International Journal of Computational Intelligence Systems, Vol. 7, No. 6 (December 2014), 1194-1221

Overview of the SLAVE learning algorithm: A review of its evolution and prospects

David García, Antonio González, Raúl Pérez

Departamento de Ciencias de la Computación e Inteligencia Artificial, University of Granada, 18071-Granada, Spain

E-mail: {A.Gonzalez,Raul_Perez,Dgarcia}@decsai.ugr.es

Received 12 November 2013

Accepted 27 August 2014

Abstract

Inductive learning has been—and still is—one of the most important methods that can be applied in classification problems. Knowledge is usually represented using rules that establish relationships between the problem variables. SLAVE (Structural Learning Algorithm in a Vague Environment) was one of the first fuzzy-rule learning algorithms, and since its first implementation in 1994 it has been frequently used to benchmark new algorithms. Over time, the algorithm has undergone several modifications, and identifying the different versions developed is not an easy task. In this work we present a study of the evolution of the SLAVE algorithm from 1996 to date, marking the most important landmarks as definitive versions. In order to add these final versions to the KEEL platform, Java implementations have been developed. Finally, we describe the parameters used and the results obtained in the experimental study.

Keywords: Classification problems, Feature selection, Fuzzy rules, Genetic algorithms

1. Introduction

In the field of knowledge acquisition through automatic learning, one of the main methods used is the inductive learning, which tries to extract the most important information of a system from a set of variables[1,2]. The extracted knowledge is usually represented using rules that establish the relationship between the input variables. For some problems, the information can be handled in a more flexible way through the discretization of the domains associated to these input variables using linguistic labels. In these cases, the use of fuzzy rules to represent knowledge has proven to be a good solution.

SLAVE (Structural Learning Algorithm in a Vague Environment) was first proposed in 1994[3] and later developed in 1996[4] with the goal of extracting a set a fuzzy rules to represent a problem. At that time there were not many algorithms for fuzzy rule learning available. The only relevant proposals were those of Wang and Mendel[5] and Jang[6], published two years and one year earlier respectively. Both algorithms were focused mainly on control rule learning (regression problems), where fuzzy modeling had proven to be useful, so they were not designed specifically for classification problems. Although these proposals used different methodologies, they had in common the Mamdani rule model. The main drawback of these proposals was the limitation in the number of variables they could handle. When working with more than five variables, the response time becomes unmanageable.

Probably the origin of SLAVE lies to a large extent on the weaknesses of these proposals and the

lack of a fuzzy rule learning proposal for classification problems. When SLAVE was developed, the main objective was to build a fuzzy rule learning algorithm targeting classification problems, featuring a performance similar to that provided by the classical learning algorithms, and able to manage problems involving many variables, missing data, etc.

Thus, SLAVE was based on an efficient classical learning technique called sequential covering strategy[7]. This strategy reduces the problem of learning a disjunctive set of rules to a sequence of simpler problems, each requiring the learning of a single conjunctive rule. Each rule is searched using a genetic algorithm. Therefore, SLAVE uses the Iterative Rule Learning Approach[8]. Moreover, SLAVE uses the DNF (Disjunctive Normal Form) rule model[1]. This rule model improves the interpretability and simplicity of the knowledge obtained, and allows eliminating irrelevant variables in a simple way. Finally, in order to extract the best rule representing the set of examples, an extension of the classical conditions of completeness and consistency[1] was proposed.

Since its development, the original implementation of the algorithm underwent frequent changes, and the resulting algorithms were not considered new versions. After some years, it was hard to tell where was the end of one version and the beginning of the next one. So, the first and the main objective of this study is to describe the evolution of SLAVE between years 1994 and 2001[*], setting the main landmarks that define the development of the algorithm. This study also considers the latest version of SLAVE, called NSLV[10] (New SLaVe).

As SLAVE is frequently used in several comparisons[†] and referenced in other works (e.g. [12,13,14,15,16,17,18,19]), we wanted to make the different versions publicly available, a task that required translating the original code of each recognized version and the code of NSLV from C++ to Java, and also adding the resulting implementations to the KEEL platform[20].

During this historical review process we were able to identify three major versions of SLAVE, which will be called **SLAVE**, **SLAVE2**, and **NSLV** in this paper. The following sections describe these algorithms and their differences. Section five contains the experimental results obtained using the algorithms mentioned above. Section six discusses some steps we are developing for the SLAVE methodology. Finally, Section seven presents the conclusions drawn from the results described in this study.

2. SLAVE

SLAVE is a fuzzy rule learning algorithm based on the use of a sequential covering strategy[7]. A prototypical description of this family of algorithms is shown below:

SEQUENTIAL-COVERING (Y,X,E,Learned-rules)

- Learned-rules ← {}
- Rule ← LEARN-ONE-RULE (Y, X, E)
- while PERFORMANCE (Rule,E) $>$ 0, do
  - Learned-rules ← Learned-rules + Rule
  - E ← E-examples correctly classified by Rule
  - Rule ← LEARN-ONE-RULE (Y, X, E)
- Learned-rules ← sort Learned-rules according to PERFORMANCE (Rule,E) over Examples
- return Learned-rules

where Y is the target attribute, X is the attribute set, E is the set of examples and *Learned-rules* is the output of the procedure containing the final set of rules. PERFORMANCE is a procedure that measures the contribution caused by the inclusion of the last rule in the rule base. It measures the increase in the degree of completeness that causes the last learned rule over the set of examples E.

To describe the implementation of this strategy in SLAVE, we must first define the type of rule and the process used to implement the LEARN-ONE-RULE procedure, together with other details that we mention below.

[*] In year 2001, a major modification of the learning algorithm [9] was published.
[†] For example, paper [11], with 234 citations in Google Scholar, 156 in Scopus, and 124 in Web of Knowledge so far.

2.1. The DNF rule model

A very well known extension of a simple rule is the DNF rule, in which each input variable takes as possible values a set of linguistic terms whose members are joined by a disjunctive operator. On the other hand, the output variable uses a common linguistic variable with a single associated value. SLAVE employs a fuzzy extension of the DNF rule model:

IF X_1 is A_1 and X_2 is A_2 and $\ldots$ and X_n is A_n

THEN Y is B with **weight w**

where $X_1, \ldots, X_n$ are the attributes, $A = (A_1, \ldots, A_n)$ are the values taken for each attribute, each A_i is a subset of D_i, the fuzzy domain of X_i, Y is the consequent variable and B is the value of the consequent variable. We denote this rule as $\boldsymbol{R_B(A)}$. Finally, w is a measure of the weight associated to the rule.

This rule model has been used in SLAVE to learn the structure of a rule, since when a variable takes all possible fuzzy values of its domain, it is not relevant for the consequent and therefore can be removed from the rule. Moreover, the DNF fuzzy rule model allows compacting the set of rules and makes it more interpretable.

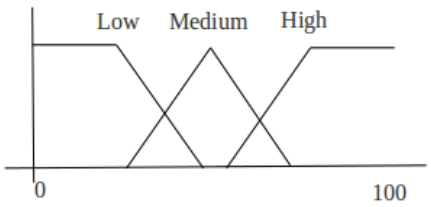

Fig. 1. Fuzzy domains for variables X_1 and X_2.

A specific example of a DNF fuzzy rule is

IF X_1 is {Low or Medium or High}

and X_2 is {Medium or High}

THEN Y is 2 with **weight w**

where the fuzzy domains of X_1 and X_2 are described in Figure 1, and Y takes its values in a discrete domain with class in the set {1, 2, 3}. Since X_1 takes all the possible values of its domain, the previous rule is equivalent to:

IF X_2 is {Medium or High}

THEN Y is 2 with **weight w**

Moreover, the label {Medium or High} can be interpreted as the convex hull of both labels [4,21].

2.2. The LEARN-ONE-RULE function

SLAVE uses a genetic algorithm (**GA**) to implement the LEARN-ONE-RULE function. The input of this **GA** is a target attribute, representing the consequent variable, the complete set of antecedent variables and the set of examples, and the output is a single rule. In SLAVE, function LEARN-ONE-RULE is called following a particular order. It sets a specific value of the consequent class and this procedure iterates until all the rules needed for describing this class are obtained; then another class is selected and the process is repeated.

One of the main components of the LEARN-ONE-RULE is the criterion to extract a single rule. The main idea is to extract the rule that offers the best representation of the set of examples. The best rule criterion is related to an extension of the classical conditions of completeness and consistency. In order to propose a fuzzy version of these concepts we first need a way to decide whether an example is positive or negative for a rule, and a way to count the number of examples in both cases.

2.2.1. Positive and negative examples

The way to calculate the number of positive and negative examples for a rule $R_B(A)$ in SLAVE changed over the years. The first approximation for calculating the number of positive and negative examples[3] was based on a possibility measure. Thus, let a and b be two fuzzy sets in a common referential set U, and $*$ a t-norm. The compatibility between a and b was defined by the function:

$$\sigma(a,b) = sup_{x \in U}\{\mu_a(x) * \mu_b(x)\} \qquad (1)$$

where $\mu_r(s)$ is the degree of membership of value s to the fuzzy set r.

In order to calculate the compatibility between two sets of fuzzy sets, let us consider Dom_1 and Dom_2, two domains consisting of fuzzy sets in a common referential set U, and $C_1 \subseteq Dom_1$ and $C_2 \subseteq Dom_2$, two sets of fuzzy sets. In this case, the compatibility between both sets follows this formula:

$$\sigma(C_1,C_2) = sup_{a\in C_1} sup_{b\in C_2} \sigma(a,b). \quad (2)$$

Considering these expressions it was possible to define the following measure of possibility:

$$Poss(A_i|e_i) = \frac{\sigma(e_i,A_i)}{\sigma(e_i,D_i)} \quad (3)$$

representing the adaptation between the i-th component of the example and the fuzzy set A_i, where $E = (e_1,e_2,\ldots,e_n)$ is an example, $A_i \subseteq D_i$ is the value of variable X_i, and D_i is the domain of this variable. This concept is critical to the interpretation of two adjacent fuzzy sets as the convex hull of both.

From these definitions we could obtain two adaptation concepts, one for the antecedent part and another for the consequent part:

- Adaptation between the example and the antecedent of $R_B(A)$:

$$U(e,A) = *_{i=1\ldots n} Poss(A_i|e_i). \quad (4)$$

- Adaptation between the example and the consequent of $R_B(A)$:

$$U(e,B) = \frac{\sigma(class(e),B)}{\sigma(class(e),F)} \quad (5)$$

where e is an example, $class(e)$ represents the consequent of the example, and * is a t-norm.

Using the previous concepts we can define the set of positive and negative examples for rule $R_B(A)$:

$$E^+(R_B(A)) = \{(e,U(e,A)*U(e,B))|e \in E\} \quad (6)$$

$$E^-(R_B(A)) = \{(e,U(e,A)*U(e,\overline{B}))|e \in E\}. \quad (7)$$

where $\overline{B}$ is the set of all the fuzzy values of D_i, except B. Finally, the number of positive examples and the number of negative examples for fuzzy rule $R_B(A)$ are:

$$n_E^+(R_B(A)) = |E^+(R_B(A))| \quad (8)$$

and

$$n_E^-(R_B(A)) = |E^-(R_B(A))| \quad (9)$$

respectively, where $|.|$ is the cardinality of a fuzzy subset.

