Cognitive Computation (2019) 11:388–399
https://doi.org/10.1007/s12559-019-09632-4

Evolutionary Design of Linguistic Fuzzy Regression Systems with Adaptive Defuzzification in Big Data Environments

Samuel López[1] · Antonio A. Márquez[2] · Francisco A. Márquez[3] · Antonio Peregrín[3]

Received: 24 April 2018 / Accepted: 5 March 2019 / Published online: 19 March 2019

Abstract
This paper is positioned in the area of the use of cognitive computation techniques to design intelligent systems for big data scenarios, specifically the use of evolutionary algorithms to design data-driven linguistic fuzzy rule-based systems for regression and control. On the one hand, data-driven approaches have been extensively employed to create rule bases for fuzzy regression and control from examples. On the other, adaptive defuzzification is a well-known mechanism used to significantly improve the accuracy of fuzzy systems. When dealing with large-scale scenarios, the aforementioned methods must be redesigned to allow scalability. Our proposal is based on a distributed MapReduce schema, relying on two ideas: first, a simple adaptation of a classic data-driven method to quickly obtain a set of rules, and, second, a novel scalable strategy that uses evolutionary adaptive defuzzification to achieve better behavior through cooperation among rules. Some different regression problems were used to validate our methodology through an experimental study developed and included at the end of our paper. Therefore, the proposed approach allows scalability while tackling applications of linguistic fuzzy rule-based systems for regression with adaptive defuzzification in large-scale data scenarios. This paper thus examines the use of some relevant techniques for cognitive computing when working with a vast volume of examples, a common occurrence when dealing with the design of artificial intelligent systems that perform reasoning in a similar way as humans.

Keywords Linguistic fuzzy modeling · Evolutionary fuzzy systems · Fuzzy regression · Defuzzification · Big data · MapReduce · Apache Spark

Introduction

The emerging discipline of cognitive computation deals with artificial reasoning systems that interact with humans in complex situations, mimicking human rational processes as autonomously as possible in order to improve people's productivity in many areas.

To do so, this interdisciplinary research area employs and combines models from various areas, many of them involving computational intelligence developments as natural and biologically inspired methodologies [1–3]. They use knowledge learned from both people and data by means of machine learning and data mining, among others. Cognitive computing systems entail many challenges [4], some of them closely related to large-scale [5], high dimensional [6] and big data [7] problems.

Fuzziness is an inherent feature of cognitive information, due to the incomplete cognition of human beings [8]. On the other hand, decision-making ability is one of the main characteristics of cognitive computation and in this sense, fuzzy logic helps bring computer reasoning closer to its human counterpart [9].

Evolutionary algorithms [10, 11] are biologically inspired methods that have been widely used, particularly together with fuzzy systems, forming the core of the soft computing area. Both together are particularly useful due to the good

✉ Antonio Peregrín
peregrin@uhu.es

Antonio A. Márquez
amarquez@dti.uhu.es

Francisco A. Márquez
alfredo.marquez@dti.uhu.es

[1] Escuela Técnica Superior de Ingeniería, University of Huelva, 21007 Huelva, Spain

[2] Departamento de Tecnologías de la Información, University of Huelva, 21007 Huelva, Spain

[3] Centro de Estudios Avanzados en Física, Matemáticas y Computación, University of Huelva, 21007 Huelva, Spain

practical results, developing the area named evolutionary fuzzy systems (EFSs). Relevant current reviews in this area can be found in [12–14]. Also, some recent papers in this area related with other current hot topics could be found in [15] which is about explainable artificial intelligence and EFS; or applications such as [16], related with intrusion detection systems with EFS; or [17], discussing the fuzzy modeling and control of micro-air vehicles using evolutionary algorithms; [18] is also about learning TSK fuzzy systems with evolutionary algorithms for high dimensional datasets.

Nonetheless, evolutionary algorithms in general terms have the drawback of a substantial computational cost, due to the iterative searching method employed. Furthermore, when learning fuzzy systems rule bases (RBs), the high number of examples (large scale problem) implies an exponential rule growth, i.e., as many defuzzification parameters as rules, significantly expanding the search space and thus the time required. Moreover, calculating the fitness function when using an enormous number of training examples take a lot of computational time.

The philosophies usually followed in the EFS area to handle large amounts of data are basically the following [14]:

(a) Staking out the algorithms: the inner mechanisms of the evolutionary algorithms should be rebuilt.
(b) Reducing the amount of data: this way, the evolutionary procedure should require lower computational effort.
(c) Using distributed computing: the use of clusters of computers to decrease the time employed.

Therefore, in the EFS design area, but not exclusively, one important challenge is the development of methods and algorithms for enormous data volumes capable of doing the same work as small data approaches [19, 20], or quality improvements due to the higher computational power.

Although many papers on EFSs for big data focused on classifiers [21–26] can be found, only a few proposals are devoted to regression problems [27–30]. However, [27, 28] are not of linguistic models but Takagi-Sugeno. The recent interest in scalability was preceded by some previous proposals around sizeable datasets [31–35] based on different kinds of approach, such as reducing the training data or decreasing the searching space. Therefore, they are not really scalable proposals, that is, offering a wide range of applicability while increasing the size of the dataset, based on increasing the computational power without varying the design of the algorithm.

Focusing on the design area of fuzzy rule-based systems (FRBSs) problems, there have been many different methods to improve their accuracy [12, 14]. Some of them are based on the use of custom aggregation operators [36, 37]. Attending to regression problems, one of the most well-known, and normally also compatible with the others, is the use of adaptive fuzzy operators in the inference system [38, 39], and particularly adaptive defuzzification [40]. The accuracy improvements it provides derive from the adaptation of defuzzification to each particular rule, fine-tuning their specific relevance, i.e., making the rule set be more cooperative. This is especially interesting when the RB has been learned using methods based on covering criteria and the best rule of its area. One of these methodologies is the widely used and well-known WM-method [41]. Therefore, the blend of this simple RB learning procedure with the adaptive defuzzification achieves a good combination, particularly when using evolutionary algorithms [40].

However, the use of this approach when the problems involve huge data volumes is a challenge. In that situation, the first step, which is RB learning through adaptation of the WM-method, was conceptually resolved in our conference paper [30]. There, a model that gets the same RB as that which could be learned with the sequential original model was presented and tested with an Apache Hadoop implementation. The second step, in [30], is a preliminary study devoted to an evolutionary adaptive defuzzification (EAD) method, but there, the proposed model suffered from lack of precision in terms of the accuracy achieved by the equivalent traditional sequential model.

Attending particularly to the aforesaid second step, in this work, we introduce an original new EAD method which improves upon the approach presented in [30]. Specifically, it definitively enhances the previous work in terms of accuracy and scalability based on a substantially different distributing scheme. Now, it is a single evolutionary process with distributed population evaluation, designated *global learning model* [42], instead of the multiple distributed evolutionary processes, called the *local learning model* [42], used in [30]. Also, a specific Apache Spark [43] implementation in place of the Apache Hadoop employed in [30] is proposed, in order to perform the iteration efficiently over the distributed loop needed by the new learning model.

