Progress in Artificial Intelligence manuscript No.
(will be inserted by the editor)

MR-DIS: Democratic Instance Selection for Big Data by MapReduce

Álvar Arnaiz-González · Alejandro González-Rogel · José-Francisco Díez-Pastor · Carlos López-Nozal

Received: date / Accepted: date

Abstract Learning algorithms and pre-processing techniques present difficulties when executed on data sets with a high number of instances. Computational time increases as the volume of information grows due to algorithm complexity: linear O(n), quadratic O(n2)... We propose apply parallelisation design to reduce complexity and test it in a case of study. Democratic Instance Selection(DIS) is a pre-processing algorithm with linear complexity and is based on design paradigm "divide and conquer". This work details how the parallelisation of DIS using MapReduce programming paradigm on Spark architecture has been implemented. To test the scalability of the new implementation two empirical experiments are defined with popular big data sets from the UCI repository. As results of the experiments, the processing time is reduced in a linear manner as the numbers of Spark executors increase. It is tested with various configurations ranging from 4 to 256 executors. In addition, the algorithm development process is publicly accessible to the scientific community.

Keywords big data · MapReduce · Apache Spark · instance selection · democratic instance selection · learning algorithms

1 Introduction

Nowadays, the large quantity of data amassed means that the techniques and tools, which have traditionally been used in the field of data mining and automatic learning, are not applicable. A new term has been coined to describe this problem: *big data*. There is no clear definition of the meaning of *big data*, although Laney [16], at the start of this century, described it as the opportunities and difficulties that appear as the data volumes, variety, and speeds increase. In this definition, the volume refers to massive data volumes that are generated and collected; variety, from the diversity of formats in which we can find the data: structured conventionally, semi-structured, or without any structure, such as video or audio.

Álvar Arnaiz-González
University of Burgos, Avda. Cantabria s/n, Burgos 09006, Burgos, Spain E-mail: alvarag@ubu.es

Finally, speed refers to the fact that data should be processed and processed in an swift way, so that its utilization and exploitation are of commercial value [7]. Another commonly accepted definition describes the problems of *big data* as the problem that appears when the quantity of data to process exceeds the capacities (memory and/or time) of a given system [20].

According to Wu et al. [25], the most acceptable approach for the treatment of massive data sets is their distribution into parallel environments and *cloud* computing platforms. The parallelization of tasks is not at all new, and the "divide and conquer" strategy has been used since the dawn of machine learning. Nevertheless, new *frameworks* have recently arisen that facilitate the task and that help programmers in their work. From among them all, *MapReduce* is, as of today, one of the most popular models of parallel communication, thanks to its robustness and transparency with regard to resource management [8,21].

Data reduction is a direct approximation to the problem of processing large volumes of information, which may be applied in conjunction with any type of algorithmic learning. To do so, instance selection is a commonly used method, which consists of selecting a subset of examples of a complete sample that is capable of maintaining or even improving the predictive capability of the original set [9]. The problem that arises at this point is that the majority of instance selection methods have a high computational complexity (quadratic order in the number of instances or higher), which means that they are impossible to use in data sets of large dimensions [22]. One of the instance selection algorithms that is not subject to this limitation is *Democratic Instance Selection* (DIS) [11].

In this article, the adaptation of the DIS algorithm to the *MapReduce* model is presented through *framework* Apache Spark [26]. In summary, the objectives of the present article are as follows:

- To design a parallelized DIS algorithm following the model *MapReduce*.
- To implement the previously parallelized DIS algorithm in Spark.
- To confirm its correct functioning in terms of accuracy and compression.
- To evaluate its scalability through its execution on large data sets.

The organization of the present article is as follows: Section 2 presents an introduction to instance selection techniques, explaining in detail the algorithm *Democratic Instance Selection* (DIS), Section 3 introduces the *MapReduce* model and its specific application on the DIS algorithm, Section 4 details the experiments conducted by running the parallel algorithm studied in this paper and, finally, Section 5 expounds the conclusions and the principal lines of future work.

2 Instance selection

A fundamental task in knowledge extraction from data sets is their preparation. *Data preparation* groups together a large number of techniques that are necessary as a preliminary step in such learning processes as classification and clustering. Among these data preparation techniques, there is a series of tools for data set size reduction while maintaining the integrity of the original data set. These techniques are grouped under the term data reduction, which, in turn, is divided into: discretization, feature extraction, instance generation, feature selection and instance selection [10]. The objective of all these is to reduce the size of the initial data set, either to reduce the number of instances or to reduce the number of attributes (avoiding the so-called *curse of dimensionality* [15]).

Moreover, both instance selection and instance generation work at the level of instances, reducing their numbers. While instance generation creates new instances and discard existing ones, instance selection (which concerns us in this article) selects the most representative of the original data set. The subset that is selected[1] should be able to maintain, or even improve, the predictive capability of the original set. One of the benefits of size reduction is the corresponding reduction in classification time (for lazy learning algorithms) and training time (for eager-learning algorithms).

