Efficient inference models for classification problems with a high number of fuzzy rules

Leonardo Jara, Rubén Ariza-Valderrama, Juan Fernández-Olivares, Antonio González, Raúl Pérez,

Dept. of Computer Science and Artificial Intelligence, University of Granada, 18017-Granada, Spain

Abstract

In data science there are problems that are not visible until you work with a sufficiently large number of data. This is the case, for example, with the design of the inference engine in fuzzy rule-based classification systems. The most common way to implement the winning rule inference method is to use sequential processing that reviews each of the rules in the rule set, to determine the best one and return the associated class. This implementation produces fast response times when the set of rules is small and is applied to a small set of examples. In this paper we explore new versions to implement this inference method, avoiding analyzing all the rules and focusing the analysis on the neighborhood of rules around the example. We study experimentally the conditions where each of them should be applied. Finally, we propose an implementation that combines all the studied versions offering good accuracy results and a significant reduction in the response time.

Keywords: big data, fuzzy rule-based classification systems, inference engine, fuzzy reasoning, soft computing

Email addresses: `ljara@correo.ugr.es` (Leonardo Jara), `rubenariza@correo.ugr.es` (Rubén Ariza-Valderrama), `faro@decsai.ugr.es` (Juan Fernández-Olivares), `a.gonzalez@decsai.ugr.es` (Antonio González), `fgr@decsai.ugr.es` (Raúl Pérez)

1. Introduction

One of the fundamental components of knowledge-based systems is the inference engine. In the design of fuzzy rule based systems this component is related with the use of fuzzy reasoning [18]. There is a large number of works that analyze, study alternatives and apply the fuzzy reasoning [4, 12, 13, 15, 16]. However, the inference for classification problems with fuzzy rules when there is a high number of fuzzy rules have not been studied enough.

The complexity of the inference of the winning rule for classification problems depends on three factors, the number of rules, the number of examples and the number of variables considered in the antecedent of the rules. Thus, when the complexity of this process is not large the time required to perform the inference is negligible and the process followed does not matter. However, when some of these factors are large, the process can be time consuming and it is convenient to analyze in detail how it can be done in an efficient way.

Although there may be a certain relationship between the three factors, for example, a large number of examples and variables may imply a large number of rules, we will focus our study on reducing the influence of the number of rules on the inference method, since both the number of variables and the number of examples depend on the problem.

The problem of inferring over a large set of rules has not been frequently raised in the literature to the best of our knowledge. In [2], an inference model combining the algorithm defined for Mamdani systems [11] and TSK systems [14] is proposed, but it is experimented on only a set of 256 rules, and it does not solve the problems that arise in classification problems when large number of rules are needed. The problem has not really become apparent until massive datasets considered as big data have been used [7, 8].

One way to reduce the influence of the factor number of rules in the process is to avoid reviewing all the rules by focusing the search for the best or winning rule on the rules in a neighborhood of the example. The idea of neighborhood, which we will define later, corresponds to those rules that are likely to have an influence on the example, and which can be constructed in a simple way from the example.

The main problem of the neighborhood of an example is that it could also contain a very large set of rules, so it is necessary to define certain heuristics that allow the search for the best rule to be performed as efficiently as possible and thus reduce the response time of the inference method.

The basic idea of using a neighborhood exploration method to propose a fuzzy rule inference algorithm was first proposed in [10]. In this paper, starting from the basic algorithm, we propose some improvements aimed at reducing the response time of inference in complex problems. In this way, the main contribution is the definition of a hybrid fuzzy rule inference method that combines the different proposals made in this work and that is obtained from the experimental study.

The process we have followed to carry out the experimental study is as follows. We select a set of datasets with different levels of complexity, from very simple datasets to datasets commonly used in big data problems. On these datasets we apply the Chi algorithm [3] to obtain a set of rules for each dataset. We use these rule sets to test the effectiveness of each proposed inference algorithm.

The organization of the paper is as follows. In the next section we describe in detail the different steps of the winning rule inference method for classification, and from that description we propose a standard algorithm to perform the inference that can be interpreted as a linear search for the best rule in the set of rules where the inference is performed, from this proposal and an improvement of it, we propose in the next section a change of model in which the search is not done directly in the set of inference rules but in the neighborhood of the rule, and different improvements of the basic proposal are proposed. In Section 5, an extensive experimental study is carried out and the results are discussed.

2. Problem definition

With the idea of analyzing the possible problems posed by the inference with fuzzy rules, first, we describe in detail this process of inference. To do this, we describe the fuzzy rule model with weight that we use and step by step the operations that are necessary to perform.

Given n linguistic variables X_i, $i = 1 \dots n$, where each variable has a fuzzy domain D_i composed of $n_i = |D_i|$ linguistic labels

$$D_i = \{A^i_1, A^i_2, \dots, A^i_{n_i}\}.$$

Given these variables, their associated domains and a consequent variable Y representing the class to be learned, the set of all weighted fuzzy rules that could be defined is named R^G, and each rule has the form

$$R : \textbf{IF } X_1 \text{ is } A^1_{j_1} \text{ and } X_2 \text{ is } A^2_{j_2} \text{ and } \dots \text{and } X_n \text{ is } A^n_{j_n}$$

THEN Y is B with **weight** $w(R)$

where $A^i_{j_i}$ is a particular value of the domain D_i, Y is a discrete variable with an associated domain D_Y representing the consequent of the rule, and B a particular value of this domain, and $w(R)$ is a measure of the weight associated with the rule.

As we have said, R^G is composed by all the possible rules that can be generated with the n antecedent variables, with the consequent variable and with the different associated weights.

Now, we consider a particular set of fuzzy rules for which we want to define the inference process, this set could be obtained, for example, as the output of a rule learning algorithm as [3]. We will note this set as

$$R^I = \{R_1, R_2, \dots, R_m\}$$

and obviously

$$R^I \subseteq R^G.$$

From this set of rules R^I and a given example e=$(e_1, e_2, \dots, e_n)$, the fuzzy reasoning allows us to obtain the class associated with this example given the set of rules. The procedure for obtaining the class associated with the example is very simple using the well-known method of the winning rule and it has the followings steps:

- First we calculate the adaptation between the example e and the antecedent of a rule $R \in R^I$ by applying a t-norm T

  $$A(R, e) = T(\mu_{A^1_{j_1}}(e_1), \dots, \mu_{A^n_{j_n}}(e_n)),$$

  where $\mu_{A^k_{j_k}}$ is the membership function of the fuzzy sets $A^k_{j_k}$.

- Next we incorporate the weight of the rule into this adaptation using a certain operator Op

  $$h(A(R, e), w(R)) = Op(A(R, e), w(R)).$$

- Finally we select the rule of R^I that maximizes this value, that is to say,

  $$max_{R \in R^I} h(A(R, e), w(R)) \qquad (1)$$

  and the class associated with that rule is assigned to the example.

Frequently, the minimum or the product are used as T-norms, and the product is used as the operator Op.

In this way the inference model of the winning rule used in classification can be solved through the optimization process described by equation 1.

Thus, given an example e=$(e_1, e_2, \ldots, e_n)$ the calculation of the class associated with the example given the rule set R^I can be solved by an algorithm that checks the adaptation of the example to each rule sequentially, calculates the weight and keeps the maximum value. This process can be understood, for each example, as a search for the best rule using the Equation 1 as optimization criteria. Algorithm 1 describes the different steps for a particular example e and the set of rules R^I.

Algorithm 1 Linear search

```
1: function STANDARD_INFERENCE($e$, $R^I$)
2:     BestCurrentMatching = 0
3:     for $i = 1$ to $m$ do
4:         CurrrentMatching = 1
5:         for $j = 1$ to $n$ do
6:             CurrentMatching = T(CurrentMatching, $\mu_{A^j_{i_j}}(e_j)$)
7:         end for
8:         CurrentMatching = Op(CurrentMatching, $w(R_i)$)
9:         if CurrentMatching > BestCurrentMatching then
10:            BestCurrentMatching = CurrentMatching
11:            Class = ClassOfRule(i)
12:        end if
13:    end for
14:    return Class
15: end function
```

This algorithm has a time complexity O(mxn), being m the number of rules where the inference is made and n the number of variables of each rule, and the whole process is repeated for each example. Obviously, the inference is very dependent on the number of rules that our system has. In simple problems really the inference can be applied without any inconvenience, however, there are problems in which the number of rules is very high.

Thus, this inference model has been used on classifiers and problems with massive data [7, 8]. These methods using the MapReduce model [5, 6], and the Chi algorithm [3] learn a fuzzy rules set in a relatively fast way but obtaining a very high number of rules. When it is necessary to perform the inference with massive datasets and on a problem with a very high number of rules, the problem is that the inference can be very slow, or even impossible to do in an acceptable time [10].

A simple improvement of the previous algorithm is to change the linear search of the best rule to a linear search with pruning (see Algorithm 2). The idea is very simple, during the search we keep the value of the best adaptation between the example and the rule calculated so far, and if the current partial calculation of the adaptation of a component of the example with part of the rule is already worse than the best adaptation we have calculated, obviously since we use a t-norm, that rule cannot compete with the best rule anymore and we can abandon the calculation of the rest of the adaptations.

Algorithm 2 Linear search with pruning

```
function STANDARD_INFERENCE_PRUNNED($e$, $R^I$)
    BestCurrentMatching = 0
    for $i = 1$ to $m$ do
        CurrentMatching = $w(R_i)$
        while $j \leq n$ and CurrentMatching > BestCurrentMatching do
            CurrentMatching = T(CurrentMatching, $\mu_{A^j_{i_j}}(e_j)$)
            $j = j + 1$
        end while
        if CurrentMatching > BestCurrentMatching then
            BestCurrentMatching = CurrentMatching
            Class = ClassOfRule(i)
        end if
    end for
    return Class
end function
```

This algorithm is an improved version of the algorithm proposed in [10] in which the own weight is used to perform pruning. Obviously the result of Algorithm 2 as opposed to Algorithm 1 depends on the order in which the rules are explored, however, in any case it is an improvement over the basic version since it reduces the number of calculations needed to obtain the class of the winning rule although in the worst case it is still O(mxn).

3. Exploring the rules in the neighborhood of the example

The idea of this section is to explore alternative ways to make the inference that can reduce the calculation time used by Algorithms 1 and 2. Fundamentally these improvements can be interpreted as alternative ways to solve the optimization problem described by the Equation 1. The idea of pruning raised in Algorithm 2 is an interesting idea that we maintain in the successive improvements that we propose. However, to make a qualitative leap in improving the inference process when we have a very high number of rules, it is necessary to change the search process radically. The central idea is to change the search space. This space is no longer R^I, it is R^G.