To verify the behavior of our proposal, we carried out an experimental study. We used 12 regression problems and measured the performance in terms of computational cost and accuracy. Furthermore, to confirm the advantages of the new model, we compared it against the preliminary distributed EAD presented in [30], applying statistical tests [44, 45] in order to confirm our hypothesis.

To organize this paper, we have planned the sections as follows: "Preliminaries" section describes the new scalable EAD method; "WM-EAD-Global" section comprises the distributed MapReduce approach developed in this work; and "Experimental Study" section shows the experimental study carried out, where we analyze the accuracy and study the speed-up of our new proposal to finally reach some conclusions presented in "Conclusions and Future Works."

Preliminaries

This section is devoted to, first of all, review definitions and notations related with adaptive defuzzification methods, followed by an introduction to the big data distributed computing frameworks.

Evolutionary Adaptive Defuzzification

Adaptive defuzzification is an easy mechanism to improve the accuracy of linguistic FRBS for fuzzy modeling, based on using the appropriate individual contribution of each rule to the inference process in order to promote cooperation between the rules [39, 40].

There are many papers devoted to parameterized defuzzifiers. Frequently, they tune the behavior of the defuzzification with a single global parameter or with one parameter for each rule, resulting in an improved accuracy.

In this paper, we opted to use the specific expression shown in (1), as it is efficient, easy to implement, and showed good behavior in previous works [40]:

$$y_0 = \frac{\sum_i^N h_i \cdot \alpha_i . CG_i}{\sum_i^N h_i . \alpha_i}, \qquad (1)$$

where h_i is the so-called matching degree, α_i is the parameter that tunes each rule R_i, $i = 1\ to\ N$, and CG_i is the gravity center of the fuzzy set inferred with the rule R_i. This is a Mode - B defuzzifier, i.e., it converts individually every inferred fuzzy set into a real value and then calculates a weighted sum.

Note that the α_i parameters are equivalent to rule weighting [46], where values of $\alpha_i \in [1,\infty)$ emphasize the contribution of that rule, whereas values $\alpha_i \in [0,1]$ penalize it.

The set of defuzzifier parameters are often learned by using an evolutionary algorithm with real coding [39, 40], following the scheme of a chromosome comprising all the parameters associated with each rule of the RB. In this way, the learning process achieves a subset of rules with improved cooperation among them [39, 40]. Therefore, the learning process described is particularly interesting for use in post-processing after the quick and simple methods of RB learning guided by examples coverage (i.e., WM-method). These methods select the best rules individually, instead of in a collaborative group, which is finally reached thanks to the evolutionary process.

Big Data and Cluster Computing Frameworks

In general terms, big data is employed to denote volumes of data out of the capabilities of the typical database resources to capture, store, manage, and analyze [47].

The current big data technologies employed to manage the aforesaid data volumes are based on three columns [48]:

- Distributed file systems that store the big files in several distributed servers, e.g., the popular Apache Hadoop Distributed File System (HDFS) [49]
- Programming paradigms such as MapReduce [50] or Pregel [51] that ease the distributed programming jobs into clusters or servers
- Frameworks for computing clusters such as Apache Hadoop [49] or Apache Spark [43] to let us organize and manage this groups of computers as storage (through distributed file systems) and data processing (implementing distributed programing models) structures efficiently

The MapReduce programming paradigm was featured by Google in 2004 [50], and its best known open-source implementation is Apache Hadoop. It is famous due to being one of the first proposals including MapReduce, and also concepts like relatively easy to use scalable storage and data processing, as well as highly fault-tolerant, high availability, automatic data redundancy and recovery, etc. Hadoop is conceived to perform in a simple one-pass batch processing over data, that is, it is not intended to implement iterations over data where it is not efficient, or for interactive data research. Some of these drawbacks have recently been resolved by the Apache Spark [43] framework.

Apache Spark is likewise an open-source distributed programming framework conceived as a step forward in flexibility and efficiency. It incorporates different computational models, MapReduce being one of them, whose implementation is significantly faster [43] than those of other frameworks. A particularly interesting advantage is that it allows efficient iterative or multi-pass data processing due to one of its key features: the use of in-memory computing. This mechanism is based on a distributed memory abstraction (called resilient distributed dataset (RDD) [43]) that reduces the middle disk access, dramatically accelerating overall performance. An RDD can be seen as a set of data split through different servers of the cluster, which can be processed in parallel. Programmers can use two categories of operations over RDDs: transformations, which take an RDD and obtain another new RDD, and actions, which obtain a value from a computation over a given RDD. The fault-tolerant capability is implemented based on the aforementioned RDDs, as their slices can be automatically reconstructed if, for any reason, they get lost.

A user program in Spark can be seen as a single driver running the main function in the cluster master machine, and a set of several parallel tasks, run by the executors, achieving the RDDs on the slave machines of the cluster and returning the results of their computations to the driver.

Finally, we can point out that Spark can also use HDFS distributed storage, but it is independent of the storage file system of the cluster, and it can be used not only through programming but also interactively by using a console command line interpreter.

The EAD proposal developed in this paper is based on the use of the MapReduce paradigm implemented in the Spark framework, due to its capacity for efficient use in data sciences problems [52–55].

WM-EAD-Global: a Linguistic Fuzzy System with Evolutionary Adaptive Defuzzification with Spark

Now, we describe the proposal, the *WM-EAD-Global* FRBS for regression. This entails two sequential phases, which benefit from the distributed approach:

- *WM-Spark*: The first phase consists of creating the RB using the scalable version of the WM-method we proposed in [30]. As we code it using Spark this time, it is designated *WM-Spark*.
- *EAD-Global-Spark:* The second phase entails the *evolutionary adaptive defuzzification* method itself. It uses a scalable global evolutionary learning model, where the defuzzification method parameters are learned using an evolutionary algorithm [40]. This time is also implemented in Spark, where it takes advantage of the in-memory data capabilities to implement an iterative global model efficiently.