Since their appearance [14], multiple instance selection algorithms have been described in the literature. We recommend the work of S. García et al. [9] for readers with an interest in this field. Nevertheless, most of these algorithms suffer from a problem that complicates their application to massive data sets: their computational complexity. The computational complexity of the majority of the above mentioned methods is quadratic, in the number of instances, or greater. Recent algorithms have been developed to overcome these drawbacks [2,3,6,13, 11], among which we will highlight *Democratic Instance Selection* [11], which is analysed in detail in the following section.

2.1 Democratic Instance Selection

The algorithm *Democratic Instance Selection* o DIS [11] is based on the classic idea of divide and conquer. Its execution consists of the completion of a number of rounds r, in each of which the original set is divided into a series of partitions (called bins), to each of which an instance selection algorithm is applied.

In greater detail, in each round, the algorithm divides the original set into a number of disjoint subsets (of approximately the same size). A classic instance selection algorithm is applied in an independent way to each subset. The instances selected by the latter algorithm to be eliminated receive a vote. This process is repeated during a predefined number of rounds (chosen by the user), after which the instances with a number of votes equal to or higher than the calculated threshold are eliminated.

The biggest advantage of DIS is the reduction in execution time thanks to its computational complexity: linear in the number of instances. The detail of how this calculation is obtained can be seen in the subsection 2.1.3.

Another plus point is the ease with which the algorithm can be adapted to a parallel environment, because the execution of each instance selection process on each partition is independent from the others. It is worth noting that, as the size of the subsets is parametrizable, the user can select the amount and the size of the subsets in such a way that it is adjusted to the capabilities of the system where the algorithm will be executed.

Pseudocode 1 presents the original algorithm, which is divided into two parts that are analysed below:

[1] The subset selected by the algorithm is indistinctly referred to as filtered and selected in the present article.

– Partitioning and voting: the input set is divided into disjoint subsets and the desired algorithm is applied to each one of them. This process is repeated as many times as there are rounds. The votes of those instances selected by each algorithm are stored for subsequent processing.
– Selection of instances: the threshold of votes is calculated in accordance with a *fitness function* and all instances with a number of votes equal to or over that threshold are removed.

Algorithm 1: Democratic Instance Selection (DIS) algorithm

```
Input: A training set $T = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, subset size $s$, number of rounds $r$
Output: Selected set $S \subseteq T$
1 for $k = 1$ to $r$ do
2     Divide instances into $n_s$ disjoint subsets $t_i$: $\bigcup_i t_i = T$ of size $s$
3     for $j = 1$ to $n_s$ do
4         Apply instance selection to $t_j$
5         Store votes of removed instances from $t_j$
6 Obtain threshold of votes, $v$, to remove an instance
7 $S = T$
8 Remove from $S$ all instances with a number of votes $\geqslant v$
9 return $S$
```

An example of how the DIS algorithm operates can be seen in Figure 1. The original set contains 30 instances, 10 rounds were performed and, finally, all those with a number of votes below 5 were selected (in other words, all those with a number of votes equal to or greater than 5 were eliminated).

2.1.1 Data set partition

An important stage in the method is partitioning of the initial set T into smaller subsets t_i, which comprise the totality of the set $\bigcup_i t_i = T$. The size of the subsets is selected by the users upon whom the execution time will, in large part, depend. It is worth highlighting that the execution time of the complete algorithm depends on the size of the subset with the greatest number of instances. Steps should therefore be taken so that each subset has, approximately, the same number of instances.

The simplest partitioning method is by random selection, in which each instance is assigned a subset in a random manner. The problem with this method is that the neighbourhoods are not maintained from one round to another, in other words, instances that are neighbours in one round do not have to be so in the following round. However, this method has been selected because of its easy implementation and speed.

2.1.2 Determining the threshold of votes

An important decision is estimation of the threshold that will determine the instances that are selected for the filtered set. Experimentation reported by García-Osorio et al. [11] demonstrated that this value depends on the data set under analysis. Therefore, it is not possible to apply a pre-established value for any set