Although it seems surprising searching in a larger space may reduce the response time, the reason is that the search will be focused on a subset of rules in the neighborhood of the example. Moreover, the search starts at a specific point in this space, that is, the rule that best fits the example.

To this end, given an example e, the set of rules that have strictly positive adaptation with the example is defined by

$$N(e) = \{R \in R^G | h(A(R), e) > 0\}.$$

$N(e)$ contains all the rules that cover the example e to a positive degree, so it contains the rules that could be triggered by that example, or in other words the set $R^G - N(e)$ corresponds to the rules that could not be triggered by the example in any way. We call $N(e)$ the neighborhood of the example. The elements of $N(e)$ do not have to be rules from the R^I set and therefore do not have to be rules that participate in the example's inference, i.e,

$$N(e) \not\subseteq R^I.$$

However, the winning rule must necessarily belong to the set $N(e)$ since it needs to have a positive adaptation with the example to be the solution to the optimization process defined by the Equation 1. Therefore the winning rule is found in the set

$$N(e) \cap R^I.$$

A relevant rule of the set $N(e)$ is the rule that contains the best adaptation of each component of the example with each antecedent variable of the rule. This rule that we call the central rule of $N(e)$ and we notice for $C(e)$ does not have to belong to the set R^I.

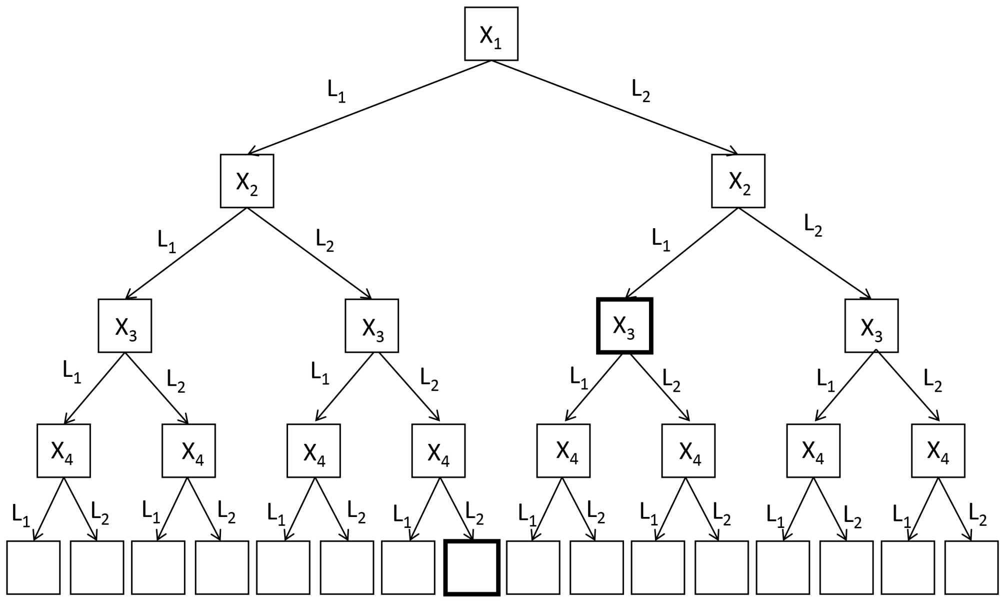

Figure 1: Tree representing the set $N(e)$ where X_i are the antecedent variables and L_1 is the label with the best adaptation to the example and L_2 is the label with the worst adaptation to the example, for problems in where for each real value of the domain there are only two labels with positive adaptation

Our idea is to define a search process on the set $N(e)$, always taking into account that if the explored rule does not belong to the set R^I it will not be considered, and taking as a starting point the central rule of the set $N(e)$.

Thus, with the idea of exploring the set $N(e)$ to select the winning rule, we represent this set by a tree. This tree has in each node a variable, and in each arc a possible value of this variable, among those values (fuzzy labels) which have positive adaptation with the example, representing the assignment of that value to the node variable. The leaf nodes in this tree represent the antecedents of all possible rules of $N(e)$.

Figure 1 represents an example in which we have four antecedent variables X_1, X_2, X_3 and X_4, where for simplicity in the representation we suppose that for each real value of the domain there are only two labels with positive adaptation (see Figure 2) except at the end points, as it happens for example when we take domains where the labels are uniformly distributed. We will notice for L_1 the label (among the fuzzy labels of the domain of

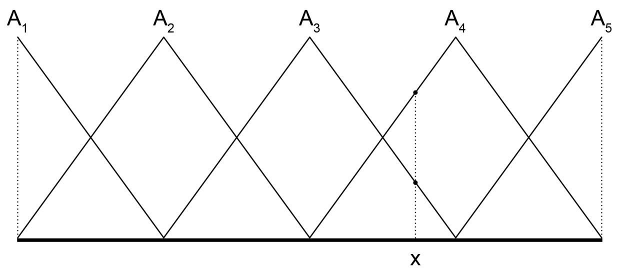

Figure 2: A domain for which each real value has positive adaptation with exactly two fuzzy labels except at the end points

the corresponding variable) with the best adaptation to the example and for L_2 the label with the worst adaptation to the example. When a value has adaptation with more than two labels, the idea is the same and we use as many literals L_i as necessary, and L_1 would represent the label with the best adaptation, L_2 the second best, and so on.

Figure 1 shows the tree and each internal node represents an antecedent of a partially assigned rule. For example, the inner node of the tree with a bold square represents the assignment to the antecedent variables of the following values

$$(X_1 = L_2, X_2 = L_1, X_3 = Unassigned, X_4 = Unassigned).$$

and therefore a rule with the following antecedent

IF X_1 is L_2 and X_2 is L_1 **THEN** ...

However, each leaf node represents a complete assignment of values to the four variables, as for example the tree leaf node with a bold square in the Figure 1 represents the assignment to the antecedent variables of the following values

$$(X_1 = L_1, X_2 = L_2, X_3 = L_2, X_4 = L_2).$$

and therefore a rule with the following antecedent

IF X_1 is L_1 and X_2 is L_2 and X_3 is L_2 and X_4 is L_2 **THEN** ...

This tree is a complete representation of all the antecedents of the rules of $N(e)$ through the tree's leaf node. We use this representation to describe the different search algorithms that represent the inference process. The description that we have made of the tree, associating each level of the tree to a new assignment of a value to a variable facilitates during the search process the inclusion of the pruning defined in the Algorithm 2.

The first proposal to calculate the winning rule based on the exploration of the set $N(e)$ that we propose is described by Algorithm 3 and it uses a backtracking search algorithm on the tree associated to $N(e)$ with the following particularities:

- In each node of the search we have a partial assignment of values to variables (the assignment is complete in the leaf nodes) and we use this partial assignment to calculate the partial adaptation between the example and that assignment, when that value is worse than the value of the best complete assignment obtained so far the node is pruned, i.e., the search process stops at that node and causes a backward movement. As we have commented before, all the rules that could be obtained from that node are worse than another one we have found.

- When during the search we reach a leaf node then we have a complete assignment, in that moment we check if there is a rule in R^I with that antecedent, if not found in this set the leaf node is ignored, otherwise we consider it in the calculation of the winning rule. We have used a hash table for the implementation of the rules of the set R^I, so checking if the antecedent of the rule in a leaf node belongs or not to R^I is done in a very efficient way.

Due to the way the previous tree was created, the backtracking search (Algorithm 3) starts by exploring the central rule of the example, then goes through the rest of the neighborhood rules, discarding the rules that do not belong to the R^I inference rule set or those that we are sure will have a worse adaptation than the best one found so far.

The last parameter $MaxRule$ establishes the maximum number of rules that this inference process will check in the neighborhood of the central rule. When $MaxRules{=}\infty$ is taken, no limitation on the number of rules is considered.

Algorithm 3 Backtracking search with pruning in the neighborhood of the example

```
function NEIGHBORHOOD_INFERENCE($e$, $R^I$, $MaxRules$)
    vector $\sigma[n]$
    for i=1 to $n$ do
        $\sigma[i] = i$
    end for
    return BACKTRACK (1, $\emptyset$, 1.0, 0.0, $\emptyset$, $MaxRules$)
end function

function BACKTRACK(i, CurrentRule, CurrentAdapt, &BestAdapt, &CurrentClass, &MaxRules)
    if (MaxRules $\leq 0$ then
    else if (CurrentAdapt $\leq$ BestAdapt) then
        return
    else if (i == n) then
        $MaxRules = MaxRules - 1$
        if (CurrentRule $\in R^I$) then
            $w$ = WeightOf(CurrentRule)
            CurrentAdapt = Op(CurrentAdapt, $w$)
            if (CurrentAdapt > BestAdapt) then
                BestAdapt = CurrentAdapt
                CurrentClass = ClassOf(CurrentRule)
            end if
        end if
        return CurrentClass
    end if
    for j=1 to $|D_i|$ do
        if ($\mu_{A_j^i}(e_i) > 0$) then Push_back($v, A_j^i$)
        end if
    end for
    Sort($v$)
    for k=1 to $|v|$ do
        CurrentClass = BACKTRACK ($\sigma[i+1]$, Append(CurrentRule, $X_i$ is $A_k^i$), $T$(CurrentAdapt,$A_k^i(e_i)$), BestAdapt, CurrentClass, MaxRules)
    end for
    return CurrentClass
end function
```

In the implementation of Algorithm 3 two functions have been used: a main non recursive function (called NEIGHBORHOOD_INFERENCE) that receives the previous parameters and uses an auxiliary recursive function (called BACKTRACK) that really implement the inference. With the idea that the recursive function can be used in the description of the following algorithms, we use a σ function that associates each index i with itself. This function is really important in the next algorithm.

The parameters used in the BackTrack function are:

- i is the index of the variable being evaluated at any given time, for $1 \leq i \leq n$. Since σ keeps the index unchanged, the variables are processed in the default order, that is, first the first variable, second the second variable, and so on. In the following algorithm we will change this order using the function σ.
- The variable *CurrentRule* stores the antecedent of the rule to be evaluated, initially is an empty string, and the antecedent is obtained by adding assignments of values to variables following the downward movement through the tree associated to $N(e)$. When i=n the assignment has been finished and we have a complete antecedent of a rule.
- The variable *CurrentAdapt* is the degree of adaptation of the antecedent described in *CurrentRule* from the first to the i-th variable.
- The variable *BestAdapt* is the best adaptation of a complete antecedent among all that has been evaluated previously. Initially its value is zero.
- The variable *CurrentClass* is used to store the consequent or class of the rule that provided the value to *BestAdapt*.
- The variable *MaxRules* is the maximum numbers of rules in the neighborhood of the example to be explored.