First Phase: WM-Spark

The WM data-driven method [41] is one of the most referenced algorithms of the FRBSs research area, to obtain, in a simple way, the RB employing a set of samples. The first phase, a distributed Spark implementation of the WM-method, which we named WM-Spark, involves a conceptually simple idea, which is to divide the original training dataset into some subsets, and apply the WM-method to each subset in a distributed way, helped by the MapReduce paradigm. Thus, in Spark, the training set is uniformly split into *n* portions and distributed alongside the computer cluster. This one-pass MapReduce schema is shown in Fig. 1, highlighting the operations carried out on the master node single computer by the *driver program*, and the ones on the group of slave computers by the executors. The functions are detailed below:

1. First, the *driver program* makes the partition of the fuzzy variables (so-called Data Base (DB)) using a uniformly allocated fixed number of triangular linguistic terms. Beforehand, the training dataset is divided into *n* disjoint subsets of training data with the same size, which are spread into the worker nodes together with the partitioned fuzzy variables.
2. Map function: *worker nodes* individually perform the classical WM-method, creating a rule for each example on its partition (naming RB_i in Fig. 1, to the set of rules of partition *i*) using for every variable the labels with the greater matching. Additionally, a matching degree is given in order to combine these generated rules. Therefore, this function creates a list of key-value pairs, where (key: *labels of the antecedents of the rules*; value: *consequent of the rules*, *and its matching*), which are returned to the driver program to be joined.
3. Reduce function: at least one or more reduce processes take the RB_i to be joined to build the final set of rules (in Fig. 1, we name it as RB_F). When rules with the same antecedents and consequent appear two or more times, they are removed keeping only a single copy; on the other hand, if there are rules with the same antecedents but dissimilar consequent, they are fixed selecting only the rule with the highest matching degree. Regarding the key-value pairs managed, this function takes a list of

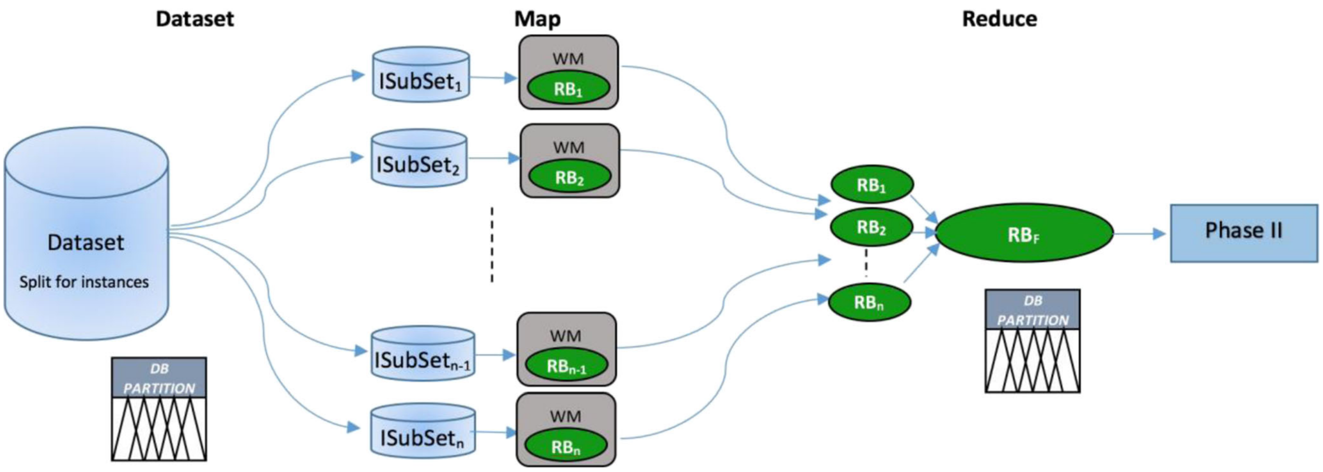

Fig. 1 First stage of the WM-EAD-Global method in MapReduce: the WM-Spark

key-value duos grouped by key/antecedent as (key: *labels of the antecedents of a rule*; list of value: *list of pairs (consequents for that antecedents, and their respective matching*)) and results in a rule. The whole of the rules generated is the final RB, (RB_F) which, together with the DB previously performed by the driver program, is the complete Knowledge Base (KB). Lastly, note that the WM-Spark described achieves the same as the WM-method so the RBs it creates are identical.

Second Phase: The EAD-Global-Spark

This phase performs the evolutionary process devoted to tune the parameters of the defuzzifier linked with each rule [39, 40] in a scalable way.

In the "Introduction" section, we commented that the proposal of this paper significantly improves the approach presented in [30]. There, we employed a local distributed evolutionary process model, each with its own different training data subset, to learn the defuzzification rule parameters. In fuzzy regression, where each output of the FRBS is computed through the aggregation or combination of the inference of some fired rules together instead of a single one, the values of rule parameters learned are closely related to each other, creating a cooperation relationship [40]. Therefore, the later complex combination of rule parameters learned within different partitions causes an inexorable decrease in the accuracy of the model.

This paper proposes to solve the aforementioned drawback by changing the local evolutionary learning model (*single-pass MapReduce schema*) for a global one (*multi-pass MapReduce schema*). In this way, the distributed computational power is not employed to perform distributed learning, but the heavy chromosome evaluation for every iteration of the evolutionary algorithm. Conceptually, the proposed model performs in the same way as the sequential model and later, in the experimental study, the differences between local and global model will be studied.

In order to describe the method proposed, we begin by describing the evolutionary schema details, and then how we use a MapReduce schema implemented in a Spark to make it scalable and efficient.

The evolutionary algorithm implemented is based on a classical CHC evolutionary algorithm [56], so here, we briefly depict its essential mechanisms:

Encoding

A real encoding equal to the one employed in our previous paper [40] was used. It consists of N genes in the interval [0,10], corresponding to the respective rule parameters α_i, of the RB, R_i.

$$C = (\alpha_1, \ldots, \alpha_N) \mid \alpha_i \in \{0, 10\}$$

Initial Population

The chromosomes of the initial population have been established with a single one with all its genes fixed to 1, in order to have an individual with the whole of its rules without weights. The rest of the chromosomes of the population were initialized randomly.

Evaluation

The evaluation process is the element of the evolutionary procedure performed in a distributed way by using Spark executors, as described later. Our approach entails minimizing the mean square error (MSE) to maximize the accuracy. The MSE expression 2 is:

$$MSE\ (S) = \frac{\frac{1}{2}\sum_{k=1}^{M}\left(y_k - S\left(x_k\right)\right)^2}{M} \tag{2}$$

where S denotes the fuzzy model considered. We use a set of evaluation data made by M pairs of numerical data $Z_k = (x_k, y_k)$, $k = 1,..,M$, with x_k being the values of the input variables, y_k the corresponding values of the output variables.

Crossover and Restart

The recombination of the chromosomes was implemented using the BLX-α operator [57] specific for real-coded genetic algorithms (fixing the parameter $\alpha = 0.5$). This involved taking into consideration that CHC algorithm only pairs chromosomes that overcome the mating threshold (they are sufficiently different, measured by using the Hamming distance after the conversion of the real numbers into strings).

The aforesaid mating threshold is set initially to *L/4*, being *L* is the number of characters of the string. The mating threshold is reduced by one unit if no offspring reach the new population.

The best chromosome of the population is maintained when the algorithm restarts, while the rest of the individuals are randomly initiated.

An Iterative MapReduce Process for Adaptive Defuzzification

Now, we shall describe the MapReduce strategy to achieve the EAD-Global-Spark. First, we describe it in terms of functions and processes (illustrated in Fig. 2), and then in terms of the computation of the MSE (showed in Fig. 3).

