QChi: A Faster Classification Algorithm Based On Wang-Mendel Algorithm

Leonardo Jara, Antonio González, Raúl Pérez

***Abstract*—To address learning problems in which many examples and variables are involved, the use of algorithms that are capable of processing the data using simple and efficient techniques is required. The Wang and Mendel algorithm applied to classification (also known as the Chi algorithm) satisfies these properties in many cases. However, in particularly complex problems, such as those provided by Big Data, it has become necessary to use architectures for parallel processing, such as MapReduce, whose base algorithm is just this algorithm. The use of these architectures is necessary because the bottleneck of the algorithm is the calculation of the weight of the rules. This calculation requires, for each selected rule, to consider the adaptation of all the examples with this rule. Although this calculation is polynomial-order, in problems with many examples and where many rules are generated, it is very time consuming. In this paper, we propose QChi, a new algorithm based on the Wang-Mendel algorithm that uses, among other changes to make it efficient, a new way of calculating the weights of the rules that is much less computationally expensive. The reduction in learning time is so significant that it allows the algorithm to tackle Big Data problems without parallel processing structures. In addition to the description of the algorithm, this paper checks experimentally on a wide variety of problems, including those that other authors have used as Big Data problems, that this algorithm drastically reduces the learning time while maintaining the classification performance.**

***Index Terms*—Fuzzy Rule-Based Classification Systems, Fuzzy rule learning, Wang–Mendel method, Fast Fuzzy Inference, Big Data.**

I. Introduction

The first fuzzy rule learning algorithms appeared in the 1990s with proposals such as the Wang-Mendel algorithm (WM) [1], or the ANFIS [2] and SLAVE [3], [4] algorithms.

Among them, the WM algorithm cannot be qualified as the best from many points of view compared to other algorithms such as NSLV [5] or FURIA [6], basically for two reasons, the first one is that it is not the best in the classification capacity, and the second one is that, since it does not have a feature selection mechanism, the rules it obtains are not interpretable in many cases, and also for the same reason it generally obtains a very high number of rules. However, it has been frequently used in several applications such as [7]–[10] or multiple versions of it have been proposed or used to compare with new proposals [11]–[16]. Among the reasons for this frequent use, we can highlight its simplicity (it is basically an assignment of examples to fuzzy regions) and its efficiency (since by its description, using an efficient implementation, the time to learn is linear in the number of examples).

Leonardo Jara, Antonio González and Raúl Pérez are with DaSCI Andalusian Research Institute and with Departament Computer Sciences and Artificial Intelligence, University of Granada, Spain. (e-mail:ljara@correo.ugr.es; a.gonzalez@decsai.ugr.es; fgr@decsai.ugr.es)

An interesting review of the WM algorithm applied to classification can be found in [17] where both the positive and negative aspects of the algorithm are analyzed. However, its behavior when working with massive data and its possible improvement has not been analyzed in the same way.

Thus, despite the efficiency of the algorithm in many situations, it is also true that when the complexity of the problem increases, either the number of examples, the number of variables, or both, the algorithm is also affected, and the complexity of the calculations increases.

A model based on the use of a hierarchical structure [18] that extends the WM algorithm with the idea of solving problems with massive data has been proposed in [19]. However, in their proposal they use a different rule model (TSK rule) and do not use weight, which is the key aspect in our proposal.

One of the versions of the WM algorithm for classification problems, called the Chi algorithm [20], has been used to solve Big Data problems [21]–[24], but it has been necessary to use a parallel architecture, such as MapReduce [25], [26], to cope with all the necessary calculations.

With the idea of checking whether it is necessary to use parallel architectures based on the WM algorithm to solve certain problems or simply to improve the performance of this algorithm to solve complex problems, in this paper we analyze the complexity of this algorithm and, in particular, we will focus on the most important aspect that generates it, which is precisely the calculation of the weight of each rule.

The weight used in this algorithm has varied over time, from the simplest model used in the original version [1], which is not operational because it does not take into account the number of examples supported by each rule, to models based on the use of frequencies [12], [20]. On the latter, there are different variations such as the one proposed in [23]. Finally, models that estimate a posteriori with weight optimization algorithms [27] have also been proposed.

In any case, all the models that have been used so far have the same thing in common, they represent a bottleneck for the WM algorithm (or its versions) to solve more complex problems.

In our opinion this complexity when using massive data sets is precisely centered on the need to use the entire set of examples in the weight calculation (irrespective of the weight model chosen).

For this purpose, we study the complexity of the algorithm by analyzing each of its component parts, and from this study we effectively verified that the bottleneck is the calculation of the weight of each rule. Thus, we focus on the possible relevance of the weight in the WM algorithm, and our objective is to check to what extent accurate calculation of the

weight affects the final results. In this way we propose a new algorithm which uses a new definition of the elements involved in the computation of the weight based mainly on a heuristic reduction of the examples required for the calculation of the weight of each rule and also the new algorithm is structured in a way that makes more efficient all calculations. The proposal is valid with the different weight models proposed [12], [20], [23].

This new algorithm, which we call QChi (*Quick Chi*), produces results in a more efficient and faster way, that is, it significantly reduces the execution time and with the same level of classification performance as the original version, so that for certain problems it is not necessary to use parallel calculation structures.

The organization of this paper is as follows. In the next section, we describe the preliminary concepts necessary to formulate our proposal, such as the rule model, the types of weight, and the rest of the details of the WM algorithm, as well as the complexity of each of its steps or stages. Section III describes the details of the QChi algorithm and discusses the time improvements it brings. Finally, Section IV shows extensive experimentation in which it is proven that the classification performance of QChi is similar to that of the original WM algorithm, but, however, the former requires significantly less learning time. It is also shown that this time reduction is very important in Big Data problems, even better than the proposals that have been made using techniques such as MapReduce.

II. Preliminaries

In this section we describe those concepts necessary to understand our proposal such as the type of fuzzy rule, the types of weights used in the rule, the description of the different elements involved in the WM algorithm and an analysis of the complexity of the different steps or stages of the algorithm.

The type of rule with which we develop our proposal is the one originally used in [20], that is,

$$\text{R: } \textbf{IF } X_1 \text{ is } A^1_{j_1} \text{ and } X_2 \text{ is } A^2_{j_2} \text{ and } \ldots \text{ and } X_n \text{ is } A^n_{j_n} \quad \textbf{THEN } \text{Y is B with } \textbf{weight w(R)} \tag{1}$$

where X_i, $i = 1 \ldots n$ are the antecedent linguistic variables and each of these variables has a fuzzy domain D_i composed of $L_i = |D_i|$ linguistic labels $D_i = \{A^i_1, A^i_2, \ldots, A^i_{L_i}\}$, being $A^i_{j_i}$ a domain value D_i. Each linguistic label $A^i_{j_i}$ has an associated membership function $\mu_{A^i_{j_i}}$. Moreover, the consequent variable Y is a discrete variable, with an associated domain D_Y representing the class to be learned, B is a particular value of this domain, and $w(R)$ is a the weight associated with the rule R.

The antecedent of the rule R is noted as

$$ANT(R) = (A^1_{j_1}, A^2_{j_2}, \ldots A^n_{j_n}). \tag{2}$$

The rules extracted by an inductive learning algorithm are obtained from a data set E. Each example e in E has the form $e = (e_1, e_2, \ldots, e_n, class(e))$, where $class(e)$ is the class associated with the example.

