Noname manuscript No.
(will be inserted by the editor)

An Incremental Approach to Address Big Data Classification Problems Using Fuzzy Rules

Antonio González · Raúl Pérez · Rocío Romero-Zaliz

Received: date / Accepted: date

Abstract INTRODUCTION: The recent emergence of massive amounts of data requires new algorithms that are capable of processing them in an acceptable time frame. Several proposals have been made, and all of them share the idea of using a procedure to break down the entire set of examples into smaller subsets, process each subset with a learning algorithm, and then combine the different partial results. Most of these models make use of a parallel process, where each learning algorithm learns independently for each subset of data.
OBJECTIVES: In our case, the goal is to propose a new model to obtain classifiers based on fuzzy rules that make use of a sequential model that can process a large number of examples and to show that, for some problems, a sequential procedure can be competitive in time and learning capacity against parallel processing proposals based on the MapReduce paradigm.
METHODS: This sequential processing uses a batch-incremental learning technique that can process each subset of examples. The incremental proposal makes use of a biologically-inspired computation method. This method is a cognitive computational model which use genetic algorithms to learn fuzzy rules.
RESULTS: The experimentation carried out shows that the incremental model is competitive with respect to a parallel model proposed for addressing big data classification using fuzzy rules.

This work has been partially funded by the Spanish MEC Projects TIN2015-71618-R, DPI2015-69585-R and co-financed by FEDER funds (European Union).

Antonio González · Raúl Pérez · Rocío Romero-Zaliz
Department of Computer Science and Artificial Intelligence, University of Granada, 18071-Granada SPAIN
Tel.: +34-958-244019
E-mail: A.Gonzalez@decsai.ugr.es

fgr@decsai.ugr.es · rocio@decsai.ugr.es ·

Keywords Big data · Fuzzy rule based classification systems · Incremental learning algorithms

1 Introduction

The term Big Data is usually applied to information that cannot be analyzed or processed using traditional methods [42]. Recently several papers that use big data techniques to solve different types of problems were published (e.g., [1,12,24,25,28,31–33]), many of which make use of cognitively inspired techniques [8].

One of the characteristics of the inability to process information is the handling of a massive amount of data that prevent the use of current algorithms. Nowadays, it is very frequent to collect a vast amount of data, and it is necessary to propose new algorithms capable of obtaining information from these data. Specifically, in this work, we want to make a proposal that allows us to tackle big data problems using fuzzy rule-based classification systems [7,18].

The most popular model for addressing big data problems is MapReduce [5,6]. MapReduce is a distributed and parallel programming model specifically proposed for processing a large amount of data. This model has been successfully applied [3,9,10,34,36,38], however, when the goal of the classification process is to obtain a single and interpretable rule base it has only been applicable when very simple rule models were used.

In this work, we propose an alternative model for working with classification problems. The fundamental idea is to change the parallel model for a sequential model in which the result of solving a subproblem is used in the solution of the next one. This sequential model will include an incremental learning algorithm as part of the overall architecture. Incremental learn-

ing algorithms [17,26,40] are algorithms in which input data is continuously used to modify the current knowledge of the model.

More specifically, the idea is to use a procedure to break down the entire set of examples into smaller subsets, process the first subset with an incremental learning algorithm to obtain a classifier, and then use the second subset along with the classifier obtained in the first step to obtain a new classifier that represents both the first and second subsets of examples, and so on.

In specialized literature two types of incremental learning are distinguished, batch-incremental versus instance-incremental [35]. In the instance-incremental learning approach the algorithm learns for each new example, while in the batch-incremental learning approach the algorithm is trained on batches of the data.

Since in our case we have all the examples from the beginning, a batch-incremental learning method is appropriate, since each new subset of examples will be considered as a new batch.

To develop this process, we make a batch-incremental proposal based on an extension of a learning fuzzy rule algorithm. There are several proposals to develop fuzzy rule learning algorithms, many of them make use of cognitive models such as genetic algorithms [20,23,27,39]. In our case, we will use NSLV-AR [15] a bio-inspired computation algorithm for learning fuzzy rules. This proposal is based on the models proposed in [14,37] where we integrate the main ideas proposed there with a new incremental algorithm, a complete formulation to define a rule model that is capable of "remembering" previously processed examples and incorporating new ones, along with extensive experimentation.

In the next section we will present briefly some related work, next we make a theoretical discussion about how to solve the problem by splitting the set of examples in either parallel or sequential processing of each subset, making special reference to the sequential model we propose. In Section 4 we will detail the model of the rule that allows us to remember the use of past examples in relation to that rule, and in Section 5 we will explain the learning algorithm. Section 6 will show the experimental study conducted, and finally, the paper ends with a conclusion section.

2 Related work

There is a wide variety of classification algorithms based on the use of Fuzzy Rule-Based System (FRBS) [4,19, 21,22,27]. However, all these models suffer a clear deterioration in performance processing massive datasets, which makes it impossible to use these classification algorithms on big data problems [36].

Different proposals have appeared in the literature to try to solve this problem. All of these proposals are based on first splitting massive data into small chunks and then process, in some way, the results obtained for each chunk.

Most of the proposals published so far are based on the use of the programming framework called MapReduce [5,6], which automatically distributes and performs parallel calculations on each chunk of the set of examples. In a very simplified way, we could say that MapReduce consists of two basic operations, the *map function* that is responsible for dividing the original database and processing each subproblem separately to generate intermediate results, and the *reduce function* that collects and aggregates the intermediate results of the map function.

In [36] one of the first linguistic fuzzy rule-based classification algorithm that makes use of the MapReduce paradigm is proposed. This proposal uses first a split of the set of examples, then uses the classification algorithm of fuzzy rules of Chi et al. [4] to obtain each classifier, finally, two different proposals for fuzzy rule fusion are studied.

A new version of MapReduce combined with Chi classification algorithm is proposed in [9]. This version removes the dependency with respect to the mapper number. The accuracy of the global model is the same that will be obtained using a single node with all the training examples.

The MapReduce paradigm has been also used to learn rule associative classifiers [3] and other classification algorithms not based on the use of rules, for example, Random Forests with fuzzy trees [38,34] or kNN and fuzzy prototypes [10].

When we analyze the previous proposals, we can conclude that the MapReduce paradigm undoubtedly helps to solve Big Data problems, and is a good alternative for the rapid construction of ensembles. However, when the objective is to obtain a single knowledge base represented by rules, it is only applicable to poorly expressive rule models. The reason is that only with this kind of rules the process of combining rules is efficient. With other more expressive rule models, such as CNF or DNF, which allows the embedded selection of variables, the aggregation process works on non-polynomial orders.

Recently, other models have been proposed that also make use of the division of the set of examples, but make use of incremental learning algorithms. These algorithms can offer an efficient alternative for obtaining a single knowledge base expressed in the form of rules,

using more expressive representation models. The main difference with the use of MapReduce is that the processing is sequential rather than parallel. In [14] a first proposal for the use of incremental learning algorithms to obtain classification rules is proposed. Later in [37] the study is extended and some parameters of the model are analyzed.

The objective of this work is to define a formal context on the use of incremental learning algorithm to address big data classification problems using fuzzy rules, also to propose a new version of the algorithm that makes use of different parameters to improve response time, as well as extensive experimentation to demonstrate that the results are competitive compared to the parallel proposal proposed in [36].

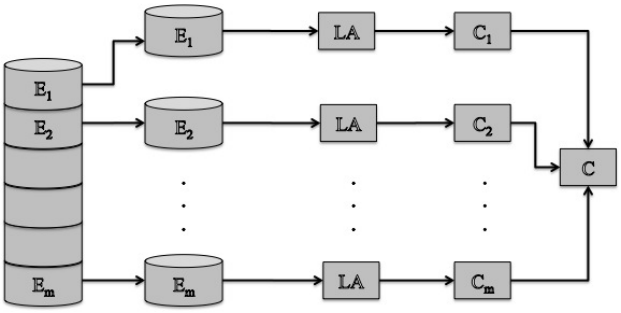

Fig. 1 A graphical view of the parallel model, where E_i represents a set of examples, called episode or batch of examples, LA represents a learning algorithm, C_i the classifier obtained by LA on the examples of the episode, and C the final classifier.

3 Parallel and sequential approaches for massive data classification

In this section, we present a brief analysis that allows us to visualize jointly the two technical groups that have so far been used to address the definition of classifiers based on fuzzy rules for the problem of processing massive data and that has been commented in the previous section.

Given a set of p examples

$$E = \{e_1, e_2, \ldots, e_p\}$$

where each example is composed by n attributes and a label value representing the particular category or class of the example

$$e_i = \{e_{i1}, e_{i2}, \ldots, e_{in}, B_i\},$$

being e_{ij} the j-th attribute value of the i-th example, and B_i the value of the class label of the i-th example.

Using the paradigm of divide and conquer, a basic proposal to process a large set of examples is to split it into m smaller subsets:

$$E = \{E_1 \cup E_2 \cup \ldots \cup E_m\} \quad (1)$$

in such a way that for each subset E_i, obtaining a classifier is a viable problem. Each new subset of examples E_i will be called an **episode** or batch of examples.

Let us suppose that, given a particular learning algorithm (LA), we are able to get a classifier C_i for each set of examples E_i (i.e., the i-th episode). Obviously, since the final idea is to obtain a single classifier that contains the information of each one of the classifiers obtained on each episode, it is necessary to define some procedure to do it. There are different ways to do this, and one of the key ideas that we can take into account is an assumption of independence when getting each classifier over the previously learned classifiers. Thus, we can develop the process in at least two different ways, the first one makes use of a hypothesis of independence and the second one does not.

In the first case, that we call **parallel model** (see Figure 1), each classifier is obtained independently of the other classifiers taking into account only the examples of the corresponding episode, and the main problem is to decide how the different classifiers should be combined to obtain a single final classifier.

In the second case, that we call **sequential model**, each classifier is obtained taking into account the examples of the new episode, along with the previously learned classifiers, and the main problem is how to adapt or modify previous classifiers given a new set of examples or episode.

One way to implement the parallel model could be for example through the MapReduce paradigm, and the algorithms discussed in the previous section are examples of this model.