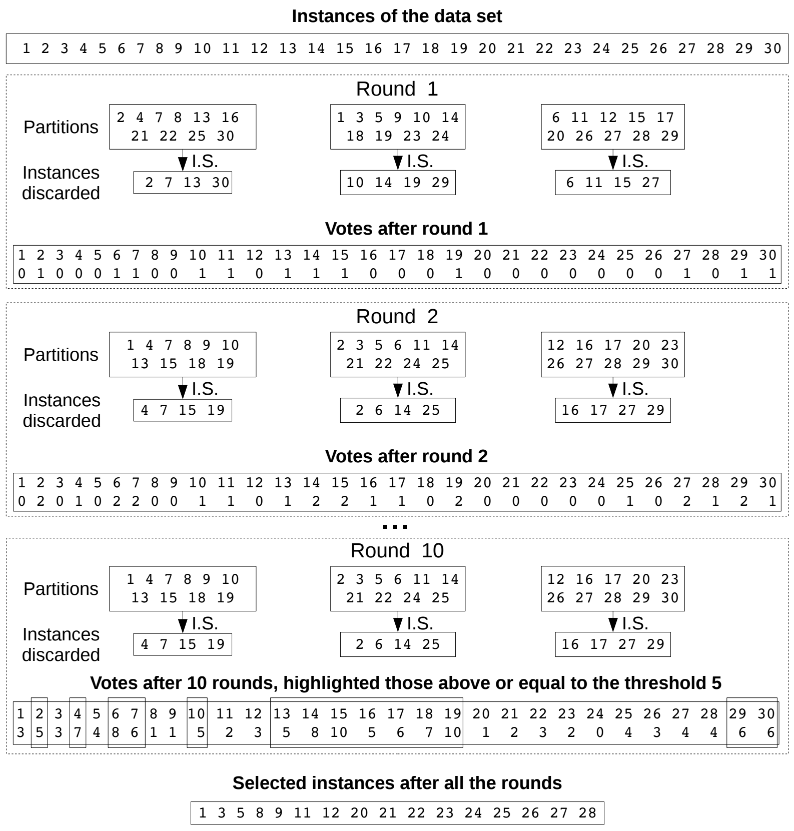


Fig. 1 Example of a DIS algorithm execution on a set of 30 instances. The algorithm executes 10 rounds and the calculated threshold of votes is 5, so that all those instances with a lower number of votes are selected for the selected set.

and a function is necessary that selects the value directly from the initial set in execution time. The most intuitive approach would be to perform a cross-fold validation on the initial set, although it would be tremendously costly in terms of time.

The chosen method is much lighter. It estimates the value of the most appropriate number of votes on the basis of the initial set. When computing the threshold, two criteria are taken into account: training error ϵ_t (on a subset of the initial set) and the resulting size m of the possible solution set. As is logical, both values should be minimized. A criterion is defined in that way, called fitness or $f(v)$, which is the combination of both, as shown in the following equation:

$$f(v) = \alpha\epsilon_t(v) + (1-\alpha)m(v)$$

where m is the percentage of instances conserved by the algorithm, ϵ_t is the training error and α is a parameter in the interval $[0, 1]$. This parameter represents the relative importance of each of the two previously mentioned criteria. The user-selected α value means the user has the choice to maximize either accuracy or reduction.

Calculation of error can be a costly process. To minimize this cost, the error is estimated by using a small percentage of the complete training set, which can be, according to the authors, 10% for large sets and 0.1% for massive sets.

The process to obtain the threshold is as follows: the votes received by each instance after r number of rounds completed. The threshold to know whether an instance should be eliminated will be the value $v \in [1, r]$ which minimizes the function $f(v)$. Once calculated, all the instances are eliminated that have the same number of votes greater or equal to the threshold v.

2.1.3 Complexity analysis

The DIS algorithm divides the data set into disjoint sets of a fixed size s, such that the instance selection algorithm used during the voting, whatever its complexity, is applied to a set of fixed size. Let K be the number of operations necessary to complete the selection of instances in a set of size s, for a data set of size n if we do r rounds the total time is proportional to $r(n/s)K$, which is lineal as K is a constant value.

In addition to the selection of instances applied in each partition, there are another two processes: the partitioning of the original set and the determination of the number of votes. The partitioning is random, so its complexity is $\mathcal{O}(1)$. With regard to the decision on the number of votes, if all the instances were considered, the cost of this step would be $\mathcal{O}(n^2)$, however the number of instance that are considered decreases as the size increases (for example 10% of the training set in medium-sized sets, 0.1% in large sets, etc.). Therefore, the complexity of this step and of the whole algorithm is maintained.

3 MapReduce

MapReduce emerged as a programming model that approaches the problem of processing large data sets from the perspective of parallel computing. Its operation is based on the use of pairs ⟨key, value⟩ and on the division of problems into two phases: *Map* and *Reduce*. The *Map* stage operates on the initial pairs ⟨key, value⟩ to generate other intermediate ⟨key, value⟩ values. Subsequently, these data are combined according to their key in the *Reduce* phase, yielding the final result [8].

The tendency to work with ever larger amounts of data, its high tolerance to failure, and its transparency in relation to resource management, have converted this model into one of the most widely used over recent years [19].

Apache Hadoop and Apache Spark are two of the most popular *frameworks* at present for massive processing in *big-data* environments. They both implement MapReduce and both use a master-slave architecture where a single node (master) is in charge of managing a variable number of worker nodes (slaves). Nevertheless, Spark has a clear advantage over Hadoop given that it is designed to make iterative operations faster through the intensive use of memory (instead of using only disk as Hadoop does). This means it is more appropriate for Data Mining and Machine Learning tasks, where it is able to multiply the performance of Hadoop by 10 or 100 times [26].