The weight of the rule has played a key role in the WM algorithm. Initially, the WM algorithm used a very simple weight model, where for each example the following was calculated

$$w(R) = \max_{e \in E} \{\mu_R(e) \mid class(e) = B\} \tag{3}$$

where E is the training set and $\mu_R(e)$ is the matching degree between the example e and the antecedent of the rule R defined by

$$\mu_R(e) = T(\mu_{A^1_{j_1}}(e_1), \mu_{A^2_{j_2}}(e_2), \ldots \mu_{A^n_{j_n}}(e_n)). \tag{4}$$

where T is a T-norm representing the conjunction of the terms, and in our case we use the product

The consequent providing the highest weight value was the one selected for that antecedent.

This weight model is extremely simple and does not consider how many examples support the weight of the rule, so several alternatives were subsequently proposed, and all of them are based on the following concepts.

The first is $n^+(R, E)$, that is, the cardinal of the fuzzy set of the all examples of the B class in training set E that have a positive adaptation with the antecedent of the rule R, that is:

$$n^+(R, E) = \sum_{e \in E} \mu_R(e) \mid class(e) = B. \tag{5}$$

In a similar way, we define $n^-(R, E)$ as the cardinal of the fuzzy set of the all examples in training set that have a positive adaptation with the antecedent of the rule and they are not of the B class

$$n^-(R, E) = \sum_{e \in E} \mu_R(e) \mid class(e) \neq B. \tag{6}$$

Finally, we define $n(R, E)$ as the cardinal of the fuzzy set of the all examples in training set that have adaptation with the antecedent

$$n(R, E) = \sum_{e \in E} \mu_R(e). \tag{7}$$

Based on these concepts, different weight models have been used, e.g. in [12], [20] a more traditional calculation of weight, already proposed in [3] for fuzzy rules, was used

$$w(R) = \frac{n^+(R, E)}{n(R, E)}. \tag{8}$$

More recently it has been used [23] the heuristic method known as the Penalized Certainty Factor (PCF) originally proposed in [28]:

$$w(R) = \frac{n^+(R, E) - n^-(R, E)}{n(R, E)} \tag{9}$$

and a new version of the PCF, called PCF-CS [23], suitable for imbalanced datasets includes in the formula a misclassification cost associated with a particular class.

Other works have used proposals with the idea of reducing the weight computation. Thus, CHI-BD-DRF [24] uses a weight that is defined by the confidence of each particular rule:

$$w(R) = \frac{\sigma(ANT(R), B)}{\sigma(ANT(R))} \tag{10}$$

where $\sigma(ANT(R), B)$ counts the number of examples that trigger the rule R (both antecedent and consequent) with any matching degree and $\sigma((ANT(R))$ counts the number of examples that trigger the antecedent of rule R with any matching degree. This weighting of the rules not only reduces the computation time, but is also an approximation of the reliability of each rule.

Finally, there are other proposals based on estimates, rather than calculations, of the weights of the rules, such as, for example, [27].

Different versions of the WM algorithm have therefore made use of different types of weights, but most are based on the use of the $n^+(R, E)$, $n^-(R, E)$ and $n(R, E)$ functions on the whole set of examples and in the last case of an optimization process to generate the weights a posteriori, in all cases the calculation of the weight for massive datasets is a problem due to the complexity of the calculations.

After reviewing some weights proposals used in different versions of the WM algorithm, we analyze in more detail the different steps of the algorithm, as well as its complexity.

The WM algorithm has been modified over time in different aspects, such as adapting it for classification (it was initially described for regression), the type of weight associated with each rule and how to calculate it, or the conflict elimination process, but in general terms its description follows the following steps regardless of the type of weight used:

Algorithm 1 WM algorithm

```
1: function WM_ALGORITHM(E, D, Rules)
2:     Rules = {∅}
3:     for e ∈ E do
4:         r = BestAdaptationRule(e, D))
5:         AddToRuleSet(r, Rules)
6:     end for
7:     for r ∈ Rules do
8:         Calculate(n+(r, E), n−(r, E), n(r, E))
9:         w = GetWeight(r, n+(r, E), n−(r, E), n(r, E))
10:        PutWeight(r, w, Rules)
11:        ConflictResolution(r, Rules)
12:    end for
13:    return Rules
14: end function
```

The algorithm 1 shows a description of the WM algorithm applied to classification problems. The algorithm has as input a set of examples E and a definition of the domains associated with the variables D involved in the classification problem. The definition of the domains also includes the number and semantics of the linguistic labels. The algorithm generates as output a set of weighted fuzzy rules, which we will note in the description as $Rules$.

In this algorithm we can distinguish two different parts or stages, a first stage of rule generation, located between steps 3 and 6 (the first cycle), and a second stage in charge of the weight calculation and elimination of unrepresentative rules, which would be between steps 7 and 14 (corresponding to the second cycle).

In relation to the first stage, there are two processes that are applied to each example. In the first of them, *BestAdaptationRule*, given an example e and the domains of the variables D, returns the rule r with better adaptation to the example. The second of the processes *AddToRuleSet* adds the previous rule r to the set of $Rules$.

Regarding the complexity of these processes, we can say that if the rule set is modeled using a hash table, the time to add a rule to that set, even verifying that it did not exist before, is of constant order. The order of complexity associated with obtaining the antecedent with the best adaptation of an example is of order n x L, where n is the number of antecedent variables and L is the maximum number of fuzzy labels that compose the different domains of the variables, i.e., $L = max_i\{L_i\}$. Normally L is usually a value less than 9 and consequently we can consider it constant, in which case we could consider that the order of complexity of this process depends exclusively on n. Since this process is repeated for each example in the set E, the complexity order of the rule generation part is O(n x $|E|$), where $|E|$ is the number of examples in the set E.

The second stage of the algorithm deals with the calculation of the weight of each of the rules; it consists of 4 steps that must be applied to each of the rules obtained in the previous part. These steps are, *Calculate* function calculates the values of $n^+(r)$, $n^-(r)$, $n(r)$ that allow establishing the exact value of the weight, given a weight model (any of those discussed above) and the values calculated in the previous step, the *GetWeight* function obtains the concrete value of the weight, the *PutWeight* function sets that value in the concrete rule (and removes the rule r from the set of rules if the weight is negative), and *ConflictResolution* function resolves possible conflicts between rules, i.e. if there are several rules with the same antecedent but different consequent, only the rule with the highest weight is included in $Rules$.

In summary, the complexity order of the last three steps of the second part of the algorithm are constants if we use appropriate data structures, but step 8, i.e. the computation of the values to establish the weight, is the most computationally expensive process. If we assume that L, the number of fuzzy labels defined in the variable domain is constant, the complexity order is n x $|E|$ for a rule, n being the number of variables involved in the antecedent of the rule. Since the process is applied for each rule, the time consumed in this process is O($|Rules|$ x n x $|E|$). In the worst case, $|Rules|$ = $|E|$ and therefore the complexity order is O(n x $|E|^2$).

Although the order of complexity of the whole process is polynomial, for Big Data problems where we work with a massive amount of data and a very high number of rules are generated, the exact calculation of the weight slows down the rule extraction process significantly. In fact, in works such as [21]–[24] use is made of the Map-Reduce model [25], [26], a parallel processing technique whose objective is to reduce this rule extraction time.

Since the main bottleneck of the algorithm is weight calculation, we want to know the importance of accurate weight calculation and whether a faster and simpler weight estimation allows us a similar prediction capability, but with a much

shorter calculation time. The answer lies in a new algorithm that we describe in the next section.

III. The QChi algorithm

With the idea of proposing a faster algorithm than the original WM algorithm, we propose to accelerate the calculation of the rule weight by reducing the set of examples on which it is calculated. The types of weights to be used are the same as those described above, but the calculation of $n^+(R,E)$, $n^-(R,E)$ and $n(R,E)$ now will be done on a subset of E.

We first define the concept of the central rule to an example e, as the rule with the best matching degree between each component of the example and each component of the rule, and will be denoted as $C(e)$. The central rule is just the rule that associates the WM algorithm to each example in step 4 of Algorithm 1.