In this second phase, an iterative process MapReduce is performed to obtain the weighs for each rule obtained in the first phase. Next, we describe this process developed using the Spark paradigm:

1. The *driver program*, which is executed in the master node, performs an evolutionary learning algorithm to learn the associated RB weights, processing the heavy computational population evaluation in a distributed way along the cluster. To do so, it takes the whole data set, splits it into as many different units as working nodes available in the cluster, and distributes it to each node, also giving them a full copy of the KB (DB + RB) previously obtained in the first phase, and, of course, a full copy of the population to be evaluated.
2. Map function carried out by the executors: each *worker node* uses its available data to do the evaluation process, that is, every chromosome is evaluated using the *worker node* set of examples. Therefore, each chromosome, which represents the set of rule weights or defuzzification parameters associated with the RB, gets in this process the partial fitness corresponding with its data partition. In terms of the key-value pairs, the Map function produces a list of intermediate key-value pairs as (key: *chromosomes* (*weights associated of each rule that have to be evaluated*); value: *fitness* (*a measure of the accumulated error obtained for the RB and DB using the associated weights of each chromosome*), which are transmitted back to the master node that joins them. Later, we describe in full how the MSE is computed with the aforesaid accumulated error.
3. Reduce function, also carried out by the executors: at least one or more reduce processes get the sorted ends of the Maps functions and merge them to construct the definitive fitness of each chromosome (we named C_i with w_i in Fig. 2). This is achievable because the final fitness of each chromosome of the population, which is the MSE of the whole data, can be computed by combining the measure of the accumulated errors computed in each partition with their subset of data examples. Using key-value pair terminology, the reduce process gets the list of midway key-value pairs aggregated for key as (key: *chromosome* (*weights associated of each rule that have been evaluated*); list of value: *list of fitness* (*accumulated errors*) *for each chromosome evaluated in each partition*) and produces a list of chromosomes with their new associated fitness. The fitness obtained for all chromosomes is then sent to *driver program* in order to follow the evolutionary process.

Specifically, the definition of the MSE shown in expression 2 for a sequential calculation is computed within the MapReduce schema in this way (see also Fig. 3):

- The Map functions compute the accumulated error ($Error_i^j$) showed in expression 3:

$$Error_i^j = \sum_{k=1}^{\frac{M}{n}} \left(y_{i_k} - S^j \left(x_{i_k} \right) \right) \tag{3}$$

where n is the number of subsets into which the training dataset has been divided, i is the subset considered (with $i =$

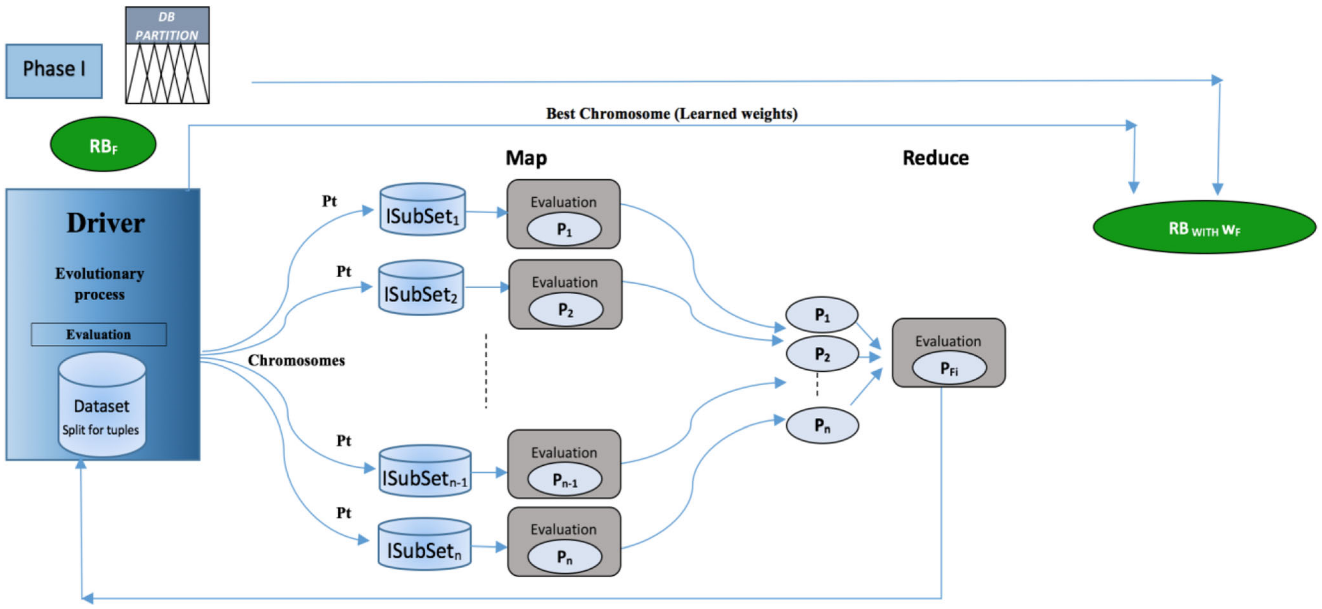

Fig. 2 Second stage of the WM-EAD-Global method in MapReduce: evolutionary adaptive defuzzification