Some of the arguments of the BackTrack function have a & symbol indicating that those arguments are passed by reference, while those that do not have it are passed by value. This distinction between both types of argument passing is relevant for the behaviour of the recursive process.

Finally, it is also used in the algorithm the example that is being inferred e, being e_i the i-th component of this example, the domains of the antecedent variables, being D_i the domain of the i-th variable and R^I the the set of rules used in the inference process.

The description of the process is as follows. There are three stop conditions,

- the first one uses the parameter $MaxRules$ and will be explained later,
- the second one applies when the adaptation obtained up to the variable i, that is $CurrentAdapt$, does not exceed the adaptation of the best existing rule in the rule base already found, that is, the value of $BestAdapt$. In this case, a pruning is done and it goes to look for another option.
- The last stop condition occurs when all the antecedent variables ($i = n$) are instantiated. In this case, it is checked if the rule with the current antecedent is in the rule set R^I. If it is, the weight of the rule is taken and the function h is calculated from the Equation 1. If the value of the function h is bigger than the one of $BestAdapt$, then $BestAdapt$ is changed with this new adaptation and the class is the class of this rule. Otherwise, this rule is not considered as an alternative to give output to the inference and is ignored.

When neither of the three stop conditions is true then all the labels of the variable X_i that have adaptation with the component i of the example are taken and ordered from greater to lesser in a vector v. For each one of the labels with non-zero adaptation of the i-th variable a recursive call is made (step 30) using now the next variable (X_{i+1}), and the following parameters:

- a new $CurrentRule$ in which to the antecedent defined by previous $CurrentRule$ a new antecedent component, such that 'X_i is L_k', is added, being L_k the k-th label of the reordered D_i domain,
- a new $CurrentAdapt$ variable defined as the t-norm between the one we already had and the adaptation between the component of the example with the k-th label,
- and finally the $BestAdapt$ and $CurrentClass$ variables.

The ranking described in lines 24 to 28 of the Algorithm 3 consists of detecting those fuzzy labels with positive adaptation with the component of the example, and once detected, sort them in a decreasing way following the structure of the tree described above.

Now it is time to explain the reason for the first stop criterion. This algorithm explores the rules in the neighborhood of the example, and as we have already said, these rules do not have to belong to the R^I set. This search is more efficient than exploring the R^I set directly in many cases, but when the example is not covered by any rule from that set the process becomes inefficient as it requires exploring all the rules in the example's neighborhood before realizing that no rule is applicable. To avoid this problem we include the parameter $MaxRules$ that limits the number of rules the algorithm explores. When the $MaxRules$ parameter is taken as infinite, the output of this algorithm must be exactly the same as the two previous algorithms, except for possible ties in the value of the function h. This may be due to the fact that in the two previous models there is an order between the rules, and that order allows solving ties. In the new algorithm we do not have an order of the rules in the previous sense, so it can be solved in different ways. For smaller values of the parameter the algorithm becomes an approximate method for making the inference.

The complexity of this algorithm does not depend now on the number of rules in the set R^I, but it depends on the number of antecedent variables n, in fact the algorithm must explore all the rules of the neighborhood of the example (when $MaxRules = \infty$), therefore it will have a complexity $O(2^n)$ (If we assume that every real value in the range of each variable is covered by exactly two linguistic labels, as is the case, for example, in domains where the labels are uniformly distributed). Although it does not actually involve all the antecedent variables, it only involves the continuous variables with an associated fuzzy domain. The possible improvement of this algorithm over the two previous ones depends on the concrete problem, and in particular on the number of rules of the set R^I and the number of antecedent variables n as we will study in the experimental section.

Algorithm 3 is the reference model for all the neighborhood search processes we have defined in this paper. A first improvement of this algorithm is related to the use of a heuristic function that allows us to explore the variables X_i is an order other than the default. The order in which the variables are explored obviously influences the pruning process, so once we have calculated the partial adaptation between the component in the example and the different fuzzy labels of the variable, if the adaptation between the best fitting label and the adaptation between the worst fitting label is very large, that means that the use of that variable is more interesting than if the difference between the adaptations is smaller. The reason is that the greater the difference, the greater the value of the best adaptation is and therefore we begin by exploring those variables with the labels that have the best adaptation. The heuristic that describes the previous idea is the following:

$$\Delta(i) = max_j(\mu_{A^i_j}(e_i)) - min_j(\mu_{A^i_j}(e_i)).$$

Algorithm 4 is an adapted version of Algorithm 3 in which we explore the variables not in a default order but in a decreasing order based on the heuristic Δ. The reordering of the variables makes use of the σ function that records the most convenient order for their exploration following the idea of heuristics.

Algorithm 4 Heuristic backtracking search with pruning in the neighborhood of the example

```
1: function HEURISTIC_NEIGHBORHOOD_INFERENCE($e$, $R^I$, $MaxRules$)
2:     vector $\sigma[n]$, $\Delta[n]$
3:     for i=1 to $n$ do
4:         $\sigma[i] = i$
5:         $\Delta(i) = max_j(\mu_{A^i_j}(e_i)) - min_j(\mu_{A^i_j}(e_i))$
6:     end for
7:     Sort($\sigma$) using $\Delta$ heuristic
8:     return BACKTRACK ($\sigma[1]$, $\emptyset$, 1.0, 0.0, $\emptyset$, $MaxRules$)
9: end function
```

The last of the algorithms we propose based on the exploration of the neighborhood of the example is based Algorithm 4, but adds a new heuristic criterion. This new algorithm will make use of a pruning process and the Δ heuristic described above, and will of course explore the neighborhood of the example, but now we introduce a new parameter that limits the neighborhood that can be explored. Since Algorithm 4 starts exploring the central rule of the example and from that rule it explores rules that we have heuristically ordered from best to worst, now instead of exploring all the neighborhood rules, the search is limited to a neighborhood near the central rule, and that proximity will be determined by a distance parameter.

Thus we define a measure of distance between rules. Given two rules from the set R^G

$$R_1 : \textbf{IF } X_1 \text{ is } A^1_{r_1} \text{ and } X_2 \text{ is } A^2_{r_2} \text{ and } \ldots \text{and } X_n \text{ is } A^n_{r_n}$$

$$\textbf{THEN } Y \text{ is } B_1 \text{ with } \textbf{weight } w(R_1)$$

$$R_2 : \textbf{IF } X_1 \text{ is } A^1_{s_1} \text{ and } X_2 \text{ is } A^2_{s_2} \text{ and } \ldots \text{and } X_n \text{ is } A^n_{s_n}$$
$$\textbf{THEN } Y \text{ is } B_2 \text{ with } \textbf{weight } w(R_2)$$

We define the distance between the two rules as follows:

$$d(R_1, R_2) = \sum_{i=1}^{n} \delta_i(R_1, R_2)$$

where

$$\delta_i(R_1, R_2) = \begin{cases} 0 & \text{if} \quad A^i_{r_i} = A^i_{s_i} \\ \\ 1 & \quad \text{otherwise.} \end{cases}$$

In this way the distance between two rules is defined as the number of antecedent variables that have different assignment in each rule. The definition of near neighborhood makes use of the distance to the central rule as follows:

$$N^d(e) = \{R \in N(e) | d(R, C(e) \leq d\}.$$

Special cases of this definition would be:

$$N^0(e) = \{C(E)\}$$

and

$$N^\infty(e) = N(e).$$

That is, when the distance is zero we only have in the set the central rule, and when the distance is infinite the set is all $N(e)$.

Well, the basic idea of the Algorithm 5 is to keep the elements that have been added in the Algorithm 3 and 4, but to make use of a parameter d that limits the search to the space $N^d(e)$ with the idea of making more efficient the process of search and making use of the heuristic criterion that the proximity to the central rule has direct influence in the detection of the winning rule.

In the description of this algorithm we changed the structure of the recursive function BackTrack to that new BackTrack2 function to include the parameter distance and use it conveniently.

Algorithm 5 Heuristic backtracking search with pruning in the nearby neighborhood of the example defined by a distance measure

```
1: function HEURISTIC_NEARBY-NEIGHBORHOOD_INFERENCE($e$, $R^I$, $d$)
2:     vector $\sigma[n]$, $\Delta[n]$
3:     for i=1 to $n$ do
4:         $\sigma[i] = i$
5:         $\Delta(i) = max_j(\mu_{A^i_j}(e_i)) - min_j(\mu_{A^i_j}(e_i))$
6:     end for
7:     Sort($\sigma$) using $\Delta$ heuristic
8:     return BACKTRACK2($\sigma[1]$, $\emptyset$, 1.0, 0.0, $\emptyset$, $d$)
9: end function

10: function BACKTRACK2(i, CurrentRule, CurrentAdapt, &BestAdapt, &Current-
    Class, Distance)
11:     if (Distance $<$ 0 then
12:         return
13:     else if (CurrentAdapt $\leq$ BestAdapt) then
14:         return
15:     else if (i == n) then
16:         $MaxRules = MaxRules - 1$
17:         if (CurrentRule $\in R^I$) then
18:             $w$ = WeightOf(CurrentRule)
19:             CurrentAdapt = Op(CurrentAdapt, $w$)
20:             if (CurrentAdapt $>$ BestAdapt) then
21:                 BestAdapt = CurrentAdapt
22:                 CurrentClass = ClassOf(CurrentRule)
23:             end if
24:         end if
25:         return CurrentClass
26:     end if
27:     for j=1 to $|D_i|$ do
28:         if ($\mu_{A^i_j}(e_i) > 0$) then Push_back($v, A^i_j$)
29:         end if
30:     end for
31:     Sort($v$)
32:     for k=1 to $|v|$ do
33:         CurrentClass = BACKTRACK2($\sigma[i+1]$, Append(CurrentRule, $X_i$ is $A^i_k$),
    $T$(CurrentAdapt, $A^i_k(e_i)$, BestAdapt, CurrentClass, Distance-(k-1) )
34:     end for
35:     return CurrentClass
36: end function
```

4. Discussions of experimental results

Once we have described the different methods to perform inference on fuzzy rules in classification problems, in this section we will analyze the behavior of these methods. This study is conducted in two steps, first we analyze the behavior of the algorithms that we can consider exact, i.e., they return exactly the class of the winning rule. This first study involves Algorithms 1, 2, 3 and 4, although Algorithm 1 finally is not considered, since its results are the same as those produced by Algorithm 2 in terms of accuracy, and the execution times are significantly worse than those obtained by this second algorithm. The second step involves Algorithm 5, since it is not an exact algorithm, it is an approximate algorithm in which the output of the algorithm may not be that of the winning rule. Once these two steps are completed we will propose a hybrid approach that combines the previous algorithms and obtains the best results.