Now, given a rule R, first we select the subset of examples of E that have in their central rule the same antecedent as the rule R, i.e.

$$E^R = \{e \in E \mid ANT(C(e)) = ANT(R)\} \tag{11}$$

and secondly in the calculation of the weight we use $n^+(R,E^R)$, $n^-(R,E^R)$ and $n(R,E^R)$ instead of $n^+(R,E)$, $n^-(R,E)$ and $n(R,E)$ respectively.

Obviously, in most cases

$$|E^R| \ << \ |E| \tag{12}$$

and therefore, the effective time needed to calculate $n^+(R,E^R)$, $n^-(R,E^R)$ and $n(R,E^R)$, and therefore the weight of the rule, is very significantly reduced.

With this change we associate to each rule an estimation of its exact weight based on taking into account only those examples that present the maximum compatibility with that rule, leaving out of that calculation the rest of the examples of the training set. So, in Algorithm 2 we present a fuzzy rule learning algorithm for classification based on the WM algorithm but using E^R subsets for obtaining the weight estimations. In addition, the new algorithm is structured differently from the original one to efficiently and incrementally perform the calculations necessary to obtain the estimated weights.

Algorithm 2 QChi algorithm

1: **function** QChi(E, D, $Rules$)
2: $AntSet$ = {∅}, $Rules$ = { ∅}
3: **for** $e \in E$ **do**
4: a = BestAdaptAntedecent (e, D))
5: UpdateAntSet (a, $AntSet$)
6: **end for**
7: **for** $a \in AntSet$ **do**
8: r = BuildRule(a, &w)
9: **if** ($w > 0$) **then**
10: AddToRuleSet(r, w, $Rules$)
11: **end if**
12: **end for**
13: **return** $Rules$
14: **end function**

The inputs of the algorithm are an example set E and a description of the domain of each variable involved in the problem D. The output is a fuzzy rule set based on the same rule model that is used by the WM algorithm.

As with the WM algorithm, we can also distinguish two stages: the first one with the generation of the potential rules that could be selected (from step 2 to 6) and the second one with the final choice of the rules.

The main difference is now the place where the "most expensive" process of weight calculation is performed. In this new algorithm that process is performed in the first part, specifically in the procedure *UpdateAntSet*. Thus, in step 4, the function *BestAdaptAntecedent*, takes an example e and returns the antecedent a that has the best adaptation with that example. The sequence of antecedents generated in this step is stored in the set $AntSet$. During the execution of the algorithm, it is *UpdateAntSet* the one in charge of keeping the base of generated antecedents updated. This means that for each antecedent, the accumulated values of the adaptations of the examples that have already been processed for each of the potential values of the class are stored. Since the calculation of the weight will be done taking into account only the examples that considered that antecedent as the best fit, this updating process has a constant order of complexity when a hash table type structure is used for its implementation.

It is easy to see how the order of complexity of this first stage of the algorithm coincides with the order of complexity of the first stage of the algorithm 1. In both cases it is O(n x $|E|$), where $|E|$ is the number of examples in the set E and n the number of variables involved in the classification problem.

Following the steps of the algorithm, at the end of the first cycle, just before step 7, in the set $AntSet$ we have all the antecedents that are generated from the central rule of each example using the complete set of training examples E (just as the original version would do). In step 7, the *BuildRule* function is responsible for giving the final rule for each antecedent considered in the previous process. That is, given an antecedent a, it completes the rule r by adding what its consequent will be and what weight it will be assigned. To determine these values, the weight value is calculated for each possible consequent value, and the one that obtains the highest weight value w (It is marked with an '&' to indicate that it is an output parameter) and its associate consequent are returned. The order of complexity associated with this procedure is constant.

In steps 9 and 10 only those rules that have weight values strictly greater than 0 will be included in the final set of $Rules$ (by means of the function *AddToRuleSet*). If an efficient data structure is used for insertion, the order of complexity is constant. Finally, the algorithm ends up returning $Rules$.

The complexity order of the algorithm, as a whole, is that of the computationally most expensive part of the algorithm, which corresponds to the first part, is O(n x $|E|$). This is a better complexity order than that of the original algorithm which would be O(n x $|E|^2$) and this difference is due, as mentioned above, to the different form of the weight calculation.

Dataset	Var	CVar	Ex
abalone	8	7	4174
banana	2	2	5300
coil2000	85	2	9822
letter	16	16	20000
newthyroid	5	5	215
phoneme	5	5	5404
saheart	9	8	462
shuttle	9	9	57999
thyroid	21	6	7200
vowel	13	11	990

Dataset	Var	CVar	Ex
adult	14	6	45222
bands	19	19	365
crx	15	6	653
magic	10	10	19020
page-blocks	10	10	5472
pima	8	8	768
satimage	50	36	6435
spambase	57	57	4597
twonorm	20	20	7400
wineq-red	11	11	1599

Dataset	Var	CVar	Ex
australian	14	7	690
bupa	6	6	345
iris	4	4	150
mammogr	5	1	830
penbased	16	16	10992
ring	20	20	7400
segment	19	19	2310
texture	40	40	5500
vehicle	18	18	846
wineq-white	11	11	4898

TABLE I
DATASETS USED TO COMPARE THE QCHI AND WM ALGORITHMS.

IV. EXPERIMENTS

In this work we have proposed a new classification algorithm, called QChi, which proposes an alternative implementation to the well-known WM algorithm for classification and whose main difference lies in the weight estimation. In this section, we experimentally prove that the use of the proposed weight estimation not only produces a very important reduction in the time needed for learning, but it is also able to maintain the prediction capacity obtained when using the weight used in the original version and based on an exhaustive calculation on all the examples.

This experimental section is divided into two parts. In the first part (subsection A) we try to experimentally show the behavior of the QChi algorithm versus the original version of the WM algorithm for classification, or Chi algorithm as it is also called, and for this purpose we make use of 30 datasets from the UCI repository [29]. The 30 selected datasets are those we can consider "small" in the sense that neither by the number of examples they contain, nor by the number of variables involved, nor by the number of rules that are generated, the exhaustive calculation of the weight is not really seen as a real drawback, since compared to other classification algorithms their learning times are very fast.

In the second part (subsection B) of this experimental section we will use datasets that have usually been considered, by experts in this field, as very time-consuming for the original version of WM and for which parallel computing versions have been proposed using the WM algorithm as the base algorithm. In particular, we will focus on proposals using the MapReduce paradigm, which were proposed to speed up the weight computation in time.

A. Comparing the QChi and WM algorithms

In the first part of this study, we use 30 datasets for classification that are described in Table I, all of which are extracted from the UCI machine learning repository [29]. For each of them is shown the global number of variables (Var), the number of continuous variables (CVar), and the number of examples (Ex). All the selected datasets are well known, frequently used and 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.

All experiments have been performed on a computer Intel Core i7-9750H CPU @ 2.60GHz and 16 Gb memory running on Ubuntu 20.04.3 LTS x86 64 operating system[1].

In [30] an efficient mechanism is proposed to perform inference. It has an algorithmic complexity that does not depend on the number of rules. This fact, makes the response time very fast compared to that offered by the usual implementation of the inference. The results obtained in the different tables have been obtained using this new inference method.

In relation to the weight in this experimentation, we use the PCF model described above in Equation 9, and in relation to the domains, all continuous variables have been discretized using a set of three uniformly distributed fuzzy labels.

In this first part of the experimentation, all the results consigned are obtained using 10-fold cross-validation, and we study three parameters: accuracy (Test), number of rules obtained by the algorithms (#Rules) and time (in seconds) needed for the algorithms for extracting the fuzzy rule set (Training Time). In Table II the obtained results are showed. In this table we can see the columns have been grouped for each one of the previous parameters and for each group, there is a column with the results of QChi and other with the results of WM. In the case of the *Training Time* parameter, an additional column is included indicating how many times faster QChi is than WM. The best results for each dataset and for each of the parameters studied are highlighted in bold.