2.2.2. *Completeness and consistency*

Completeness and consistency are two conditions typically used to extract rules. The completeness condition states that every example of some class must satisfy some rule from this class. On the other hand, the consistency condition states that if an example satisfies a description of some class, then it cannot be a member of a training set of any other class. Both conditions provide the logical foundation of algorithms for concept learning from examples.

From the classical definitions of completeness and consistency[1], the following two extensions were defined in[21]. Given an example set E, the degree of completeness of a rule $R_B(A)$ is:

$$\Lambda(R_B(A),E) = \frac{n_E^+(R_B(A))}{n_{E_B}} \quad (10)$$

where

$$n_{E_B} = \sum_{e\in E} U(e,B) \quad (11)$$

is the number of examples of class B in the training set.

In relation to consistency, we propose a soft extension of this measure by allowing some noise in the rules. Thus, to define the soft consistency degree we use the following set:

$$\triangle_E^k = \{R_B(A) | n_E^-(R_B(A)) < k n_E^+(R_B(A))\} \quad (12)$$

with

$$\triangle_E^0 = \{R_B(A) | n_E^-(R_B(A)) = 0\} \quad (13)$$

which represents the set of rules having a number of negative examples strictly less than a percentage (that depends on k) of the positive examples, and being

$$\triangle = P(D_1) \times P(D_2) \times \ldots \times P(D_n) \times F \quad (14)$$

where $P(\chi)$ denotes the set of subsets of χ, D_i is the domain of antecedent variables, and F is the domain of the consequent variable.

Thus, the degree to which a rule $R = R_B(A)$ satisfies the soft consistency condition is:

$$\Gamma_{k_1 k_2}(R,E) = \begin{cases} 1 & \text{if } R \in \triangle_E^{k_1} \\ \frac{k_2 n_E^+(R) - n_E^-(R)}{n_E^+(R)(k_2 - k_1)} & \text{if } R \in (\triangle_E^{k_2} - \triangle_E^{k_1}) \\ 0 & \text{otherwise} \end{cases} \quad (15)$$

where $k_1, k_2 \in [0,1]$ and $k_1 < k_2$, and $n_E^-(R)$, $n_E^+(R)$ are the number of positive and negative examples for rule R, respectively.

This definition uses two parameters: k_1 is the lower bound of the noise threshold, and k_2 is the upper bound. The above formula gives degree 1 to rules in $\triangle_E^{k_1}$, that is, rules having an admissible number of negative examples (measured as a percentage in k_1 of the number of positive examples). It gives degree 0 to rules out of $\triangle_E^{k_2}$, that is, rules having an excessive number of negative examples (measured as a percentage in k_2 of the number of positive examples). If $k_1 < k_2$ then $\triangle_E^{k_1} \subseteq \triangle_E^{k_2}$, so a linear variation is assigned to rules between both extremes.

After an experimental study, and from the publication of [11], the values for k_1 and k_2 were set to $k_1 = 0$ and $k_2 = 1$. In this case, the above expression is equivalent to:

$$\Gamma_{0,1}(R_B(A),E) = \frac{n_E^+(R_B(A)) - n_E^-(R_B(A))}{n_E^+(R_B(A))}. \quad (16)$$

The criterion to select the best rule given a set of examples in SLAVE is finally

$$\Lambda(R_B(A),E) \times \Gamma_{0,1}(R_B(A),E) = \\ = \frac{n_E^+(R_B(A)) - n_E^-(R_B(A))}{n_{E_B}}. \quad (17)$$

2.2.3. λ-covering concept

The adaptation between an example and a rule should not be a crisp concept when working with fuzzy information. In this way, an example has a degree of adaptation to a fuzzy rule. An important task is to know the minimum degree of adaptation that should exist between an example and a rule for considering that the example satisfies the rule. In order to solve this problem, SLAVE uses a parameter called $\lambda \in [0,1]$. The meaning of λ is exactly the minimum adaptation required to consider that the example is covered by the rule. λ is an input parameter of the learning algorithm, and it plays an important role in the design of the algorithm. Low values of λ imply a low requirement for adaptation between examples and rules, and this fact has two consequences: The first one is that the system searches a small number of very general rules, and the second one is that there is probably too much overlap among rules competing for classifying, which often leads to less predictability. On the other hand, values of λ very close to 1 tend to force a very "crisp" rule behavior in relation to examples, which results in a greater number of more specific rules.

The concept of λ-covering is used for the removal of examples correctly classified in the sequential covering algorithm and in the termination condition.

From the work published in [3] and after an experimental study, the value of this parameter was set to $\lambda = 0.8$.

2.3. The genetic algorithm used in SLAVE

SLAVE, using the above criterion for selecting the best rule, employs a genetic algorithm to implement the LEARN-ONE-RULE function. Next we will describe the main components that define the behavior of the genetic algorithm.

2.3.1. Representation of a population element

Related to the genetic representation, SLAVE uses a binary codification. If the database has n antecedent variables

$$X_1, \ldots, X_n$$

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

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

and it is encoded as a vector of $m_1 + \ldots + m_n$ zero-one components. The value of each component $(m_1 + \ldots + m_{r-1} + s)$ is 1 if the s-th element in domain D_r is a value of variable X_r, and 0 otherwise.

In order to better understand the representation, let us consider the following example:

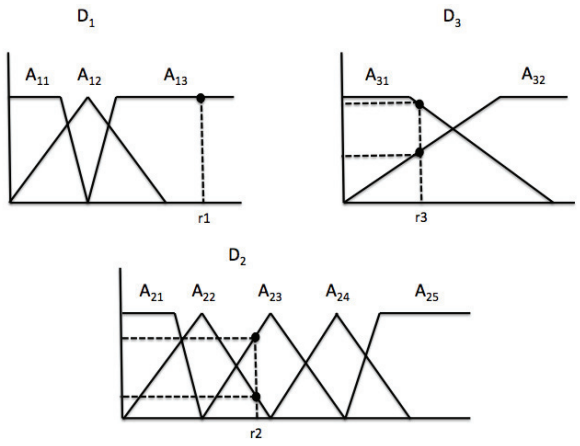

Fig. 2. Domains of variables X_1, X_2 and X_3.

Example 1 *Let us assume that we have three variables X_1, X_2, and X_3, such that their associated fuzzy domains are*

$$D_1 = \{A_{11}, A_{12}, A_{13}\}$$

$$D_2 = \{A_{21}, A_{22}, A_{23}, A_{24}, A_{25}\}$$

$$D_3 = \{A_{31}, A_{32}\}.$$

These domains are represented in Figure 2. In this case, a vector 1100010111 represents the following antecedent:

X_1 is $\{A_{11}, A_{12}\}$, X_2 is $\{A_{23}, A_{25}\}$ and X_3 is $\{A_{31}, A_{32}\}$.

Since X_3 takes all possible values from domain D_3, the antecedent is equivalent to:

X_1 is $\{A_{11}, A_{12}\}$ and X_2 is $\{A_{23}, A_{25}\}$.

2.3.2. Generation of the initial population

The generation of the initial population is an important aspect in the process of getting antecedents with a high possibility of guiding the search process toward good solutions. The procedure used in SLAVE consists in randomly taking a subset of examples among those with the current consequent that have not been eliminated yet. For each of these examples, the most specific antecedent with the highest adaptation to the example is selected.

The procedure for selecting the initial population is similar to the one used in AQ algorithms[22], as the generation of the initial antecedent for a class uses examples of this class in the training set as a starting point, and the genetic process can be considered as a generalization process over the chosen antecedents.

Example 2 *Let X_1, X_2, and X_3 be variables with the associated domain shown in Figure 2, and let (r1, r2, r3) be the randomly selected example from the training set for a class. The most specific antecedent that best represents it would be:*

X_1 is A_{13} and X_2 is A_{23} and X_3 is A_{31}

with the binary representation 0010010010.

2.3.3. *Evaluation function*

The main purpose of the function LEARN-ONE-RULE is to find the best rule verifying the completeness and consistency conditions to the highest possible degree, and for this task we use a genetic algorithm. According to that, given a rule R and a set of training examples E, we define the fitness function of the genetic algorithm in the following way:

$$fitness(R,E) = \Lambda(R,E) \times \Gamma_{0,1}(R,E) \qquad (18)$$

where $\Gamma_{0,1}(R,E)$ is the degree to which the rule R satisfies the soft consistency condition and $\Lambda(R,E)$ is its degree of completeness, previously defined.

The combination of these two values provides rules that are simultaneously complete and consistent to the highest degree. Furthermore, we use the product t-norm, as the combination through this operator is very interactive compared to other proposals.

2.3.4. *Genetic operators*

The following genetic operators are used in SLAVE to generate new populations:

- **Selection operator**
  This is a selection model that sorts the elements in the population using its fitness valuation and assigns a probability selection to each position in the population.
- **Crossover operator over two points**
  This type of crossover establishes two cutoff points between two elements in the population and exchanges the central segment, like the one shown in Figure 3.

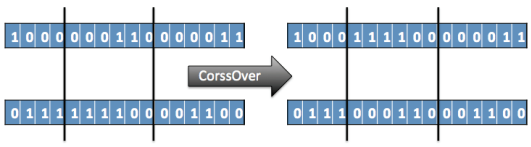

Fig. 3. Crossover operator over two points.

- **Mutation operator**
  In the genetic algorithm used in SLAVE, this operator changes one gene of an element in the population with a certain probability.
- **Generalization operator**
  This operator tries to clarify the rules returned by the learning algorithm and make them more understandable, and only acts over those variables that have an associated ordered domain. We say that an antecedent variable with an associated ordered domain is stable if there is a single continuous sequence of 1-values in the binary representation of its value. We say that a variable is unstable if there are several continuous sequences of 1-values in the binary representation of its value. Taking all this into account, the generalization operator tries to obtain stable variables by removing their unstable regions. The behavior is shown below (Figure 5).

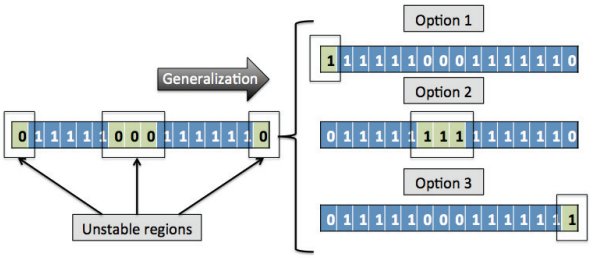

Fig. 4. Generalization operator.

2.3.5. *Termination condition*

The implementation of this condition is necessary to distinguish between a class that has at least one rule and a class that does not. The idea is to make a more exhaustive search when we want to find the first rule of a class, and relax this search process when we already have some rules for a class.

The genetic algorithm returns the best rule for the last population if one of the following conditions is verified:

- The number of iterations is greater than a fixed limit.
- The fitness function of the best rule in the population does not increase the value during at least a fixed number of iterations and there are some other rules for the class we are working with.
- No rules with this value of the consequent have been obtained before, but the fitness function does not increase the value during a fixed number of iterations and the current best rule λ-covers at least one example from the training set.

2.4. Other components of SLAVE

2.4.1. The role of PERFORMANCE

The sequential covering previously described uses a *PERFORMANCE(Rule,E)* function. In SLAVE this function is related to the concept of completeness. Specifically, when a new rule is added to the set of learned rules, the completeness of the set increases. When there is a non-zero increase, the new rule is considered useful and relevant. The rule is added to the set of learned rules and the algorithm starts a new iteration.