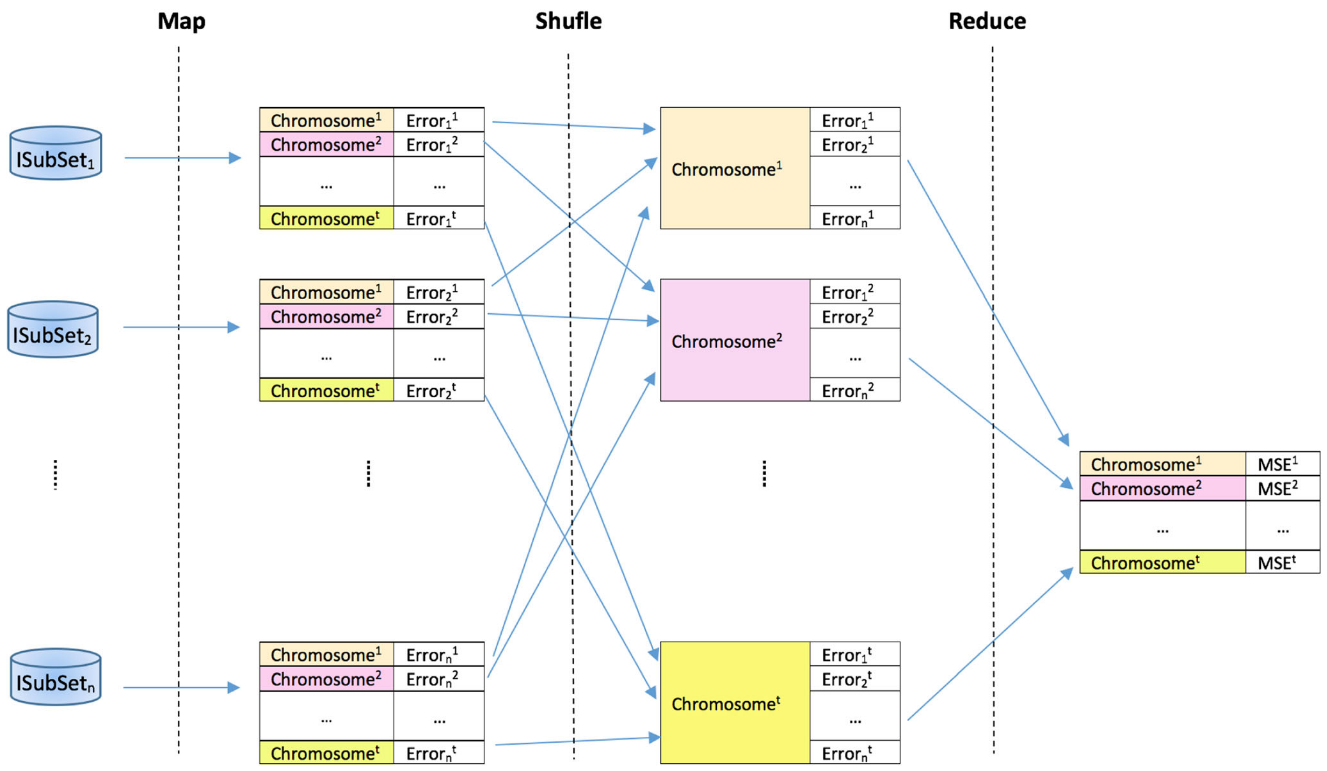

Fig. 3 Detailed schema used to compute the MSE by the proposed WM-EAD-Global approach with the MapReduce programming model

$1,\ldots,n$), j is the chromosome contemplated (with $j = 1,\ldots,t$, and t being the number of chromosomes), and M denotes the number of instances or examples of the original dataset. S^j is the fuzzy model employing the chromosome j, that is, using the j set of defuzzifier parameters, and in the same way that in (2), (x_{ik},y_{ik}) are the numerical pairs of values of the i-subset of examples for the input variables (x) of the example k (with $k = 1,\ldots,M$), and the theoretical output (y).

- The reduce functions compute the MSE shown in expression 4 using the previously computed accumulated errors, $Error_i^j$:

$$MSE^j(S) = \frac{\frac{1}{2}\sum_{i=1}^{n}\left(Error_i^j\right)^2}{M} \tag{4}$$

with j being the chromosome considered, so $MSE^j(S)$ is the set of MSEs computed returned as the fitness of each chromosome to the evolutionary algorithm.

Experimental Study

In this section, we present a study of the behavior of the proposed model in terms of accuracy and scalability. As we commented above, both WM-Spark and the new proposal EAD-Global-Spark perform exactly in the same way as the sequential WM and EAD. Therefore, the WM-EAD-Global is alike to its sequential ancestor in terms of accuracy. Thus, in the present experimental study, we attend to the scalability of the new model and compare the accuracy of the presented model against its predecessor proposed in [30].

This section is organized as follows: first, we show the datasets selected for the experimental study. Then, the "Experimental Setup" subsection describes how has been configured the experimental study developed and the non-parametrical statistical test employed to do a comparative study. Then, the "Results Obtained and Analysis" subsection is devoted to the examination of the results. Finally, the "Scalability" subsection studies the speed-up achieved by our proposal.

Table 1 shows a summary of the main characteristics of the *12* regression problems that we selected to carry out our experimental study. They have different complexities, different numbers of instances and variables and can be found in the KEEL [58] data repository, UCI Machine Learning Repository, and the complementary material website of the paper. We also included two particularly complex and great datasets (ETHY2 and YPRE) in order to observe the behavior with them. Particularly ETHY2 has been synthetically built from its corresponding original. It is one of the two time series *ethylene_methane gas sensor array under dynamics gas mixtures* of 4.178.504 instances from UCI. It has 19 attributes including the time and 2 outputs, so we have selected the ethylene-methane output and the 17 input variables for 12 h. Therefore, the set of datasets selected goes from a lesser to greater number of variables, comprising a range from *8192* to *1,044,625* examples and from *6* to *90* input variables.

Experimental Setup

Regarding the KBs, we have used three triangular linguistic terms for each variable in each problem. Both the conjunction and inference operator were the minimum t-norm. Concerning defuzzifiers, WM-Spark models use the center of gravity weighted by the matching degree, while the later global model EAD uses the defuzzifier showed in expression (1).

Datasets were previously separated into training and test, also considering a *5*-fold cross-validation model. A total of 30 runs for each problem (5 partitions and 6 different seeds for the random number generator of the genetic algorithms) were carried out. A population of 50 chromosomes, a crossover probability of 1, and a maximum number of *100,000* evaluations were the rest of the evolutionary process setup considered.

An exception to the aforesaid setup is the YPRE dataset. It was added in order to use a greater and complex dataset, but due to its features, we have exceptionally used a single partition and a single seed. Regarding the evolutionary process, in this particular case, we set the number of evaluations to *30,000*, because of the relatively modest testing platform available we describe next.

We employed a Spark cluster of *17* virtual servers with *4* cores and *8* GB of RAM each, the first of them being the one that executes the driver program, and the other *16* the worker nodes where the executors act. The host computer hardware is a server with *4* CPUs Intel Xeon E7–4850 with *10* cores per CPU and hyperthreading (thus, capable of *80* threads), and *192* GB of RAM. The number of cores was set in two different ways:

- Using a *2* core setup to measure the times needed with a basic machine.
- Using the cluster mode with the whole *16* servers, with *16* and *32* cores setups.

Note that this is not a true HPC cluster for big data problems but a research platform to test and validate algorithms, so the absolute values of time obtained are not truly representative, but interesting from a scalability point of view and for comparison between them. Our objective is not to show the well-known usefulness of the EADs for fuzzy regression applications, but to study the ways to adapt them to the current MapReduce distributed paradigm, to take advance of computer clusters.

Our study has been validated using statistical testing [44, 45]. In fact, we compared the performance approaches using a Wilcoxon signed-rank test [59]. In this sense, we performed a nonparametric statistical test of pairwise comparisons, the Wilcoxon signed rank test [59], to compare performance approaches. For this purpose, the test first performs the absolute value of the differences between the two FRBSs compared and classifies the results in ascending order, establishing a range for each of them. Once done, a sum of the R+ ranges is calculated when the first model exceeds the second, and vice versa when the opposite occurs R-. To conclude, a *p* value related with the statistical distribution is calculated so then, the null hypothesis of equality of means can be rejected if it is under a pre-specified level of significance.