Analyzing the data in Table II, we can see that in all cases QChi obtains a similar or slightly higher number of rules than WM. Specifically, on average, the number of rules obtained by QChi is 6% higher than that obtained by WM. The cause lies in the resolution of the conflict and is due to the weight model used in QChi, which does not allow us to eliminate all the rules that WM eliminates. The study of Figure 1 will help us to understand the reason for the increase in QChi rules[2].

The table included in the upper part of Figure 1 shows some data that can help us to understand how the final value of the weight changes for each rule when we use our proposal. Specifically, the mean and standard deviation of the 30 databases studied in Table II of the following three data are described in columns A to C. Column A indicates the

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

[2]A document with the whole results for each dataset of this study can be found at https://github.com/Raul-PerezR/QChi in AdditionalInformation-PaperTFS folder.

	Test		#Rules		Training Time (seconds)		
Dataset	QChi	WM	QChi	WM	QChi	WM	Faster
abalone	**11.48**	0.02	**26.6**	1.2	**0.038691**	0.092450	2.39
adult	67.54	**67.97**	14577.8	14577.8	**0.621978**	115.650221	185.94
australian	69.86	**71.45**	350.0	350.0	**0.008228**	0.053709	6.53
banana	60.11	**60.21**	7.9	7.9	**0.020787**	0.034530	1.66
bands	**65.75**	64.93	292.4	292.4	**0.009184**	0.058442	6.36
bupa	**58.55**	57.97	45.7	45.7	**0.003249**	0.011463	3.53
coil2000	24.32	24.32	7392	7392	**0.959500**	12.142169	12.65
crx	53.29	**53.75**	435.2	435.2	**0.007456**	0.059354	7.96
iris	**94.67**	92.67	14.7	14.7	**0.000737**	0.001083	1.47
letter	**50.22**	34.07	**1320.9**	381.8	**0.304632**	11.143852	36.58
magic	**80.64**	76.75	354.1	354.1	**0.182082**	2.224813	12.22
mammogr	**79.28**	78.92	120.4	120.4	**0.005345**	0.018071	3.38
newthyroid	**89.77**	84.65	20.5	20.5	**0.001017**	0.001867	1.84
page-blocks	**92.54**	91.89	**53.3**	51.6	**0.055508**	0.154406	2.78
penbased	97.35	**97.67**	**3333.8**	3303.5	**0.153424**	7.068353	46.07
phoneme	**72.35**	71.82	58.4	58.4	**0.034246**	0.088160	2.57
pima	71.88	**72.79**	110.7	110.7	**0.010259**	0.033721	3.29
ring	**88.24**	55.19	1080.0	1080.0	**0.121407**	3.822583	31.49
saheart	69.48	**72.29**	190.9	190.9	**0.003201**	0.034588	10.81
satimage	**74.37**	47.47	**1458.6**	1228.7	**0.234403**	12.964517	55.31
segment	**86.62**	86.06	**318.3**	282.8	**0.037196**	0.281023	7.56
shuttle	**82.47**	80.16	**28.8**	26.7	**0.597692**	0.984147	1.65
spambase	**72.92**	71.50	372.1	372.1	**0.256961**	1.638114	6.37
texture	**77.38**	70.80	**1187.7**	907.2	**0.223541**	6.355889	28.43
thyroid	**92.35**	92.03	**463.0**	461.9	**0.130387**	0.785351	6.02
twonorm	88.84	**90.54**	1539.6	1539.6	**0.160099**	5.414788	33.82
vehicle	58.63	**59.69**	**405.6**	262.6	**0.014785**	0.177454	12.00
vowel	**64.24**	49.60	**279.6**	153.8	**0.015012**	0.066303	4.42
wineq-red	**52.03**	51.84	**219.7**	129.2	**0.019048**	0.156741	8.23
wineq-white	41.26	**48.43**	**260.9**	76.6	**0.064612**	0.678111	10.50
Average	**69.61**	65.91	**1210.6**	1141.0	**0.14**	6.07	42.42

TABLE II
RESULTS OBTAINED BY QCHI VS WM ALGORITHMS USING UNIFORMLY DISTRIBUTED DOMAIN WITH 3 LABELS ON ALL CONTINUOUS VARIABLES

percentage of rules that satisfy the requirement to have exactly the same weight value with both ways of calculating the weight. It indicates that there is a 20 percent, but with a very high deviation, which shows that there is a large variability, in fact, on the banana or letter data set there is no match at all, and on coil2000 the match is 100 percent. Column B indicates in how many cases the weight, although not exactly the same in both calculations, the difference between the one obtained by the QChi algorithm is not more than 10% different from the one obtained by the WM algorithm. An average result of 38.94% is obtained, but also with a fairly high deviation of 31.04, again showing that the results in the weights obtained in both cases are substantially different. Finally, column C shows the percentage in which the value obtained by the QChi algorithm is greater than or equal to that obtained by the WM algorithm for the same rule. In this case, the value of 91.58% clearly indicates that the calculation proposed here tends to obtain larger weight values than those obtained by exhaustive calculation. In this case, the standard deviation shows much less variability in this parameter. A diagram in the Figure 1 shows the conclusions of this study more clearly.

	A (==)	B ($\mid \leq 10\% \mid$)	C ($\geq$)
Mean	20.00%	38.94%	91.58%
Std	28.92	31.04	9.65

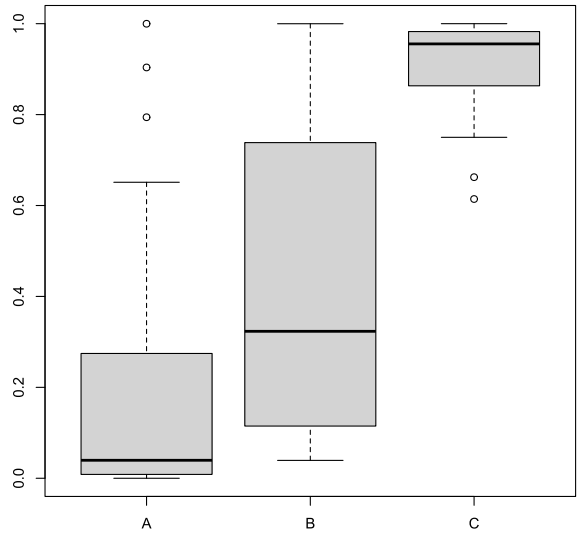

Fig. 1. Comparative between both rule weight models

Higher weight values make it more likely that rules tend to be considered more viable and therefore are less affected by both the conflict resolution mechanism and the specific weight model used. This justifies the increase in the overall number of rules obtained by the QChi algorithm as compared with WM.

In relation to learning time, it is very evident that QChi is significantly faster than WM for all the problems studied, reaching 185 times faster for datasets with many examples and that generate many rules, as in the case of *adult*. On average, for all the datasets studied, it is more than 40 times faster.

With respect to predictive capacity, the conclusion is not so simple. We can see in Table II that in some cases QChi is better and in others WM is better. If we go into more detail, we can

WM vs QChi	W/T/L	p-value
3 labels	10/1/19	0.03995
5 labels	14/3/13	0.69180
7 labels	21/3/6	0.01125

TABLE III
STATISTICAL TESTS COMPARING THE RESULTS OBTAINED BY QCHI AND WM FOR DIFFERENT NUMBER OF FUZZY LABELS IN THE DOMAINS ASSOCIATED WITH CONTINUOUS VARIABLES.

observe that in the case of the data set iris, for example, both algorithms have the same number of rules; in fact, they obtain exactly the same rules but with different weights. However, the accuracy result is better in QChi than in WM. Something similar happens in other datasets, such as magic or newthyroid, among others. However, we can see the opposite effect in datasets such as australian, saheart, or twonorm.