2.4.2. Removing examples

Another important aspect of the sequential covering is the removal of examples correctly classified by the learned rule. In each iteration, SLAVE removes the examples that were λ-covered by the last learned rule (i.e., the examples considered as sufficiently well classified).

2.4.3. Inference Model

SLAVE uses a typical fuzzy inference mechanism for classification: The winner rule. This model has a simple description. Let $Rules = \{R_{B_1}(A_1), \ldots, R_{B_q}(A_q)\}$ be the set of rules and e an example. The inference engine assigns to the examples the class B_j related to the associated rule $R_{B_j}(A_j)$ verifying

$$U(e,A_j) \geqslant \max_{0 \leqslant i \leqslant q} \{U(e,A_i)\}. \quad (19)$$

When working with rules, it may occur that two or more rules describing different concepts can be applied to the same input system. In this case, it is necessary to establish a way to decide which of the possible rules should be applied.

The conflict resolution mechanism used in this version is based on rules confidence. This confidence degree is calculated taking into account the behavior of each rule over the training set, and it is related to their prediction capability. Thus, we define the weight of a rule as a value in [0, 1] that measures the relation between the examples correctly represented over the training set and the total set of examples covered.

Let E be a set of examples and $R_B(A)$ a rule. We can define the weight of rule $R_B(A)$ as:

$$\omega_E(R_B(A)) = \frac{n_E^+(R_B(A)) + 1}{n_E^+(R_B(A)) + n_E^-(R_B(A)) + 1}. \quad (20)$$

Therefore, the two criteria taken into account to break the tie between two rules are (in the order of appearance):

- The rule with higher weight.
- The rule that was learned first.

2.5. Main advantages and disadvantages of SLAVE

The first advantage of SLAVE is that it works with DNF rules, which are widespread in the field of classical machine learning (since they have a greater ability to represent information than the typical rules used in the soft computing field), and keeps the interpretability of the linguistic rules.

In the area of fuzzy sets, SLAVE also incorporates feature selection, an important concept that was already used in classical learning. In particular, SLAVE defines an embedded feature selection, a very useful characteristic for tackling problems with high dimensionality in the number of attributes.

Finally, SLAVE was defined assuming that the rule learning can be performed independently for each class. This independence allows us to implement SLAVE in a parallel way for each class of the consequence variable. Obviously, this results in an improved response time of the algorithm when working with multi-processor computers.

The two main disadvantages of the method are:

- While it addresses the issue of the partial relevance of input attributes, the mechanism to detect the irrelevance of an attribute is relatively weak for two reasons. The first one is that the fitness function does not encourage the detection of irrelevant variables, and the second one is that the mechanism to eliminate irrelevant variables is slow and requires increasing the cycles of the genetic algorithm to converge towards a solution.
- The dependence on the λ parameter. This parameter is essential for the algorithm, since it is related to both the number of rules needed to describe a concept and the classification ability of the knowledge acquired. Additionally, this parameter is not easy to estimate a priori, since it depends on the distribution of the examples in the training set.

3. SLAVE2

SLAVE2[11,23,24] was developed with the objective of improving the behavior of SLAVE when working with high dimensional data and also with the idea of improving the interpretability of the rules obtained during the learning process. To achieve this, it introduces three fundamental changes with respect to SLAVE:

1. A new criterion to penalize examples.
2. A new codification of the individuals in the population.
3. A new evaluation function.

These changes are aimed to correct the problems identified in SLAVE. The first one reduces the dependence on the λ-covering parameter. The second change will allow us to establish a more efficient mechanism to eliminate irrelevant variables and thus to be able to work with databases with a large number of variables. Finally, the third change is motivated by the idea of giving priority to the most simple and interpretable rules.

3.1. First change: A new criterion to penalize examples

SLAVE was not able to appropriately consider the interaction between the rule that was just being learned and the rules already learned. The cause of this problem is that the degree of interaction among rules of different classes is determined by the value of the parameter λ previously described. Let us remember that the value of parameter λ is fixed throughout the process.

The main idea implemented in SLAVE2 to fix this problem was to provide some flexibility in the use of the λ-covering parameter. Thus, to consider that an example is covered by a rule not only requires having an adaptation value for the rule higher than parameter λ (SLAVE), it also requires that the adaptation value of a rule with the correct class exceeds the adaptation value for rules of other classes (SLAVE2).

So, the key idea is that now an example has a certain degree of positiveness (negativeness) only if that degree is enough to correctly (incorrectly) classify the example. In this sense, it is necessary to modify the criterion for penalizing examples: On each stage of the sequential covering strategy, once the first class is learned, the penalization criterion is adapted for each particular example.

The formal description of this penalization model[11] uses the following values. Let $\lambda^{+}_{Rules}(e)$ be the best adaptation between example e and the learned rules that have the same class as the example, and $\lambda^{-}_{Rules}(e)$ the best adaptation between example e and the learned rules that have a different class:

$$\lambda^{+}_{Rules}(e) = max\{U(e,A) * U(e,B) | \forall R_B(A) \in Rules \text{ and } Class(e) = B\} \quad (21)$$

$$\lambda^{-}_{Rules}(e) = max\{U(e,A) * U(e,B) | \forall R_B(A) \in Rules \text{ and } Class(e) \neq B\}. \quad (22)$$

From these expressions, we can write new definitions for the number of positive and negative examples of a rule $R_B(A)$:

$$n^{+}_{E}(R_B(A)) = |\{(e, U(e,A) * U(e,B))\}| \text{ such that } \lambda^{-}_{Rules}(e) > \lambda^{+}_{Rules}(e) \text{ and } U(e,A) * U(e,\overline{B}) > \lambda^{+}_{rules}(e) \quad (23)$$

$$n^{-}_{E}(R_B(A)) = |\{(e, U(e,A) * U(e,\overline{B}))\}| \text{ such that } \lambda^{+}_{Rules}(e) > \lambda^{-}_{rules}(e) \text{ and } U(e,A) * U(e,\overline{B}) > \lambda^{+}_{rules}(e) \quad (24)$$

where $Rules$ is a set of rules previously learned, and $U(e,A)$, $U(e,B)$ are the adaptation values between example e and the antecedent A or the consequent B respectively, which were previously defined.

So, an example e of the training set E is positive for a new rule $R_B(A)$ if e was incorrectly classified before selecting $R_B(A)$ and it can be correctly classified using this rule. In the same sense, an example is negative for a rule if the example was correctly classified and the inclusion of the rule misclassifies it.

Now, during the learning process of a certain class B, the examples removed from the training set E are those $e \in E$ that satisfy

$$Class(e) = B \text{ and } \lambda^{+}_{Rules}(e) \geqslant \lambda \text{ and } \lambda^{+}_{Rules}(e) > \lambda^{-}_{Rules}(e) \quad (25)$$

That is, those examples that are correctly classified for class B and their best match for the rules of their class is equal to, or greater than, λ.

3.2. *Second change: A new rule codification for the genetic algorithm*

The genetic algorithm used in SLAVE can obviously eliminate irrelevant variables. However, the genetic representation of the information used by SLAVE makes more difficult to eliminate a variable than to add it, and the system might take a long time to find a good solution. Consequently, if we want to improve the detection of irrelevant variables, we first need to change the genetic representation. So, our goal consists in obtaining a better representation of the genetic solutions to make a feature selection for each possible rule. The idea is to include information associated to each antecedent variable in order to discover whether the variable should be considered as part of the antecedent of the rule or not.

In this way, the second change of SLAVE was a new codification of rules[23]. This new codification implemented in SLAVE2 tries to improve the capacity for detecting irrelevant attributes. The idea is to expand the representation with a new level to codify the variables in the rule. Now each individual consists of two substructures or levels, one codifying the values and the other one codifying the variables.

Therefore, a genetic algorithm with two levels maintains two different representations: One for determining the subset of relevant variables associated to a particular class (the variable level) and another one (the value level) to find the best variable-value assignation for that class. On each representation, a process of genetic co-evolution is applied, where each level has its own genetic operators, but the goodness of the solution is calculated by considering the collaboration between both levels of the individual.

Thus, the variable level codifies the variables from the initial set that are considered to be relevant for inclusion in the rule. This information is modified during the evolutionary process. The value level codifies the particular values not only for the variables considered relevant, but also for those considered irrelevant, being the latter not considered when calculating the goodness of the rules.

The separation of these learning tasks (variable

learning and value learning) has proven to be very efficient for problems involving a large number of variables. When the genetic algorithm obtains the best description for a concept, it can select a subset of variables and start the search process using this information. Moreover, during the evolutionary process, the subsets of selected variables can be changed with the inclusion of new variables or the removal of some of the variables that were initially selected.

The genetic algorithm has a single population, a single selection criterion and a single fitness function, but it works in a different way for each component of the chromosome through different genetic operators.

The next section briefly describes the main components of the genetic algorithm.

3.2.1. Representation of a population element

The representation of an individual in the genetic population is encoded using two structures (see Figure 6): One codifies the relevance of the variables and the other codifies the variable-value assignments. With this decomposition, the GA representation has a complex chromosome composed of two structures: A variable chromosome and a value chromosome. This division allows us to clearly distinguish between the two different tasks that are simultaneously carried out by the genetic algorithm (search for the appropriate variables and search for the appropriate value assignment), and we can associate and apply the most appropriate set of genetic operators on each structure.

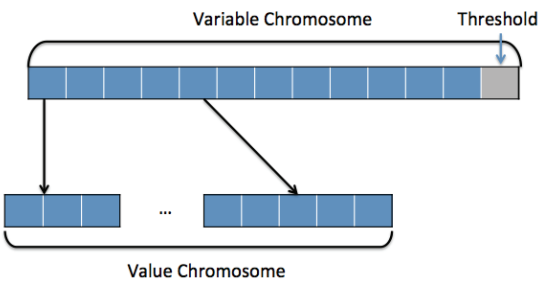

Fig. 5. Representation of a population individual in SLAVE2.

Let us suppose there are n possible antecedent variables $X_1,\ldots,X_n$, each with an associated fuzzy domain D_i containing m_i components. In order to find the best rule, SLAVE2 fixes a class and then searches for the best antecedent for this class. Therefore, the genetic code must contain information about the relevant variables of the rule and also information about the values of these variables:

- A variable chromosome.
- A value chromosome.

The variable chromosome codifies the relevant/irrelevant variables for the particular rule. It uses a real code with $n+1$ components, in which the i-th element of the j-th chromosome $\tau_C(X_i^j)$ $(i=1,\ldots,n)$ contains a real value between 0 and 1, representing the relevance degree of the i-th variable with respect to class C, that is, a number indicating the possibility of being a member of the relevant variable set for a rule. The $n+1$ value, named T_j, is a real value between 0 and 1 representing an activation threshold associated to the j-th chromosome. A variable X_i will be considered as a component of the rule antecedent for a particular class if $\tau_C(X_i^j) \geqslant T_j$. Otherwise, the variable will be considered irrelevant for the rule. The next subsection explains how to obtain these values for the first population. The genetic algorithm will change these initial values in order to obtain a better estimation of them.

The value chromosome codifies any elements of $P(D_1)\times\ldots\times P(D_n)$ and it is exactly the same one used in SLAVE.