Currently, there is a a lack of instance selection methods in Spark [23]. This is one of the reasons that brings added value to the present paper, providing

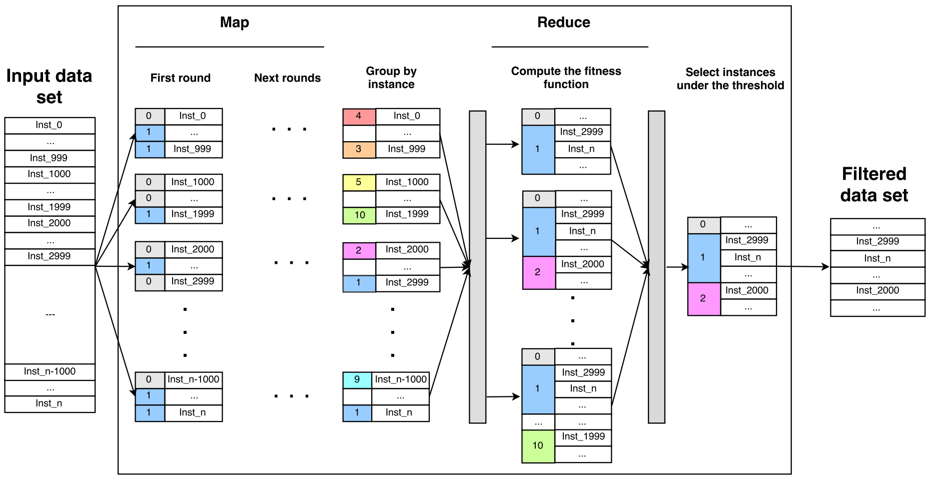


Fig. 2 Example of a DIS algorithm execution following the MapReduce paradigm.

the community with a new parallel algorithm, and its implementation, for the preprocessing of large volumes of information in Spark.

3.1 MR-DIS: the MapReduce design for Democratic Instance Selection

The *MapReduce* parallelization technique explained earlier was used for the adaptation of the DIS algorithm to the *big data* environment.

With regard to the structure of the data, pairs ⟨votes, inst⟩ were used to operate throughout the *Map* and the *Reduce* phases. The value "inst" corresponds to a concrete instance, while the value "votes" refers to the number of times that the instance that accompanies it has been selected during the voting phase.

The algorithm is described with a dynamic view in figure 2, and with two static views in pseudocodes 2 and 3. Figure 2 presents a possible scenario where 10 votes are given over n instances and a threshold of 3 votes is calculated for the final selection. In both cases, the two characteristic phases of the *MapReduce* model are differentiated, which are analysed as follows:

– **Map phase:** Completes the rounds of voting, updating the value of the key "votes" of the pairs in each iteration. Each *mapper* receives a partition of the original set, upon which to apply an instance selection algorithm, updating the votes of those instances that the algorithm selects. The input of each one of the algorithms is, therefore, one of the disjoint sets described in line 4 of the pseudocode 1. In the current implementation, the distribution of the data is random, but the size of the partitions is selected by the user. The user also decides on the instance selection algorithm and it is applied in an independent manner in each partition. The number of maps (num_maps) that are executed is obtained with the following formula:

$$\text{num\_maps} = r \times \left\lceil \frac{|T|}{|t_i|} \right\rceil$$

where, r is the number of rounds, $|T|$ is the number of instances of the original set and $|t_i|$ is the size of each one of the partitions.

Algorithm 2: MR-DIS: Map function

```
  Input: A subset of the original training set $T_j$
  Output: Array of pairs $V_j = \{\langle \text{votes}_1, \mathbf{x}_1 \rangle, ..., \langle \text{votes}_n, \mathbf{x}_n \rangle\}$
1 Apply instance selection algorithm to $T_j$
2 foreach instance $\mathbf{x} \in T_j$ do
3     if $\mathbf{x}$ has been selected then
4         Add to $V_j$ the pair $\langle 1, \mathbf{x} \rangle$
5     else
6         Add to $V_j$ the pair $\langle 0, \mathbf{x} \rangle$
7 return $V_j$
```

An intermediate grouping step occurs between the map and the reduce functions[2]. In this process, the ⟨votes, inst⟩ pairs are grouped by instance, adding through the summaries the votes that each of them have received in the successive rounds. The instances are distributed to the *reducers* in accordance with the number of their votes, creating as many groups as there are rounds r that have been completed.

– **Reduce phase:** Includes the calculation of the threshold of votes. Each *reducer* completes the calculation of the fitness function for a certain number of votes $k \in [1, r]$ (where r is the number of rounds). The input data set of each *reducer* is determined by the number of votes of each instance, in other words, the times that the instance has been selected by the algorithm in the map phase. An estimation of the accuracy of the subset and of the compression rate is necessary to calculate the fitness function. Compression is directly calculated, taking into account the number of instances of the original set and of the selected subset. It is necessary to apply a classification algorithm to calculate the estimation of accuracy, which could imply a problem with large data sets. A parallel implementation of the algorithm k-NN [19] over a percentage of the intial set of instances was used to avoid the possible bottleneck in this calculation. The number of *reducers* that will be run will be determined by the number of rounds r.