Moreover, in the study we will distinguish between the algorithm based on a sequential search over the set of rules, i.e. Algorithm 2, which we will refer to as sequential approach (Algorithm 1 is also included in this approach), and the algorithms based on a search in the neighborhood of the central rule of the example (Algorithm 3 and 4) which we will refer to as recursive approach.

Both algorithms of the recursive approach make use of the *MaxRules* parameter in the function BackTrack, which determines the maximum number of rules that are explored in the neighborhood of the central rule of the example before abandoning the search. This parameter will take the value 1024 throughout the experimentation, and we will justify this value later.

In this experimentation we have used the datasets shown in Table 1. This table shows the 50 classification problems that we have considered, all of them are extracted from the UCI machine learning repository [1]. All the selected datasets are well known and are frequently used. These datasets also represent a wide spectrum of different situations to be considered in machine learning such as binary problems, multiclass problems, multiclass problems with a large number of classes, problems with few or many variables, problems for which a small or large number of rules are obtained, problems with few or many examples, etc.

In all the above datasets we will use to discretize the continuous domains of the variables a set of uniformly distributed fuzzy sets with cutoff at 0.5.

Dataset	Var	CVar	Ex	Dataset	Var	CVar	Ex
abalone	8	7	4174	magic	10	10	19020
adult	14	6	45222	mammogr	5	1	830
australian	14	7	690	movement	90	90	360
automobile	25	15	159	newthyroid	5	5	215
balance	4	4	625	page-blocks	10	10	5472
banana	2	2	5300	penbased	16	16	10992
bands	19	19	365	phoneme	5	5	5404
bupa	6	6	345	pima	8	8	768
census	41	9	142521	ring	20	20	7400
cleveland	13	13	297	saheart	9	8	462
coil2000	85	2	9822	satimage	50	36	6435
connect-4	42	0	67557	segment	19	19	2310
covtype*	54	10	495141	shuttle	9	9	57999
crx	15	6	653	sonar	60	60	208
fars*	29	5	62123	spambase	57	57	4597
flare	11	0	1066	spectfheart	44	44	267
german	20	3	1000	susy	18	18	5000000
heart	13	5	270	texture	40	40	5500
hepmass	28	28	10500000	thyroid	21	6	7200
higgs	28	28	11000000	twonorm	20	20	7400
ionosphere	35	32	351	vehicle	18	18	846
iris	4	4	150	vowel	13	11	990
kddcup*	41	27	4856150	wine	13	13	178
led7digit	7	7	500	wineq-red	11	11	1599
letter	16	16	20000	wineq-white	11	11	4898

Table 1: Datasets used in this experimental study. The columns represent the following information: *Dataset* is the name of the dataset, *Var* indicates the number of predictive attributes, *CVar* indicates the number of predictive variables with a continuous domain and *Ex* the total number of examples. The datasets' name *mammogr*, *movement*, *wineq-red*, and *wineq-white* correspond to datasets mammographic, movement_libras, winequality-red and winequality-white respectively. The dataset with a $\star$ (*covtype*, *fars* and *kddcup*) correspond with versions of the original dataset transformed for binary classification that have been used on big data paper as such [9].

Throughout this experimental study we will use five triangular fuzzy sets in this discretization in the different datasets considered (domains similar to the one described in Figure 2).

The order of complexity of a single inference of the algorithms of the sequential approach depends on the number of rules and the number of variables involved in those rules. The order of complexity of the algorithms of the recursive approach depends exponentially on the number of continuous variables. In particular the order is $O(2^C)$, where C is the number of continuous variables of the particular problem, since given the discretization model we have used a real value has an adaptation with at most two fuzzy sets of its domain.

All experiments have been performed on a computer with 8 IntelCore i7-6700 processors and 15.6 Gb memory running on Ubuntu 20.04.2 64-bit operating system [1].

To obtain the set of rules with which to perform the inference in the different datasets we will use the so-called Chi algorithm [3], which is no more than the version of the Wang and Mendel algorithm [17] but applied to classification problems to obtain the sets of rules for each of the selected datasets. The choice of this algorithm is justified for the following reasons:

(a) It is a simple learning algorithm that can run very fast and in fact has been used as an algorithm to tackle big data problems where the number of examples in the problem is very high [7].

(b) It generates fuzzy rules that take into account all the attributes of the problem, and this fact is important to illustrate the influence of the number of attributes on the time consuming inference.

(c) It can generate a very large number of rules for some problems. This is very important to study the efficiency of the different approaches to perform inference.

Table 2 reports the results obtained by the Chi algorithm using a 10-fold cross-validation. The average number of rules obtained, the accuracy over the training set and the average time consumed by the learning algorithm

[1]The algorithms described here have been implemented in C++ and can be accessed through the web page https://github.com/Raul-PerezR/Inference-Recursive-Classification-FuzzyRules

have been considered for each of the datasets as relevant information for this study.

Dataset	Rules	Train	Time(s)	Dataset	Rules	Train	Time(s)
abalone	207.3	31.15	0.030	magic	1912.4	84.01	0.139
adult	19796.4	89.99	0.502	mammog	160.3	87.35	0.004
australian	500.9	96.81	0.007	movement	294.6	100	0.030
automobile	124.3	100	0.003	newthyroid	47.2	95.35	0.001
balance	562.5	100	0.003	page-blocks	163.8	93.82	0.041
banana	19.6	77.08	0.018	penbased	7739	99.87	0.122
bands	326.1	100	0.005	phoneme	228.2	82.38	0.028
bupa	120.8	76.52	0.002	pima	426.6	90.49	0.007
census	80968.1	99.26	4.509	ring	5060.3	99.34	0.108
cleveland	258.4	98.99	0.003	saheart	354	96.75	0.004
coil2000	7532.2	98.74	0.709	satimage	2149.3	78.21	0.164
connect-4	60801.3	100	1.647	segment	888.4	95.76	0.029
covtype*	23087.9	84.91	16.231	shuttle	73.8	91.59	0.413
crx	522.7	98.47	0.007	sonar	187.2	100	0.010
fars*	39933.7	100	1.560	spambase	1379.8	86.67	0.173
flare	269.8	85.27	0.007	spectfheart	240.3	100	0.008
german	898.4	100	0.012	susy	274477	70.39	60.012
heart	235.6	100	0.003	texture	3697.8	97.69	0.167
hepmass	9404420	99.9	203.699	thyroid	943.9	94.25	0.080
higgs	8706010	95.54	215.924	twonorm	6654.2	99.99	0.103
ionosphere	273.4	100	0.009	vehicle	712.7	98.58	0.011
iris	41.8	95.33	0.001	vowel	649.7	98.69	0.010
kddcup*	2868	99.95	127.538	wine	159.4	100	0.002
led7digit	82.1	79.6	0.004	wineq-red	771	83.99	0.016
letter	7585.2	88.15	0.237	wineq-white	1086.8	66.35	0.040

Table 2: Results obtained by the Chi algorithm using a 10-fold cross-validation. Column *Rules* shows the number of rules, *Train* the accuracy on the training set and *Time(s)* the number of seconds consumed by the learning algorithm.

The data shown in the *Train* column are obtained directly from the learning algorithm and we are not using an inference algorithm. The Chi algorithm places a grid over the search space and counts the examples that fall into each grid. When all the examples have been placed in the corresponding grid, it looks over each grid and associates as a class, the majority class among the examples that are in that grid. In this way it is known which examples are well classified (those that are of the majority class), and those that will be misclassified.

Some conclusions we draw from the experimentation collected by Table

2 are:

- The Chi algorithm is a learning method that is not very time consuming. The time is proportional to the number of examples in the problem, for this reason, problems that have more examples take longer, although a problem like *higgs* that has 11 million examples takes on average less than 3 minutes to learn.

- The Chi algorithm can generate a very large number of rules. The most outstanding examples where we can observe this high number of rules are *hepmass* which has more than 9 million rules and *higgs* with more than 8 and a half million.

- If we calculate the ratio of examples per rule, we can see that most of the datasets have a value of less than 2 (specifically, 27 of the 50 datasets). This fact indicates that for these problems the rules are supported by very few examples. Only 8 of the 50 have a value greater than 15 for this indicator.

As already discussed in the previous sections where the inference algorithms have been presented, the two different approaches put the computational effort in different areas of the search space, while the algorithms of the sequential approach search over the set of rules and therefore, their order of complexity is associated with the number of variables of the problem and the number of rules, the versions of the recursive approach search in the neighborhood of the central rule to the example, and therefore their order of complexity is exponential to the number of continuous variables of the problem. Therefore, to study the efficiency of these approximations it seems interesting to make a division by cases taking into account the number of rules and the number of continuous variables involved in the problem.

Figure 3 shows the division made for this experimentation. The 50 datasets have been divided into 4 groups:

- Group A: It has 14 datasets and is composed of those with a number less than or equal to 10 continuous variables and a number of rules less than or equal to 500.

- Group B: It has 12 datasets and is composed of those that have a number less than or equal to 10 continuous variables and a number of rules greater than 500.