Example 3 *Let us suppose that we have three variables, X_1, X_2, and X_3; the fuzzy domain associated with each one is shown in Figure 2. In this case, $m_1=3$, $m_2=5$, and $m_3=2$. Let us consider that the relevance degrees for class C of a population individual are:*

$$\tau_C(X_1)=0.5,\quad \tau_C(X_2)=0.7,\quad \tau_C(X_3)=0.1$$

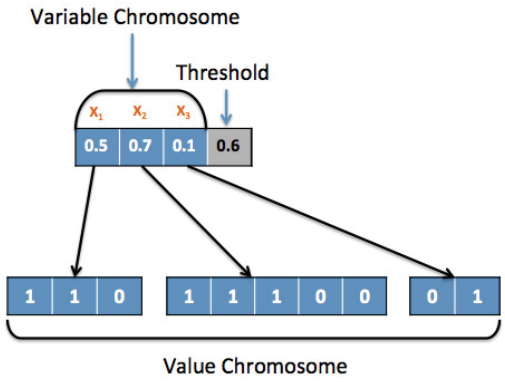

Fig. 6. Combination variable-value for an individual.

If the combination variable-value for both chromosomes is defined by Figure 7 and we take the value 0.6 as the threshold, the antecedent representation would be:

$$X_2 \text{ is } \{A_{21}, A_{22}, A_{23}\}$$

since

- *(110) represents* $\{A_{11}, A_{12}\}$*, but* X_1 *is not included in the antecedent since it is not activated in the variable chromosome* $(0.5 \leqslant 0.6)$.
- *(11100) represents* $\{A_{21}, A_{22}, A_{23}\}$ *and is included in the antecedent since* X_2 *is activated in the variable chromosome* $(0.7 \geqslant 0.6)$.
- *(01) represents* $\{A_{32}\}$*, but* X_3 *is not included in the antecedent since it is not activated in the variable chromosome* $(0.1 \leqslant 0.6)$.

Obviously, changing the threshold also changes the current description of the antecedent.

3.2.2. *Generation of the initial population*

The initial population is generated following a procedure similar to that used in SLAVE for the chromosome value. Thus, each value chromosome is obtained by randomly selecting examples from the class that must be learned and assigning the most specific antecedent that better covers it. This antecedent is made up of only one label for each antecedent variable and the selected label is the one that gives the highest degree of membership for each component in the example. If we consider the domains and variables given in Figure 2, the generated chromosome would be

$$(001)(00100)(10)$$

and would correspond to the following antecedent:

$$X_1 \text{ is } A_{13} \quad \text{and} \quad X_2 \text{ is } A_{23} \quad \text{and} \quad X_3 \text{ is } A_{31}.$$

On the other hand, the variable chromosome is built up by using an information function τ_C for each variable with respect to the fixed class on the training examples. The value $\tau_C(X)$ is calculated using the following expression:

$$\tau_C(X) = \frac{I(X, Y = C)}{H(X, Y = C)} \tag{26}$$

where the information measure I for variables X, Y is given by the following expression:

$$I(X,Y) = \sum_x \sum_y p(x,y)\ log_2 \left(\frac{p(x,y)}{p(x)p(y)} \right) \tag{27}$$

where x and y are specific values of variables X and Y, C is a fixed class of the consequent variable, and

$$H(X,Y) = \sum_x \sum_y p(x,y)\ log_2 p(x,y) \tag{28}$$

is the Shannon entropy over two variables, where **p** is a probability measure.

The value $\tau_C(X)$ measures the degree of dependence or independence between variable X and the value C of the consequent variable. It can be interpreted as the relevance value of each variable X with respect to class C [23].

When using linguistic variables, it is necessary to define the calculation of this value through the definition of the probability of X when taking a value a_i on its domain $\{a_1, a_2, \ldots, a_s\}$:

$$p(X = a_i) = \frac{1}{m} \sum_{j=1}^{m} \left(\frac{\mu_{a_i}(e_j)}{\sum_{t=1}^{s} \mu_{a_t}(e_j)} \right) \tag{29}$$

where m is the number of examples from the training set E, e_j is an example from E, and μ_w is the membership function of the fuzzy set w.

In this formula it is assumed that all the examples are crisp. The bi-dimensional probability is similar to the previous formula, but requires combining the information on two variables using a *t-norm* (denoted by the symbol *).

$$p(X = a_i, Y = b_j) = \frac{1}{m} \sum_{j=1}^{m} \left(\frac{\mu_{a_i}(e_k) * \mu_{b_j}(e_k)}{\sum_{t,h} \mu_{a_t}(e_k) * \mu_{b_h}(e_k)} \right) \tag{30}$$

where the domain of variable Y is $\{b_1, b_2, \ldots, b_r\}$ and the t-norm is defined by $a * b = min\{a, b\}$.

Moreover, we need to define an initial value for the activation threshold. T_j takes a random value in the following interval:

$$[min_i \ \ \tau_C(X_i^j), \ \ max_i \ \ \tau_C(X_i^j)].$$

Both $\tau_C(X_i^j)$ and T_j are affected by the genetic operators during the evolution of the genetic algorithm. Therefore, the initial relevance degree, calculated using the above mentioned formulas, is modified during the evolution process until it reaches an appropriate value.

3.2.3. *Genetic operators*

As a consequence of the two levels used to codify each individual in the population, one with real codification (variable chromosome) and the other one with binary codification (value chromosome), it was necessary to use different genetic operators for each structure. Taking into account the ones used in SLAVE, the value level inherited them as it uses the same codification. As for the variable level, after some experimental tests the genetic operators that performed better were the uniform mutation, the crossover, and the selection operators. In summary, the genetic operators employed for both levels were:

- Variable level:
  1. Real uniform mutation.
  2. Crossover operator over two points.
- Value level:
  1. Binary uniform mutation.
  2. Crossover operator over two points.
  3. Generalization operator.

It is important to note that we also made some tests using the BLX_α crossover operator at the variable level, but it showed no improvement over the results obtained using the two-point crossover operator.

3.3. *Third change: A new evaluation function based on the simplicity criteria*

As mentioned above, the fitness function in SLAVE combines the completeness and consistency measures using a product operator. This fitness function has proven to be very useful for learning on different kinds of problems[4,25,26]. However, with this heuristic criterion, many different rules can have the same evaluation function value. The SLAVE genetic algorithm randomly selects one of these rules. SLAVE2 includes a new multicriteria method to discriminate among these rules, instead of using random selection. Among the best rules, the algorithm prefers those which are simpler and more understandable.

This new criterion was included to achieve two different goals:

- To improve the comprehension of the acquired knowledge.
- To obtain a set of rules with a higher degree of accuracy over unseen examples.

The following definitions, proposed in [24], will be useful to introduce new concepts in order to formalize these ideas.

Definition 1 *Let $R_B(A)$ be a rule with antecedent $A = (A_1, \ldots, A_n)$ and $A_i \in P(D_i)$. A variable X_i, with $i = 1, \ldots, n$, is considered to be irrelevant in this rule if $A_i = D_i$. The number of irrelevant variables of a rule will be denoted as $i(R_B(A))$.*

This definition is based on the treatment of rules used in SLAVE, where the disjunction of adjacent values is taken as the convex hull of the fuzzy labels [4,21].

This definition allows us to propose the concept of rule simplicity: A rule is simpler than other if it has a lower number of relevant variables. Therefore, the following definition is proposed.

Definition 2 *Let $R_B(A)$ be a rule. The simplicity degree in variables of this rule is:*

$$svar(R_B(A)) = \frac{i(R_B(A))}{n} \tag{31}$$

where n is the number of possible antecedent variables.

The second concept is presented through the following example.

Example 4 *Let X_2 be a variable with an associated ordered domain D_2 (see Figure 2). Let $A = \{A_{23}, A_{24}, A_{25}\}$ and $A' = \{A_{23}, A_{25}\}$ be two possible values for X_2. The first value is equivalent to "X_2 is greater than or equal to A_{23}", using the adaptation concept of SLAVE[21], whereas the second one does not have a similar interpretation. If both values are equally appropriate for describing the value of a variable, it is preferred the first one since it is easier to understand. The tie situation is generated by the lack of examples covered by label A_{24}. The preference of the second antecedent is directly related to the generalization principle applied when there is a lack of information.*

Let us consider the definition of stable value:

Definition 3 *Given a specific value $A_i \in P(D_i)$ for variable X_i, we say that A_i is stable if and only if A_i is composed of a unique sequence of adjacent values of D_i.*

Definition 4 *Let $R_B(A)$ be a rule with $A = (A_1, \ldots, A_n)$ and $A_i \in P(D_i)$. We define $e(R_B(A))$ as the number of X_i variables required to make D_i an ordered domain and to make A_i or $\overline{A}_i$ a stable value.*

The use of the complementary in the previous definition is justified since a unique sequence of adjacent values in the complementary corresponds to a simple description as NOT A, with A being a stable value. By using this concept, we can define the concept of simplicity regarding the values of a rule.

Definition 5 *Let $R_B(A)$ be a rule. We define the simplicity degree in values of a rule as:*

$$sval(R_B(A)) = \frac{1 + e(R_B(A))}{1 + p} \tag{32}$$

where $p \leqslant n$ is the number of variables with an associated ordered domain.

With all these elements, we can define a multi-criteria fitness function (consisting of three components) for rule R[24]:

$$fitness(R,E) = \\ = (\Lambda(R,E) \times \Gamma_{01}(R,E), svar(R), sval(R)). \tag{33}$$

Finally, the selection of the best rule responds to a multi-criteria evaluation function guided by a lexicographical order; that is, the initial criterion (consistency and completeness) is maintained. In case of a tie situation, the simplicity criterion in variables is used; if the tie situation remains, then we appeal to the simplicity criterion in values. Thus, the lexicographical order uses ***(max, max, max)*** as the optimization criteria. That is, to maximize on the first component, then on the second component in case of tie, and finally to maximize on the last component in case of a new tie situation.

Example 5 *Let us suppose we have three variables X_1, X_2, and X_3 with domain D_1, D_2, and D_3 respectively (see Figure 2). Let us consider a fixed consequent B and three possible antecedents:*

$$A = (\{A_{11}, A_{13}\}, \{A_{23}, A_{25}\}, \{A_{31}\})$$

$$A' = (\{A_{11}, A_{12}, A_{13}\}, \{A_{23}, A_{24}, A_{25}\}, \{A_{31}\})$$

$$A'' = (\{A_{11}, A_{12}, A_{13}\}, \{A_{23}, A_{25}\}, \{A_{31}\})$$

*with the same value of **(consistency $\times$ completeness)**.*

In this case, the fitness function uses the previous concepts to decide the best antecedent:

- *Antecedent A corresponds to*

$$X_1 \text{ is } \{A_{11}, A_{13}\} \text{ and } X_2 \text{ is } \{A_{23}, A_{25}\} \text{ and}$$

$$and \quad X_3 \text{ is } \{A_{31}\}$$

with values $svar(R_B(A)) = 0$ *and* $sval(R_B(A) = \frac{3}{4})$, *since* $\{A_{11}, A_{13}\}$ *is equivalent to NOT* (A_{12}).

- *Antecedent* A' *corresponds to*

$$X_1 \text{ is } \{A_{11}, A_{12}, A_{13}\} \quad and \quad X_2 \text{ is } \{A_{23}, A_{24}, A_{25}\} \quad and$$

$$and \quad X_3 \text{ is } \{A_{31}\}$$

with values $svar(R_B(A)) = \frac{1}{3}$ *and* $sval(R_B(A)) = 1$.