The sequential model uses a division of the global set of examples as shown in Equation (1). Obviously, this step is similar to the use of a map function. In relation to the process of obtaining a new classifier, we have said that this depends on the previously obtained classifiers, and this could be reflected in the following expression

$$C_i = LA(E_i, C_1, \ldots, C_{i-1}) \quad (2)$$

where LA is the particular learning algorithm used. Thus, in order to obtain a new classifier C_i the learning algorithm must take into account the new episode E_i together with the previously learned classifiers

$$\{C_1, \ldots, C_{i-1}\}.$$

Assuming that the knowledge of a classifier already takes into account the knowledge of previous classifiers,

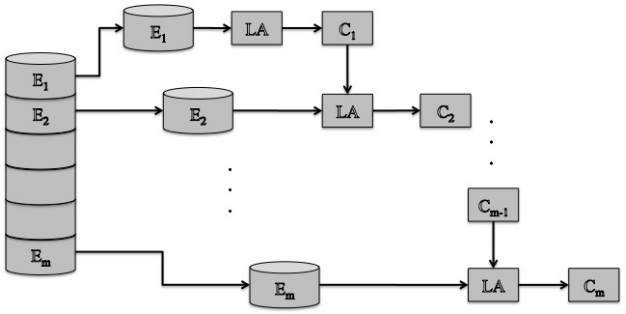

Fig. 2 A graphical view of the sequential model described by equation 3, where E_i represents a set of examples, called episode, LA represents a learning algorithm, C_i the classifier obtained by LA on the examples of the episode or batch of examples, and C_m the final classifier.

we can simplify the above expression and we can use a simplifying assumption of the sequential model to learn a new classifier C_i based only on the examples of the new episode E_i and of the previous classifier C_{i-1}. That is

$$C_i = LA(E_i, C_{i-1}). \tag{3}$$

This assumption obviously simplifies the problem but can lead to a possible loss of knowledge.

This model is shown in Figure 2. A reasonable approach to implement the model described by Equation (3) is to use a batch-incremental learning algorithm.

A batch-incremental learning algorithm is an algorithm able to update the knowledge learned (C_{i-1}) when new examples (E_i) are obtained over time.

One of the most frequent applications of incremental learning is the "concept drift" problem [41]. This problem consists in learning the change that the description of a concept can have over time. This usually happens because there are variables that affect the concept, and that has not been considered in the learning problem. In our case, there is no change in the concept to learn and incremental learning, therefore, does not have to consider this problem. This idea will be taken into account in the later design of the learning algorithm.

In relation to the stored examples used by an incremental learning algorithm we can consider three types of algorithms [26]:

- a learner with full instance memory, when all previously encountered examples are stored,
- a learner with partial-memory, when only some selected examples are stored and used to learn a new classifier, and
- a learner with no concept memory, when only the examples of the new episode and no additional examples are considered for learning the new classifier.

In [16] a full instance memory incremental algorithm of fuzzy rules, based on the use of NSLV-AR, was proposed. However, a full instance memory algorithm is not useful for learning classifier for big data problems since our hypothesis is that the learning algorithm is unable to learn when there is a massive number of examples. However, a learner with a partial-memory provided that the memory requirements are limited, can be useful to develop a sequential model.

Thus, our goal is to propose a batch-incremental learning algorithm with partial memory. This proposal includes NSLV-AR as an essential part of the algorithm.

Once we have shown two models to generate classifiers for big data problems, the parallel and the sequential, it would be necessary to analyze some characteristics of them. We have already commented in the previous section that the parallel model may not be very appropriate for learning models using expressive rules when we are interested in obtaining a single and interpretable rule base, whereas the sequential model is. Another important aspect to take into account is the efficiency of each of the models. A priori, a parallel model might seem more efficient than a sequential model, and in general, it is true when we are working with an unlimited number of processing units on a decomposable problem. However, since this assumption is not always true, there may be problems where a sequential method is better. Parallel processing implies independence in how to obtain the classifiers for each subset of examples, in the sense that the partial results obtained are only considered in the aggregation of fuzzy rules, but never in obtaining each classifier. However, incremental models will be more efficient in some cases, and these cases depend on the possible redundancy in the set of examples. We understand that there is redundancy in the particular problem when the information contained in the subset of examples is similar to the information in the global set of examples. In this way, when the database has sufficient redundancy, the sequential model can be much more efficient since instead of learning in each subset, it learns with the first subsets of examples, and from there it only has to make small modifications to the rule base. Thus, considering an extreme case, it is possible that the parallel model is learning in each node, again, and again the same knowledge, while the sequential model will only learn for the first subset and for the rest of the subsets will only verify that this knowledge was applicable to the whole set of examples. However, when the redundancy in the data set is very small, it is perfectly possible for a parallel model to work better.

4 Rule model

The basic model of a rule used in [36] is

IF X_1 is A_1 and X_2 is A_2 and ... and X_n is A_n

THEN Y is B with **weight w**

where $X_1, \ldots, X_n$ are the attributes, $A = (A_1, \ldots, A_n)$ are the linguistic terms taken for each attribute, Y is the consequent variable, B is the value of the consequent variable and w is a measure of the weight associated with the rule. We denote this rule as $R_B(A, w)$.

However, in this paper, we use an extended and more expressive rule model that simplifies the obtained knowledge. This model allows the value assigned to each variable (i.e., A_i) to be a set of fuzzy labels of the domain. This is interpreted like assigning a disjunction of linguistic terms to an attribute.

As mentioned above our goal is to develop an incremental learning algorithm with partial memory, and apply it to big data classification problems. For this purpose, we first need to have a flexible model of fuzzy rules that can contain a memory of its performance with examples used in the past, but which are no longer available. This kind of memory must be modified with the addition of new examples since all the examples are equally important. This fact is completely different from other kinds of problems, as the previously mentioned concept drift problem, where the algorithm must take into account the time when the examples are incorporated. To this end, first, we present a rule model, as well as a binary coding of the rule.

4.1 Coding simple rules

We use the extended fuzzy rule model described in the previous section, and we use a binary code for representing the antecedent of each rule. As each rule has n antecedent variables

$$X_1, \ldots, X_n$$

each one having an associated fuzzy domain D_i with p_i components, the antecedent of a rule is an element of

$$P(D_1) \times \ldots \times P(D_n),$$

where $P(D_i)$ is the set of subsets of D_i.

A way to encode the antecedent $A = (A_1, A_2 \ldots, A_n)$ of a rule $R_B(A, w)$ is to use a vector of $p_1 + \ldots + p_n$ zero-one components. This vector is noted as BinaryCode(A) and the value of component $(p_1 + \ldots + p_{i-1} + j)$ of the vector is 1 if the j-th element in domain D_i is a value of the variable X_i, and 0 otherwise, with i=1...n and j=1...p_i.

Finally, the consequent of the rule is represented as an integer value which refers to the position of the particular value of the domain, that is, for example, we use the number 3 to represent the third value of the domain. The weight of the rule w is usually calculated from the number of positive and negative examples of the rule, and it is coded as a real number representing its value. The last two elements that we include in the coding are the number of positive and negative examples covered by the rule.

Let us suppose that we are working with the IRIS database [13], which contains 4 real predictive attributes ($SepalLength$ [SL], $SepalWidth$ [SW], $PetalLength$ [PL], $PetalWidth$ [PW]) and a classification variable with 3 classes ($Setosa$, $Versicolour$, $Virginica$). Furthermore, let us suppose that we use for each predictive attribute five uniformly distributed fuzzy labels on its ranges ($Very\_Low$ [VL], Low [L], $Medium$[M], $High$ [H], $Very\_High$ [VH]). So, each variable is coded with five different values, and therefore, the complete antecedent needs 20 values to be coded. An example of the antecedent of a rule could be the following one:

SL	SW	PL	PW
11111	00001	00011	11111

This antecedent is

SW is $High$ and PL is $Medium$ or $High$

as SL and PW are irrelevant variables since it has assigned all the possible values of its respective domains.

To encode a full rule we just have to add to the antecedent coding an additional symbol to encode the class of the rule by an integer value, the weight coding as a real number, and the number of positive and negative examples to the rule.

4.2 Coding sets of rules

With the idea of creating a structure that represents a set of rules and serves as a memory for an incremental learning algorithm, first, we are going to define an order relation between the antecedents of the fuzzy rules.

Definition 1 Let $A_1 = (A_{11}, A_{12}, \ldots, A_{1n})$ and $A_2 = (A_{21}, A_{22}, \ldots, A_{2n})$ be two antecedents of rules. We say

$A_1 \preceq A_2$ if and only if $A_{1i} \subseteq A_{2i}$ $\forall$ i=1 to n.

This order relation represents the idea of inclusion of antecedents, A_1 is included in A_2, since all example satisfying in a positive degree the antecedent A_1, also satisfy in a positive degree the antecedent A_2. On the other hand, using the binary coding of the antecedent, this relation is equivalent to a component to component comparison of the zero one vector, that is, using the IRIS database, the following relation is verified

$$10000\ 00001\ 00011\ 11111 \preceq 11111\ 00001\ 01111\ 11111.$$

Obviously, $\preceq$ verifies the reflexivity, antisymmetry and transitivity properties, then it is an order relation. However, it is not a total order relation.

From this definition, it is very simple to extend the relationship to rules.

Definition 2 Let $A_1 = (A_{11}, A_{12}, \ldots, A_{1n})$ and $A_2 = (A_{21}, A_{22}, \ldots, A_{2n})$ the antecedents of rules $R_{B_1}(A_1, w_1)$ and $R_{B_2}(A_2, w_2)$ respectively. We say

$$R_{B_1}(A_1, w_1) \preceq R_{B_2}(A_2, w_2) \text{ if and only if}$$

$$A_1 \preceq A_2 \ \wedge \ B_1 = B_2$$

and the number of positive and negative examples covered by both rules is equal.

That is, a rule is less than or equal to another rule if both have the same consequent, the same covered examples, and therefore the same weight, but the antecedent of the first rule is less than or equal to the antecedent of the second one.

The relation $\preceq$ for rules is not a total order relation and represents a comparison of rules from the point of view of the length of the description of the rule antecedent.