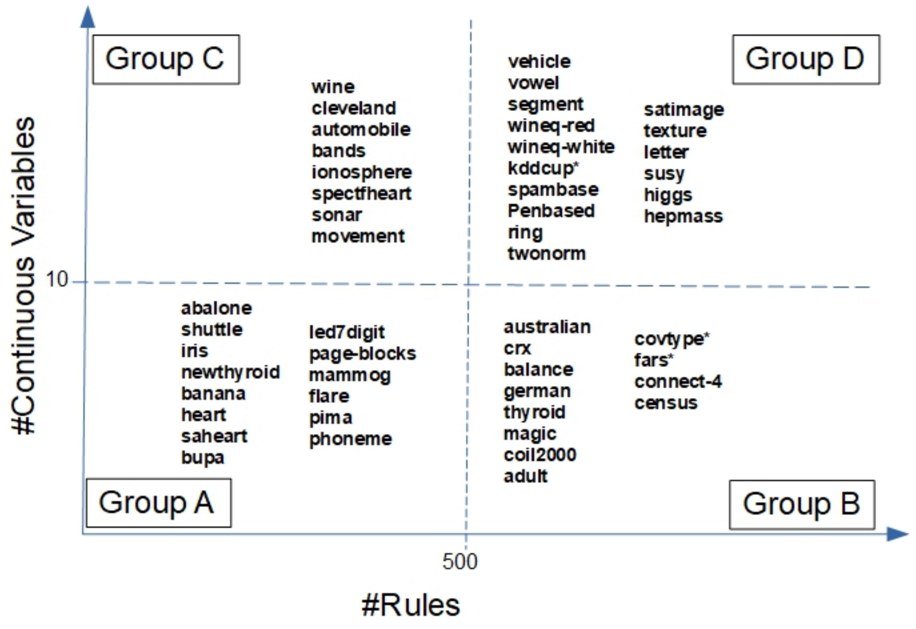

Figure 3: The four groups (A,B,C and D) considered in this experimentation, taking into account the number of rules obtained by the Chi algorithm and the number of continuous variables of each dataset.

- Group C: It has 8 datasets and is composed of those with more than 10 continuous variables but less than or equal to 500 rules.
- Group D: It has 16 datasets and is composed of those with more than 10 continuous variables and more than 500 rules.

A specific study for each of these groups will follow.

4.1. Study on Group A datasets

In this subsection we study datasets with a low number of rules and a low number of continuous variables.

Table 3 reports the results obtained by Algorithms 2, 3 and 4 on the group A datasets. As previously mentioned, Algorithm 1 has been removed from the study to facilitate the readability of the tables since its results in time are always greater than those of Algorithm 2 and its results in terms of accuracy are exactly the same.

Since we have taken as the value of the *MaxRules* parameter the value $2^{10} = 1024$ and as the number of continuous variables is less than or equal to 10, this means that in these cases, the whole neighborhood is explored, and consequently, the results of the sequential inference (Algorithm 2) coincide with those of the recursive versions (Algorithms 3 and 4), in terms of accuracy, and for that reason only one column for *Test* and one column for *%NC* will appear in Table 3.

dataset	Test	%NC	Alg2(μs)	Alg3(μs)	Alg4(μs)
abalone	16.70	0.12	29.00	19.23	16.46
shuttle	92.42	0.01	23.74	13.86	10.74
iris	95.33	0.00	12.29	3.17	4.72
newthyroid	95.35	0.93	13.42	3.79	3.79
banana	79.38	0.02	6.97	1.34	1.70
heart	22.22	74.07	57.04	33.23	13.71
saheart	60.82	6.49	77.26	27.40	17.59
bupa	55.65	2.90	38.58	10.21	8.11
led7digit	63.60	15.20	22.86	2.60	4.42
page-blocks	93.70	0.20	42.58	6.83	8.09
mammog	76.87	7.47	26.91	2.44	3.85
flare	61.54	25.14	40.02	3.26	4.94
pima	66.02	3.26	82.90	13.33	10.10
phoneme	80.59	0.00	44.44	4.39	4.64
Average	68.58	9.7	37	10.36	8.06

Table 3: Results obtained on the Group A (datasets with low number of rules and continuous variables). The columns show the following information: *Test* accuracy on the test set, *%NC* percent of examples from the test set not triggered by any rule and *Alg2(μs)*, *Alg3(μs)* and *Alg4(μs)* average time consumed (in microseconds) by each algorithm for doing the inference of one example. The last row shows the average of each one of the column.

The obvious difference between these algorithms is the time used in performing the inference. The last three columns show the average time it takes to perform the inference of an example for each of the algorithms. It can be observed that in all cases, the Algorithm 2 presents a higher time consumption which on average can be estimated as 4 times more than that of the other algorithms. Comparing the two recursive versions, there is no a clear conclusion, although Algorithm 4 was slightly better. In any case, we are measuring in microseconds (using the C/C++ "time.h" library functions) and therefore the difference between both is not really significant.

In a slightly deeper analysis we can observe that the time reduction is greater for datasets with few continuous variables (such as *flare* or *mammog* with 0 and 1 respectively) being of the order of 10 times faster. This result confirms the analysis on the order of complexity discussed above. Algorithms 3 and 4 will be more efficient and the advantage of their use will be reflected more with problems with a low number of continuous variables.

Despite all the above, and the advantage in efficiency demonstrated in the experimentation, for problems in this group A, there is no real difference in their use. An average time of 37 microseconds per inference is a very reasonable time and for most problems (other than big data problems with a large number of examples) the classical approach will perform well.

4.2. Study on Group B datasets

In this subsection we study datasets with a high number of rules and a low number of continuous variables.

Table 4 reports the results obtained by Algorithms 2, 3 and 4 on the group B datasets. As in the previous group, the column *Test* shows the prediction ability on the test set and the column *%NC* shows the percentage of examples in the test set that are not triggered by any of the learned rules coincide for all algorithms. The reason for this is exactly the same as stated above since the number of continuous variables for the datasets contained in this group is again less than or equal to 10.

A priori this group represents the most favorable situation for the recursive approach and the most unfavorable for the sequential approach, since the efficiency of the former depends on the number of continuous variables and in this group it is less than 10, and the complexity of the algorithm of the second approach depends on the number of rules, and in all these datasets the number of rules is greater than 500.

The results empirically confirm the above and the execution time obtained by Algorithms 3 and 4 are significantly better than that obtained by Algorithm 2. On average, the results of the recursive approach are on the order of about 200 times faster than the sequential version. This difference is also related to the number of continuous variables. For example, in the *connect-4* dataset which has no continuous variables, the recursive version is of the order of more than 700 times faster than the time of Algorithm 2. For the *autralian* dataset, more similar inference times are obtained, and

dataset	Test	%NC	Alg2(μs)	Alg3(μs)	Alg4(μs)
australian	54.64	34.06	97.87	27.35	12.88
crx	40.74	52.37	103.55	25.26	14.81
balance	0.00	100.00	77.14	1.52	2.33
german	0.60	99.40	142.84	25.19	13.70
thyroid	91.99	1.36	209.86	13.20	14.62
magic	81.63	0.09	399.34	24.21	14.28
coil2000	24.27	73.66	2108.63	75.94	81.93
adult	63.32	21.37	4733.32	20.89	13.16
covtype*	85.69	0.01	8644.59	64.92	54.62
fars*	44.88	55.12	14934.89	33.82	35.16
connect-4	0.00	100.00	21070.95	14.50	27.31
census	48.09	50.08	30766.76	108.33	48.39
Average	44.65	48.96	6940.81	36.26	27.77

Table 4: Results obtained on the Group B (datasets with low number of continuous variable and high number of rules). The columns show the following information: *Test* accuracy on the test set, *%NC* percent of examples from the test set not triggered by any rule and *Alg2(μs)*, *Alg3(μs)* and *Alg4(μs)* average time consumed (in microseconds) by each algorithm for doing the inference of one example. The last row shows the average of each one of the columns.

despite that, it goes from 97 microseconds to 27 microseconds in the worst case, i.e. the recursive versions are more than three times faster than those of Algorithm 2.

The percentage of examples not triggered by any rule, column *%NC*, theoretically disadvantages the recursive approach, since it forces it to explore the whole neighborhood to find no rule to trigger. But when the set of continuous variables is low, that time is assumable, as it happens in this group.

On the other hand, it can be observed how the number of rules negatively affects the efficiency of the sequential approach. The number of rules ranges from 500 in the *australian* dataset to 80968 in the *census* dataset. The results show that for this group, the recursive versions is significantly better, producing very important time reductions in the inference. As an example, the time consumed by the inference in Algorithm 2 to do the 10-fold cross-validation for the *census* dataset (that has a total of 142521 examples) was 73 minutes, while Algorithm 4 needed less than 7 seconds to perform the same task.

In relation to the behavior of the two algorithms of the recursive version for the datasets of this group we can say that Algorithm 4 performs better in most cases (it is the best in 8 of the 12 datasets) than Algorithm 3, but no statistically significant differences are found between them.

4.3. Study on Group C dataset

In this subsection we study datasets with a low number of rules (less than 500) and a high number of continuous variables (more than 10). From the point of view of complexity orders, this group represents the most favorable case for the sequential approach. Here we will show whether the empirical results support this theoretical behavior.

	Alg2		Alg3		Alg4	
Dataset	Test	Time(μs)	Test	Time(μs)	Test	Time(μs)
wine	75.28	48.38	46.07	471.56	58.43	232.68
cleveland	18.86	61.86	18.52	46.92	18.86	24.13
automobile	42.14	30.42	27.67	515.86	40.25	294.08
bands	33.42	71.20	8.49	756.22	16.44	350.86
ionosphere	54.99	91.12	20.23	594.07	34.19	318.15
spectfheart	23.97	120.77	0.00	690.15	0.00	538.99
sonar	16.83	109.09	0.00	781.63	0.00	639.81
movement	63.33	171.56	18.33	892.87	22.78	769.66
Average	41.10	88.05	17.41	593.66	23.87	396.05

Table 5: Results obtained on the Group C - part one (datasets with low number of rules and high number of continuous variables). The two first columns are associated with the Algorithm 2, next two columns with Algorithm 3 and the last two columns with Algorithm 4. The last row shows the average of each one of the columns. For each algorithm, the accuracy on the test set (*Test*) and the time consumed by the inference of one example in microseconds (*Time(μs)*) are shown.

The parameter *MaxRules* of the function BACKTRACK is taken as 1024 $= 2^{10}$ as in the previous groups. However, since now in this group the number of continuous variables is greater than 10, by taking this value we assume that not all the neighborhood of the central rule of the example is explored and therefore, the result given by the sequential approach does not necessarily match the one given by the recursive approach (since not all rules are explored).

Moreover, as Algorithms 3 and 4 differ in the heuristics used to explore the neighborhood, the results between them are also different. For the sake of clarity of the results and to provide more readable tables, we have divided the relevant information on the behavior of these algorithms into two tables. Table 5 reports the results in accuracy over the total of test examples (*Test*) and the time consumed by an inference for each of the algorithms, while Table 6 reports the accuracy over the examples of the test set but only over those examples that trigger some rule (*Test**) as well as the percentage of examples that are not triggered by any rule (*%NC*).

The results of the table 5 show that the algorithms of the recursive approach are not very appropriate for this type of datasets since they present high inference time and worse accuracy than the sequential approach. Under these characteristics, the sequential model is preferable. On average, Algorithm 2 offers more than twice the accuracy and is more than 5 times faster than the other two algorithms.