- *Antecedent* A'' *corresponds to*

$$X_1 \text{ is } \{A_{11}, A_{12}, A_{13}\} \quad and \quad X_2 \text{ is } \{A_{23}, A_{25}\} \quad and$$

$$and \quad X_3 \text{ is } \{A_{31}\}$$

with values $svar(R_B(A)) = \frac{1}{3}$ *and* $sval(R_B(A)) = \frac{1}{2}$.

Then, the best choice would be antecedent A', *as it complies with:*

$$fitness(A) \leqslant fitness(A'') \leqslant fitness(A').$$

3.4. *Inference model of SLAVE2*

Apart from the previously described changes, SLAVE2 includes a modification in the inference mechanism related to the rule weight. In SLAVE, the weight of the rule is used as a secondary criterion for solving conflicts in order to select the winner rule (see Section 2.4.3). However, SLAVE2 uses the classical inference mechanism of the winner rule, but in contrast with SLAVE, the weight of each rule plays a relevant role in this process. A simple description of this new inference process is shown. Let us consider:

$$Rules = \{R_{B_1}(A_1), \ldots, R_{B_q}(A_q)\}$$

the set of rules, $\Omega = \{\omega_1, \ldots, \omega_q\}$ the weight associated to each rule, and e an example. The examples are assigned the class B_j of the rule $R_{B_j}(A_j)$ verifying

$$j = \text{argmax}_{0 \leqslant i \leqslant q}\{U(e, A_i) * \omega_i\}. \qquad (34)$$

In a similar way to SLAVE, the conflict resolution mechanism used keeps the two criteria mentioned above to break possible ties among the rules, that is,

- The rule with higher weight.
- The rule that was first learned.

3.5. *Main advantages and disadvantages of SLAVE2*

SLAVE2 provides two important advantages with respect to SLAVE. The first one is related to a lower dependency on the λ parameter. This fact causes an increment in the collaboration/competition among rules during the learning process. So, the rules selected previously in the iterative strategy are used to guide the search mechanism and allow reducing inappropriate interactions on the knowledge obtained. The second advantage is a significant improvement of the embedded feature selection, allowing the algorithm to obtain simpler and more general rule sets.

On the other hand, reduce the dependency on the λ parameter presents a computational inconvenient, since the algorithm does not admit a parallel version. Another important problem is that the algorithm maintains a strong bias caused by the need to learn the rules in a particular order. This order is associated with the choice, probably arbitrary, of a class before another.

4. NSLV

A simple way to describe NSLV[10] would be to say that it is exactly the same as SLAVE2, but with the difference that it learns fuzzy rules without fixing the class of the consequent variable.

This simple description is valid, but hides important nuances that must be highlighted. On the one hand, many SLAVE2 drawbacks are solved by not fixing the class during the learning process: The bias due to class selection order disappears; furthermore, it removes the dependence on parameter λ,

since the consideration that an example is covered by a rule in its class is determined by the interaction among rules already learned. This parameter is adjusted automatically as new rules are included in the knowledge base.

Moreover, the inclusion of the rule consequent in the search process forces a major redefinition of the algorithm, although the required changes are natural extensions that keep the usual format of SLAVE and SLAVE2.

However, a major problem arises, related to the diversity of the population associated with the genetic algorithm. It is well known that the convergence of GAs can result in a final population where most of the individuals are very similar. To solve this problem, the iterative approach followed in SLAVE and SLAVE 2 resets the initial population to enhance the rule in the next iteration.

However, alternative solutions are possible, and NSLV uses an alternative approach. This approach is based on the use of subpopulations. Thus, NSLV maintains a subpopulation for rules of each class and a combination operator adapted to maintain the diversity in classes. After completing one iteration step, the final population obtained by the GA contains the best rule, but also other promising rules that could be useful and therefore can be used as the starting point for the next iteration.

The following subsections describe in more detail the implementation of these ideas in the algorithm.

4.1. *The genetic algorithm*

The genetic algorithm of NSLV maintains the basic structure of SLAVE2, with some differences in the representation of individuals, the search mechanism (now we are looking for complete rules), the genetic operators, and the termination condition.

4.1.1. *Representation of a population element*

Unlike SLAVE and SLAVE2, NSLV does not learn classes in a specific order. That is why the representation of an individual must include a new level which allows considering the class that is being learned in each moment. This new level is called *consequent level*. So, the complete structure of an individual (and therefore of a rule), would be (See Figure 8):

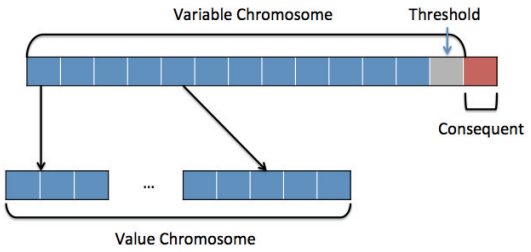

Fig. 7. Representation of a population individual in NSLV.

The *consequent level* codifies the value of the classification variable of the rule. This level is composed by one gene that is represented through an integer value and is randomly generated in the initial population.

A real example extracted from database *Glass*[20] that could be helpful to better understand the codification of each level in the rule representation is shown in Figure 9. The *Glass* database, created by the USA Forensic Science Service, classifies 6 types of glass which can be found in a crime scene, defined in terms of their oxide content (i.e. Na, Fe, K, etc). The first attribute measures the refractive index, while the remaining attributes in the antecedent measure the weight percentage in corresponding oxide.

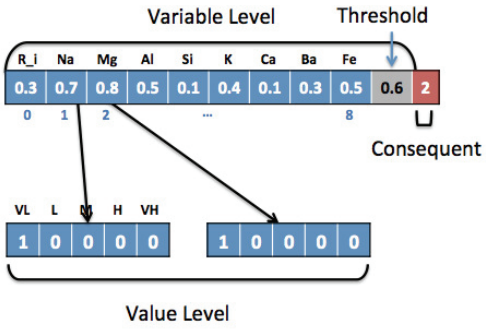

Fig. 8. Example of a rule codification.

According to this figure, and focusing our attention on the variable level, we can see that the only variables that exceed the threshold are "*Sodium*" (Na) and "*Magnesium*" (Mg), so they are considered in the rule with their respective value levels. For variable "*Na*", the value level has only one active position related to label "*Very Low*". Regarding to variable "*Mg*", the active label is also "*Very Low*". Thus, this codification corresponds to the rule:

IF Na is {VeryLow} and Mg is {VeryLow} THEN

TypeOfGlass is BuildingWindowsNonFloatProcessed

with **weight 0.93**.

The weight is also encoded in the rule, but since it does not change with the evolution model, it is not included in the representation of a population individual in the genetic algorithm.

So, the previous rule would be easily interpreted as:

"**IF** Sodium is Very Low and Magnesium is Very Low

THEN

TypeOfGlass is BuildingWindowsNonFloatProcessed

with **weight 0.93**"

4.1.2. ***Keeping diversity in the genetic population***

In order to obtain "good individuals", the genetic algorithm must maintain the diversity in the different classes. That is, it must ensure that there are always individuals of all classes. To achieve this, NSLV uses a population composed of subpopulations or niches (one for each class to be learned) and a modified version of a steady state genetic algorithm whose selection process ensures that no niche disappears from the population.

The selection process is as follows: Two individuals in the population are selected; the crossover operator on each level is applied between them obtaining two new individuals. The mutation operator is used for modifying the new individuals. A standard genetic algorithm replaces these two new individuals with the two worst in the population. This procedure could lead to the elimination of all individuals in a niche. Thus, it is necessary to modify the standard criterion. In this way, the genetic algorithm of NSLV takes an alternative approach which consists in replacing the two new individuals with the two worst of those subpopulations or niches that are not at risk of being removed from the population. It is considered that a subpopulation is not at risk of being eliminated if it maintains at least m individuals in its niche, where $m = N_{Population}/(N_{classes}+1)$, $N_{Population}$ is the number of individuals in the population, and $N_{classes}$ is the number of classes evolved in the problem.

This steady state algorithm provides some important advantages. Unlike SLAVE and SLAVE2, which after each iteration had to calculate the initial relevant degree of each variable for defining the variable level for the next population, NSLV defines the variable level of the next initial population as the variable level of the last population. This modification has two advantages:

- A runtime reduction, since the time required for obtaining the initial relevant degree is expensive when there is a large number of examples or variables.
- The last population keeps the best individuals found, and this setting is a good starting point for the next search.

4.1.3. ***Genetic operators***

As for genetic operators, NSLV uses different ones depending on the level involved. As occurred with SLAVE2, we can distinguish between:

- Variable level:
  1. Real uniform mutation.
  2. Crossover operator over two points.

- Value level:
  1. Binary uniform mutation.
  2. Crossover operator over two points.

- Consequent level:
  1. Integer uniform mutation.

4.1.4. *Penalization of examples and termination condition*

NSLV uses a new module for the penalization of examples. SLAVE and SLAVE2 removed the examples covered by a rule to avoid considering them in a later step. In contrast, NSLV marks these examples so they are considered by rules of other classes. As the evaluation of a rule is guided by the examples not covered by any previous one, the marked examples do not affect any rule positively, but they influence the evaluation of rules belonging to other classes.

With respect to the termination condition, the iterative process ends under the consideration of the completeness degree over the whole fuzzy rule set. Thus, if a new rule is added and the completeness degree does not improve, then the learning process ends. The final solution obtained by the algorithm is the complete set of extracted rules, excluding the last one.

4.1.5. *Rule Filtering module*

According to our experimental tests, NSLV can add irrelevant rules during the learning process. A rule is considered irrelevant if it is not used at least once for classifying correctly an example. This situation can occur when using an iterative rule learning approach frequently, since it can happen that very specific rules are subsumed by more general rules obtained in later steps. In order to simplify the final rule set, NSLV uses a module for removing irrelevant rules from the learned rule set.

4.2. *Main advantages and further improvements of NSLV*

As already mentioned, NSLV has two major advantages over its predecessors: The elimination of bias in the presentation order of the classes during the learning process, and the elimination of the dependence on the λ parameter of minimum coverage rules. These two advantages have two practical consequences for the algorithm. First, we get knowledge bases with fewer rules, and secondly, the learning time is lower. The reason we obtain fewer rules is that, when the algorithm learns all the rules (after fixing one class), it must ensure that most of the examples of this class are well covered by the rules. In general, this involves adding additional rules in the knowledge base, in anticipation of possible conflicts with the rules of the classes that are to be learned. This does not happen in NSLV, as it does not have to anticipate future conflicts. The improvement in learning time is related to the reuse of the final population of an iteration as the initial population of the next iteration, since this initial population is now closer to the solutions.

NSLV is an algorithm that demonstrates good performance compared to other classification algorithms that work with knowledge bases expressed by fuzzy rules. However, there are many features that can be improved; they have been studied and still are in development, such as improving the efficiency of the algorithm [27] or using a knowledge representation model more complex than the DNF rule[28].

5. Experimental study

In this section, we study the performance of the re-implemented methods: SLAVE, SLAVE2, and NSLV[‡]

The experimental study has been carried out on the KEEL platform[20] using 37 datasets of this

[‡]The implementation of these algorithms, along with instructions for incorporating them to the KEEL platform, can be downloaded from http://isg.ugr.es/descargas/descargar/272/

Table 1. Database used for the experimental study.

