlin, Germany, 2009.

[13] A. Srinivasan, T. A. Faruquie, and S. Joshi, "Data and task parallelism in ILP using MapReduce," *Machine Learning*, vol. 86, no. 1, pp. 141–168, 2012.

[14] M. Zaharia, M. Chowdhury, T. Das et al., "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, pp. 1–14, USENIX Association, 2012.

[15] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song, "Design and evaluation of a real-time URL spam filtering service," in *Proceedings of the IEEE Symposium on Security and Privacy (SP '11)*, pp. 447–462, Berkeley, Calif, USA, May 2011.

[16] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: SQL and rich analytics at scale," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*, pp. 13–24, ACM, June 2013.

[17] M. S. Wiewiorka, A. Messina, A. Pacholewska, S. Maffioletti, P. Gawrysiak, and M. J. Okoniewski, "SparkSeq: fast, scalable and cloud-ready tool for the interactive genomic data analysis with nucleotide precision," *Bioinformatics*, vol. 30, no. 18, pp. 2652–2653, 2014.

[18] S. García, J. Luengo, and F. Herrera, *Data Preprocessing in Data Mining*, Springer, 1st edition, 2015.

[19] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," *Data Mining and Knowledge Discovery*, vol. 6, no. 2, pp. 153–172, 2002.

[20] J. A. Olvera-López, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, and J. Kittler, "A review of instance selection methods," *Artificial Intelligence Review*, vol. 34, no. 2, pp. 133–143, 2010.

[21] J. S. Sánchez, "High training set size reduction by space partitioning and prototype abstraction," *Pattern Recognition*, vol. 37, no. 7, pp. 1561–1564, 2004.

[22] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[23] H. Liu and L. Yu, "Toward integrating feature selection algorithms for classification and clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 4, pp. 491–502, 2005.

[24] Y. Saeys, I. Inza, and P. Larrañaga, "A review of feature selection techniques in bioinformatics," *Bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 2007.

[25] H. Liu and H. Motoda, *Computational Methods of Feature Selection*, Chapman & Hall/CRC Data Mining and Knowledge Discovery Series, Chapman & Hall/CRC Press, 2007.

[26] J. A. Lee and M. Verleysen, *Nonlinear Dimensionality Reduction*, Springer, 2007.

[27] B. de la Iglesia, "Evolutionary computation for feature selection in classification problems," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 3, no. 6, pp. 381–407, 2013.

[28] L. J. Eshelman, "The CHC adaptative search algorithm: how to have safe search when engaging in nontraditional genetic recombination," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed., pp. 265–283, Morgan Kaufmann, San Mateo, Calif, USA, 1991.

[29] MLlib Website, https://spark.apache.org/mllib/.

[30] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[31] M. Tan, I. W. Tsang, and L. Wang, "Towards ultrahigh dimensional feature selection for big data," *Journal of Machine Learning Research*, vol. 15, pp. 1371–1429, 2014.

[32] T. White, *Hadoop: The Definitive Guide*, O'Reilly Media, 3rd edition, 2012.

[33] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[34] I. Triguero, D. Peralta, J. Bacardit, S. García, and F. Herrera, "MRPR: a MapReduce solution for prototype reduction in big data classification," *Neurocomputing*, vol. 150, pp. 331–345, 2015.

[35] C.-T. Chu, S. Kim, Y.-A. Lin et al., "Map-reduce for machine learning on multicore," in *Advances in Neural Information Processing Systems*, pp. 281–288, 2007.

[36] Pascal large scale learning challenge, http://largescale.ml.tu-berlin.de/instructions/.

[37] Epsilon in the LIBSVM website, http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#epsilon.

[38] ECBDL14 dataset: Protein structure prediction and contact map for the ECBDL2014 Big Data Competition, http://cruncher.ncl.ac.uk/bdcomp/.

[39] S. del Río, V. López, J. M. Benítez, and F. Herrera, "On the use of MapReduce for imbalanced big data using Random Forest," *Information Sciences*, vol. 285, pp. 112–137, 2014.

[40] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and Their Applications*, vol. 13, no. 4, pp. 18–28, 1998.

[41] D. W. Hosmer Jr. and S. Lemeshow, *Applied Logistic Regression*, John Wiley & Sons, 2004.

[42] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, vol. 3, Wiley, New York, NY, USA, 1973.

[43] MLlib guide, http://spark.apache.org/docs/latest/mllib-guide.html.