Once the values of the fitness function have been calculated for each of the values $k \in [1, r]$, the lowest value is selected. The last step of the algorithm is the generation of the set result, which is constructed by selecting all those instances with a number of votes below the threshold (less fitness), in other words, all those instances whose number of votes is equal to or higher than the threshold.

[2] In Spark, the process located between the map and the reduce phases is usually referred to as *shuffle*.

Algorithm 3: MR-DIS: Reduce function

```
**Input:** Array of pairs $V_k = \{\langle \text{votes}_1, \mathbf{x}_1\rangle,..., \langle \text{votes}_n, \mathbf{x}_n\rangle\}$, test set $X \subset T$, round key
       $k$, $\alpha$ value
**Output:** Fitness value $f_k$
1 Train $k$NNIS with $V_k$
2 $\varepsilon \leftarrow \mathsf{Error}(k\texttt{NNIS}, X)$
3 $m \leftarrow \frac{|V_k|}{|T|}$
4 $f_k = \alpha\varepsilon + (1-\alpha)m$
5 **return** $f_k$
```

4 Experimental set-up

The objective of this section is the study of the scalability of the DIS algorithm implemented in Spark. It was taken into account that the results for both accuracy (% accuracy) and compression (% of instances of the original group selected in the filtered set) were similar to those obtained through sequential implementation, in order to validate that the parallel implementation functions in a similar way to its sequential version. The k-nearest neighbour was used for the calculation of accuracy (in its version available for Spark [19]).

The implementation of the DIS was done with Scala on Spark 1.6.1 and is publicly accessible at BitBucket[3].

The hypothesis that is proposed in the experiment is as follows:

Hypothesis 1 *The increase in the number of processors in the execution of the MR-DIS algorithm reduces execution time in large data sets.*

Two case studies have been developed, with the intention of proving this hypothesis:

- Small data set: the algorithm was executed on various configurations of *executors*[4], from 4 up to 256, on a medium-size data set. Accuracy, compression and time of filtering were all measured.
- Big data sets: The filtering times of the MR-DIS algorithm with configurations of *executors* from 64 to 256 were measured on two data sets of up to one million instances.

The structure of the present section is as follows: subsection 4.1 defines the framework of experimentation; subsection 4.2 details the instrumentation; and, finally, the case studies are presented in subsections 4.3 and 4.4.

4.1 Experimental framework

The experimental study is centred on verifying the scalability of the algorithm that is implemented. To do so, the algorithm was executed with a varying number of executors. It is expected that parallelization will not affect the general operation

[3] Author: Alejandro González-Rogel, `https://bitbucket.org/agr00095/tfg-alg.-seleccion-instancias-spark`.

[4] in the Spark framework, each worker node has one or more executors, each one of which completes a task. A processor was assigned to each executor in the experimental work.

of the algorithm. The following measures were extracted from a 10 *fold* cross validation without repetition:

- Mean accuracy: Percentage accuracy calculated on a 1-NN classifier trained with the subset selected by the MR-DIS algorithm.
- Average Compression: Percentage of instances of the original set present in the selected subset.
- Filtering time: Time spent by the instance-selection algorithm on the filtering (subset selection).
- Speed up: Measures the efficiency of the parallel version with respect to the sequential version. It is calculated by dividing the execution time of the sequential version by the time of the parallel version. The speed up of a single processor is one, such that it is easy to guess that the maximum theoretical speed up that may be obtained is the same as the number of processors in use. Nevertheless, the speed up is normally lower than the number of processors as the system tends to become over-saturated [12]. In other words, efficiency falls as the numbers of processors increase, which is known as Amdahl's law [1].

$$\text{speed\_up} = \frac{\text{base\_line}}{\text{parallel\_time}}$$

In all the case studies, the parameters used in the execution of the MR-DIS algorithm were as follows:

- Algorithm used: Condensed Nearest Neighbor (CNN) [14].
- Number of rounds: 10.
- Number of instances by partition: 1 000.
- Percentage of instances for the error calculation: 1%[5].
- Alpha Value: 0.75.

4.2 Instrumentation

The experimentation was performed using the Apache Spark 1.6.1 distribution on the Google DataProc service[6]. Various clusters were deployed, each one with twice the number of processing cores than the earlier ones, from 4 to 256. The cluster computation nodes were formed of Intel Xeon E5 with 16 cores and 60 GB of RAM. Additionally, each cluster has a master node in charge of administration and management of the *executors*. The master node was an Intel Xeon E5 with 2 cores and 7.5 GB of RAM.