Database	#E	#V	#R	#I	#N	#Cl	M
Appendicitis	106	7	7	0	0	2	No
Australian	690	14	3	5	6	2	No
Automobile	205	25	15	0	10	6	Yes
Balance	625	4	4	0	0	3	No
Banana	5300	2	2	0	0	2	No
Bands	539	19	13	6	0	2	Yes
Breast	286	9	0	0	9	2	Yes
Bupa	345	6	1	5	0	2	No
Car Evaluation	1728	6	0	0	6	4	No
Chess	3196	36	0	0	36	2	No
Cleveland	303	13	13	0	0	5	Yes
Contraceptive Method	1473	9	0	9	0	3	No
Credit Approval	690	15	3	3	9	2	Yes
Dermatology	366	34	0	34	0	6	Yes
Ecoli	336	7	7	0	0	8	No
Glass	214	9	9	0	0	7	No
Haberman	306	3	0	3	0	2	No
Hayes-Roth	160	4	0	4	0	3	No
Heart	270	13	1	12	0	2	No
Hepatitis	155	19	2	17	0	2	Yes
Housevotes	435	16	0	0	16	2	Yes
Ionosphere	351	33	32	1	0	2	No
Iris	150	4	4	0	0	3	No
Monk-2	432	6	0	6	0	2	No
Movement Libras	360	90	90	0	0	15	No
New Thyroid	215	5	4	1	0	3	No
Pima	768	8	8	0	0	2	No
Ring	7400	20	20	0	0	2	No
Segment	2310	19	19	0	0	7	No
Sonar	208	60	60	0	0	2	No
Thyroid	7200	21	6	15	0	3	No
Vehicle	846	18	0	18	0	4	No
Vowel	990	13	10	3	0	11	No
Wisconsing Diagnostic	569	30	30	0	0	2	No
Wine	178	13	13	0	0	3	No
Wisconsing	699	9	0	9	0	2	Yes
Zoo	101	16	0	0	16	7	No

Table 2. Specific conditions for SLAVE. NGVL means the number of genes in the value level.

	Specific Conditions SLAVE
Size of genetic population	20
λ parameter	0.8
Number of iterations	500
Generalization prob. (Value level)	0.1
Mutation prob. (Value level)	0.5/NGVL
Crossover prob. (Value level)	0.1

Table 3. Specific conditions for SLAVE2. NGVL means the number of genes in the value level and NAV means the number of antecedent variables.

	Specific Conditions SLAVE2
Size of genetic population	20
λ parameter	0.8
Number of iterations	500
Generalization prob. (Value level)	0.1
Mutation prob. (Value level)	0.5/NGVL
Mutation prob. (Variable level)	1/NAV
Crossover prob. (Value level)	0.1
Crossover prob. (Variable level)	0.1

Table 4. Specific conditions for NSLV. NAV means the number of antecedent variables.

	Specific Conditions NSLV
Size of genetic population	100
Number of iterations	500
Mutation prob. (Value level)	0.01
Mutation prob. (Variable level)	1/NAV
Mutation prob. (Consequent level)	0.01
Crossover prob. (Value level)	1
Crossover prob. (Variable level)	1

platform, with the same partitions in all databases and the recommended settings. Table 1 describes these databases, where each row represents the number of examples (#E), variables (#V), real variables (#R), integer variables (#I), nominal variables (#N), classes (#Cl), and missing values (M) respectively.

For all databases we have considered five uniformly distributed linguistic labels to define the domain of the continuous variables. The results have been obtained using ten-fold cross validation and the same partitions for all the algorithms. All the parameters used for each algorithm in this experimentation are detailed in Tables 2, 3 and 4.

We can see that the parameters used for SLAVE and SLAVE2 (Tables 2 and 3) are very similar. The difference between them stems in the new codification level added to SLAVE2, with an associated probability value for the crossover and mutation operators. The GA of these two proposals tries in a special way the probability values associated to the mutation operator, which are linked to the code length of each level. So, in the case of a value level, the factor NGVL represents the Number of Genes in the Value Level. In the same sense, in a variable level the NAV factor represents the Number of Antecedent Variables involved in the example set.

Comparing these tables with the parameters used by NSLV (Table 4), we can find some important differences. The first one is that the latter table does not include the λ parameter nor the generalization genetic operator, as the algorithm does not require them. The second difference is related to the new GA used for this proposal. NSLV uses a GA based on a steady state approach, keeping niches (one for each class) in the genetic population. These niches are the reason for increasing the size of the population (from 20 to 100 individuals). The use of a steady state approach requires changing the probabilities associated to the genetic operators.

5.1. SLAVE, SLAVE2, and NSLV

We have focused this analysis on the study of four parameters: The accuracy on training and test, the

Table 5. Results obtained by SLAVE, SLAVE2, and NSLV on 37 databases. The Table shows the accuracy on training and test.

Data	Training			Test		
	SLAVE	SLAVE2	NSLV	SLAVE	SLAVE2	NSLV
appendicitis	91.7 (2.5)	91.7 (2.5)	**93.2 (1)**	**84.2 (1)**	84.1 (2.5)	84.1 (2.5)
australian	88.9 (3)	89.4 (2)	**90.8 (1)**	83.4 (3)	84.2 (2)	**85 (1)**
automobile	94.4 (3)	98.4 (2)	**98.5 (1)**	72.9 (3)	77.8 (2)	**78.5 (1)**
balance	94 (2)	**94.4 (1)**	87.5 (3)	66.2 (3)	66.8 (2)	**76.4 (1)**
banana	78 (2)	75.2 (3)	**79.2 (1)**	78 (2)	75.3 (3)	**79.2 (1)**
bands	84.9 (2)	**85.3 (1)**	81.4 (3)	65 (2)	**69.1 (1)**	64.7 (3)
breast	**94.2 (1)**	91.8 (2)	86.9 (3)	62.1 (3)	62.4 (2)	**68.3 (1)**
bupa	65 (3)	65.2 (2)	**71.3 (1)**	59.3 (3)	61.3 (2)	**62.1 (1)**
car	70 (3)	**72.5 (1)**	70.4 (2)	69.8 (3)	**71.3 (1)**	70 (2)
chess	97.5 (2)	**98.5 (1)**	94.9 (3)	97.1 (2)	**98.2 (1)**	94.8 (3)
cleveland	**93.8 (1)**	90.6 (3)	91.6 (2)	**54.7 (1)**	53.7 (2)	49.4 (3)
contraceptive	**63.6 (1)**	58.9 (2)	52.4 (3)	**44.6 (1)**	42 (2)	38.3 (3)
crx	**94.4 (1)**	94.2 (2)	90.4 (3)	84.2 (3)	84.4 (2)	**84.9 (1)**
dermatology	**99.9 (1)**	99.6 (2)	99.1 (3)	85.7 (3)	**93.2 (1)**	92.3 (2)
ecoli	**88.7 (1)**	88.4 (3)	88.6 (2)	82.7 (2)	**83 (1)**	80.6 (3)
glass	72.6 (2)	71.6 (3)	**80.4 (1)**	62.8 (2)	61.7 (3)	**65.4 (1)**
haberman	77.7 (2)	77.4 (3)	**78.4 (1)**	71.5 (2.5)	71.5 (2.5)	**72.1 (1)**
hayes	89.6 (2)	89.5 (3)	**90.2 (1)**	77.5 (3)	78.1 (2)	**78.7 (1)**
heart	**96 (1)**	95.6 (2)	91.1 (3)	75.9 (3)	76.6 (2)	**79.6 (1)**
hepatitis	98.7 (2)	**98.8 (1)**	96.2 (3)	87 (3)	88.9 (2)	**89.4 (1)**
housevotes	**100 (1)**	99.9 (2)	97.2 (3)	**97.4 (1)**	96.2 (2)	96 (3)
ionosphere	96.9 (2)	**98.7 (1)**	95.9 (3)	85.4 (3)	**90.9 (1)**	90.6 (2)
iris	97.2 (3)	97.3 (2)	**97.7 (1)**	96 (2.5)	96 (2.5)	**96.6 (1)**
monk-2	**100 (1.5)**	**100 (1.5)**	98 (3)	**100 (1.5)**	**100 (1.5)**	98.4 (3)
movement libras	92.3 (3)	97 (2)	**98.3 (1)**	70.2 (3)	78.8 (2)	**79.1 (1)**
new thyroid	93.4 (2)	93.2 (3)	**95.2 (1)**	92.1 (2)	91.6 (3)	**93 (1)**
pima	78.8 (2)	78.3 (3)	**80 (1)**	**74.8 (1)**	73.4 (2)	73.3 (3)
ring	92.5 (3)	93.3 (2)	**93.8 (1)**	92.1 (2)	**93.3 (1)**	92 (3)
segment	87.7 (3)	88.1 (2)	**93 (1)**	87 (3)	87.7 (2)	**91.3 (1)**
sonar	**99.3 (1)**	98.7 (2.5)	98.7 (2.5)	71.7 (3)	**79.8 (1)**	77.3 (2)
thyroid	92.9 (3)	**93 (1.5)**	**93 (1.5)**	92.8 (3)	**93 (1.5)**	**93 (1.5)**
vehicle	71.1 (2)	68.3 (3)	**84.6 (1)**	63.9 (2)	62.8 (3)	**67.5 (1)**
vowel	82.4 (2)	77.5 (3)	**96.1 (1)**	76.9 (2)	71.7 (3)	**85.7 (1)**
wdbc	96.2 (2.5)	96.2 (2.5)	**98.2 (1)**	94.5 (3)	**94.7 (1.5)**	**94.7 (1.5)**
wine	99 (3)	**99.8 (1)**	99.5 (2)	**96 (1)**	95.4 (2)	93.2 (3)
wisconsin	**99.7 (1)**	99.1 (2)	98.8 (3)	93.1 (3)	**93.8 (1)**	93.4 (2)
zoo	**100 (2)**	**100 (2)**	**100 (2)**	90.2 (3)	95.8 (2)	**96.1 (1)**
Rnk	2.01351	2.09459	1.89189	2.36486	1.89189	1.74324
Pos	2	3	1	3	2	1
Mean	89.5405	89.3351	90.0135	79.4243	80.5	81.2162

Table 6. Results obtained by SLAVE, SLAVE2 and NSLV on 37 databases. The Table shows the average number of rules per database and the time needed to get the model.