To understand this idea, let us realize that the rule model we are using must be interpreted as a discriminant description, in the sense given in [29], i. e. a rule specifies a single way or several alternative ways to distinguish the class of the rule from the rest of classes. As indicated in [29] one of the most interesting descriptions is the *minimal discriminant descriptions* that are the shortest in the sense of the length of the description. This minimal discriminant description corresponds to a maximum of the order relation $\preceq$ of all the descriptions valid for a class. This is because the maximum is the one that includes the most values in each variable, and therefore the maximum is the one that eliminates the most variables. In the same way, we can obtain the maximal discriminant descriptions as the minimum of the order relation $\preceq$ of all the descriptions valid for a class.

Thus, given a rule R we could calculate the complete set of rules related to R through the relation $\preceq$ and we can represent this set through its lower and upper bounds.

The lower end of this set of rules can be obtained by selecting the rule $\underline{R}$ that verifies

$$\underline{R} \preceq R \text{ and } \nexists R'/R' \preceq \underline{R}$$

and this lower end corresponds to the maximal discriminant description of the rule R, that we call, large discriminant rule.

Similarly we can obtain the upper end of this set of rules as the rule $\overline{R}$ that verifies

$$R \preceq \overline{R} \text{ and } \nexists R'/\overline{R} \preceq R'$$

and this upper end corresponds to the minimal discriminant description of the rule R, that we call, short discriminant rule.

Somehow all the rules that are found between $\underline{R}$ and $\overline{R}$ through the relation $\preceq$, are rules equivalent to R from the point of view that are similar descriptions for a problem, given the current set of examples.

Using the binary description of the rule's antecedent, we could say that in the maximal discriminant description a number one is added in the description if there is no negative evidence to add the value that represents this number one. Instead, in the minimal discriminant description, number one is added, only when there is positive evidence that allows us to add that value.

As we have said before, we want to extend the rule model so that in some way it serves as a memory of how the rule works in past examples. Our idea is to move from the simple rule to a new rule set structure, represented by the short description and large description of the rule. Each large and short description rule pair, through the order relation, defines a set of rules that are adapted correctly to the examples processed so far. This set of rules is defined by all those rules that are located between the large description rule and the short description rule, using the order relation between rules. That "interval" of rules is just the type of memory we were looking for and it will be the one that the incremental learning algorithm will use.

Specifically, our learning algorithm will always obtain large description rules. From those rules, we will calculate their short description versions.

Using the previous example of the IRIS database, let us suppose we obtain a maximal discriminant description as

SL	SW	PL	PW
11100	01111	11011	11000

being its associated linguistic description

SL is $\{VL, L, M\}$ and SW is $\{L, M, H, VH\}$ and

$$PL \text{ is } \{VL, L, H, VH\} \text{ and } PW \text{ is } \{VL, L\}$$

and from this rule we calculate an equivalent minimal discriminant description

SL	SW	PL	PW
11111	11111	11011	11000

being its associated linguistic description

$$PL \text{ is } \{VL, L, H, VH\} \text{ and } PW \text{ is } \{VL, L\},$$

the assumption in both cases is that rules have the same consequent, number of positive and negative examples and weight. Both rules represent a lower and upper boundary respectively of the rule set compatible with the current information.

The code associated with this set of rule will be an extension of the previous rule code. The main component of the representation is the antecedent part and the idea is to extend the zero-one coding of an antecedent to a code with three values:

- 1 the value is present in the antecedent,
- 0 the value is not present in the antecedent,
- ? the value may or may not be present in the antecedent.

So, we establish a new codification for the antecedent of a set of rules in the following way:

$$Code(\widetilde{A}) = \{Code_i(\widetilde{A})\}_{i=1\ldots p}$$

being now $Code_i(\widetilde{A}) \in \{0, 1, ?\}$, p $= p_1 + p_2 + \ldots + p_n$ and $\widetilde{A}$ a representation of a set of antecedents between the minimal and maximal discriminant description using the order relation.

Given the large discriminant rule

$$\underline{R} = R_B(\underline{A}, w)$$

and the associated short discriminant rule

$$\overline{R} = R_B(\overline{A}, w)$$

where $\underline{A}$ and $\overline{A}$ are the antecedents of $\underline{R}$ and $\overline{R}$ respectively, we can generate the code of the set of antecedent between both rules as:

- If $BinaryCode_i(\underline{A}) = 1$ and $BinaryCode_i(\overline{A}) = 1$ then $Code(\widetilde{A}) = 1$.
- If $BinaryCode_i(\underline{A}) = 0$ and $BinaryCode_i(\overline{A}) = 0$ then $Code(\widetilde{A}) = 0$.
- If $BinaryCode_i(\underline{A}) = 0$ and $BinaryCode_i(\overline{A}) = 1$ then $Code(\widetilde{A}) =?$.

Note that the option

$$BinaryCode_i(\underline{A}) = 1$$

and

$$BinaryCode_i(\overline{A}) = 0$$

is not possible since $\underline{A} \preceq \overline{A}$. $Code(\widetilde{A})$ represents the antecedent of the set of rules.

Using the previous example of the IRIS database, the following description:

SL	SW	PL	PW
111??	?1111	11011	11000

represents a set of antecedent of rules. Specifically, the set of antecedents between the minimal and maximal discriminant rule.

The representation of the antecedent of $\underline{A}$ is obtained by instantiating all "?" values with "0", and the representation of the antecedent of $\overline{A}$ is obtained by instantiating all "?" values with '1", as long as the restrictions of the $\preceq$ relation between rules are verified.

The number of antecedents contained in a particular set of antecedent of rules $\widetilde{A}$, which will be called $Cardinal(\widetilde{A})$, can be calculated from the number of "?" symbols in the coding. Particularly, this number is 2^k, being k the number of "?". So, in the previous example, the subset is composed by $2^3 = 8$ antecedents, where the antecedents of the short and large description rules establish the upper and lower ends in this representation.

To codify a set of rules, we first codify the set of antecedents, as shown above, and add the consequent rule and the weight, along with the number of positive and negative examples of the associated short and large discriminant rule.

$$Code(R_B(\widetilde{A}), w) = (Code(\widetilde{A}), B, w, N^+_{inf}, N^-_{inf}, N^+_{sup}, N^-_{sup})$$

where $Code(\widetilde{A})$ is the coding of the set of antecedents, B is the consequent or class of the rule, w is the weight, N^+_{inf} and N^-_{inf} represent the number of positive and negative examples values of the large discriminant rule respectively, and in a similar way N^+_{sup} and N^-_{sup} refer to the values of the number of positive and negative examples of the short discriminant rule.

The set of rules defined by this model flexibly represents all rules compatible with the current information, that is, with the examples that have been processed by the algorithm so far (even when they are no longer stored).

4.3 Inference using sets of rules

It is necessary to define the process of inference when we work with rules such as those described in the previous subsection, that is, we need to define the inference for a set of rules, where each one of these rules represent a set of rules again, precisely all those included between the large and short discriminant rules.

A large discriminant rule is the most specific rule consistent with the information, so if an example verifies this rule it means that there are examples in the training set similar to the current example, and we should use this rule to classify. However, if that large discriminant rule does not classify it could mean that you need more general rules for classifying. Thus, our proposal is to use first the large discriminant rules and if one of them is triggered, then the class of that rule is assigned to the example. If no one is triggered, then the short discriminant rules are used, and the class corresponding to the rule triggered with a higher value is assigned to the example. Otherwise, no class is assigned.

5 Learning algorithm

Since our goal is to use incremental learning algorithms for solving big data classification problems using fuzzy rules, we need to take into account some common characteristics of the incremental learning algorithms for its application. These characteristics are that the concept to be learned may change over time (concept drift) and that the full set of examples is not known. However, in our case, as stated above the concept drift problem can be ignored as there is no need to consider a change of concept to learn and the size of the example set is large but it is known. The idea is to take advantage of these two peculiarities by proposing a batch-incremental learning scheme for classification problems.

In [16] a first approach towards an incremental learning algorithm based on the sequential covering strategy was proposed. This algorithm is based on a fuzzy rule learning algorithm called NSLV-AR [15]. One of the main features of this algorithm is its ability to learn a set of fuzzy rules based on a previous set of fuzzy rules and new examples. However, the proposed algorithm stores all the examples, that is, it is a full instance memory incremental learning algorithm, and this makes it not directly applicable to big data problems as we explained before. Therefore we need to define an incremental learning algorithm with partial memory. The proposed algorithm is based on NSLV-AR, and in the next section, we briefly explain the main ideas of this algorithm.

5.1 NSLV-AR

NSLV-AR is a fuzzy rule learning algorithm based on the use of the sequential covering strategy [30]. This strategy uses a decomposition model in order to iteratively learn a single rule on each step. Thus, each learned rule is added to the rule base, being part of the rule base until the end of the learning process. However, the sequential covering strategy of NLSV-AR described in Algorithm 1 is characterized by the revision of already learned rules in each iteration. This constant revision of rules allows NSLV-AR to modify/adapt previously learned rules on a new set of examples, and this behavior is the key to incremental learning algorithms.

Algorithm 1 NSLV-AR

```
1: function NSLV-AR(E,Learned_rules)
2:     Action ← Return_One_Action (E, f,
3:             Learned_rules, new_rule, rules_to_remove)
4:     while (Action ≠ STOP) do
5:         Learned_rules ← Add (Learned_rules,
6:                                 new_rule)
7:         if (Action == REPLACE) then
8:             Learned_rules ← Remove
9:                         (Learned_rules, rules_to_remove)
10:        end if
11:        E ← Penalize (Learned_rules, E)
12:        Action ← Return_One_Action (E, f,
13:             Learned_rules, new_rule, rules_to_remove)
14:    end while
15:    return Learned_rules
16: end function
```

The sequential covering algorithm of NSLV-AR described in Algorithm 1 has as inputs a set of examples E and a set of rules *Learned_rules*, which may be empty. It uses the procedure Return_One_Action, implemented through a genetic algorithm and outputs one of these actions: AGGREGATE, REPLACE or STOP. Return_One_Action analyzes two different fuzzy rules, the best rule increasing the accuracy and the best rule replacing rules, and it makes a decision about which one is the best choice. Depending on the decision, the procedure has a different output,

- AGGREGATE when the decision is adding a new rule,
- REPLACE when the decision is removing one or more rules replaced by a new rule also returning the list of replaceable rules (*rules_to_remove*), and finally
- STOP when the rule base cannot be improved and no action must be carried out.