To determine if there are significant differences between the two algorithms in predictive capability a Wilcoxon Signed Rank Test [31] is applied. The results of this test, for different number of fuzzy labels in the domains, are shown in table III, which represents the results obtained in these comparisons, indicating the wins (W), draws (T) and losses (L) of WM against QChi along with the calculated p-value.

In the case of domains with three labels, if we formulate the null hypothesis that the predictive capacity of both algorithms is equal, the test returns a p-value less than 0.05, therefore rejecting the hypothesis with confidence of 95%. By discarding the equality, we can assume that the values obtained by QChi are higher than those obtained by WM. Therefore, it can be inferred that QChi is significantly better in predictive capacity than WM in these datasets when using 3 labels.

In order to consolidate these conclusions we extend the experimentation using uniformly distributed domains with 5 and 7 fuzzy labels. The new results regarding the number of rules and learning time reinforce the previous conclusions. In the case of 5 labels, QChi generates on average a 5% larger number of rules and is 25 times faster, and with 7 labels, it generates a 1% larger number of rules and is 60 times faster.

In relation to the learning capacity it is convenient to extend the study to a larger number of fuzzy labels of the domains to verify the result. When we use 5 fuzzy labels, both algorithms generate exactly the same rules for the iris dataset (the same as for 3 labels) but in this case WM obtains an accuracy of 95.33 and QChi of 94.66 (just the opposite of what happened for 3 fuzzy labels). However in magic or newthyroid, QChi still shows better results and it still holds that both have the same rules[3].

Thus we performed a Wilcoxon Signed Rank Tests for the accuracy parameter whose results are shown in the rows with value 5 and 7 of the Table III. It can be seen that for 5 labels it is accepted that the test results of both algorithms are equal. For 7 labels, however, the hypothesis of equality is rejected and we can assume that the predictive capability of WM is better than that of QChi is accepted.

[3]Tables with the whole results on 5 and 7 labels for each dataset can be found at https://github.com/Raul-PerezR/QChi in the AdditionalInformation-PaperTFS folder.

In short, it cannot be concluded that the exhaustive calculation of the weight value used in the WM algorithm leads to a significant improvement in the predictive capability results and therefore, a simpler and more selective calculation model that provides a quick estimation of these values, such as the one proposed here, can significantly reduce the learning time and produce good predictive capability.

B. QChi versus MapReduce proposals

Some authors have seen the possibility of using the WM algorithm to address Big Data problems. In the section IV-A we show the learning time of the WM algorithm for datasets containing less than 60 thousand examples and we can see that its times, compared to those of other well-known classifiers are really low, although in more complex problems where many rules are generated, the learning time increases significantly.

If we look at the table II, we can see that the dataset shuttle, which contains 57,999 examples, has a learning time of less than one second (0.9841 on average). The time is very low and the reason is that it generates on average only 26.7 rules. However, other datasets, such as adult, with 45,222 examples, generates 14,578 rules and that makes the learning time close to two minutes. Obviously, these time results increase for datasets with many more examples and generating many more rules.

With the idea of tackling more complex problems using the WM algorithm some authors have proposed to insert this algorithm in a parallel processing scheme known as MapReduce [25] and the idea of this experimental part is to know the relationship between QChi and these methods.

As already described in the Algorithm 1 the bottleneck of the WM algorithm is in the calculation of the weight whose order of complexity depends on the product of the number of examples by the number of rules. The proposed versions using this scheme aim to alleviate this calculation by using parallel processing.

Dataset	Var	CVar	Ex	(P_{maj}:P_{min})
kddcup*	41	26	4,856,151	(3,883,370 : 972,791)
poker*	10	10	946,799	(513,702 : 433,097)
covtype*	54	10	495,141	(283,301 : 211,840)
census	41	13	142,521	(134,359 : 8,162)
fars*	29	5	62,123	(42,116 : 20,007)

TABLE IV
DATASETS USED IN THE EXPERIMENTAL STUDY OF [22]

In [21] appear the first proposals that combine the WM algorithm (specifically, the version for classification known as the Chi algorithm) and MapReduce, and result in an algorithm called Chi-FRBCS-BigData. This algorithm divides the set of training examples into as many subsets as parallel processing is to be performed. On each subset, the WM algorithm is applied and a set of rules is obtained. In the final process, each set of rules obtained from each subset of examples is taken and the final set of rules is constructed by merging them all. In this merging process, there may be rules with the same antecedent and different consequent whose conflict is resolved by including the rule with the consequent with the highest associated weight and rules with the same antecedent

		Accuracy (on Test)			Runtine (seconds)			
		Chi-FRBCS-BigData			Chi-FRBCS-BigData			
Dataset	#Labels	32 maps	512 maps	QChi	32 maps	512 maps	QChi	Faster
kddcup*	3	99.92498	99.94120	**99.95502**	7890.87	9602.99	**161.32**	48.9
poker*	3	58.92973	56.79368	**61.42863**	2210.13	4492.45	**10.35**	213.5
covtype*	3	74.61723	73.65237	**77.13470**	391.40	3880.21	**20.02**	19.5
census	3	93.48355	92.66282	**95.96784**	388.64	714.42	**5.20**	74.7
fars*	3	94.25642	93.56599	**100.00000**	141.92	377.25	**1.71**	83.0
Average		84.24	83.32	**86.90**	2204.60	3813.5	**39.72**	55.5
kddcup*	5	99.95345	99.95085	**99.95745**	18710.18	15784.56	**167.11**	94.5
poker*	5	56.87383	56.79511	**57.88643**	8675.26	4475.27	**12.06**	371.1
covtype*	5	83.24885	81.34559	**85.24857**	4663.55	4910.73	**20.06**	232.5
census	5	94.68176	94.18538	**96.33379**	650.44	762.38	**5,66**	114.9
fars*	5	99.86419	99.84249	**100.00000**	180.77	376.45	**1.91**	94.6
Average		86.92	86.42	**87.88**	6576.04	5261.88	**45.27**	116.2
kddcup*	7	99.95500	99.95425	**99.95809**	25638.19	22540.5	**180.76**	124.7
poker*	7	59.98018	59.92133	**60.35220**	8286.67	4387.05	**12.24**	358.4
covtype*	7	87.43990	86.15001	**88.91371**	6891.16	5571.43	**20.94**	266.1
census	7	95.54433	95.36667	**96.59426**	658.55	760.92	**5.41**	121.7
fars*	7	99.94522	99.94522	**100.00000**	182.27	376.11	**1.96**	92.9
Average		88.57	88.27	**89.16**	8331.37	6727.20	**44.26**	152.0

TABLE V
QCHI VERSUS CHI-FRBCS-BIGDATA

and the same consequent but different weights. In these cases, two different versions are defined: Chi-FRBCS-BigData-Max which takes the weight of the rule with the highest weight and Chi-FRBCS-BigData-Ave which includes the rule with the average weight among the rules involved.

The results obtained with these two versions are not exactly the same as those obtained by Chi on the same datasets, but the authors conclude in this respect with *"In terms of the accuracy achieved by the algorithms considered in the study we can see that, in general, the Chi-FRBCS-BigData versions are able to provide better classification results than the serial version"*, which shows that the exact calculation of the weight does not ensure better predictive capacity.