	Rules			Time		
Data	SLAVE	SLAVE2	NSLV	SLAVE	SLAVE2	NSLV
appendicitis	6.4 (3)	5.9 (2)	**5.5 (1)**	**0 (1.5)**	**0 (1.5)**	0.1 (3)
australian	**13.7 (1)**	15.1 (3)	14.1 (2)	**5.4 (1)**	10.2 (3)	6.4 (2)
automobile	27.2 (3)	23.1 (2)	**19.3 (1)**	2.1 (2)	**1.3 (1)**	3.8 (3)
balance	71.8 (3)	71.3 (2)	**21.6 (1)**	**0.3 (1)**	2.3 (2)	4 (3)
banana	8.5 (3)	**7.7 (1.5)**	**7.7 (1.5)**	20.4 (2)	36 (3)	**10.5 (1)**
bands	25.5 (2)	31 (3)	**11.6 (1)**	12.6 (2)	21.1 (3)	**6 (1)**
breast	20.4 (3)	19.4 (2)	**11.8 (1)**	**0.2 (1)**	0.4 (2)	1.2 (3)
bupa	9.6 (3)	8.4 (2)	**6.8 (1)**	**0 (1)**	1.4 (3)	1.2 (2)
car	3.6 (2)	9.8 (3)	**2.2 (1)**	4.1 (2)	12.4 (3)	**0.7 (1)**
chess	12.7 (3)	12 (2)	**5.4 (1)**	98.9 (3)	62.3 (2)	**10.7 (1)**
cleveland	57.1 (3)	51.7 (2)	**49.8 (1)**	**3.7 (1)**	4.9 (2)	17.4 (3)
contraceptive	77.9 (3)	60.7 (2)	**54.9 (1)**	80.6 (2)	125.9 (3)	**63.9 (1)**
crx	20.8 (2)	21.9 (3)	**8.6 (1)**	7.5 (2)	10.1 (3)	**3.8 (1)**
dermatology	25.1 (3)	10.8 (2)	**10.1 (1)**	10.8 (3)	3.8 (2)	**2.5 (1)**
ecoli	19.7 (3)	17.1 (2)	**16.4 (1)**	**0 (1)**	1.1 (2)	2.8 (3)
glass	19.3 (3)	16.5 (2)	**16.4 (1)**	**0 (1.5)**	**0 (1.5)**	2.4 (3)
haberman	8.7 (3)	7.9 (2)	**5.6 (1)**	**0 (1.5)**	**0 (1.5)**	0.2 (3)
hayes	9.4 (2)	9.7 (3)	**8.9 (1)**	**0 (2)**	**0 (2)**	**0 (2)**
heart	19.5 (2)	20.6 (3)	**10.7 (1)**	**0.4 (1)**	1.2 (2)	1.4 (3)
hepatitis	7 (3)	6.5 (2)	**3.4 (1)**	**0 (2)**	**0 (2)**	**0 (2)**
housevotes	6.4 (3)	6 (2)	**2.5 (1)**	**0 (2)**	**0 (2)**	**0 (2)**
ionosphere	39.8 (3)	17.8 (2)	**9.1 (1)**	13.7 (3)	11.1 (2)	**3.5 (1)**
iris	6 (3)	5.9 (2)	**3.3 (1)**	**0 (2)**	**0 (2)**	**0 (2)**
monk-2	4 (2.5)	4 (2.5)	**3 (1)**	**0 (2)**	**0 (2)**	**0 (2)**
movement libras	114.7 (3)	91 (2)	**50.3 (1)**	334.2 (2)	378.2 (3)	**105.6 (1)**
new thyroid	9.4 (3)	8 (2)	**5.9 (1)**	**0 (1.5)**	**0 (1.5)**	0.1 (3)
pima	14.1 (3)	**11.6 (1)**	13.4 (2)	**6.8 (1.5)**	11.1 (3)	**6.8 (1.5)**
ring	**12 (1)**	16.2 (2)	20.4 (3)	470.4 (2)	3081.7 (3)	**334.8 (1)**
segment	19.2 (2)	**19 (1)**	20.3 (3)	139.9 (2)	142.7 (3)	**59.9 (1)**
sonar	58.4 (3)	21.2 (2)	**10.7 (1)**	30.8 (3)	28.9 (2)	**8.8 (1)**
thyroid	5.1 (3)	4.8 (2)	**3.2 (1)**	76 (3)	75.7 (2)	**10.2 (1)**
vehicle	23 (2)	**18.7 (1)**	46.6 (3)	64.5 (3)	**61.1 (1)**	63.5 (2)
vowel	85.1 (2.5)	**78.1 (1)**	85.1 (2.5)	**101.3 (1)**	143.3 (2)	148.4 (3)
wdbc	9.3 (2)	11.2 (3)	**7.7 (1)**	26.7 (3)	20.8 (2)	**8.2 (1)**
wine	12.3 (2)	13.7 (3)	**6.8 (1)**	**0.3 (1)**	0.4 (2)	1 (3)
wisconsin	9.4 (2)	9.8 (3)	**9.1 (1)**	**4.1 (1.5)**	5.7 (3)	**4.1 (1.5)**
zoo	8.5 (3)	8.1 (2)	**7.8 (1)**	**0 (2)**	**0 (2)**	**0 (2)**
Rnk	2.59459	2.13514	1.27027	1.86486	2.21622	1.91892
Pos	3	2	1	1	3	2
Mean	24.3405	20.8703	16.1081	40.9649	115.003	24.1595

average number of rules, and the time required to learn them. To perform the comparison we used the Bonferroni-Dunn test[29]. This test establishes that two classifiers have differences if they differ at least in one critical distance. This distance depends on the number of databases, the number of learning algorithms to compare and a critical value q_α, with error α. In this case we used $\alpha = 0.05$, being $q_\alpha = 1.960$. For 37 databases and three classifiers, the critical distance (CD) was **0.456**.

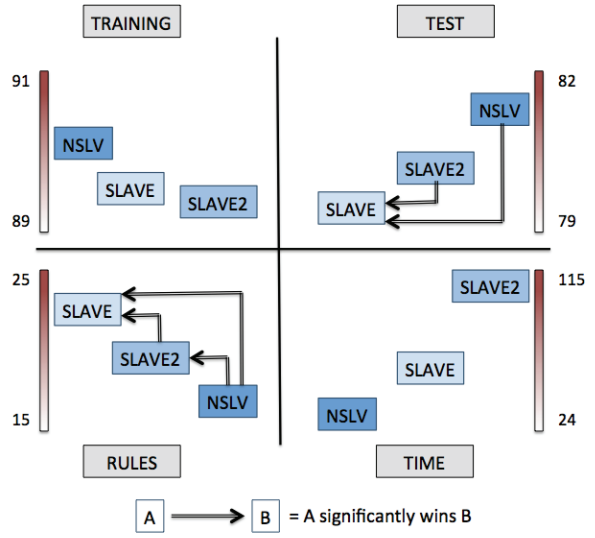

Fig. 9. Representation of the significant differences between the algorithms.

Now, using the information of Table 5, Table 6 and Figure 10, we can detect the main differences among the three algorithms.

First, we have to highlight the main differences between SLAVE and SLAVE2. SLAVE2 features significant differences in accuracy and number of rules compared to SLAVE, but with the penalty of needing more time to learn. The likely cause of this improvement in the number of rules and accuracy along with a degradation of the time required, is the codification used for a population individual. As previously mentioned, SLAVE2 introduced a new codification in order to improve the interpretability when working with more complex databases.

Second, we can see that the three algorithms do not present significant differences in training, being NSLV the one that obtains the best results. SLAVE and SLAVE2 have a very similar behavior. On the other hand, according to the results obtained in test, we have to note that both SLAVE2 and NSLV obtain significant differences with respect to SLAVE.

Third, the results show that NSLV obtains a smaller number of rules than SLAVE and SLAVE2, with significant differences with respect to these two algorithms. If we consider the time parameter, the best algorithm is SLAVE, followed by NSLV and SLAVE2, but without significant differences between them. It is important to remark that the fastest algorithm with regard to means is NSLV, which invests half the time required by the best one (SLAVE). This means that SLAVE wins more frequently than NSLV, but for the databases where NSLV wins, it does so by investing considerably less time than SLAVE.

In summary, as shown in Figure 10, the ability to remove the bias caused by establishing a predefined order in the way classes are learned allows NSLV to improve the prediction capability (greater accuracy), when compared to the two competing algorithms. Moreover, it allows to reduce the final number of rules in the model obtained, being significantly smaller than the rule sets returned by SLAVE and SLAVE2. Finally, one important point is that, unlike SLAVE and SLAVE2 (which are generational algorithms), NSLV is a stationary algorithm, a fact that explains why the overall time (taking into account mean values) employed to generate the model is much smaller.

5.2. *SLAVE, SLAVE2, and NSLV vs other learning algorithms*

In this section we will compare the previously described versions of SLAVE with other learning algorithms. In order to do so, we use the KEEL platform [20] with recommended settings, and the databases shown in Table 1. We show below a brief description of the algorithms used for the comparison (in brackets, the short name used in the KEEL platform):

Table 7. Results obtained by several learning algorithms on 37 databases. The Table shows the accuracy on test parameter.

	Logitboost	C4.5	Naive Bayes	SGERD	SLAVE	SLAVE2	NSLV
appendicitis	80.2 (7)	83.2 (5)	**87 (1)**	83.1 (6)	84.2 (2)	84.1 (3.5)	84.1 (3.5)
australian	**86 (1)**	85.7 (2)	83.3 (7)	85.5 (3)	84.2 (6)	85 (4.5)	85 (4.5)
automobile	43.6 (7)	**80.9 (1)**	75.6 (4)	47.5 (6)	72.9 (5)	77.8 (3)	78.5 (2)
balance	**88.4 (1)**	76.7 (2)	71.2 (5)	75.9 (4)	66.2 (7)	66.8 (6)	76.4 (3)
banana	82.1 (2)	**89.1 (1)**	64.2 (6)	60.5 (7)	78 (4)	75.3 (5)	79.2 (3)
bands	64.1 (6)	66.5 (3)	68.3 (2)	63.7 (7)	65 (4)	**69.1 (1)**	64.7 (5)
breast	71.2 (3)	**76.9 (1)**	74.3 (2)	70.4 (4)	62.1 (7)	62.4 (6)	68.3 (5)
bupa	**69.5 (1)**	69.3 (2)	60.5 (5)	57.3 (7)	59.3 (6)	61.3 (4)	62.1 (3)
car	88.2 (2)	**91.5 (1)**	85.9 (3)	67.1 (7)	69.9 (5.5)	69.9 (5.5)	70 (4)
chess	52.2 (7)	**99.4 (1)**	87.7 (5)	61.4 (6)	97.1 (3)	98.2 (2)	94.8 (4)
cleveland	53.1 (4)	54.4 (2)	53 (5)	49.4 (6.5)	**54.7 (1)**	53.7 (3)	49.4 (6.5)
contraceptive	**53.9 (1)**	53.8 (2)	46.9 (4)	47.9 (3)	35.7 (7)	37 (6)	38.3 (5)
crx	83.4 (6)	85.2 (2)	82.3 (7)	**86.2 (1)**	84.2 (5)	84.4 (4)	84.9 (3)
dermatology	31 (7)	94.3 (2)	**95.2 (1)**	81.2 (5)	63.9 (6)	91 (4)	92.3 (3)
ecoli	78 (5)	79.4 (4)	77.3 (6)	74.6 (7)	82.7 (2)	**83 (1)**	80.6 (3)
glass	66.9 (2)	**67.4 (1)**	62.5 (5)	61 (7)	62.8 (4)	61.7 (6)	65.4 (3)
haberman	71.2 (6)	73.1 (2)	71.1 (7)	**73.5 (1)**	71.5 (4.5)	71.5 (4.5)	72.1 (3)
hayes	75 (5)	**80 (1)**	60 (6)	44.9 (7)	77.5 (4)	78.1 (3)	78.7 (2)
heart	76.6 (4.5)	78.5 (2)	75.9 (6.5)	77 (3)	75.9 (6.5)	76.6 (4.5)	**79.6 (1)**
hepatitis	81.9 (6)	83.9 (5)	85.9 (4)	77.3 (7)	87 (3)	88.9 (2)	**89.4 (1)**
housevotes	58 (7)	97 (2)	91 (5)	87.4 (6)	**97.4 (1)**	96.2 (3)	96 (4)
ionosphere	64.3 (7)	**91.1 (1)**	89.4 (4)	80.9 (6)	85.4 (5)	90.9 (2)	90.6 (3)
iris	94.6 (6)	96 (3)	94 (7)	95.3 (5)	96 (3)	96 (3)	**96.6 (1)**
monks	96.5 (5)	**100 (2)**	51.2 (7)	80.6 (6)	**100 (2)**	**100 (2)**	98.4 (4)
movement libras	5.5 (7)	69.4 (4)	50 (5)	42.5 (6)	70.2 (3)	78.8 (2)	**79.1 (1)**
new thyroid	**95.3 (1.5)**	92.5 (4)	**95.3 (1.5)**	87 (7)	92.1 (5)	91.6 (6)	93 (3)
pima	**75.7 (1)**	73.9 (3)	72.5 (7)	73 (6)	74.8 (2)	73.4 (4)	73.3 (5)
ring	**97 (1)**	90.2 (6)	91.8 (5)	71.4 (7)	92.1 (3)	93.3 (2)	92 (4)
segment	85.4 (5.5)	**97.4 (1)**	85.4 (5.5)	77.1 (7)	87 (4)	87.7 (3)	91.3 (2)
sonar	56.2 (7)	70.5 (4)	68.7 (5)	66.6 (6)	71.7 (3)	**79.8 (1)**	77.3 (2)
thyroid	92.8 (5.5)	**99.5 (1)**	97.4 (2)	91.9 (7)	92.8 (5.5)	93 (3.5)	93 (3.5)
vehicle	66.7 (3)	**74.1 (1)**	61.9 (6)	51.8 (7)	63.9 (4)	62.8 (5)	67.5 (2)
vowel	43.7 (5)	81.5 (2)	26.9 (7)	38.6 (6)	76.9 (3)	71.7 (4)	**85.7 (1)**
wdbc	94 (5.5)	**95.2 (1)**	94 (5.5)	90.6 (7)	94.5 (4)	94.7 (2.5)	94.7 (2.5)
wine	**97.7 (1)**	94.9 (5)	97.1 (2)	92 (7)	96 (3)	95.4 (4)	93.2 (6)
wisconsin	95.5 (2)	94.7 (3)	**96.5 (1)**	93.4 (5.5)	93.1 (7)	93.8 (4)	93.4 (5.5)
zoo	46.3 (7)	92.8 (4)	94.4 (3)	84.5 (6)	90.2 (5)	95.8 (2)	**96.1 (1)**
Rnk	4.33784	2.40541	4.59459	5.72973	4.18919	3.55405	3.18919
Pos	5	1	6	7	4	3	2
Mean	71.9378	83.2405	76.3676	71.6216	78.6189	80.2892	81.2162