The multi-criteria evaluation function used in NSLV-AR is:

$$f(R) = [\Psi_\delta(R),\ |\Omega(R)|,\ svar(R),\ sval(R)]$$

where:

- $\Psi_\delta(R)$ is the product of consistency and completeness conditions.
- $|\Omega(R)|$ is the number of rules that R can replace from the set of learned rules.
- $svar(R)$ is the number of irrelevant variables.
- $sval(R)$ is the number of understandable assignments.

Two rules are compared through the previous multicriteria evaluation function along with the use of lexicographical order.

The weight of a rule R in NSLV-AR is defined as:

$$\omega_E(R) = \frac{n_E^+(R) + 1}{n_E^+(R) + n_E^-(R) + 1}.$$

5.2 Adapting NSLV-AR to the incremental learning framework

The main feature of the NSLV-AR algorithm is that it allows us to remove rules included in the initial rule set or to add new rules (if needed) for obtaining a new rule set with a better representation of the examples included in the training set.

However, direct use of the algorithm for incremental learning is not recommended, as in its original design it assumes that all examples are available. When we use the algorithm on a batch of examples, obtained from the partitioning of the complete set of examples, some drawbacks may appear:

- The weight of each rule learned by the algorithm takes its value on the current training set. This is a problem because a rule with a high success rate for a particular subset of examples could be a rule with a low success rate for the whole set of examples. Rules with a high success rate frequently have a weight close to 1 and they are better than the rest of rules in the inference process. So, it is necessary to additionally include in NSLV-AR a specific process in the incremental learning algorithm that allows changing the weight of a rule.
- The remove rule process only takes into account the current training set. So, rules that are included in the algorithm that were learned in a previous subset are removed when they are not relevant for the classification of the current examples. Because this behavior may not be correct when the whole set of examples is not available, the rules that are added in previous episodes must be protected during the current iteration and they cannot be replaced. Obviously, this restriction reduces the ability to eliminate irrelevant rules. Therefore, it will be necessary to include an additional process in the incremental learning algorithm for removing rules with low weight.

Both issues will be tackled in the algorithm described below. In addition to all of the above, it is also necessary to make some modifications to NSLV-AR, specifically:

- Distinguish between the initial set of rules that are input in the algorithm and the rules that are added by the algorithm to the rule set so that only the latter can be modified.
- Decrease the dependence of this value on the current episode In order to make a better estimation of the weight of the rules. To do that we will use an extended set of examples that will include the current episode along with some previous episodes. Precisely making use of this extended set makes the learning algorithm be classified as a partial memory incremental algorithm.
- Extend the algorithm so that it can learn sets of rule pairs, a large discriminant rule, and a short discriminant rule, as described in Section 4. These pairs of rules try to capture those rules for the same class that have similarities in their description and that therefore represent adjacent zones in the space of the input variables. However, the original algorithm works with a non-paired rule set. The extension of the algorithm for considering paired rule sets is simple: during learning, only large discriminant rules are considered. In the end, for each new large discriminant rule, an associated short discriminant rule is obtained using a hill-climbing search algorithm.

5.3 Describing the incremental learning algorithm

As mentioned above, we initially have a set of examples E. Moreover, we assume that we have a partition of this set of examples

$$E = \{E_0 \cup E_2 \cup \ldots \cup E_{m-1}\}$$

where in each episode E_i there are $size$ examples, except perhaps the last one. $size$ will be a fixed parameter from the beginning. We have started the index i to zero in the first episode to facilitate the description of the algorithm, but we still maintain a partition with m sets.

We can add different elements of the partition using the following notation

$$AE(i,j) = \{E_i \cup E_{i+1} \cup \ldots \cup E_j\}$$

where we suppose that

$$E_i = \emptyset$$

when $i < 0$ or $i > m - 1$, and $i \leq j$.

The incremental algorithm includes three fundamental elements:

- A learning algorithm (NSLV-AR adapted) that from a new episode generates the new knowledge that did not exist using the previous examples.
- An aggregation procedure that combines the new knowledge gained with the episode with the existing knowledge associated with the previous examples.
- A procedure for adjusting the weights of a rule.

Our proposal of incremental learning algorithm is described in Algorithm 2.

Algorithm 2 An incremental algorithm adapted to big data

```
1: function INCREMENTAL (E, size, t, k)
2:     Thld = 1.00
3:     R = ∅
4:     i = 0
5:     E_i = E.Select(i, i + 1, size)
6:     while (E_i ≠ ∅) do
7:         R.UpdateWeight(E_i)
8:         R.FilterRule()
9:         E_i^→(t) = E.Select(i, i + t, size)
10:        if (R.Predict(E_i^→(t)) ≥ Thld) then
11:            Thld = UpdateThreshold(E_i^→(t), R, Thld)
12:            i = i + 1 + t
13:        else
14:            E_i^←(k) = E.Select(i − k, i, size)
15:            R_s = NSLV-AR(E_i, R.SubSet(E_i), E_i^←(k))
16:            R = R.CombineRule(R_s)
17:            Thld = UpdateThreshold(E_i, R, Thld)
18:            i = i + 1
19:        end if
20:        E_i = E.Select(i, i + 1, size)
21:    end while
22:    return R
23: end function
```

The algorithm has as inputs the complete set of examples and three parameters. These parameters allow you to generate three subsets of examples in each episode. The first subset is E_i which is the training set that will use the NSLV-AR algorithm in the i-th episode and which contains $size$ examples. The second one

$$E_i^{\rightarrow}(t) = AE(i, i + t - 1)$$

is the set that we will use to check if the set of rules previously obtained by the incremental learning algorithm satisfy the examples of several successive episodes. This set is formed by the union of the examples of the episode i-th and the following $t-1$ episodes, where t is a parameter. If we use t=1 then we only check the performance of the rules on the i-th episode examples. Finally, the subset

$$E_i^{\leftarrow}(k) = AE(i - k + 1, i)$$

is the set that we will use to estimate the weight of the new rules, and is composed by the union of the examples of the current episode and the k-1 episodes previous to the current one.

The algorithm begins by initializing four variables, the first one $Thld$ is the threshold, which represents a value between 0 and 1, from which it is considered that the accuracy of the set of rules obtained in previous episodes is sufficient and it is not necessary to learn using the examples of the new episode (or extended episode). It will initially take the value 1. The second variable is R which represents the complete set of rules. It's initially empty. The third one is i which represents the index of the current episode, and finally, E_i which is the current episode, which is initiated through the $Select$ procedure. $E.Select(a, b, c)$ extracts the subset of examples of the E set from the $a \times c$ position to the $(b \times c) - 1$ position.

Just after the initialization of the four variables, a cycle begins. This cycle represents an iterative procedure that is repeated until there are no more examples to process, and therefore the current episode is empty. Within the cycle, the first thing to do is to update the weight of the rules obtained in previous episodes with the examples of the current episode (line 7) and to remove bad rules (line 8). The idea is to know the capacity to predict the current set of rules on new examples. These two processes are related to those described in subsection 5.2. The procedure R.FilterRules() has a simpler implementation, it removes all the rules in R with weight less than

$$\frac{0.9}{\#classes}$$

being $\#classes$ is the number of classes involved in the classification problem.

We then select an extended set of examples (line 9) from the current episode (determined by parameter t). The R.Predict($E_i^{\rightarrow}(t)$) procedure returns a number between 0 and 1, which represents the capacity of prediction of the rule set over the extended set of examples. This value is compared with the threshold. When the value is larger than or equal to the threshold then it is not necessary to learn with these examples, as the rules sufficiently represent them. In this case, the threshold is recalculated taking into account the new examples, then all the episodes corresponding to the extended set of examples are skipped.

In the other case, when the value obtained is less strict than the threshold, it is necessary to learn again. Thus, on line 15, the NSLV-AR algorithm will produce new rules. To do this, R.SubSet(E_i) first extracts the subset of rules of the complete set of rules R that are triggered on the examples of E_i and we use the extended set of examples $E_i^{\leftarrow}(k)$ (using parameter k) to estimate the weights.

The result of applying NSLV-AR is a new set of rules R_s. This set represents those new rules that need to be added to the complete set of rules in order to represent the new examples. But, before adding them, it is necessary (line 16) to combine the new rules with the old ones. At this point it is important to remember that the algorithm is working with short discriminant rules, but each one of these rules has associated a large discriminant rule. In this way, the rule model we are considering is an interval of rules contained between these two extreme rules. Thus, when we have a previously obtained rule interval, and another rule interval obtained in the current iteration, the idea of the combination is to keep the intersection of both intervals, which represents the consistent information between the previously learned rules and the new rules.

To do this, the R.CombineRule(R_s) procedure will first add all new rules that are not related to those previously obtained, and then add those combined rules in the intersection between the intervals corresponding to the previously obtained rules and the new rules. This procedure will be described in Subsection 5.3.2.

Next (line 17) we update the threshold value taking into account only the examples of the current episode, and we move on to the next episode and repeat the loop. Finally, the algorithm ends up returning the R rule set.

There are two aspects of the description of the above algorithm that need to be described in more detail, the first being how to update the threshold and the second is how to combine rules.

5.3.1 Threshold estimation

A correct update of the threshold is fundamental for the previous incremental algorithm. In fact, the threshold determines whether the current rules correctly represent current examples and we do not need to learn with these examples, or whether it is necessary to learn new rules. Therefore, the threshold value will definitely affect the final accuracy of the rule set and the time needed to obtain it.

The most appropriate value for the threshold would be the accuracy on the complete set of examples E, but obviously, this value is not known. There are undoubtedly several ways to estimate the threshold parameter. In our experimentation, we have used as initial value 1 and subsequently, for each iteration, the value is updated by taking a 10% reduction in the average individual predictive capacity over each of the previous episodes. This value can be considered as a pessimistic estimation of the accuracy.