In [22] the study of the behavior of Chi-FRBCS-BigData-Max is extended by changing the granularity of the fuzzy partitions used. For such study, the 5 datasets described in Table IV are used, where kddcup* is a binary version of the dataset KDDCup1999 taking only examples of the denial-of-service (DOS) class vs. normal class, poker is a binary version of the dataset poker* taking only classes 0 and 1, covtype* is a binary version of the dataset CoveType taking only classes 2 and 1, and fars * is a binary version of the dataset Fars taking only classes Fatal_Injury and No_Injury.

Table V shows a summary of the results obtained by Chi-FRBCS-BigData (corresponding to Chi-FRBCS-BigData-Max) and offered by its authors in [22] to which are added the results obtained by QChi in which we have reproduced the same conditions used by the former and for the same datasets.

Two parameters are studied: the predictive ability on the test set, using the standard measure of accuracy, and the learning time. The results shown in the table correspond to the mean of a 10-fold cross-validation. The weight model used is the PCF (equation 9) and the reasoning method is the winning rule. An important parameter for the Chi-FRBCS-BigData version is the number of maps used in the MapReduce scheme. In the table, we have taken the extreme values that were considered in the experiment proposed by the authors in [22], which correspond to 32 and 512 maps. The infrastructure used to obtain the results was a cluster with 16 nodes connected with a 40Gb/s Infiniband. Each node is equipped with two Intel E5-2620 microprocessors (at 2 GHz, 15 MB cache) and 64GB of main memory running under Linux CentOS 6.6. The head node of the cluster is equipped with two Intel E5645 microprocessors (at 2.4 GHz, 12 MB cache) and 96 GB of main memory. Furthermore, the cluster works with Hadoop 2.6.0 (Cloudera CDH5.4.0), where the head node is configured as name-node and job-tracker, and the rest are data-nodes and task-trackers.

The results of QChi were performed on an Intel Core i7-9750H CPU @ 2.60GHz and 16 Gb memory running on Ubuntu 20.04.3 LTS x86 64 operating system.

In Table V the best results for each of the two parameters studied are highlighted in bold. It can be seen that for all datasets and all different granularities, the learning time offered by QChi is much lower than those offered by the Chi-FRBCS-BigData versions, regardless of the number of maps used. The last column indicates how many times faster QChi is than the fastest version of Chi-FRBCS-BigData (between 32 and 512 maps). It can be seen that QChi becomes comparatively more efficient as the granularity increases.

In relation to the prediction capacity, we can observe that our proposal obtains the best results in all the studied granularities. These results show that the Chi-FRBCS-BigData approach is not better in accuracy and is largely outperformed by the learning times obtained by QChi.

In [23], an alternative algorithm to Chi-FRBCS-BigData is proposed. The new proposal, called CHI-BD, takes Chi as the base algorithm and also uses a MapReduce architecture. Although the two approaches are very similar in the way they approach the classification problem using parallel processing, they do have two important differences. The first one is that CHI-DB reproduces the same result that would be obtained by applying the original CHI algorithm, since it faithfully reproduces the weight that the original algorithm would associate to each of the rules provided as output. The second difference is that a weight preprocessing method is included

Dataset	Var	CVar	Ex	(P_{maj}:P_{min})
census	41	13	142,521	(134,359 : 8,162)
cov_1	54	10	581,012	(369,172 : 211,840)
cov_2	54	10	581,012	(297,711 : 283,301)
cov_3	54	10	581,012	(545,258 : 35,754)
cov_7	54	10	581,012	(560,502 : 20,510)
far_Fat	29	5	100,968	(58,852 : 42,116)
far_Inc	29	5	100,968	(85,896 : 15,072)
far_No	29	5	100,968	(80,961 : 20,007)
far_Nin	29	5	100,968	(87,078 : 13,890)
higgs	28	28	11,000,000	(5,829,123 : 5,170,877)

Dataset	Var	CVar	Ex	(P_{maj}:P_{min})
KDD_dos	41	26	4,898,431	(3,883,370 : 1,015,061)
KDD_nor	41	26	4,898,431	(3,925,650 : 972,781)
KDD_prb	41	26	4,898,431	(4,857,329 : 41,102)
KDD_r21	41	26	4,898,431	(4,897,305 : 1,126)
pok_0	10	10	1,025,009	(513,701 : 511,308)
pok_1	10	10	1,025,009	(591,912 : 433,097)
pok_2	10	10	1,025,009	(976,181 : 48,828)
pok_3	10	10	1,025,009	(1,003,375 : 21,634)
skin	3	3	245,057	(194,198 : 50,859)
susy	18	18	5,000,000	(2,712,173 : 2,287,827)

TABLE VI
DATASETS USED IN THE EXPERIMENTAL STUDY OF [23].

during the learning process, which speeds up and accelerates the calculation of the weight.

The experimental study proposed in [23] makes use of 20 binary classification datasets that were generated from eight datasets from the UCI repository. Table VI shows the datasets considered for experimentation. The table collects various data such as the total number of examples (Ex), the number of examples of the majority and minority class ((P_{maj}:P_{min}), the total number of variables involved in the problem (Var) and how many of them are continuous (CVar).

The datasets census, higgs, skin and susy are binary and were therefore used in their original version. The rest are multiclass and were transformed to binary by considering the examples of one of the classes as the positive class and the rest as the negative class. The abbreviations in the name of the datasets are as follows: cov (Covtype), rar (Fars), KDD, (KDDCup1999), and pok (Poker). For multiclass datasets, the positive class identifier has been appended to the dataset name. For data sets where the values associated with the class are numerical, they are identified with the number associated with the class that is considered as positive. For the other datasets, the following association was performed: for Far, Fat is the class Fatal_Injury, Inc is Incapaciting_Injury, No is No_Injury and Nin is Nonincapaciting_Evident_Injury. In the case of KDD, the four datasets considered are Two, Nor, prb and r21.

This study sets the following conditions:

- A 5-fold cross-validation is used.
- A configuration of the CHI algorithm used as the base algorithm uses a 3-label discretization of the continuous variables, the winning rule inference method, and the PCF-CS as the weight model.
- Because most of the datasets used are strongly unbalanced, the Area Under the ROC Curve (AUC) and the Geometric Mean (GM) are used as a measure of classification quality.
- All the parallel methods have been executed in an 8 nodes cluster connected via 1Gb/s Ethernet LAN network. For more details, please refer to the paper.

Table VII shows the results obtained by Chi-FRBCS-BigData (in Table CHI^{BD}_{Local}), Chi-BD (in Table CHI^{BD}_{Global}) and QChi for the predictive capability parameters (measured using geometric mean and AUC) and learning time. For both CHI^{BD}_{Local} and CHI^{BD}_{Global} we take the values provided in [23] for the case of 32 mappers since as the authors themselves indicate it is the maximum that can be executed in parallel in their cluster. The results related to QChi shown in the table have been obtained using the same configuration as those obtained in that work but in this case it was executed on an Intel Core i7-9750H CPU @ 2.60GHz and 16 Gb memory running on Ubuntu 20.04.3 LTS x86 64 operating system, the same used in the previous studies.

The table highlights in bold the best results obtained for each of the parameters. Also included is a row with the average of those datasets that have been used to generate more than one problem. It should also be noted that the higgs dataset has not been considered in the calculation of the mean because its high execution time (even more so in the case of models using MapReduce) distorted its value.

It can be observed that the number of rules obtained by QChi in most cases is higher than those obtained by the other two approaches, being CHI^{BD}_{Global} the one that tends to obtain the lowest value. A similar behavior (more accentuated in databases with many examples) in relation to the number of rules can be observed with the results obtained in Table II. As already shown in the conclusions of Table 1, this behavior of obtaining a higher number of rules is due to the tendency that the new weight estimation has to have a higher value than the one that the version with an exhaustive calculation would offer, which is the case in this experimentation of CHI^{BD}_{Global}.