	Alg2		Alg3		Alg4	
Dataset	Test*	%NC	Test*	%NC	Test*	%NC
wine	96.40	21.91	97.62	52.81	98.11	40.45
cleveland	65.88	71.38	66.27	72.05	65.88	71.38
automobile	88.16	52.20	100.00	72.33	95.52	57.86
bands	66.67	49.86	72.09	88.22	68.97	76.16
ionosphere	97.97	43.87	98.61	79.49	98.36	65.24
spectfheart	48.12	50.19	0.00	100.00	0.00	100.00
sonar	97.22	82.69	0.00	100.00	0.00	100.00
movement	93.06	31.94	100.00	81.67	100.00	77.22
Average	81.69	50.51	66.82	80.82	65.86	73.54

Table 6: Results obtained on the Group C - part two (datasets with low number of rules and high number of continuous variables). The two first columns are associated by the algorithm 2, next two columns with algorithm 3 and the last two columns with algorithm 4. For each algorithm, the accuracy on the test set considering only the examples that trigger any rule (*Test**) and the percentage of examples that do not trigger any rule (*%NC*) are shown.

A more detailed analysis reveals that the difference in accuracy between the two approaches is due to the high number of examples that do not trigger any rule when the recursive version is used. The column *%NC* associated with Algorithm 2 of the Table 6 shows the percentage of examples that are not triggered by any rule. Comparing this column with the same information but from Algorithms 3 and 4, it can be seen that except for the *cleveland* dataset where the percentages are similar, the increase in this parameter is always higher than 17 points. There are two extreme situations to be highlighted, the *specfheart* and *sonar* datasets where the recursive approach fails to match any example with any rule.

From Table 6 we can draw the following conclusions:

- The difference in the parameter *%NC* between Algorithms 3, 4 and Algorithm 2 shows the proportion of examples that are triggered by rules that are not among the 1024 best-fitting ones. In addition, algorithms 3, 4 use different heuristics to estimate this order in the evaluation and that is different in both.

- From the above fact, it follows that the degrees of adaptation with which the examples trigger the rules take low values.

- When the examples of the test set are correctly distributed over the clusters that were recognized by the learning algorithm over the training examples, the recursive approach works well, even better than the sequential approach, giving similar values in *cleveland* dataset.

- If we compare the results obtained by Algorithms 3 and 4 we can observe that in all the datasets the latter obtains better results in the two parameters studied. This indicates that the heuristic used in this algorithm improves experimentally to the one proposed in Algorithm 3. On average the improvement is 6 % in accuracy and about 200 microseconds less per inference.

The conclusion for the group C datasets is that the sequential approach of Algorithm 2 is the best option since nothing can be assured about the behavior of the test examples.

4.4. Study on Group D datasets

In this subsection we study datasets with a high number of rules (more than 500) and a high number of continuous variables (more than 10).

In the three previous groups we have concluded that the empirical results have demonstrated what could be intuited from the order of complexity of each algorithm. On the one hand, group B with many rules would favor the recursive approach, group C with many continuous variables made the sequential approach the recommended one. In the case of group A, where the time-increasing variables were at low values for both approaches, experimentation showed that the recursive approach offered better results.

In this last group we find datasets with a high number of rules and a high number of continuous variables, that is, this group contains datasets where the variables that affect high time consumption are found with high values for the both approaches and therefore it is more difficult to estimate which one can offer better results.

In this group we have 16 datasets, and as in group C we have divided the results obtained in two tables. Table 7 reports the results on accuracy and time consumed in the inference of an example for each one of the algorithms.

	Alg2		Alg3		Alg4	
Dataset	Test	Time(μs)	Test	Time(μs)	Test	Time(μs)
hepmass$^{0.1\%}$	75.77	3907504.76	13.83	1539.08	29.71	1072.97
higgs$^{0.1\%}$	58.54	3860736.36	27.58	996.74	49.40	503.74
kddcup*	99.95	650.58	99.95	22.91	99.95	31.10
letter	77.74	2195.16	76.13	150.19	77.43	72.81
penbased	95.58	1316.72	91.11	186.34	94.01	67.62
ring	68.92	1804.92	49.34	396.78	54.64	240.84
satimage	65.25	2025.13	56.25	439.11	64.83	313.37
segment	91.73	190.59	87.36	137.19	90.52	70.71
spambase	81.31	872.66	77.25	215.23	80.64	98.74
susy	73.81	104224.60	73.68	139.22	73.80	79.05
texture	92.85	2166.64	51.84	658.25	87.44	382.74
twonorm	89.15	2088.24	19.84	634.60	34.14	385.67
vehicle	60.28	223.55	42.44	448.42	52.01	208.21
vowel	93.64	127.75	93.64	53.22	93.43	22.06
wineq-red	56.85	205.61	56.79	76.43	56.79	30.72
wineq-white	49.92	352.38	49.84	58.75	49.92	24.60
Average	76.96	492917.85	60.43	384.53	68.04	225.31

Table 7: Results obtained on the Group D - part one (datasets with high number of rules and high number of continuous variables). The two first columns are associated with the Algorithm 2, next two columns with Algorithm 3 and the last two columns with Algorithm 4. The last row shows the average of each one of the column. For each algorithm, the accuracy on the test set (*Test*) and the time consumed by the inference in microseconds of an example (*Time(μs)*) are shown.The value $^{0.1\%}$ on the name of *higgs* and *hepmass* datasets indicates that the results have been obtained using ten cross-validation but using only the 0.1% of the examples of the test set in each one of the ten validations.

As in the experimentation carried out in all the previous groups, for this group we will use 1024 as the value of the parameter *MaxRules* of the procedure BackTrack and for the same reason already discussed in group C, the output of the algorithms based on the recursive approach does not have to match the output returned by the sequential approach, nor does the output of the two algorithms of the recursive approach have to match.

In addition, within this group, there are two datasets that are particularly complex and require a very efficient inference since they have more than 10 million examples, generate more than 8 million rules and has 28 variables, all continuous. These datasets are *higgs* and *hepmass*. Due to the high values of these parameters, the inference take a long time. For this reason, both in this study and in the following ones we make a special treatment of these two datasets. If we look at the Table 7, both datasets have above their name the value $^{0.1\%}$. This value refers to the fact that the 10-fold cross validation has been performed as in all the datasets of this study, but in this case, in the inference, instead of taking 100 percent of the examples, only 1 per thousand (0.1 percent) of the examples has been taken in each one of the 10 validations.

Under these conditions, in our case, the inference performed by the three algorithms studied here learn the same models and test on exactly the same examples, although this testing is only on 0.1 % of the total number of examples that should be tested. From the point of view of the time consumed by the inference, it only affects the fact that this average has been made with a lower number of examples, but the average that would be obtained and applied on all the examples should be very similar. If we look at the value obtained by Algorithm 2 on the dataset *hepmass*, we can observe that each inference consumes almost 4 seconds on the computer on which the experimentation has been performed, (3907505 μ seconds exactly). The dataset has 10.5 million examples. In a complete cross validation, it is necessary to make inferences on all the examples, i.e., it is necessary to make 10.5 million inferences consuming an average of about 4 seconds only on the inference. That means that 42 million seconds (1.3 years) are required to perform the experiment. Applying the 0.1 % reduction we have managed to go from those 1.3 years to about 4.86 days. Similarly, the time needed for *higgs* was also about 5 days compared to the 1.4 years that would have been needed taking all the examples for testing.

Regarding the time analysis, it can be observed that the recursive approach obtains the lowest execution times for all datasets, with the only exception of *vehicle* when using Algorithm 3. On the other hand, Algorithm 4 obtains the lowest times except for the dataset *kddcup**. In this comparative study, algorithm 4 is about 1/3 faster than Algorithm 3. Comparing Algorithm 2 and Algorithm 4, it is observed that with datasets with a high number of rules, the time differences of both approaches are more pronounced. If we take all the datasets, except the ones that take more rules *susy*, *higgs* and *hepmass*, Algorithm 4 is on average 7 times faster than Algorithm 2. If only the last 3 datasets are considered, on average, Algorithm

4 is five thousand times faster than Algorithm 2.

Although algorithms of the recursive approach are faster in time, on average there is a significant reduction in the overall accuracy, from 76.96 of Algorithm 2 to 68.04 of Algorithm 4. This average loss of 8 % is not uniform over all the datasets. We can indicate that there are 10 datasets where the difference in accuracy is less than 2 % (*vowel*, *segment*, *wineq-red*, *wineq-white*, *kddcup**, *spambase*, *penbased*, *satimage*, *letter* y *susy*), and yet in *twonorm* and *hepmass* that difference is over 45 %.

	Alg2		Alg3		Alg4	
Dataset	Test*	%NC	Test*	%NC	Test*	%NC
hepmass$^{0.1\%}$	77.65	2.42	79.56	82.62	78.35	62.08
higgs$^{0.1\%}$	58.59	0.08	60.82	54.66	58.47	15.50
kddcup*	99.95	0.00	99.95	0.01	99.95	0.00
letter	77.75	0.02	77.18	1.36	77.64	0.27
penbased	98.85	3.31	98.93	7.91	98.92	4.96
ring	72.18	4.51	85.26	42.14	79.07	30.91
satimage	65.25	0.00	64.16	12.32	65.23	0.61
segment	93.22	1.60	93.21	6.28	93.52	3.20
spambase	84.70	4.00	84.51	8.59	84.63	4.72
susy	73.81	0.00	73.77	0.11	73.81	0.01
texture	92.92	0.07	91.61	43.42	93.11	6.09
twonorm	91.88	2.97	91.29	78.27	90.99	62.49
vehicle	61.74	2.36	66.48	36.17	63.86	18.56
vowel	93.73	0.10	93.73	0.10	93.72	0.30
wineq-red	57.53	1.19	57.58	1.38	57.61	1.44
wineq-white	50.01	0.18	49.96	0.24	50.02	0.20
Average	78.11	1.43	79.25	23.47	78.68	13.21

Table 8: Results obtained on the Group D - part two (datasets with high number of rules and high number of continuous variables). The two first columns are associated with the Algorithm 2, next two columns with Algorithm 3 and the last two columns with Algorithm 4. The last row shows the average of each one of the column. For each algorithm, the accuracy on the test set considering only the examples that trigger any rule (*Test**) and the percentage of examples that do not trigger any rule (*%NC*). The value $^{0.1\%}$ on the name of *higgs* and *hepmass* indicates that the results have been obtained using ten cross-validation but using only the 0.1% of the example of the test part in each one of the ten validations.