5.3.2 Combining rules

The incremental learning algorithm will use sets of rules instead of simple rules. A key element in this algorithm is the process of adapting sets of rules learned in previous episodes to the information given by the new set of examples. To do this we define a method of combining compatible sets of rules.

Definition 3 We say that two sets of rules $R_B(\widetilde{A}_1, w)$ and $R_B(\widetilde{A}_2, w)$ are compatible if and only if the associated intervals to each rule overlap, that is, there is at least one simple rule included in both sets of rules.

In relation to the code of the set of rules the compatibility can be easily checked:

$R_B(\widetilde{A}_1, w)$ and $R_B(\widetilde{A}_2, w)$ are two sets of rules compatible when

$$\forall i \; Code_i(\widetilde{A}_1) = Code_i(\widetilde{A}_2)$$

or

$$\text{if } Code_i(\widetilde{A}_1) \neq Code_i(\widetilde{A}_2)$$

then

$$Code_i(\widetilde{A}_1) =? \text{ or } Code_i(\widetilde{A}_2) =?.$$

When $R_B(\widetilde{A}_1, w)$ and $R_B(\widetilde{A}_2, w)$ are compatible we can combine both pattern rules in a new pattern rule that includes those rules contained in both pattern rules. Thus, we define

$$combine(R_B(\widetilde{A}_1, w), R_B(\widetilde{A}_2, w)) = R_B(\widetilde{A}_3, w)$$

where

$$Code(R_B(\widetilde{A}_3)) = (Code(\widetilde{A}_3), B, w, N_{inf}^{3+}, N_{inf}^{3-}, N_{sup}^{3+}, N_{sup}^{3-})$$

and

$$Code_i(\widetilde{A}_3) = \begin{cases} ? & \text{if } Code_i(\widetilde{A}_1) = Code_i(\widetilde{A}_2) =? \\ Code_i(\widetilde{A}_1) & \text{if } Code_i(\widetilde{A}_1) \neq? \\ Code_i(\widetilde{A}_2) & \text{otherwise} \end{cases}$$

and if $Cardinal(Code(\widetilde{A}_1)) \geq Cardinal(Code(\widetilde{A}_2))$ then

$$N_a^{3*} = N_a^{1*}$$

in other case,

$$N_a^{3*} = N_a^{1*} + N_a^{2*}$$

where $a = \{inf, sup\}$ and $* = \{+, -\}$.

6 Experimental section

In this section, we study the behavior and scalability of the proposal introduced in the previous sections, as well as the adjustment of some of its parameters. Subsection 6.1 includes a study on the granularity of the domain. Subsection 6.2, 6.4 and 6.3 analyze the influence of the *size*, t and k parameter, respectively. Subsection 6.5 shows an scalability study. Finally. Subsection 6.6 contains a comparison of the proposal with parallel models.

In this experimentation, we will use seven databases from the UCI dataset repository [2], which are described in Table 1. In this table, $Nemo$ refers to the short name for each dataset, $\#Ex$ represents the number of examples and $\#Atts$ shows the number of attributes, where the C component reflects the number of continuous attributes and N the number of nominal attributes. As occurred in [14] and [36], some of the selected databases have multiple classes, so we maintain the same criteria reducing these problems to two classes. To perform the comparison we applied the Wilcoxon's test in each experiment. In every case, we used a significance level of $\alpha = 0.05$.

Table 1 Summary of Databases. The names of several databases have been changed to match the ones in [14].

Datasets	Nemo	#Ex	#Atts (C/N)
Census	cens	141544	41 (8/32)
Covtype_2_vs_1	covt	495141	54 (10/44)
Fars_Fatal_Inj_vs_No_Inj	fars	62123	29 (5/24)
Kddcup_DOS_vs_normal	kddc	4856151	41 (26/15)
Poker_0_vs_1	poke	946799	10 (0/10)
Susy	susy	5000000	18 (18/0)
HepmassAll	hepm	10500000	28 (27/1)

With respect to the infrastructure used to perform the experiments, we have used the cluster of the research group Soft Computing and Intelligent Information Systems (http://sci2s.ugr.es). This cluster has 16 nodes connected with a 40Gb/s Infiniband. Each node is equipped with two Intel E5-2620 microprocessors (at 2 GHz 15MB cache) and 64GB of main memory running under Linux CentOS 6.5. The head node of the cluster is equipped with two Intel E5645 microprocessors (at 2.4 GHz, 12MB cache) and 96GB of main memory. In the experimentation, all the processes have been executed in a sequential fashion using only one node.

6.1 Study on the granularity of the domain

The first study is dedicated to analyzing the best granularity associated with the fuzzy domain. To do so, we have studied the behavior of the algorithm on the databases we have previously selected and with the use of three, five, seven, nine and eleven fuzzy linguistics labels uniformly distributed over the universe of values of each continuous variable.

Table 2 Average prediction capacity (Test) for the different granularity values

	3 labels	5 labels	7 labels	9 labels	11 labels
cens	94.20	94.10	94.22	94.28	94.31
covt	71.19	75.46	75.33	75.59	75.85
fars	100.00	100.00	100.00	100.00	100.00
kddc	99.82	99.74	99.76	99.81	99.77
poke	54.52	54.52	54.52	54.52	54.52
susy	64.90	68.76	69.36	68.43	68.16
hepm	73.83	78.74	80.35	80.24	79.97
mean	79.78	81.62	81.93	81.84	81.80

Table 3 Running time (in seconds) for the different granularity value

	3 labels	5 labels	7 labels	9 labels	11 labels
cens	88.40	101.02	94.63	102.69	102.19
covt	135.72	355.90	456.47	580.64	942.74
fars	13.60	13.49	11.69	13.48	12.34
kddc	139.89	144.63	150.52	175.30	160.36
poke	417.14	417.14	417.14	417.14	417.14
susy	676.17	1582.33	1899.73	3998.54	2346.73
hepm	6691.17	6657.74	4284.89	5485.56	7336.40
Mean	1166.01	1324.61	1045.01	1539.05	1616.84

Table 4 Average number of rules for the different granularity values

	3 labels	5 labels	7 labels	9 labels	11 labels
cens	15.10	16.89	15.10	14.50	14.40
covt	22.60	67.00	85.40	97.90	150.50
fars	2.90	2.80	2.70	3.00	2.80
kddc	5.90	5.60	5.50	6.10	5.20
poke	209.00	209.00	209.00	209.00	209.00
susy	59.60	126.00	129.40	238.30	133.90
hepm	208.40	193.80	113.90	128.30	158.90
Mean	74.79	88.73	80.14	99.59	96.39

Table 5 Average number of variables per rule for the different granularity value

	3 labels	5 labels	7 labels	9 labels	11 labels
cens	29.59	31.54	32.19	31.60	31.24
covt	39.81	44.34	45.21	44.83	46.01
fars	19.38	20.20	21.88	22.06	22.93
kddc	37.42	37.95	37.95	38.23	37.95
poker	6.99	6.99	6.99	6.99	6.99
susy	16.38	16.85	17.22	17.52	17.67
hepm	23.86	25.41	26.18	26.62	27.13
Mean	24.78	26.18	26.80	26.83	27.13

The rest of the parameters used are the following:

- the *size* parameter that establishes the number of examples in each episode, fixed to 1000,
- the t parameter which corresponds to the number of episodes used for testing the threshold value, fixed to 1,
- the k parameter indicating the number of episodes used for obtaining an approximation to the weight of each new rule, fixed to 10.
- default configuration for the parameters of the NSLV-AR algorithm.

Tables 2, 3, 4 and 5 show the accuracy on unseen examples, running time, the number of obtained rules and the average number of variables per rule, respectively on each database and each granularity value. In all tables, the final row shows the average value of each column.

In relation to the average prediction capacity (Table 2) we can observe that the results are quite similar except in the case of 3 labels. This fact does not indicate that the number of labels is an irrelevant factor, but that the appropriate value for each problem is different. For example, for cens and covt the best value is 11 while for hepm is 7. In fars and poke all values are the same. In the case of poke, it is natural, since all its variables are nominal or integer.

The increase in the number of labels directly affects the size of the solution space. In the case of a problem with n continuous variables and l labels, the solution space associated with the rule model used in this approach is $2^{n \times l}$ for each rule. Therefore, an increase in l implies an exponential increase in the solutions space. Table 3 shows how this increase in solution space affects runtime. Both the global mean behavior and the particular behavior of each database do not show changes in execution times that are related to exponential increases in the solution space.

Table 4 shows the average number of rules obtained for each number of labels. In this case, the general trend indicates a direct correlation between the number of rules and the number of labels. However, this increase does not have an exponential rate.

Table 5 shows that the behavior in relation to the average number of variables per rule is very stable in relation to the use of a different number of labels to discretize the domain, although a slight increase of the value can be observed when the granularity is increased.

Using the average results of all the previous tables, we can observe that the case of 7 labels is the one that shows a good compromise. This number of labels will be used from now on in the following experiments.

6.2 Studying on the influence of the *size* parameter

The second study aims to determine the possible influence of episode size on accuracy and time required to learn.

The *size* parameter establishes the number of examples that are presented to the learning algorithm in each iteration of the incremental process. The learning algorithm (NSLV-AR in our case) is the most time-consuming task of our proposal. The time spent by NSLV-AR grows with the number of examples in the training set.

The other two tasks that use high amount of time are **UpdateWeight** and **CombineRule**. The latter is not directly dependent on the number of examples, as it only depends on the number of learned rules. The time consumed by the former depends on three factors: number of examples (*size*), number of rules ($\#rule$) and number of variables involved in the learning problem ($\#Attribute$), achieving a complexity order of $O(size \times \#rule \times \#Attribute)$.

In Tables 6 and 7 the results obtained by the incremental proposal using different values for the *size* parameter on the different databases are shown. In relation to the rest of the parameters of this experiment, we have considered 7 labels for continuous variables, t=1 and k=10.

From these tables, we can observe that the running time grows with the size of the episode, except for the initial value *size*=500. For this size, the algorithm obtains a larger running time since it produces a large number of rules that increase the needed time for the tasks **UpdateWeight** and **CombineRule**.