Each *executor* was assigned a core. The collections of data used by Spark (RDDs) were stored under the persistence level `MEMORY_ONLY` of the *framework*. This storage implies that the data were stored without serialization in memory and, in case of there being no space, some partitions of these structures can be eliminated and recalculated whenever necessary.

[5] In [11] the percentage of instances used for error estimation in massive data sets was 0.1%, but the use of a parallel implementation of 1-NN permits an increase in this percentage, improving the precision of the estimation.

[6] Google Cloud Platform: `https://cloud.google.com/dataproc/`

Table 1 Results of the experimentation with 10% of the Susy data set.

# executors	Comp. (%)	Acc. (%)	Filt. time (s)	Speed up
4	21.79	66.58	6 877.92	1.81
8	21.80	66.61	3 774.22	3.30
16	21.81	66.56	1 641.45	7.58
32	21.81	66.52	995.02	12.50
64	21.80	66.52	640.44	19.42
128	21.80	66.53	342.74	36.29
256	21.78	66.61	276.70	44.95

4.3 Small experiment

One of the objectives of the present article is to test the scalability and the correct distribution of the work between the nodes, in such a way that the processing time is reduced in a linear manner as the numbers of executors increase. It was tested with various configurations ranging from 4 to 256 executors. A subset of 10% of the Susy data set [18] was selected to be able to run the job on such a small number of executors within a reasonable time. Susy comprises 5 000 000 instances produced by a physics particle simulation. Each instance is formed of 18 attributes and is a binary classification problem.

Table 1 shows the results of those executions. The speed up was calculated by using the execution time of MR-DIS with a single executor as the lineal base. This arrangement means that the linear base is a lot quicker than a possible sequential external implementation with Spark, as access to the data sets is done in a more efficient manner (access is not sequential line by line but it is able to read and to load blocks of instances in memory) [19].

Additionally, speed up is shown in graph form in figure 3. It may be seen that, as the numbers of executors increase, the speed up slope increases up until a limit at 128. As from that point, speed up is lower than might be expected. This lower speed is because 10% of the Susy data set is not sufficiently large to be able to give work to all of the available nodes, and part of the parallelization power is not exploited.

It is worth mentioning that the slight variations in reduction and accuracy that may be seen in table 1, are due to the indeterminism generated by parallel execution. In other words, there is no exact order in which the operations are executed and, this indeterminate process usually means that the partitioning or selection of instances from the initial set is different in each iteration.

4.4 Big data sets

In the previous subsection, the experiment carried out on the subset of 10% of the Susy data set has demonstrated that the implementation maintains accuracy and compression regardless of the numbers of executors that are used. In this subsection, the execution was done with clusters of 64, 128, and 256 executors on two data sets of up to one million instances.

Two popular data sets from the UCI [18]repository were used: Cover type and Poker Hand.

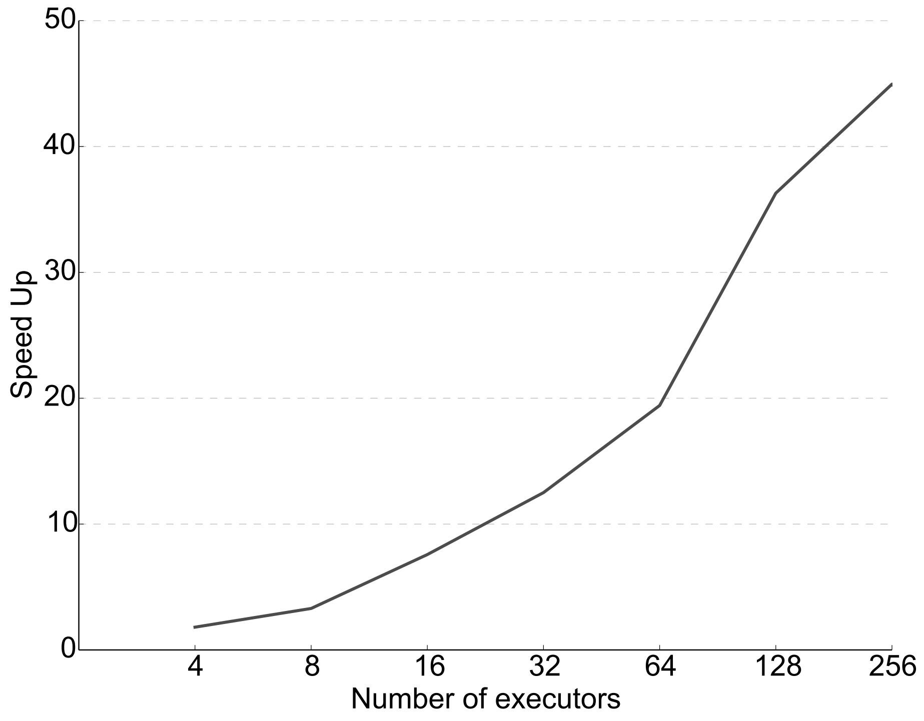


Fig. 3 Evolution of speed up with 10% of Susy as the numbers of executors increase. The scale of the horizontal axis is not linear and its number is duplicated at each step.