Regarding learning time, we observe that QChi obtains the best results on 16 of the 20 problems, only surpassed by CHI^{BD}_{Global} for the problems related to the KDD data set. Looking at the average times, it can be seen that it is on the order of twice as fast as CHI^{BD}_{Global} and more than 23 times as fast as CHI^{BD}_{Local}. In the special case of the higgs dataset these differences are even larger, being more than 35 and 50 times faster than them.

Regarding the test performance using Geometric Mean, the best results are obtained mostly by QChi (10 out of 20) and CHI^{BD}_{Global} (9 out of 20), while CHI^{BD}_{Local} is only the best in higgs. Looking more closely at these results on groups of datasets that were generated from the same dataset, we can see that for the Poker dataset, CHI^{BD}_{Global} clearly obtains the best results, for the Fars dataset, QChi is overall better, while the results obtained by both approaches on KDDCup1999 and covtype datasets are very similar.

In relation to the test performance using AUC. Unlike the previous one, the best results are obtained by CHI^{BD}_{Global} (14 out of 20), while CHI^{BD}_{Local} equals as best with CHI^{BD}_{Global} in 2 of those 16. For this measure, QChi is the best only in 6.

	Test (using Geometric Mean)			Test (using AUC)			Runtine (in seconds)			Rules		
Dataset	CHI^{BD}_{Local}	CHI^{BD}_{Global}	QChi	CHI^{BD}_{Local}	CHI^{BD}_{Global}	QChi	CHI^{BD}_{Local}	CHI^{BD}_{Global}	QChi	CHI^{BD}_{Local}	CHI^{BD}_{Global}	QChi
census	0.403	**0.5231**	0.36442	0.5757	0.6220	**0.6541**	26	96	**5.3**	64137	**63598**	64166.8
cov_1	0.7528	0.7531	**0.7714**	0.7530	0.7532	**0.7592**	68	76	**24.8**	8275	**7940**	8859.2
cov_2	0.7296	0.7291	**0.7700**	0.7379	0.7373	**0.7730**	70	75	**25.7**	8372	**8108**	8862.2
cov_3	0.9551	**0.9565**	0.9517	0.9553	**0.9572**	0.7926	69	74	**22.2**	8438	**8249**	8866
cov_7	0.9089	0.9281	**0.9313**	0.9109	**0.9285**	0.8225	70	75	**22.5**	8181	**7917**	8859.7
cov avg	0.8366	0.8417	**0.8561**	0.8393	**0.8441**	0.7868	69.3	75.0	**23.8**	8317	**8053**	8861.7
far_Fat	0.5871	0.5871	**0.9123**	0.6723	0.6722	**0.9908**	24	81	**2.9**	**49707**	**49707**	49727.4
far_Inc	0.5572	0.6919	**0.7419**	0.6370	**0.7123**	0.7037	24	80	**2.9**	49692	**49130**	49725.2
far_No	0.8336	0.8675	**0.9049**	0.8438	0.8728	**0.9139**	23	82	**2.6**	49700	**49584**	49720.6
far_Nin	0.5307	**0.7139**	0.7108	0.6195	**0.7283**	0.6728	24	81	**2.8**	49686	**48984**	49711.6
far avg.	0.6272	0.7151	**0.8175**	0.6931	0.7464	**0.8203**	23.8	81.0	**2.8**	49696	**49351**	49721.2
KDD_dos	0.9991	0.9991	**0.9992**	**0.9991**	**0.9991**	0.9989	4362	**78**	155.7	5753	5747	**5387.4**
KDD_nor	0.9992	0.9992	**0.9993**	**0.9992**	**0.9992**	0.9985	4317	**76**	154.1	5755	5734	**5379.8**
KDD_prb	0.9911	0.9924	**0.9925**	0.9911	**0.9925**	0.9587	4402	**77**	153.4	5750	5701	**5379.6**
KDD_r2l	0.9251	**0.9840**	0.9837	0.9279	**0.9841**	0.5529	4412	**75**	155.9	5744	5686	**5381.8**
KDD avg.	0.9786	**0.9937**	**0.9937**	0.9793	**0.9937**	0.8773	4373.3	**76.5**	154.8	5751	5717	**5382.1**
pok_0	0.6183	**0.6336**	0.6224	0.6206	**0.6360**	0.6224	58	107	**10.3**	56023	**54523**	56181.5
pok_1	0.5616	**0.5848**	0.5679	0.5658	**0.5859**	0.5698	59	110	**10.7**	55986	**54254**	56181.5
pok_2	0.3713	**0.6703**	0.5411	0.5473	**0.6709**	0.5414	57	109	**10.0**	55408	**46618**	56177.4
pok_3	0.272	**0.7387**	0.5954	0.5302	**0.7388**	0.5349	59	106	**11.0**	55480	**44632**	56190
poker avg.	0.4558	**0.6569**	0.5817	0.5660	**0.6579**	0.5671	58.3	108.0	**10.5**	55724	**50007**	56182.6
skin	0.9595	**0.9597**	0.9381	0.9604	**0.9605**	0.9246	22	53	**1.3**	**23**	**23**	25
susy	0.5477	0.5524	**0.6722**	0.6210	0.6242	**0.6860**	2019	103	**90.5**	9675	**9505**	10516.6
AVG.	0.7117	0.7823	**0.7879**	0.7718	**0.8085**	0.7676	1061.3	84.9	**45.5**	29041	**27665**	29338.3
higgs	0.5772	**0.5847**	0.5639	0.5776	**0.5848**	0.5642	15115	9658	**263.2**	762443	**666068**	832577

TABLE VII
RESULTS OBTAINED COMPARING QCHI VERSUS CHI^{BD}_{Local} AND CHI^{BD}_{Global}

Study	CHI^{BD}_{Local} vs QChi	W/T/L	p-value
1	Test GM	4/0/16	0.00679
2	Test AUC	9/0/11	0.64766
3	Time	20/0/0	0.00010

Study	CHI^{BD}_{Global} vs QChi	W/T/L	p-value
4	Test GM	10/0/10	0.89603
5	Test AUC	14/0/6	0.15365
6	Time	16/0/4	0.01687

TABLE VIII
STATISTICAL TESTS COMPARING THE RESULTS OBTAINED BY QCHI, CHI^{BD}_{Global} AND CHI^{BD}_{Local}

A statistical study is included in [23] which shows that there are significant differences in favor of CHI^{BD}_{Global} in classification performance. In order to find significant differences among the classification performance and training time of these three approximations, we have used the Wilcoxon Signed Rank Tests to compare previous two MapReduce methods with QChi.

Table VIII shows the 6 studies performed, which represents the results obtained in these comparisons, indicating the wins (W), draws (T) and losses (L) of CHI^{BD}_{Local} against QChi along with the calculated p-value. The first one studies the classification performance using the Geometric Mean between CHI^{BD}_{Local} and QChi, where the hypothesis of equality is rejected at a confidence level of 95%. So we can assume that the classification capacity is higher in QChi. Considering the AUC measure, in the second study no significant differences are found between the two algorithms, since the hypothesis of equality cannot be rejected.

The third study also considers the same methods but in terms of learning time. In this case, the equality hypothesis is again discarded, but the hypothesis that CHI^{BD}_{Local} requires significantly more time to learn is accepted.

In the lastest three studies, QChi is compared with CHI^{BD}_{Global}. In relation to predictive capability (with both, Geometric Mean and AUC), the test accepts that both have the same capability and therefore no significant differences can be found between them. Although in the case of the AUC measure the win/loss balance is very favorable to CHI^{BD}_{Global}, the p-value returned by the test indicates that the differences found between the algorithms are not sufficient to rule out the hypothesis of equality. Finally, clearly QChi obtains a significantly lower learning time than CHI^{BD}_{Global} for the databases studied.