Almost the same conclusion (growths as *size* grows) can be detected when the predictive capacity is examined, although the increase is smaller and does not always occur.

For determining an appropriate value for the *size* parameter that combines an acceptable prediction ca-

Table 6 Accuracy on test set (in %) for different values of the *size* parameter

	500	1000	2000	3000	4000	5000	10000	20000
cens	94.22	94.22	94.31	94.37	94.36	94.39	94.54	94.71
covt	75.11	75.33	76.65	76.95	76.94	77.01	78.49	78.61
fars	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
kddc	99.74	99.76	99.79	99.86	99.89	99.92	99.91	99.94
poke	54.33	54.52	54.50	55.78	55.38	55.61	57.11	56.13
susy	65.20	69.36	69.30	71.51	72.09	73.37	71.41	72.71
hepm	80.23	80.35	80.84	79.28	80.16	81.46	80.96	81.65
Mean	81.26	81.93	82.42	82.54	82.69	83.11	83.20	83.39

Table 7 Running time (in seconds) for different values of the *size* parameter

	500	1000	2000	3000	4000	5000	10000	20000
cens	51.33	94.63	233.35	388.53	612.74	823.14	2139.14	6240.90
covt	420.65	456.47	752.27	1026.79	1296.73	2081.81	4027.96	7260.38
fars	7.88	11.69	18.56	25.18	34.90	44.74	80.30	149.56
kddc	149.02	150.52	226.16	242.24	290.39	355.31	604.74	1276.29
poke	1493.88	417.14	182.12	306.20	141.38	298.81	270.66	421.61
susy	11457.40	1899.73	1869.81	2246.53	3048.81	4243.24	3606.05	6445.82
hepm	8957.19	4284.89	6662.83	9767.55	7834.32	12619.70	22557.70	50928.40
Mean	3219.62	1045.01	1420.72	2000.43	1894.18	2923.82	4755.22	10388.99

Table 8 Average number of rules for different values of the *size* parameter

	500	1000	2000	3000	4000	5000	10000	20000
cens	14.90	15.10	21.80	23.30	27.30	26.90	34.29	48.80
covt	114.30	85.40	84.10	80.40	84.60	107.10	99.30	84.60
fars	2.90	2.70	2.80	2.80	2.90	2.90	2.60	2.20
kddc	5.40	6.10	7.20	6.90	7.80	8.30	7.40	9.60
poke	1197.20	209.00	58.60	74.50	27.70	49.70	20.39	14.80
susy	1166.77	238.30	97.20	102.00	129.00	156.69	86.80	98.30
hepm	312.60	113.90	183.28	262.37	182.60	292.25	377.00	481.66
Mean	402.01	80.14	65.00	79.00	65.99	91.98	89.68	105.71

pacity in an acceptable time we can calculate the ratio:

$$\frac{PredictionCapacity(\%)}{Time(sg)}$$

Figure 3 graphically shows the value of this ratio for different values of *size*, and we can observe that in $size = 1000$ the maximum value is obtained, that is, for the databases considered in this experimentation the best value for the size parameter that combines the accuracy on non-seen examples and running time is 1000.

6.3 Studying on the influence of the t parameter

The t parameter in the incremental proposal is related to the number of episodes used to detect whether the current set of rules correctly represents the new examples. If this is the case, it is not necessary to do anything with the examples of such episodes, otherwise, the algorithm returns to work with a single episode and learn again. Obviously, parameter t is intended to accelerate the learning process by skipping batches of correctly represented examples.

So far in our studies, we have used only one episode ($t = 1$), but we want to study the influence of using this parameter. We maintain *size*=1000 in all cases.

In Tables 9 and 10 the accuracy and running time obtained for different values of the t parameter, respectively, are shown.

Table 9 Accuracy on test set (in %) for different values of the t parameter fixing $size = 1000$

	1	5	10	15
cens	94.21	94.21	94.22	94.22
covt	75.39	75.57	75.44	75.45
fars	100.00	99.99	99.99	99.99
kddc	99.70	99.66	99.66	99.66
poke	54.50	54.17	54.45	54.14
susy	69.30	69.64	69.54	69.36
hepm	80.30	80.32	79.94	80.07
Mean	81.90	81.94	81.89	81.84

From the previous tables, we can conclude that the t parameter has no effect on the prediction capacity. It is possible to observe a slight reduction when t increases but we can check that the prediction capacity is

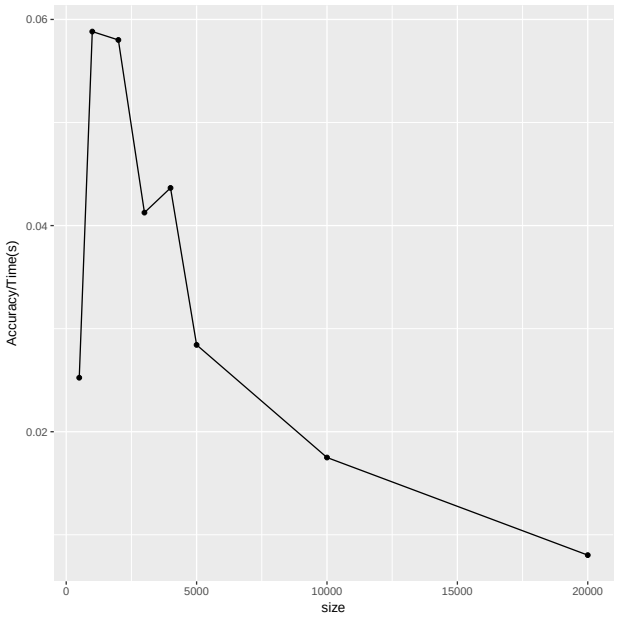

Fig. 3 Graphical representation of the rate between prediction capacity and time required to learn for different values of *size*

Table 10 Running time (in seconds) for different values of the t parameter fixing $size = 1000$

	1	5	10	15
cens	94.60	94.44	93.04	100.98
covt	456.40	322.70	278.99	276.54
fars	11.60	10.67	10.49	11.01
kddc	150.50	88.32	83.69	83.34
poke	417.10	131.93	245.78	159.20
susy	1899.70	794.85	854.77	965.80
hepm	4284.80	3771.92	3257.22	3294.09
Mean	1044.96	744.98	689.14	698.71

Table 11 Number of rules for different values of the t parameter fixing $size = 1000$

	1	5	10	15
cens	15.00	15.00	15.00	15.00
covt	92.60	74.30	66.70	66.70
fars	2.70	2.70	2.70	2.70
kddc	5.70	5.70	5.70	5.70
poke	186.80	86.90	160.50	107.30
susy	175.90	75.70	93.40	115.10
hepm	177.50	167.20	157.90	161.69
Mean	93.74	61.07	71.70	67.74

practically constant. However, it is easy to verify that it has a great influence on the time required to learn.

This influence can be clearly appreciated in Figure 4, where the ratio between accuracy and runtime is shown for different t values. It can be observed that for values of t less than 10 the ratio increases progressively. From $t = 10$ that value is maintained and even decreases a little. This behavior shows that when the size of $E_i^{\rightarrow}(t)$ is greater than the size of an episode the learning time is reduced while accuracy is maintained. This improvement is due to the fact that the value of the estimated threshold triggers less re-learning processes since it has more global information of how the following examples are that you will have to process. An excessive value in this parameter would not provide this advantage since the time consumed in the verification does not compensate for the reduction in time.

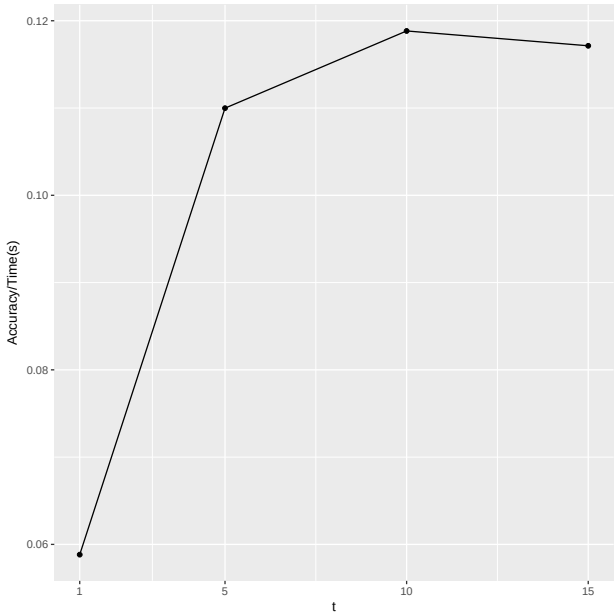

Fig. 4 Graphical representation of accuracy/running time for different values of the t parameter, fixing $size = 1000$

Therefore, the experiments performed show that to take $t = 10$ can significantly reduce the learning time, without affecting the prediction capacity.

6.4 Study on the influence of the k parameter

The k parameter determines the size of the set of examples ($E_i^{\leftarrow}(k)$) used to estimate the weight of the rules learned in the $i-th$ episode. In this subsection, we want to illustrate the influence of this parameter on the behavior of the proposal. Thus, for this experiment, we set the number of labels to 7, $size$=1000 and $t = 10$. The rest of the parameters, that are not subject of this study, remain with the same value.

Tables 12 and 13 show the prediction capacity and the execution time of the proposal for 4 different values of k. Among these values, $k = 1$ has been considered, that is, $E_i^{\leftarrow}(k) = E_i$.

Table 12 Accuracy on test set (in %) for different values of the k parameter fixing $size = 1000$ and $t = 10$

	1	10	20	30
cens	92.62	94.22	94.21	94.22
covt	69.14	75.44	75.57	75.45
fars	99.99	99.99	99.99	99.99
kddc	99.66	99.66	99.66	99.66
poke	54.53	54.45	54.17	54.14
susy	57.17	69.54	68.64	69.36
hepm	62.43	79.94	80.32	80.07
Mean	76.51	81.89	81.79	81.84

Table 13 Running time (in seconds) for different values of the k parameter fixing $size = 1000$ and $t = 10$

	1	10	20	30
cens	132.16	93.04	95.49	99.95
covt	630.45	278.99	301.37	303.68
fars	12.92	10.49	10.67	11.50
kddc	145.73	83.69	86.04	92.39
poke	831.83	245.78	259.30	271.84
susy	3238.92	854.77	850.91	884.82
hepm	7134.20	3257.22	3667.22	3792.01
Mean	1732.32	689.14	753.00	779.46

The obtained results show that for $k = 1$ the proposal has a significantly lower prediction capacity and a significantly longer execution time than for the rest of the values considered of k. This fact shows the usefulness of the $E_i^{\leftarrow}(k)$ set to update the weight of the rules taking into account previous examples.