Results and Analysis

The results achieved by the two models (WM-EAD-Local [30] (in that contribution, named *Scalable WM-EAD*) and the new proposal, WM-EAD-Global) with each problem are shown in Table 2, where the two main columns show the average MSE obtained by the two models compared, both in training (MSE_{tra}) and test (MSE_{tst}). The table also shows information on the number of rules of the models, and the best test accuracies are highlighted.

In the same sense, Table 3 shows the Wilcoxon test results of both models, where it can be concluded that there are significant differences between the two methodologies compared because the *p* values are lower than the fixed level of significance of $\alpha = 0.1$.

Consequently, when analyzing Table 2 and the statistical results found in Table 3, we can highlight that:

Table 1 Datasets considered. Available at KEEL (https://sci2s.ugr.es/keel/datasets.php) and UCI (https://archive.ics.uci.edu/ml/datasets.html) repositories, and also YPRE and ETHY2, which can be downloaded from http://www.uhu.es/gisimd/papers/WM-EAD-Global/)

Problem	Abbreviation	Instances	Input variables
Delta-elv	DELV	9517	6
California	CAL	20,640	8
Mv	MV	40,768	10
House	HOU	22,768	16
Ethy_meta1M	ETHY2	1,044,625	17
Elevators	ELV	16,599	18
Compactiv	CA	8192	21
Pole	POL	14,998	26
Puma32	PUM	8192	32
Airelons	AIL	13,750	40
Tic	TIC	9822	85
YearPrediction	YPRE	515,345	90

Table 2 Reference values of average number of rules and MSE of the FRBSs built with WM-EAD-Local and WM-EAD-Global. Values of MSE in this table must be multiplied by, 10^{-6}, 10^{8}, 10^{9}, 10^{-6}, 10^{-8}, and 10^{-4} for DELV, CAL, HOU, ELV, AIL, and TIC respectively

Datasets		WM-EAD-Local		WM-EAD-Global	
	#R	MSE_{tra}	MSE_{tst}	MSE_{tra}	MSE_{tst}
DELV	129.4	1.623575	1.646743	1.597499	1.625437
CAL	123.4	3.209802	3.214411	3.194159	3.203725
MV	3677.8	6.639665	6.922148	6.242279	6.592111
HOU	756.4	9.908656	10.971230	9.536912	10.210971
ETHY2	1099.8	990.032050	990.133439	989.463110	989.549671
ELV	508.8	10.135433	10.373326	9.650917	9.871129
CA	424.8	5.889947	6.290382	5.201201	5.985796
POL	1087.0	183.906543	189.593453	172.921691	179.290236
PUM	6553.6	0.000248	0.000587	0.000224	0.000586
AIL	1072.0	2.009592	2.086157	1.910770	2.017002
TIC	5802.4	156.035968	495.464697	113.327201	518.080663
YPRE	5322,6	70.923407	70.941918	71.087112	71.103495

- The global learning model presented, WM-EAD-Global, shows better average results for both training and test than the local learning model implemented in WM-EAD-Local, so the new proposal improves on the previous one in terms of accuracy, as it does not suffer any deterioration of the rule cooperation. But there are two exceptions with the TIC and YPRE datasets: the results of TIC in test are slightly worse due to overfitting, which is always a situation that can occur when using evolutionary algorithms. Attending to the particular case of the YPRE dataset, we observed that the WM-EAD-Local gets slightly better accuracy results than the WM-EAD-Global. This is likely due to the lower number of evaluations of the evolutionary algorithm fixed (only *30,000*), together with its greater complexity, so the evolutionary algorithm did not converge and take advantage of the global learning model like the rest of the datasets.
- Finally, it is also interesting to point out that the accuracy of the global methodology proposed is independent of the size of the partition employed (sometimes conditioned by the distributed computing resources available). In other words, this proposal is not only more accurate than the preliminary one presented in [30], WM-EAD-Local, but also independent of the computational resources. Thus, greater computational power only affects the time needed, but not the quality of the solution found.

Table 3 Wilcoxon test to compare the accuracy of the WM-EAD-Local vs. WM-EAD-Global. R+ corresponds to the sum of the ranks for WM-EAD-Global and R- to WM-Spark-Local

Comparison	R+	R-	*p* value
WM-EAD-Global vs WM-EAD-Local	17	61	0.09228

Scalability

In this section, we observe the behavior of the proposed approach attending to the times obtained and its speed-up when the number of computing resources, in terms of cores, grows. We have selected the basic setup of the cluster using *2* cores, and then setups with *16* and *32* cores. Table 4 shows the runtime results obtained in hours, minutes, and seconds, spent by the EAD-Global-Spark (the second phase comprising of a multi-pass algorithm). It is important to note that, as we have commented before, although many of the times shown are sizeable, they are due to the available experimental platform employed, based on virtual servers running inside a big host computer. Nevertheless, the times are not important in

Table 4 Average runtime elapsed in hours:minutes:seconds and speed-ups for the WM-EAD-Global using 16 and 32 cores

Dataset	2-cores	16-cores		32-cores	
	Time	Time	Speed-up	Time	Speed-up
DELV	000:19:15	00:04:06	4.69	00:03:01	6.38
CAL	000:40:57	00:07:42	5.31	00:05:16	7.77
MV	052:38:43	09:04:24	5.80	06:12:53	8.47
HOU	008:24:41	01:18:37	6.42	00:54:20	9.29
ETHY2	594:41:23	101:41:06	5.85	67:27:23	8.82
ELV	003:58:51	00:42:08	5.67	00:27:11	8.79
CA	002:10:04	00:25:03	5.19	00:17:14	7.54
POL	011:12:12	01:48:22	6.20	01:07:28	9.96
PUM	047:48:21	08:12:06	5.83	05:18:42	9.00
AIL	015:56:45	02:36:11	6.13	01:42:32	9.33
TIC	118:14:11	20:42:28	5.71	12:48:54	9.23
YPRE	–	1151:24:57	–	764:24:23	–

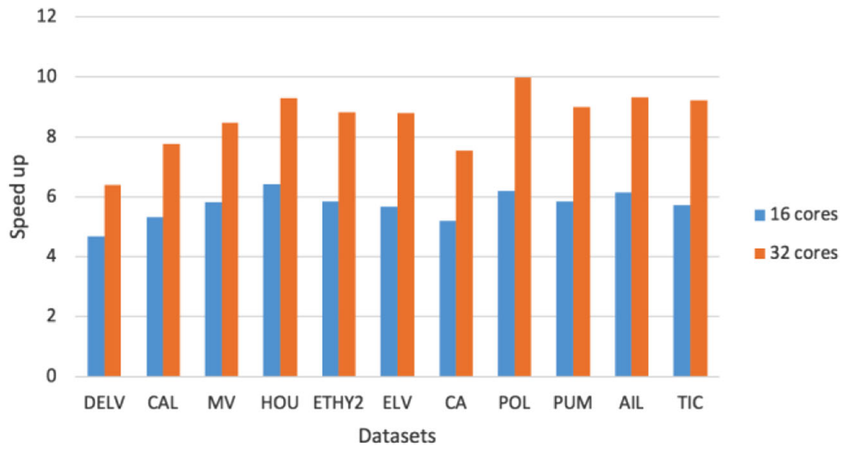

Fig. 4 Speed-up obtained with each dataset (except YPRE)

absolute terms, but in relative terms (*speed-up*), some with respect to others. We did not include in Table 4 the measures of YPRE dataset for the *2* cores setup because its complexity makes it close to impossible to compute with this specific cluster setup, but the results obtained with *16* and *32* cores show a similar scalability than the one obtained with the other datasets.