This fact again shows that performing an exact and exhaustive weight calculation does not guarantee a reduction of the error compared to the proposal made here of a computationally inexpensive weight estimation.

Finally, and in relation to learning time, the tests show that CHI^{BD}_{Global} requires significantly more learning time. As an example, to indicate that in the time that CHI^{BD}_{Global} needed to tackle the higgs problem using a cluster of 32 mappers, QChi using a single computer could have solved a higgs but with 400 million examples.

V. Conclusion

The WM algorithm, and in particular its version for classification, has been seen by some authors as a real option for tackling Big Data problems. In real applications, it has been observed that when the problems to which this algorithm is applied return very large rule sets, it is no longer a useful option. The difficulty encountered by the algorithm is with the exact calculation of the weight of the rules. In the worst case, this process can be of the order of the square of the number of examples. Even being a polynomial problem, for massive datasets, this time is also enormous. To solve this difficulty, some authors have proposed the MapReduce architecture for parallel processing. Indeed, this architecture allows one to significantly reduce the time caused by the excess of rules, but it requires the use of a cluster to deal with these problems.

The proposal presented by QChi is ultimately a modification of the WM algorithm whose main difference with the original is the modification in the calculation of the weights of the rules and a restructuring of the algorithm that allows to obtain the set of rules in a much more efficient way and with a similar classification capacity. In the experimental part, the results offered by QChi have been compared with the original WM algorithm on problems with a non-massive number of data. The results show that the new proposal significantly reduces the learning time and that the calculation of the weight using its exact values does not guarantee a better classification capacity. The results of QChi have also been compared with some of the algorithms using the MapReduce scheme. In this case, it also shows that the learning time is significantly lower while maintaining the predictive capability.

Acknowledgments

Research supported by Grant PID2022-142976OB-I00 funded by MCIN/AEI/ 10.13039/501100011033 and by "ESF Investing in your future".

References

[1] L.-X. Wang and J. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 6, pp. 1414–1427, 1992.

[2] J.-S. Jang, "Anfis: adaptive-network-based fuzzy inference system," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 3, pp. 665–685, 1993.

[3] A. González, R. Pérez, and J. L. Verdegay, "Learning the structure of a fuzzy rule: A genetic approach," *Fuzzy Systems and A.I.*, vol. 1, no. 3, pp. 57–70, 1994.

[4] A. González and R. Pérez, "A learning system of fuzzy control rules," *In: Herrera, F., Verdegay, J. L. (Eds.), Genetic Algorithms and Soft Computing, Physica-Verlag, Wurzburg*, pp. 202–225, 1996.

[5] A. González and R. Pérez, "Improving the genetic algorithm of slave," *Mathware & Soft Computing*, vol. 16, no. 1, pp. 59–70, 2009.

[6] M. Karabulut, "Fuzzy unordered rule induction algorithm in text categorization on top of geometric particle swarm optimization term selection," *Knowledge-Based Systems*, vol. 54, pp. 288–297, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705113003079

[7] M. E. Cintra, C. H. de Arruda, and M. C. Monard, "Fuzzy feature subset selection using the wang & mendel method," in *2008 Eighth International Conference on Hybrid Intelligent Systems*, 2008, pp. 590–595.

[8] X. Yang, J. Yuan, J. Yuan, and H. Mao, "An improved wm method based on pso for electric load forecasting," *Expert Systems with Applications*, vol. 37, no. 12, pp. 8036–8041, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417410004963

[9] M. Monfared, M. Fazeli, R. Lewis, and J. Searle, "Fuzzy predictor with additive learning for very short-term pv power generation," *IEEE Access*, vol. 7, pp. 91 183–91 192, 2019.

[10] P.-C. Chang, J.-C. Hieh, and T. Liao, "Evolving fuzzy rules for due-date assignment problem in semiconductor manufacturing factory," *Journal of Intelligent Manufacturing*, vol. 16, pp. 549–557, 10 2005.

[11] D. Kim, "Improving the fuzzy system performance by fuzzy system ensemble," *Fuzzy Sets and Systems*, vol. 98, no. 1, pp. 43–56, 1998. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165011496003569

[12] L.-X. Wang, "The wm method completed: a flexible fuzzy system approach to data mining," *IEEE Transactions on Fuzzy Systems*, vol. 11, no. 6, pp. 768–782, 2003.

[13] J. Gou, F. Hou, W. Chen, C. Wang, and W. Luo, "Improving wang-mendel method performance in fuzzy rules generation using the fuzzy c-means clustering algorithm," *Neurocomputing*, vol. 151, pp. 1293–1304, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231214014738

[14] Y. Jin, W. Cao, M. Wu, and Y. Yuan, "Accurate fuzzy predictive models through complexity reduction based on decision of needed fuzzy rules," *Neurocomputing*, vol. 323, pp. 344–351, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092523121831186X

[15] Y. Zhai, Z. Lv, J. Zhao, W. Wang, and H. Leung, "Data-driven inference modeling based on an on-line wang-mendel fuzzy approach," *Information Sciences*, vol. 551, pp. 113–127, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025520310082

[16] Y. Wang, H. Liu, W. Jia, S. Guan, X. Liu, and X. Duan, "Deep fuzzy rule-based classification system with improved wang–mendel method," *IEEE Transactions on Fuzzy Systems*, vol. 30, no. 8, pp. 2957–2970, 2022.

[17] D. Alvarez-Estevez and V. Moret-Bonillo, "Revisiting the wang–mendel algorithm for fuzzy classification," *Expert Systems*, vol. 35, no. 4, p. e12268, 2018.

[18] L.-X. Wang, "Fast training algorithms for deep convolutional fuzzy systems with application to stock index prediction," *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 7, pp. 1301–1314, 2020.

[19] H. Wang and J. Yao, "An improved deep convolutional fuzzy system for classification problems," in *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2020, pp. 1–6.

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

[21] S. del Río, V. López, J. M. Benítez, and F. Herrera, "A mapreduce approach to address big data classification problems based on the fusion of linguistic fuzzy rules," *International Journal of Computational Intelligence Systems*, vol. 8, no. 3, pp. 422–437, 2015.

[22] A. Fernández, S. del Río, A. Bawakid, and F. Herrera, "Fuzzy rule based classification systems for big data with mapreduce: granularity analysis," *Advances in Data Analysis and Classification*, vol. 11, no. 4, pp. 711–730, Dec 2017.

[23] M. Elkano, M. Galar, J. Sanz, and H. Bustince, "Chi-bd: A fuzzy rule-based classification system for big data classification problems," *Fuzzy Sets and Systems*, vol. 348, no. 1, pp. 75–101, 2018.

[24] F. Aghaeipoor, M. M. Javidi, I. Triguero, and A. Fernández, "Chi-bd-drf: Design of scalable fuzzy classifiers for big data via a dynamic rule filtering approach," in *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2020, pp. 1–7.

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

[26] ——, "Mapreduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[27] D. Chen, W. Tong, Y. Huang, L. Li, and J. Zhang, "Flowfs: Fast learning-algorithm with optimal weights for fuzzy systems," *International Journal of Fuzzy Systems*, vol. 24, 07 2022.

[28] H. Ishibuchi and T. Yamamoto, "Rule weight specification in fuzzy rule-based classification systems," *IEEE Transactions on Fuzzy Systems*, vol. 13, no. 4, pp. 428–435, Aug 2005.

[29] K. Bache and M. Lichman, "Uci machine learning repository," 2013.

[30] L. Jara, R. Ariza-Valderrama, J. Fernádez-Olivares, A. González, and R. Pérez, "Efficient inference models for classification problems with a high number of fuzzy rules," *Applied Soft Computing*, vol. 115, p. 108164, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494621010231

[31] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, pp. 80–83, 1945.