In table 8 we can observe a somewhat more detailed study of where these discrepancies in accuracy come from and we can formulate the following conclusions.

- The difference in the accuracy of the 10 datasets mentioned above is caused by the existence of the examples not covered by the rules.

- The discrepancy between *hepmass* and *twonorm* is also due to uncovered examples. In this case, there is a very large difference in uncovered examples.

- The percentage of accuracy on the examples covered is very similar. Only the difference in favor of Algorithm 4 in the *ring* dataset is noteworthy. The reason must be that there are rules farther away from the one with maximum adaptation on the example with a higher weight than closer rules. The sequential approach takes these farther away, while in the recursive version the rule triggered by the other approach falls outside its search space. A large but minor difference occurs in *vehicle* (2.12 %), for the rest the differences are less than 1 %.

- It seems that the use of rules closer to the optimum may give an advantage in inference, since although the differences are usually small (less than 1 %), they tend to favor the recursive approach.

- Algorithm 4 has a better prediction capability on examples that trigger some rule than Algorithm 3. In the cases, where this does not happen (*vehicle*, *ring*, *higgs* and *hepmass*), it is because there is a significant difference in the percentage of examples that do not trigger any rule between the two algorithms. For example, in *higgs*, Algorithm 3 has a 2-point higher hit percentage than Algorithm 4, but there is a 40-point difference in fewer examples covered by Algorithm 3. These high differences in the percentage of examples not covered between the two algorithms cause the heuristics implemented in Algorithm 4 to produce better results in time, rule coverage and accuracy compared to Algorithm 3.

Two things can be concluded from the above analysis:

- Algorithm 4 presents a prediction capability similar to that offered by the sequential approach with a very significant reduction in time when the former triggers a rule, but

- the high percentage of examples that are not triggered by the recursive approach versus the sequential one, makes this approach not usable for all problems.

With these two conclusions in mind, we now study the behavior of Algorithm 5, which is a modified version of Algorithm 4 in which a new heuristic is added consisting of using a distance between rules to perform the exploration of the neighborhood of the central rule to the example.

4.5. Study of Algorithm 5

In the previous study on the group of datasets containing a high number of rules and a high number of continuous variables, it has been seen that the recursive approach has a behavior that produces a very significant reduction of time, but for some datasets, the high number of examples that are not triggered by any rule produces an undesirable result in some cases. Frequently the problem is due to the fact that the triggered rules have very low values in the degree of adaptation between example and antecedent.

In order to see to what extent these degrees of theoretical adaptation affect the results obtained by the recursive approach, we study now the Algorithm 5. This algorithm is a version of Algorithm 4 with an additional constraint of limiting the neighborhood where to search for the rule. This restriction is defined by the value of the parameter d that takes integer values and was defined in the description of the Algorithm 5.

Table 9 reports the results on accuracy, both on the total number of examples (*Test*) and on only the examples that are triggered by rule (*Test**) for 5 values of the distance parameter d.

	Test					Test*				
Dataset	0	1	2	3	5	0	1	2	3	5
hepmass$^{0.1\%}$	0.76	6.26	20.67	38.78	63.73	81.09	80.06	79.94	79.62	78.33
higgs$^{0.1\%}$	13.25	32.45	46.46	53.52	57.74	64.63	59.27	58.4	58.4	58.63
kddcup*	99.92	99.95	99.95	99.95	99.95	99.95	99.95	99.95	99.95	99.95
letter	57.06	74.44	77.21	77.54	77.62	83.01	77.96	77.71	77.64	77.64
penbased	33.77	67.41	84.53	91.31	95.16	99.41	99.22	99	98.94	98.88
ring	29.03	44.88	50.73	55.00	62.97	96.67	90.96	84.32	78.1	72.54
satimage	46.95	59.47	63.68	64.88	65.27	66.37	65.27	65.28	65.27	65.29
segment	69.57	85.71	90.09	90.91	91.52	94.36	93.97	93.87	93.54	93.42
spambase	62.69	75.79	79.68	80.81	81.23	81.23	84.28	84.69	84.74	84.71
susy	66.71	73.43	73.80	73.80	73.81	68.71	73.65	73.83	73.81	73.81
texture	33.64	66.18	82.56	88.80	91.85	92.55	92.86	92.71	92.92	92.85
twonorm	0.14	3.34	15.09	36.97	75.15	100	89.17	91.63	91.63	91.66
vehicle	9.46	28.13	45.86	55.44	59.57	78.43	69.79	66.9	65.14	61.99
vowel	44.14	81.01	91.62	93.43	93.64	95.62	95.02	94.48	93.81	93.73
wineq-red	43.65	52.78	55.66	56.54	56.85	68.1	58.86	57.53	57.62	57.57
wineq-white	40.83	48.55	50.04	49.92	49.92	57.14	49.55	50.26	50.03	50.01
Average	40.72	56.24	64.23	69.23	74.75	82.95	79.99	79.41	78.82	78.19

Table 9: Results obtained in accuracy on the total number of test examples *Test* and accuracy considering only the examples of the total that are triggered by some rule *Test** for the set of distance values 0,1,2,3 and 5.

In Table 9 and in column 0 of *Test*, Algorithm 5 uses a distance d=0 so it checks if the central rule to the example belongs to the set of inference rules and if so it uses it, otherwise it returns no output. Thus, results of percentages close to 100 % would indicate that the data used for testing move in the same grids as the data where the rules were learned, and furthermore, those grids contain examples that are mostly of the class from which they were learned. This is the case of the dataset *kddcup**. On the other hand, datasets with values close to 0 % would indicate the opposite, i.e., either the test data are not on the grids that were created in the training or, if they are, in these grids there was not a good discrimination of classes or the class of the test examples does not coincide with those of the training. To try to understand what actually happens we can look at column 0 of *Test**. Here we have the accuracy of the rule base when the inference triggered some rule. We can observe that in this case the accuracy values are high. Therefore the fundamental reason for low values in column 0 of *Test* is because no rules are triggered, since the results when rules are triggered are quite acceptable, for example, if we look at *hepmass* it ranks at 0.76 % over all examples, but it has a hit probability of 81.09 % when any rule is triggered. Analogously, *twonorm* goes from 0.14 % to 100 % when it triggers a rule.

In addition, if we look at the trend of the values of *Test** as the value of d increases, we can observe that the values tend to be maintained, although they fall slightly. In the average behavior we can observe the smoothness of this drop. However we can observe that when the value of d increases the values of *Test* increase considerably as well. In conclusion, when d increases and therefore we are allowing in the search rules that are further away from the central rule to the example, the overall accuracy percentage goes up and the accuracy percentage when the system matches a rule with the example goes down a little bit.

In Figure 4 we can observe the behavior of the inference on the percentage of examples that are not triggered by any rule as the value of d increases. A significant decrease can be observed in all datasets. This fact, together with the two previous ones, indicates that by increasing the value of d we also increase the accuracy on all the examples, decreasing the percentage of examples on which no rules are triggered and maintaining (although slightly decreasing) the accuracy of the rules when the system uses one of them.

As a conclusion of the above, we could increase the value of d until we find an inflection point where the drop of *Test** is important or the percentage of examples not triggered by any rule is reduced to the maximum, but this is not possible, because increasing d implies increasing exponentially the search space of the rules and that implies an extraordinary consumption of time.

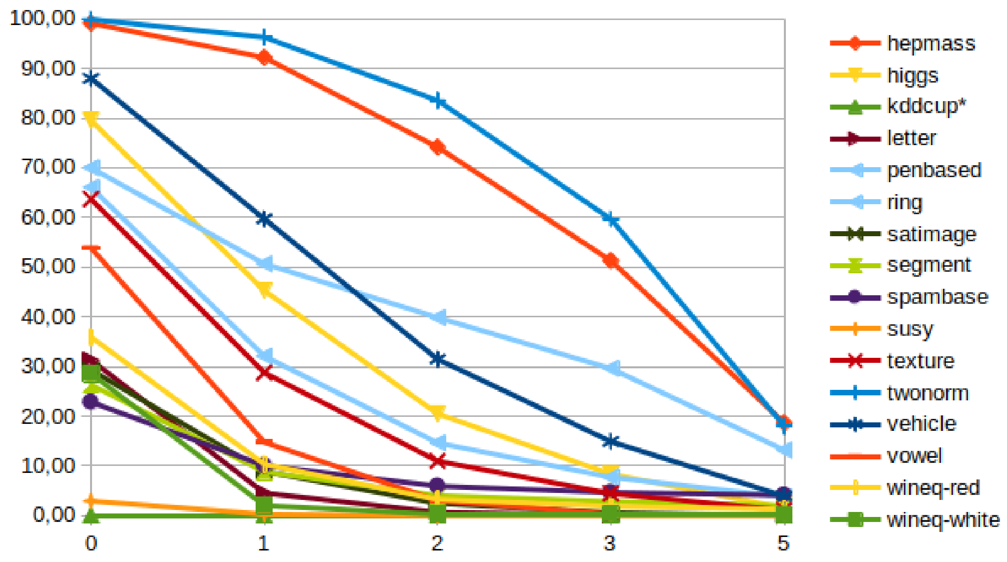

Figure 4: Percentage of examples not triggered by any rule for the set of distances values 0, 1, 2, 3 and 5.

Table 10 reports the average time (expressed in microseconds) consumed when performing the inference of an example for each of the datasets studied and for the different values of d. We can observe that for values of $d < 3$ the consumption is very small and therefore very fast inferences are produced, with value 3 the inferences begin to consume a lot of time, and from it, the time shoots exponentially.

From these results, the use of this algorithm with a low value of d produces good prediction results in many cases requiring a very small amount of time. However, the fact that the percentage of examples not triggered by any rule is high makes it unfeasible as a single method for performing inference. But, it does seem that it could be used as an agile method with a high degree of accuracy in a first stage of an inference process combining other techniques.

In the following subsection we propose a hybrid inference algorithm that combines several of the algorithms presented here to obtain a new model in inference that could obtain results similar to classical inference (sequential approach) while reducing inference times and therefore be of great use in problems with a very high number of rules and continuous variables.