Except for k=1, for the rest of values of k there are no significant differences in time or prediction capacity. Also, it can be observed that while in precision the average values remain constant, in the case of the execution time a slight increase can be appreciated as k grows. For this reason, we consider that $k = 10$ is an appropriate value for this parameter.

6.5 Study on scalability

Now we want to show how the proposal works as the number of examples in the training set increases. Table 14 shows the results obtained at runtime and number of rules for the susy database considering $size = 1000$, $t = 10$ and $k = 10$.

Table 14 Size-up on the Susy database

	1%	10%	20%	30%	100%
time	68.5	235.9	409.6	412.9	689.1
rules	61.9	63.1	61.7	74.5	93.4

The figure 5 shows the growth in runtime taking different percentages of examples from the database. Two tendencies can be clearly observed, on the one hand

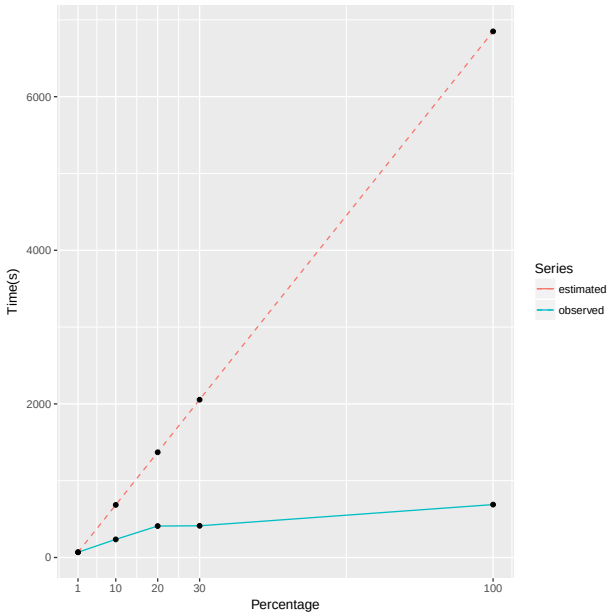

Fig. 5 Graphical representation of the relation between percentage of data used of the Susy database and running time

in the initial phase of the process (in the case of susy up to 20%) the time grows with a steeper slope and a second part where the growth is maintained with a much milder slope. The first part reflects that the algorithm needs time to find a set of rules that accommodate the new episodes. During this time the process of re-learning new rules to readjust the knowledge is executed more times, which causes a higher time consumption. In the second part, the slope of growth is lower because it is not necessary to re-learn so many times.

The dashed line reflects the estimated time for execution of the algorithm assuming that each episode learns independently without making use of the knowledge generated in previous episodes. Compared to that theoretical time, the actual execution consumes only about 10%.

In relation to the number of rules, as shown in Table 14, we can observe a slight growth depending on the number of examples used, but in no case an exponential growth.

From these results, we can deduce that the algorithm presents a good efficiency in relation to the increase in the size of the sets of examples both in execution time and in the number of rules obtained.

6.6 Comparison of the proposal with parallel models

Now, we want to study the performance of this proposal in comparison with parallel models. We will carry out

the comparison using the same databases described in the paper, which coincide with the first five of those initially described. Specifically, we first compare with the proposal describe in [36]. In this paper, a classification algorithm (called **Chi-FRBCS-BigData approach**) for addressing the Big Data problems was proposed. This algorithm combines the MapReduce approach with a fuzzy classification algorithm called **Chi** [4]. In [36], a study of the influence of the number of *maps* in the accuracy and running time on the Chi-FRBCS-BigData approach is presented. In [11] a new study on the same algorithm is proposed. In that work, an analysis on the most appropriate number of fuzzy labels used for discretizing the domain of continuous variables is included. The second paper uses the same databases that have been used in this work. Furthermore, in this second work, a new value of *maps* 512 is included. The global results obtained for each value of *maps* taking 7 fuzzy labels (that is, with the same discretization used in the global experimental part) is shown in Table 15.

Table 15 Results in prediction capacity and running time of the overall average of the datasets for different value of *maps* using the algorithm based on the MapReduce approach proposed in [11] using a discretization of 7 fuzzy labels

	#Maps				
	32	64	128	256	512
Test	88.57	88.51	88.41	88.33	88.27
Time	8331.4	4886.5	5248.4	6012.56	6727.2

This table shows that the classification algorithm based on the MapReduce model presents an accuracy similar that our proposal. When the number of examples presented to the algorithm increases the global accuracy of the knowledge obtained also increases. In this case, a small value in *maps* represents a training set with a high number of examples.

Using the same rate between prediction capacity and time used in the previous subsection we can determine the most convenient number of maps. In Figure 6 the distribution of these values obtained for each *maps* value is shown.

Figure 6 shows that the best combined value is found for $maps = 64$.

To make the comparison of the prediction capability we will use the Area Under the ROC Curve (AUC) used in [9].

Table 16 presents the result obtained by both the parallel and incremental version (this last with $size = 1000$, $k = 10$ and $t = 10$).

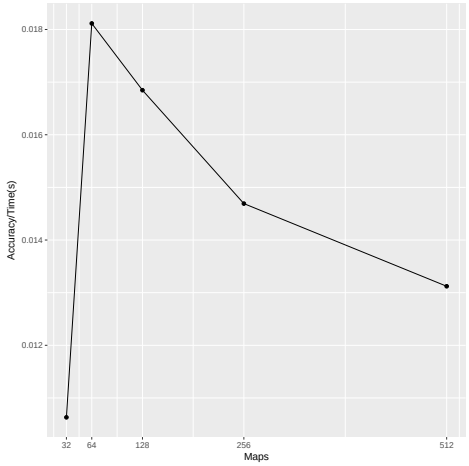

Fig. 6 Graphical representation of the rate between accuracy(%) and running time (secs) obtained for each value of *maps* in [11] using a discretization of 7 fuzzy labels

Table 16 Comparison on prediction capacity between the incremental algorithm and parallel approach proposed in [11]. The version of the incremental algorithm with the following parameter $size = 1000$, $k = 10$ and $t = 10$. Two aspects have been studied, capacity of prediction (AUC) and runtime (time).

	Incremental		Parallel	
	AUC	time	AUC	time
cens	0.56	94.4	0.68	406.8
covt	0.92	322.7	0.87	4282.9
fars	1.00	10.7	0.99	159.4
kddc	0.99	88.3	0,99	15169.5
poke	0.54	131.9	0.59	4414.0

Table 16 shows that the incremental version is significantly better in learning time, and gets the best result on all databases studied. As for the prediction capability, there are no significant differences. The MapReduce version clearly gets the best result on poke and cens, while the rest of the results get very similar values.

In [9] CHI-BD an improved version of the parallel model proposed in [11,36] is described. This model also uses Chi as a learning algorithm. The advantage proposed in this work is that the set of rules obtained after the learning process is independent of the number of maps, that is to say, it would be the same set of rules obtained if you worked with all the examples in a single node. We will carry out the comparison using the same databases described in the paper, which coincide with the first six of those initially described.

Table 17 shows the results of the incremental algorithm and CHI-BD, considering as aspects to study, the area under the ROC curve, the time required for learning and the number of rules. The star added in some of

Table 17 Comparison on prediction capacity between the incremental algorithm and CHI-BD. The incremental algorithm uses the following paramenters $size = 1000$, $k = 10$ and $t = 10$. Three aspects have been studied, capacity of prediction (AUC), runtime (time) and number of rules (rules). The star added in some of the databases indicates that the databases used for the results obtained with the incremental version are not exactly the same as those used in the study proposed in [9]

	Incremental			Parallel		
	AUC	time	rules	AUC	time	rules
cens	0.56	94.4	15.0	0.62	96	63598
covt*	0.92	322.7	74.3	0.74	75	8108
fars*	1.00	10.7	2.7	0.87	82	49584
kddc*	0.99	88.3	5.7	0,99	76	5734
poke*	0.54	131.9	86.9	0.59	100	54254
susy	0.72	794.9	75.7	0.62	103	9505

the databases indicates that the databases used for the results obtained with the incremental version are not exactly the same as those used in the study proposed in [9]. In all cases, they are reductions from a multiclass problem to a binary problem. In the ones used in this work, one class is considered against another (OVO), not considering the examples of the rest of classes. In the other case, it is considered a class against the rest of the classes (OVA). In both cases, the class considered as positive for all the databases used coincides, although obviously, the OVA case includes more examples of the negative class. Therefore, the data contained in the table should be considered as indicative. In any case, the database cens and susy do coincide exactly.

The following conclusions can be drawn from the results shown in Table 17:

- The number of rules obtained by the incremental version is much lower than the number obtained by the parallel approximation. Obviously, the rule model used in our proposal allows greater expressiveness and a single rule summarizes the information of many rules of the basic rule model. Especially noteworthy is the result of the fars database, where the incremental model with an average of 2.7 rules allows to classify all the examples of the positive class, while the parallel model gets 49584 rules. The complete set of fars with their 8 classes contains 100968 examples. This implies that each rule contains an average of 2 examples. The fact of having a high number of rules, not only implies a problem of loss of interpretability of the acquired knowledge but also entails the additional problem of causing high time costs during the inference process. In these cases, when there are a very high number of rules, it is possible that the basic inference algorithm on fuzzy rule sets is not very efficient and must be adapted, either in parallel (with a MapReduce philosophy, for example) or by developing a new more efficient inference algorithm, in cases where parallelization is not a possible option.
- Regarding the execution times, CHI-BD obtains very good results, being remarkable those obtained in covt and susy where it improves with clarity to those obtained by the incremental model. In the rest, the times are similar, except in fars, a unique database where the incremental clearly obtains the best result.
- As for the AUC results, the incremental model obtains the best result in half of the databases. As it happened in the comparison with the previous parallel model, with this one the parallel model outperforms the incremental model in cens and poke.