Finally, in order to easily compare the different speed-ups of Table 4, Fig. 4 graphically shows the relative speed-up between the *16* and *32* cores setup for each dataset of the experimental study. The speed-up is defined as the ratio between the time spent with the *16* and *32* core setups respectively, and the time spent by the simple *2* core setup. The speed-up for the YPRE dataset is not computed due to the aforesaid lack of time measured for the *2* core setup.

The time reduction when increasing the number of cores is remarkable, as was expected. However, this time reduction is not completely proportional to the number of cores, due to the extra computational load of the Spark framework.

Conclusions

The purpose of this paper is to propose a new completely linguistic FRBSs for regression with adaptive defuzzification in large-scale environments created using MapReduce distributed paradigm and a global learning model. Although this paper is focused on the evolutionary adaptive defuzzification proposal, we include a distributed scalable version of the very well-known Wang and Mendel approach [41] to learn the fuzzy RB from examples. Both models are implemented in Apache Spark. The most remarkable aspect of this paper is that both algorithms produce the same results as their sequential ancestor, which was not achieved in our preliminary approach [30] where the learning model was local. The interest of the use of evolutionary adaptive defuzzification approaches is that it is compatible with most other methodologies to improve the accuracy in linguistic FRBSs for regression and control.

The proposal presented in this work is interesting not only in terms of large-scale problems (i.e., dataset with huge volume of examples) but also with smaller datasets in more reasonable execution times utilizing distributed processing, despite the heavy computational cost of the evolutionary technique.

Funding Information This work was supported by grant from the Spanish Ministry of Science under project TIN2017-89517-P.

Compliance with Ethical Standards

Conflict of Interest The authors declare that they have no conflict of interest.

Ethical Approval This article contains no studies with human participants or animals performed by any of the authors.

References

1. Siddique N, Adeli H. Nature inspired computing: an overview and some future directions. Cogn Comput. 2015;7(6):706–14.
2. Nobakhti A. On natural based optimization. Cogn Comput. 2010;2(2):97–119.
3. Wang D, Shan H, Tian Y, Liu L. Emergent face orientation recognition with internal neurons of the developmental network. Prog Artif Intell. 2018;7(4):359–67.
4. Dragoni M, Rospocher M. Applied cognitive computing: challenges, approaches, and real-world experiences. Prog Artif Intell. 2018;7(4):249–50.
5. Fan M, Zhou Q, Abel A, Fang Zheng T, Grishman R. Probabilistic belief embedding for large-scale knowledge population. Cogn Comput. 2016;8(6):1087–102.
6. Zhang HG, Wu L, Song Y, Su CW, Wang Q, Su F. An online sequential learning non-parametric value-at-risk model for high-dimensional time series. Cogn Comput. 2018;10(2):187–200.