Dataset	0	1	2	3	5
hepmass$^{0.1\%}$	0.22	1.05	8.42	3355.04	26151.90
higgs$^{0.1\%}$	0.23	0.97	5.38	1414.23	4947.68
kddcup*	0.34	0.27	0.32	0.29	29.42
letter	0.16	0.34	0.63	0.78	93.12
penbased	0.16	0.25	0.42	0.65	95.10
ring	0.19	0.57	2.01	6.56	3316.88
satimage	0.35	1.91	14.05	79.32	96419.89
segment	0.22	0.38	0.65	0.95	225.42
spambase	0.53	0.69	0.96	1.49	651.02
susy	0.15	0.33	0.65	0.81	87.59
texture	0.40	2.44	24.34	168.44	238850.91
twonorm	0.19	0.71	3.03	13.34	9585.85
vehicle	0.22	0.46	1.46	3.81	1175.22
vowel	0.15	0.17	0.25	0.24	38.61
wineq-red	0.12	0.20	0.30	0.35	43.28
wineq-white	0.14	0.21	0.28	0.32	41.17
Average	0.23	0.68	3.95	315.41	23859.57

Table 10: Time consumed in microseconds (mus) by the execution of the inference of an example for the set of distance values 0,1,2,3 and 5.

4.6. A hybrid approach to inference in problems with many continuous variables and many rules

In the previous section, the results offered by the Algorithm 5 have been presented with the objective of discovering the behavior of the Algorithm 4 when the neighborhood for searching rules is reduced. The results show that the proposed algorithm is not a solution for any problem defined in group D in general, but that it could be a component of a more complex inference algorithm since it allows to find rules with good accuracy for the classification in an efficient way.

The main drawback of the previous algorithm is that the percentage of examples that are not triggered by any rule tends to be very high due to the restricted search space of the rule. In this case, Algorithm 4 has been shown to offer a high level of accuracy on the examples that trigger any rule, while also offering a reduced time to find such a rule.

In the same way as Algorithm 5 (called previously HEURISTIC_NEARBY-NEIGHBORHOOD_INFERENCE) when we select a low d value, Algorithm 4 (also called previously HEURISTIC_NEIGHBORHOOD_INFERENCE) tends to have a high percentage of examples not triggered by any rule. This arises on many occasions when the number of continuous variables is high in relation to the value given to the *MaxRules* parameter. This percentage is high, but lower than the percentage offered by Algorithm 4, and therefore we use it when Algorithm 5 does not trigger any rule. On the other hand, for those examples that are not triggered by either Algorithm 4 or Algorithm 5, we will use the sequential approach inference (Algorithm 2 or also called previously STANDARD_INFERENCE_PRUNNED).

An algorithmic description of this process is shown in Algorithm 6.

Algorithm 6 A hybrid inference algorithm that combines heuristic and standard inferences

```
1:  function HYBRID_INFERENCE($e$, $R^I$, $d$, $MaxRules$)
2:      class = HEURISTIC_NEARBY-NEIGHBORHOOD_INFERENCE($e$, $R^I$, $d$)
3:      if (class != $\emptyset$) then
4:          return class
5:      end if
6:      class = HEURISTIC_NEIGHBORHOOD_INFERENCE($e$, $R^I$, $MaxRules$)
7:      if (class != $\emptyset$) then
8:          return class
9:      end if
10:     class = STANDARD_INFERENCE_PRUNNED($e$, $R^I$)
11:     return class
12: end function
```

On the datasets of group D, an experimental study has been performed under the same conditions as in the previous studies, setting the value of d = 2 in algorithm 5 and *MaxRules* = 1024 in algorithm 4.

The results obtained are reported in Table 11. Comparing them with those obtained with the standard inference (see results of Algorithm 2 in Tables 7 and 8) we can highlight that:

- In both versions the percentage of examples that do not trigger any rule coincide (column *%NC*), and not only in the mean value **1.43** but it coincides in all the datasets. This indicates that the same examples are triggered in both versions and consequently, the hybrid inference algorithm does not trigger any example different from the one triggered by the sequential approach.

- The accuracy over the total of the examples (the column *Test*), offers on average a very similar result, even a little bit higher for the new hybrid proposal. If we analyze this value for each dataset, we can see

dataset	Test	Test*	%NC	Time (μs)
hepmass$^{0.1\%}$	75.89	77.77	2.42	2151904.76
higgs$^{0.1\%}$	58.60	58.65	0.08	404635.45
kddcup*	99.95	99.95	0.00	29.57
letter	77.65	77.66	0.02	59.91
penbased	95.52	98.80	3.31	136.37
ring	68.76	72.01	4.51	787.38
satimage	65.25	65.25	0.00	1455.80
segment	91.69	93.18	1.60	67.18
spambase	81.31	84.70	4.00	138.78
susy	73.84	73.84	0.00	71.31
texture	92.44	92.50	0.07	2695.45
twonorm	89.15	91.88	2.97	1775.08
vehicle	61.94	63.44	2.36	264.18
vowel	93.54	93.63	0.10	20.61
wineq-red	56.66	57.34	1.19	29.09
wineq-white	50.08	50.17	0.18	24.32
Average	77.02	78.17	1.43	160255.95

Table 11: Results obtained on the Group D of datasets (high number of rules and continuous variables)in the hybrid inference algorithm. There are four columns associated to the name of the dataset: *Test* and *Test** are the percentage of examples correctly classified on the total examples and only on the covered examples respectively, *%NC* percentage of examples do not triggered by any rules and *Time(μs)* the average time consumed in the inference of a example. The last row shows the average result of each column.

that the differences are much lower than 0.5 points except for the case of *vehicle* where the difference is 1.66 points in favor of the new proposal. Unlike all the previous versions of inference, this is the only one whose results are comparable (even better, although not significantly better) than the sequential approach on this parameter.

- The accuracy on the examples that are triggered is very similar between the two versions and even similar with the values of the *Test* column, since the percentage of examples not triggered by any rule is low.

- As for the time used by the inference, the hybrid version obtains the best result on 14 of the 16 databases. The maximum time reduction occurs on the *susy* database where the hybrid version is 1400 times faster than the algorithm of the sequential approach. In any case, this result is an abnormality, since it is not normal to obtain such a high increase. If we remove this database in the average calculation, the speed gain is more than 8 times faster.

In conclusion, and from the previous analysis, it is clear that this hybrid inference algorithm is a very good alternative to the standard inference in situations where there are many rules and many continuous variables, since it obtains results very similar to what the original one would return with a significant gain in time.

5. Conclusions

In this paper we have addressed the problem of efficiency in reasoning methods that make use of fuzzy rules. In particular, we have analyzed different implementations of the inference method known as the winning rule used for classification problems. The implementations described in this paper can be grouped into two approaches: those that, given an example, search over the set of rules for the rule that best fits that example (which we have called sequential approach) and those that search among the rules in the neighborhood of the example and check if any of them is a rule that is in the set of rules (which we have called recursive approach). We have studied the behavior of these alternatives in four different situations affecting the two relevant parameters in the calculation of the complexity orders of these two approaches: the number of rules and the number of continuous variables. The experimental study has shown that the version known as standard and which solves the problem using the sequential approach is significantly better when the number of rules is low and the number of continuous variables is high. For the cases where the number of rules is low, the recursive approach obtains the best results. In the group of problems where there is a large number of both rules and continuous variables, neither approach was adequate. For the latter case, by integrating the methods, the standard and two of the neighborhood exploration algorithms into a hybrid inference algorithm, we have managed to define an inference method suitable for problems with a high number of rules and a high number of continuous variables.

Acknowledgements

This work has been partially funded by the Spanish Project RTI2018-098460-B-I00 and co-financed by FEDER funds (European Union).

References

[1] Bache, K. and Lichman, M. (2013). Uci machine learning repository.

[2] Bustan, D., Moodi, H., Pariz, N., and Azmoodeh, N. (2006). A fuzzy inference method for systems with large number of rules. In *NAFIPS 2006 - 2006 Annual Meeting of the North American Fuzzy Information Processing Society*, pages 397–402.

[3] Chi, Z., Yan, H., and Pham, T. (1996). *Fuzzy algorithms: with applications to image processing and pattern recognition*, volume 10. World Scientific.

[4] Cordón, O., del Jesus, M. J., and Herrera, F. (1999). A proposal on reasoning methods in fuzzy rule-based classification systems. *International Journal of Approximate Reasoning*, 20:21–45.

[5] Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Commun ACM 51(1), 107-113*.

[6] Dean, J. and Ghemawat, S. (2010). Mapreduce: a flexible data processing tool. *Communications of the ACM*, 53(1):72–77.

[7] del Río, S., López, V., Benítez, J. M., and Herrera, F. (2015). 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.

[8] Elkano, M., Galar, M., Sanz, J., and Bustince, H. (2018). Chi-bd: A fuzzy rule-based classification system for big data classification problems. *Fuzzy Sets and Systems*, 348(1):75–101.

[9] Fernández, A., del Río, S., Bawakid, A., and Herrera, F. (2016). Fuzzy rule based classification systems for big data with mapreduce: granularity analysis. *Advances in Data Analysis and Classification*, pages 1–20.

[10] González, A., Pérez, R., and Romero-Zalíz, R. (2019). Reasoning methods in fuzzy rule-based classification systems for big data problems. In *Proceedings of the 4th International Conference on Internet of Things, Big Data and Security DOI:10.5220/0007709002550261*, pages 255–261.

[11] Mamdani (1977). Application of fuzzy logic to approximate reasoning using linguistic synthesis. *IEEE Transactions on Computers*, C-26(12):1182–1191.

[12] Michell, K. and Kristjanpoller, W. (2020). Strongly-typed genetic programming and fuzzy inference system: An embedded approach to model and generate trading rules. *Applied Soft Computing*, 90:106169.

[13] Mizumoto, M. and Zimmermann, H.-J. (1982). Comparison of fuzzy reasoning methods. *Fuzzy Sets Syst.*, 8(3):253-283.

[14] Takagi, T. and Sugeno, M. (1993). Fuzzy identification of systems and its applications to modeling and control. In Dubois, D., Prade, H., and Yager, R. R., editors, *Readings in Fuzzy Sets for Intelligent Systems*, pages 387 – 403. Morgan Kaufmann.

[15] Tan, C. H., Yap, K. S., Ishibuchi, H., Nojima, Y., and Yap, H. J. (2014). Application of fuzzy inference rules to early semi-automatic estimation of activity duration in software project management. *IEEE Transactions on Human-Machine Systems*, 44(5):678–688.

[16] Tanaka, K. (1997). *An Introduction to Fuzzy Logic for Practical Applications.* Springer.

[17] Wang, L. . and Mendel, J. M. (1992). Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6):1414–1427.

[18] Zadeh, L. (1975). The concept of a linguistic variable and its application to approximate reasoning—i. *Information Sciences*, 8(3):199 – 249.