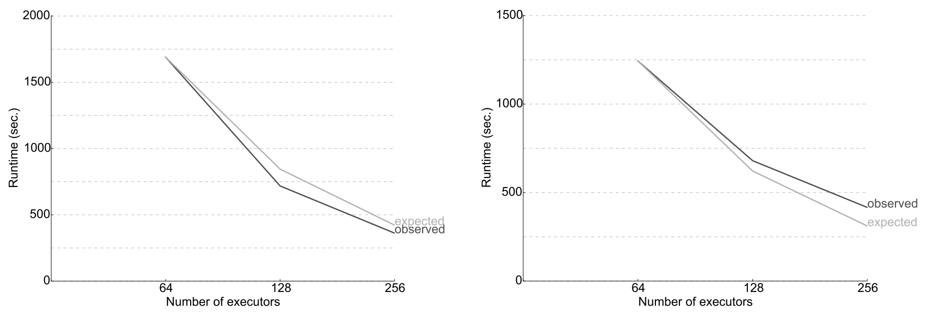
Cover type is formed of 581 012 instances, which describe 30×30 square meter regions in terms of cartographic variables. The value to predict is forestry coverage. There are 7 types of forestry cover, in other words, it is a problem with 7 classes. The attributes collected for each sample are summarized in 54 nominal and numeric attributes.

Poker Hand is composed of 1 025 010 instances, each represents an example of a hand consisting of five of the 52 possible cards from the deck. Each card is represented by two attributes (suit and rank), so the set has ten attributes: five numeric (from 1 to 13) and five categorical with four possible values (Hearts, Spades, Diamonds, Clubs). It has ten classes that indicate the possible game: one pair, two pairs, flush, and so on.

Table 2 shows the evolution over time of filtering and classification, in seconds, for Cover type and Poker. The time to apply the selection (filtered) of instances and the classification time (using MR-kNN) over the selected set. In both cases, it may be appreciated that the time is shortened in a lineal way as the number of executors is doubled. In addition, the algorithm execution time is shown in a graph in figure 4. A line (expected) has been drawn to facilitate the comparison, which takes the time with 64 executors as its origin and is halved as the numbers of executors increase.

Table 2 Average filtered and classification time for the data set types Poker and Cover Type.

# executors	Filtering time (s)		Classification time (s)	
	Cov.Type	Poker	Cov.Type	Poker
64	1 689.39	1 243.38	125.98	330.21
128	717.85	684.33	42.99	160.48
256	361.93	415.75	27.61	78.61



(a) Cover type (b) Poker

Fig. 4 Evolution over time of filtering in the data sets Cover type and Poker as the number of executors increases. The filtering time (observed) and the expected linear progression are shown. In order to facilitate the comparison, a gray line (expected) is drawn that shows the time with 64 executors as its origin and is halved as the number of executors is doubled. The scale of the horizontal axis is not linear, and at each step its number is doubled.

5 Conclusions and further work

In this work, a Spark implementation of the algorithm of selection of instances known as Democratic Instance Selection has been presented, the implementation has been done according to the model of MapReduce. This algorithm was selected for two reasons: on the one hand its complexity, lineal in the number of instances, and on the other, its intuitively parallelizable design.

During the implementation of the method, Spark's ability to distribute work between different machines has become clear. The work of the developer is to implement the algorithm following the MapReduce model, but Spark framework is responsible for assigning the work load of each node in a transparent way.

The experimentation has demonstrated the scalability of the MR-DIS implementation and the possibility of its execution on large dimensional data sets. Since it uses the parallel implementation of kNN internally, the MR-DIS is totally scalable on massive data sets without having any bottleneck that weakens its scalability. In addition, the code is publicly available at the Bitbucket repository.

Despite this first version only having implemented and tested one instance-selection algorithm (CNN), the immediate project in which we are working is the addition of new algorithms such as ICF [5], DROP [24], LSBo [17]... that are more novel, rapid and efficient.

Moreover, the random partitioning method is not the most intelligent. In the original paper where DIS was presented, the authors proposed another partitioning alternative based on *Grand Tour* theory [4]. Its execution under the *MapReduce* theory is a line of investigation that is proposed for the future.

Acknowledgements This work was funded by the Ministry of Economy and Competitiveness, project TIN 2015-67534-P.