7. Abdullah A, Hussain A, Khan IH. Introduction: dealing with big data - lessons from cognitive computing. Cogn Comput. 2015;7(6): 635–6.
8. Zhang HY, Ji P, Wang JQ, Chen XH. A neutrosophic normal cloud and its application in decision-making. Cogn Comput. 2016;8(4): 649–69.
9. Tao Z, Han B, Chen H. On intuitionistic fuzzy copula aggregation operators in multiple- attribute decision making. Cogn Comput. 2018;10(4):610–24.
10. Molina D, LaTorre A, Herrera F. An insight into bio-inspired and evolutionary algorithms for global optimization: review, analysis, and lessons learnt over a decade of competitions. Cogn Comput. 2018;10(4):517–44.
11. Pino A, Shin K, Velázquez-Rodríguez C. Improving the genetic bee colony optimization algorithm for efficient gene selection in microarray data. Prog Artif Intell. 2018;7(4):399–410.
12. Herrera F. Genetic fuzzy systems: taxonomy, current research trends and prospects. Evol Intell. 2008;1(1):27–46.
13. Fazzolari M, Alcalá R, Nojima Y, Ishibuchi H, Herrera F. A review of the application of multi-objective evolutionary systems: current status and further directions. IEEE Trans Fuzzy Syst. 2013;21(1): 45–65.
14. Fernández A, López V, del Jesus MJ, Herrera F. Revisiting evolutionary fuzzy systems: taxonomy, applications, new trends and challenges. Knowl Based Syst. 2015;80:109–21.
15. Fernández A, Herrera F, Cordón O, del Jesus MJ, Marcelloni F. Evolutionary fuzzy systems for explainable artificial intelligence: why, when, what for, and where to? IEEE Comput Intell Mag. 2019;14(1):69–81.
16. Elhag S, Fernández A, Alshomrani S, Herrera F. Evolutionary fuzzy systems: a case study for intrusion detection systems. In: Bansal J, Singh P, Pal N, editors. Evolutionary and swarm intelligence algorithms. Studies in Computational Intelligence, vol. 779. Cham: Springer; 2019. p. 169–90.
17. Ferdaus MM, Anavatti SG, Garratt MA, Pratama M. Development of C-means clustering based adaptive fuzzy controller for a flapping wing micro air vehicle. J Artif Intell Soft Com Res. 2019;9(2):99–109.
18. Cózar J, dela Ossa L, Gámez JA. Learning compact zero-order TSK fuzzy rule-based systems for high-dimensional problems using an Apriori + local search approach. Inform Sci. 2018;433–434:1–16.
19. Zikopoulos P, Eaton C, De Roos D, Deutsch T, Lapis G. Understanding big data: analytics for enterprise class Hadoop and streaming data. New York City: McGraw-Hill; 2011.
20. García-Pedrajas N, de Haro-García A. Scaling up data mining algorithms: review and taxonomy. Progr Artif Intell. 2012;1(1):71–87.
21. Río S, López V, Benítez JM, Herrera F. A MapReduce approach to address big data classification problems based on the fusion of linguistic fuzzy rules. Int J Comp Intel Syst. 2015;8(3):422–37.
22. Peralta D, Río S, Ramírez-Gallego S, Triguero I, Benítez JM, Herrera F. Evolutionary feature selection for big data classification: a MapReduce approach. Math Probl Eng. 2015:501–246139.
23. Fernandez A, Carmona CJ, del Jesus MJ, Herrera F. A view on fuzzy systems for big data: progress and opportunities. Int J Comp Intel Syst. 2016;9(1):69–80.
24. Ferranti A, Segatori A, Antonelli M, Ducange P. A distributed approach to multi-objective evolutionary generation of fuzzy rule-based classifiers from big data. Inf Sci. 2017;415(416):319–40.
25. Ducange P, Marcelloni F, Segatori A. A MapReduce-based fuzzy associative classifier for big data. In Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). 2015;1–8.
26. López V, del Río S, Benítez JM, Herrera F. Cost-sensitive linguistic fuzzy rule based classification systems under the MapReduce framework for imbalanced big data. Fuzzy Sets Syst. 2015;258:5–38.
27. Rodriguez-Fdez I, Mucientes M, Bugarin A. A genetic fuzzy system for large-scale regression. In Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). 2016; 1421–1428.
28. Rodriguez-Fdez I, Mucientes M, Bugarin A. SFRULER: scalable fuzzy rule learning through evolution for regression. Knowl Based Syst. 2016;110:255–66.
29. Rodriguez-Mier P, Mucientes M, Bugarín A. Scalable modeling of thermal dynamics in buildings using fuzzy rules for regression. In Proceedings of the IEEE International Conference on Fuzzy System (FUZZ-IEEE). 2017; 1–6.
30. Márquez AA, Márquez FA, Peregrín A. A scalable evolutionary linguistic fuzzy system with adaptive defuzzification in big data. In Proceedings of the IEEE International Conference on Fuzzy System (FUZZ-IEEE). 2017; 1–6.
31. Alcalá R, Gacto MJ, Herrera F. A fast and scalable multiobjective genetic fuzzy system for linguistic fuzzy modelling in high dimensional regression problems. IEEE Trans Fuzzy Syst. 2011;19(4): 666–81.
32. Márquez AA, Márquez FA, Roldán AM, Peregrín A. An efficient adaptive fuzzy inference system for complex and high dimensional regression problems in linguistic fuzzy modelling. Knowl Based Syst. 2013;54:42–52.
33. Antonelli M, Ducange P, Marcelloni F. Genetic training instance selection in multiobjective evolutionary fuzzy systems: a coevolutionary approach. IEEE Trans Fuzzy Syst. 2012;20(2):276–90.
34. Antonelli M, Ducange P, Marcelloni F. An efficient multi-objective evolutionary fuzzy system for regression problems. Int J Approx Reason. 2013;54(9):1434–51.
35. Gacto MJ, Galende M, Alcalá R, Herrera F. METSK-HDe: a multiobjective evolutionary algorithm to learn accurate tsk-fuzzy systems in high-dimensional and large scale regression problems. Inf Sci. 2014;276:63–79.
36. Liu P, Li H. Interval-valued intuitionistic fuzzy power Bonferroni aggregation operators and their application to group decision making. Cogn Comput. 2017;9(4):494–512.
37. Garg H, Arora R. Dual hesitant fuzzy soft aggregation operators and their application in decision-making. Cogn Comput. 2018;10(5):769–89.
38. Alcala-Fdez J, Herrera F, Márquez FA, Peregrín A. Increasing fuzzy rules cooperation based on evolutionary adaptive inference systems. Int J Intell Syst. 2007;22(9):1035–64.
39. Márquez FA, Peregrín A, Herrera F. Cooperative evolutionary learning of linguistic fuzzy rules and parametric aggregation connectors for Mamdani fuzzy system. IEEE Trans Fuzzy Syst. 2007;15(6):168–1178.
40. Cordón O, Herrera F, Márquez FA, Peregrín A. A study on the evolutionary adaptive defuzzification methods in fuzzy modelling. Int J Hybrid Intell Syst. 2004;1(1):36–48.
41. Wang L, Mendel J. Generating fuzzy rules by learning from examples. IEEE Trans Syst, Man, Cybern. 1992;22(6):1414–27.
42. Ramirez-Gallego S, Fernández A, García S, Chen M, Herrera F. Big data: tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce. Inf Fusion. 2018;42:51–61.
43. Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, et al. Apache spark: a unified engine for big data processing. Commun ACM. 2016;59(11):56–65.
44. Demšar J. Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res. 2006;7:1–30.
45. García S, Herrera F. An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons. J Mach Learn Res. 2008;9:2579–96.

46. Cho JS, Park DJ. Novel fuzzy logic control based on weighting of partially inconsistent rules using neural network. J Intel Fuzzy Syst. 2000;8:99–100.
47. Laney D. 3D data management: controlling data volume, velocity and variety. META Group Research Note 6. 2001; 70.
48. Fernández A, del Río S, López V, Bawakid A, del Jesus MJ, Benítez JM, et al. Big data with cloud computing: an insight on the computing environment, MapReduce, and programming frameworks. Wiley Interdiscip. Rev. Data Mining Knowl. Discov. 2014;4(5):380–409.
49. White T. Hadoop: the definitive guide. Sebastopol: O'Reilly Media, Inc.; 2012.
50. Dean J, Ghemawat S. MapReduce: a flexible data processing tool. Commun ACM. 2010;53(1):72–7.
51. Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, et al. Pregel: a system for large-scale graph processing. In Proceedings of the ACM SIGMOD International Conference on Management of Data 2010;135–146.
52. Padillo F, Luna JM, Ventura S. Exhaustive search algorithms to mine subgroups on big data using Apache Spark. Prog Artif Intell. 2017;6(2):145–58.
53. Pulgar-Rubio F, Rivera-Rivas AJ, Pérez-Godoy MD, González P, Carmona CJ, del Jesus MJ. MEFASD-BD: multi-objective evolutionary algorithm for subgroup discovery in big data environments - a MapReduce solution. Knowl Based Syst. 2017;117:70–8.
54. Arnaiz-González A, González-Rogel A, Díez-Pastor JF, López-Nozal C. MR-DIS: democratic instance selection for big data by MapReduce. Prog Artif Intell. 2017;6(3):211–9.
55. Luna-Romera JM, García-Gutiérrez J, Martínez-Ballesteros M, Riquelme JC. An approach to validity indices for clustering techniques in big data. Prog Artif Intell. 2018;7(2):81–94.
56. Eshelman LJ. The CHC adaptive search algorithm: how to safe search when engaging in nontraditional genetic recombination. In G.J.E. Rawlings (Ed.), Foundations of genetic algorithms. 1991;1: 265–283.
57. Herrera F, Lozano M, Sánchez A. A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study. Int J Intell Syst. 2003;18:309–38.
58. Alcala-Fdez J, Sánchez L, García S, del Jesus M, Ventura S, Garrell J, et al. Keel: a software tool to assess evolutionary algorithms for data mining problems. Soft Comput. 2009;13(3):307–18.
59. Sheskin D. Handbook of parametric and nonparametric statistical procedures. Boca Raton: Chapman & Hall/CRC; 2006.