From the comparative study of our proposal with respect to the two parallel models, we can conclude that there are databases where it is advisable to use a parallel approach since it obtains better results both in accuracy and time of execution, in our study, the databases cens and poke. However, there are others where the incremental model is presented as a better alternative. The fars database is an example of this. We think that databases where the redundancy of the examples can be easily expressed, through the rule model presented in this paper, are favorable to the incremental model. In databases where this is not verified or is only partially verified, the parallel model based on MapReduce might be more appropriate.

7 Conclusion

Working with a massive amount of data is a problem for most learning algorithms. In this work, we have first analyzed the use of the paradigm of divide and conquer in order to solve big data problems. The MapReduce approach appears as a possibility within this model that makes use of parallel computing. Another approach within the previous paradigm is the sequential model, that it is clearly an alternative to the classical MapReduce approach, which is explored in this work. The sequential model makes use of a batch-incremental learning algorithm based on the use of a cognitive computational model of learning fuzzy rules. We have given a complete description of this model and have carried out an experimental study that has allowed us to adjust some of the parameters. A comparison has also been made with a parallel model, proving that the results are quite competitive. However, there are some points that will require further study to improve the model. In particular, the estimation of the threshold parameter is very important for the algorithm and requires ad-

ditional study, as well as the calculation of the weight of the rules, currently, an estimate is made on a set of episodes, but an estimate closer to the real weight would probably improve the final results.

Acknowledgements The authors would like to thank the research group Soft Computing and Intelligent Information Systems (SCi^{2}S) [http://sci2s.ugr.es] for their collaboration, permitting to us to access to the cluster for making the experimental part of this paper.

8 Compliance with Ethical Standards

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

Ethical Approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Arnaiz-González, Á., González-Rogel, A., Díez-Pastor, J.F., López-Nozal, C.: Mr-dis: democratic instance selection for big data by mapreduce. Progress in Artificial Intelligence **6**(3), 211–219 (2017). DOI 10.1007/s13748-017-0117-5. URL https://doi.org/10.1007/s13748-017-0117-5
2. Bache, K., Lichman, M.: Uci machine learning repository (2013)
3. Bechini, A., Marcelloni, F., Segatori, A.: A mapreduce solution for associative classification of big data. Information Sciences **332**, 33–55 (2016)
4. Chi, Z., Yan, H., Pham, T.: Fuzzy algorithms: with applications to image processing and pattern recognition, vol. 10. World Scientific (1996)
5. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun ACM 51(1), 107?113 (2008)
6. Dean, J., Ghemawat, S.: Mapreduce: a flexible data processing tool. Communications of the ACM **53**(1), 72–77 (2010)
7. Delgado, M., González, A.: An inductive learning procedure to identify fuzzy systems. Fuzzy Sets and Systems **55**, 121–132 (1993)
8. Dragoni, M., Rospocher, M.: Applied cognitive computing: challenges, approaches, and real-world experiences. Progress in Artificial Intelligence **7**(4), 249–250 (2018). DOI 10.1007/s13748-018-0166-4. URL https://doi.org/10.1007/s13748-018-0166-4
9. Elkano, M., Galar, M., Sanz, J., Bustince, H.: Chi-bd: A fuzzy rule-based classification system for big data classification problems. Fuzzy Sets and Systems **348**(1), 75–101 (2018)
10. Elkano, M., Galar, M., Sanz, J., Bustince, H.: Chi-pg: A fast prototype generation algorithm for big data classification problems. Neurocomputing **287**(26), 22–33 (2018)
11. Fernández, A., del Río, S., Bawakid, A., Herrera, F.: Fuzzy rule based classification systems for big data with mapreduce: granularity analysis. Advances in Data Analysis and Classification **11**(4), 711–730 (2017)
12. Fernández, A., del Río, S., López, V., Bawakid, A., del Jesus, M.J., Benítez, J.M., Herrera, F.: Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **4**(5), 380–409 (2014). DOI 10.1002/widm.1134
13. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Annals of eugenics **7**(2), 179–188 (1936)
14. Gámez, J.C., Garcia, D., González, A., Pérez, R.: On the use of an incremental approach to learn fuzzy classification rules for big data problems. In: 2016 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2016, Vancouver, BC, Canada, July 24-29, 2016, pp. 1413–1420 (2016)
15. García, D., Gámez, J.C., González, A., Pérez, R.: An interpretability improvement for fuzzy rule bases obtained by the iterative rule learning approach. International Journal of Approximate Reasoning **67**, 37–58 (2015)
16. García, D., Gámez, J.C., González, A., Pérez, R.: Using a sequential covering strategy for discovering fuzzy rules incrementally. In: Proceedings of the IEEE International Conference on Fuzzy Systems (2015)
17. Gepperth, A., Karaoguz, C.: A bio-inspired incremental learning architecture for applied perceptual problems. Cognitive Computation **8**(5), 924–934 (2016)
18. González, A., Pérez, R.: Completeness and consistency conditions for learning fuzzy rules. Fuzzy Sets and Systems **96**, 37–51 (1998)
19. González, A., Pérez, R.: SLAVE: a genetic learning system based on an iterative approach. IEEE T. Fuzzy Systems **7**(2), 176–191 (1999)
20. González, A., Pérez, R.: Selection of relevant features in a fuzzy genetic learning algorithm. IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society **31 3**, 417–25 (2001)
21. González, A., Pérez, R.: Improving the genetic algorithm of slave. Mathware and Soft Computing **16**, 59–70 (2009)
22. Hühn, J., Hüllermeier, E.: Furia: an algorithm for unordered fuzzy rule induction. Data Mining and Knowledge Discovery **19**(3), 293–319 (2009). DOI 10.1007/s10618-009-0131-8. URL https://doi.org/10.1007/s10618-009-0131-8
23. Ishibuchi, H., Yamamoto, T., Nakashima, T.: Hybridization of fuzzy gbml approaches for pattern classification problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **35**(2), 359–365 (2005). DOI 10.1109/TSMCB.2004.842257
24. Luna-Romera, J.M., García-Gutiérrez, J., Martínez-Ballesteros, M., Riquelme Santos, J.C.: An approach to validity indices for clustering techniques in big data. Progress in Artificial Intelligence **7**(2), 81–94 (2018). DOI 10.1007/s13748-017-0135-3. URL https://doi.org/10.1007/s13748-017-0135-3
25. Mahmud, M., Kaiser, M.S., Rahman, M.M., Rahman, M.A., Shabut, A., Al-Mamun, S., Hussain, A.: A brain-inspired trust management model to assure security in a cloud based iot framework for neuroscience applications. Cognitive Computation (2018). DOI 10.1007/s12559-018-9543-3. URL https://doi.org/10.1007/s12559-018-9543-3
26. Maloof, M.A., Michalski, R.S.: Incremental learning with partial instance memory. Artificial Intelligence **154**(1-2), 95–126 (2004)
27. Mansoori, E.G., Zolghadri, M.J., Katebi, S.D.: Sgerd: A steady-state genetic algorithm for extracting fuzzy classification rules from data. IEEE T. Fuzzy Systems **16**(4), 1061–1071 (2008)

28. Mao, W., Cai, Z., Yang, Y., Shi, X., Guan, X.: From big data to knowledge: A spatio-temporal approach to malware detection. Computers & Security **74**, 167 – 183 (2018)
29. Michalski, R.: A Theory and Methodology of Inductive Learning. Symbolic Computation. Springer Berlin Heidelberg (1983)
30. Mitchell, T.M.: Machine Learning, 1 edn. McGraw-Hill, Inc., New York, NY, USA (1997)
31. Oneto, L., Bisio, F., Cambria, E., Anguita, D.: Slt-based elm for big social data analysis. Cognitive Computation **9**, 259–274 (2016)
32. Oneto, L., Bisio, F., Cambria, E., Anguita, D.: Semi-supervised learning for affective common-sense reasoning. Cognitive Computation **9**(1), 18–42 (2017)
33. Park, S.Y., Pan, B.: Identifying the next non-stop flying market with a big data approach. Tourism Management **66**, 411 – 421 (2018)
34. Ramírez-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. Information Fusion **42**, 51 – 61 (2018). DOI https://doi.org/10.1016/j.inffus.2017.10.001
35. Read, J., Bifet, A., Pfahringer, B., Holmes, G.: Batch-incremental versus instance-incremental learning in dynamic and evolving data. In: International Symposium on Intelligent Data Analysis IDA 2012: Advances in Intelligent Data Analysis XI, pp. 313–323 (2012)
36. del Río, S., López, V., Benítez, J.M., Herrera, F.: A mapreduce approach to address big data classification problems based on the fusion of linguistic fuzzy rules. International Journal of Computational Intelligence Systems **8**(3), 422–437 (2015)
37. Romero-Zalíz, R., González, A., Pérez, R.: Incremental fuzzy learning algorithms in big data problems: A study on the size of learning subsets. In: Proceedings of the 2017 IEEE International Conference on Fuzzy Systems, pp. 1–6 (2017)
38. Segatori, A., Marcelloni, F., Pedrycz, W.: On distributed fuzzy decision trees for big data. IEEE Transactions on Fuzzy Systems **26**(1), 174–192 (2018). DOI 10.1109/TFUZZ.2016.2646746
39. Shi, Y., Eberhart, R., Chen, Y.: Implementation of evolutionary fuzzy systems. IEEE Transactions on Fuzzy Systems **7**(2), 109–119 (1999). DOI 10.1109/91.755393
40. Utgoff, P.: Incremental induction of decision trees. Machine Learning **4**(2), 161–186 (1989)
41. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning **23**(1), 69–101 (1996). DOI 10.1023/A:1018046501280. URL https://doi.org/10.1023/A:1018046501280
42. Zikopulos, P., Eaton, C., DeRoos, D., Deutsc, T., Lapis, G.: Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data. Mac Graw Hill (2012)