References

1. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, AFIPS '67 (Spring), pp. 483–485. ACM, New York, NY, USA (1967). DOI 10.1145/1465482.1465560
2. Angiulli, F., Folino, G.: Distributed Nearest Neighbor-Based Condensation of Very Large Data Sets. IEEE Transactions on Knowledge and Data Engineering **19**(12), 1593–1606 (2007). DOI 10.1109/TKDE.2007.190665
3. Álvar Arnaiz-González, Díez-Pastor, J.F., Rodríguez, J.J., García-Osorio, C.I.: Instance selection of linear complexity for big data. Knowledge-Based Systems **107**, 83 – 95 (2016). DOI 10.1016/j.knosys.2016.05.056
4. Asimov, D.: The grand tour: A tool for viewing multidimensional data. SIAM J. Sci. Stat. Comput. **6**(1), 128–143 (1985)
5. Brighton, H., Mellish, C.: Advances in instance selection for instance-based learning algorithms. Data Mining and Knowledge Discovery **6**(2), 153–172 (2002). DOI 10.1023/A:1014043630878
6. Cano, J.R., Herrera, F., Lozano, M.: Stratification for scaling up evolutionary prototype selection. Pattern Recognition Letters **26**(7), 953 – 963 (2005). DOI 10.1016/j.patrec.2004.09.043
7. Chen, M., Mao, S., Liu, Y.: Big Data: A Survey. Mobile Networks and Applications **19**(2), 171–209 (2014). DOI 10.1007/s11036-013-0489-0
8. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Commun. ACM **51**(1), 107–113 (2008). DOI 10.1145/1327452.1327492
9. Garcia, S., Derrac, J., Cano, J., Herrera, F.: Prototype Selection for Nearest Neighbor Classification: Taxonomy and Empirical Study. Pattern Analysis and Machine Intelligence, IEEE Transactions on **34**(3), 417–435 (2012). DOI 10.1109/TPAMI.2011.142
10. García, S., Luengo, J., Herrera, F.: Data Preprocessing in Data Mining. Springer Publishing Company, Incorporated (2014)
11. García-Osorio, C., de Haro-García, A., García-Pedrajas, N.: Democratic instance selection: A linear complexity instance selection algorithm based on classifier ensemble concepts. Artificial Intelligence **174**(5–6), 410 – 441 (2010). DOI 10.1016/j.artint.2010.01.001
12. Grama, A.Y., Gupta, A., Kumar, V.: Isoefficiency: Measuring the scalability of parallel algorithms and architectures. IEEE Parallel Distrib. Technol. **1**(3), 12–21 (1993). DOI 10.1109/88.242438
13. de Haro-García, A., García-Pedrajas, N.: A divide-and-conquer recursive approach for scaling up instance selection algorithms. Data Mining and Knowledge Discovery **18**(3), 392–418 (2009). DOI 10.1007/s10618-008-0121-2
14. Hart, P.: The condensed nearest neighbor rule (corresp.). Information Theory, IEEE Transactions on **14**(3), 515 – 516 (1968)
15. Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC '98, pp. 604–613. ACM, New York, NY, USA (1998). DOI 10.1145/276698.276876
16. Laney, D.: 3-d data management: controlling data volume, velocity and variety. Tech. rep., META Group Research Note (2001)
17. Leyva, E., González, A., Pérez, R.: Three new instance selection methods based on local sets: A comparative study with several approaches from a bi-objective perspective. Pattern Recognition **48**(4), 1523 – 1537 (2015). DOI 10.1016/j.patcog.2014.10.001
18. Lichman, M.: UCI machine learning repository (2013). URL `http://archive.ics.uci.edu/ml`

19. Maillo, J., Ramírez, S., Triguero, I., Herrera, F.: kNN-IS: An iterative spark-based design of the k-nearest neighbors classifier for big data. Knowledge-Based Systems (2016). DOI 10.1016/j.knosys.2016.06.012
20. Minelli, M., Chambers, M., Dhiraj, A.: Big data, big analytics: emerging business intelligence and analytic trends for today's businesses. John Wiley & Sons, Inc. (2012). DOI 10.1002/9781118562260.fmatter
21. Ramírez-Gallego, S., García, S., Mouriño Talín, H., Martínez-Rego, D., Bolón-Canedo, V., Alonso-Betanzos, A., Benítez, J.M., Herrera, F.: Data discretization: taxonomy and big data challenge. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **6**(1), 5–21 (2016). DOI 10.1002/widm.1173
22. Triguero, I., Peralta, D., Bacardit, J., García, S., Herrera, F.: Mrpr: A mapreduce solution for prototype reduction in big data classification. Neurocomputing **150, Part A**, 331 – 345 (2015). DOI 10.1016/j.neucom.2014.04.078
23. Tsai, C.F., Lin, W.C., Ke, S.W.: Big data mining with parallel computing: A comparison of distributed and mapreduce methodologies. Journal of Systems and Software **122**, 83 – 92 (2016). DOI 10.1016/j.jss.2016.09.007
24. Wilson, D.R., Martinez, T.R.: Instance pruning techniques. In: Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97), pp. 404–411. Morgan Kaufmann (1997)
25. Wu, X., Zhu, X., Wu, G.Q., Ding, W.: Data mining with big data. IEEE Transactions on Knowledge and Data Engineering **26**(1), 97–107 (2014). DOI 10.1109/TKDE.2013.109
26. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster Computing with Working Sets. HotCloud **10**, 10–10 (2010)