- Logitboost (GFS-LogitBoost-C): In [33], Otero and Sánchez proposed Logitboost, a back fitting algorithm which repeatedly invokes a learning algorithm to successively generate a committee of simple, low-quality classifiers. In this algorithm, each of the weak hypotheses is a fuzzy rule extracted from data by means of a genetic algorithm. Each time, a new simple classifier is added to the compound one, and the examples in the training set are re-weighted.
- C4.5 (C4.5-C) [34]: Proposed by R.S. Quinlan, it is an inductive classification algorithm that represents knowledge using a decision tree.
- Naive Bayes (NB-C): Naive Bayes was proposed in [35] by Domingos and Pazzani. It is a probabilistic classifier based on a Bayesian model.
- SGERD (SGERD-C) [36]: It is a steady-state genetic algorithm to extract a compact set of fuzzy rules from numerical data.

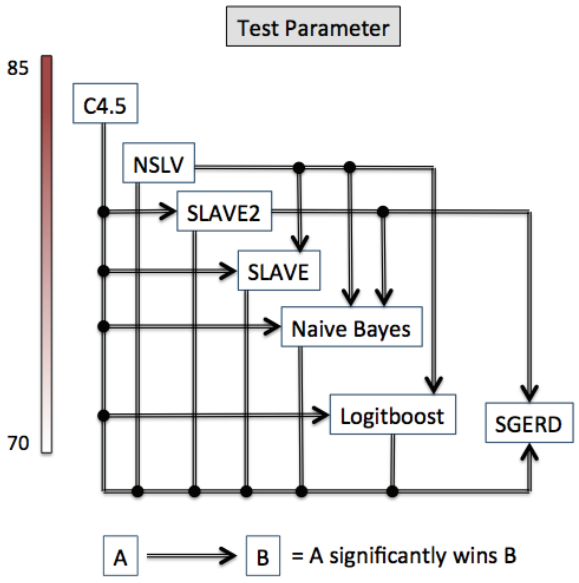

Fig. 10. Representation of the significant differences between the algorithms.

According to the Bonferroni-Dunn test, the critical distance (CD) in this experimental study is **0.984**.

Looking at the results shown in Table 7, we can see that NSLV is the second best algorithm in test, only surpassed by C4.5. In fact, NSLV obtains significant differences in relation to the rest of algorithms except C4.5 and SLAVE2. On the other hand, SLAVE2 significantly outperforms the Naive Bayes and SGERD algorithms, while SLAVE outperforms SGERD. They both follow NSLV in the ranking, improving the results obtained by Logitboost, Naive Bayes, and SGERD. Figure 11 shows a graphical interpretation of the results, where these significant differences can be seen.

Anyway, the versions presented in these paper have been the basis of later works in which these results have been substantially improved.

6. Steps under development for the SLAVE methodology

The SLAVE methodology has been—and still is—the basis of the work we are developing. NSLV allowed us to develop several new improvements and contributions to the learning algorithms based on fuzzy rules.

In a first stage, we worked on feature construction for genetic learning algorithms based on the iterative learning rule approach. The main difficulty we had to address was the need to develop techniques to convert the search over a huge space of possible new good features into an approachable problem. We studied different alternatives to incorporate a method of feature construction to NSLV, and we implemented an approach based on a filter model with very promising results. This approach follows an idea which is similar to the feature selection used in SLAVE2. Specifically, the algorithm takes two different paths for generating new features: (a) from a reduced set of possible relations or (b) from a reduced set of possible arithmetical functions among input variables. Furthermore, to determine the relevance of the new features, an information measure is used to select only the most relevant new features. From an initial set of new attributes, the evolution is ultimately responsible for deciding the attributes that will be included in each fuzzy rule. This process requires the definition of

a new rule model and a reasoning mechanism for working with this kind of rules[30,31].

In a second stage, we are interested in modifying the iterative rule learning approach used in the SLAVE proposals—particularly in NSLV—to allow reviewing the knowledge obtained in each iteration. We selected the iterative rule learning approach because it provides better efficiency. However, this approach also presents some problems. One of them is related to the fact that it always adds a new rule in each iteration. The sequential covered strategy assumes that the process is closer to the final solution each time that a new rule is added to the rule set. So, each iteration of the algorithm implies an increase in the rule count of the knowledge base. At some point, it may be better to replace a rule by another, or delete any of the rules learned instead of adding a new rule. If these situations occur, and we have experimentally observed that they do, then the sequential covered strategy must be modified. The idea is to improve the sequential covering algorithm to include the ability to add rules, among other capabilities, such as the possibility of reviewing the learned knowledge in order to remove or merge rules, or even add new rules if necessary. We are currently analyzing a proposal that replaces the main step of the sequential covering strategy previously described (adding a new rule) with a new function which is able to decide the best action (add rule, remove rule, replace rule,...) that should be taken in each situation. This proposal[32] tries to break away from the idea that the problem solution is the simple aggregation of the partial solutions obtained (a particular rule), and it introduces a more intelligent decision process where the algorithm itself chooses the solution which offers the best alternative for the current situation. As a natural consequence of this ability, if the current situation changes (e.g. the set of examples is changed), the algorithm should perform incremental learning, which is precisely the challenge we are currently working on.

7. Conclusions

In this work we have described the evolution of the SLAVE learning algorithm considering three stages that show an inflection point in its development.

SLAVE is the oldest of the three proposals, and defines the basic principles and the general framework that were used by later extensions. It was one of the first algorithms that represented knowledge using fuzzy rules, with a structure similar to that of the classical learning algorithms.

SLAVE2 is the next step in the evolution and incorporates two interesting aspects: A new criterion for considering the positivity and negativity of the examples, and model feature selection built in the genetic algorithms.

NSLV is the most recent version, and the first to learn complete rules without having to fix the consequent variable. Furthermore, it uses a steady state genetic algorithm and a niche structure in the population to reduce the learning time.

The experimental study shows that NSLV presents the best overall behavior, taking into account the prediction capacity parameters studied on training and test sets, the number of rules and the learning time. NSLV shows significant differences in test compared with SLAVE and in number of rules compared with the two competing algorithms, and no differences for the remaining parameters. On the other hand, SLAVE2 outperforms SLAVE in prediction for test sets and in number of rules, and therefore its learning time is worse.

Acknowledgments

This work has been partially funded by the Andalusian Regional Government project P09-TIC-04813, the Spanish MEC project TIN2012-38969 and cofinanced by FEDER funds (European Union).

References

1. Michalski, R.S., “A theory and methodology of inductive reasoning”, in R.S. Michalski, J. Carbonell and T.

Mitchel (Eds.), *Machine Learning: An artificial intelligence approach*, **1**, San Mateo, CA: Morgan Kaufmann (1983).
2. Quinlan, J.R., "Induction of decision trees", *Machine learning*, **1 (1)**, 81–106 (1986).
3. González, A., Pérez, R., Verdegay, J. L., "Learning the structure of a fuzzy rule: A genetic approach", *Fuzzy Systems and A.I.*, **3 (1)**, 57–70 (1994).
4. González, A., Pérez, R., "A learning system of fuzzy control rules", in: Herrera, F., Verdegay, J. L. (Eds.), *Genetic Algorithms and Soft Computing, Physica-Verlag, Wurzburg*, 202–225 (1996).
5. Wang, L. and Mendel J.M., "Generating fuzzy rules by learning from examples", *IEEE Transactions on Systems, Man, and Cybernetics*, **22 (6)** (1992).
6. Jang J.R., "ANFIS: Adaptive-Network-Based Fuzzy Inference System", *IEEE Transactions on Systems, Man, and Cybernetics*, **23 (3)** (1993).
7. Mitchell, T., "Machine Learning", *Ed. MacGraw-Hill* (1997).
8. González, A., Herrera F., "Multi-stage genetic fuzzy systems Based on the iterative rule learning approach", *Mathware and Soft Computing*, **4**, 233–249 (1997).
9. Castillo, L., Gonza lez, A., Pe rez, R., Including a simplicity criterion in the selection of the best rule in a genetic fuzzy learning algorithm, *Fuzzy Sets and Systems*, **120 (3)**, 309–321 (2001).
10. González, A., Pérez, R., "Improving the genetic algorithm of SLAVE", *Mathware & Soft Computing*, **16**, 59–70 (2009).
11. González, A., Pérez, R., "SLAVE: A genetic learning system based on an iterative approach", *IEEE Trans. on Fuzzy Systems*, **7 (2)**, 176–191 (1999).
12. Cordón O., Gomide, F., Herrera, F., Hoffmann, F., Magdalena, L., "Ten years of genetic fuzzy systems: Current framework and new trends", *Fuzzy Sets and Systems*, **141 (1)**, 5–31 (2004).
13. Herrera, F., "Genetic fuzzy systems: Taxonomy, current research trends and prospects", *Evolutionary Intelligence*, **1 (1)**, 27–46 (2008).
14. Ishibuchi, H., Nakashima, T., "Effect of rule weights in fuzzy rule-based classification systems", *IEEE Transactions on Fuzzy Systems*, **9 (4)**, 506–515 (2001).
15. Del Jesús, M.J., Hoffmann, F., Navascués, L.J., Sánchez, L., "Induction of fuzzy-rule-based classifiers with evolutionary boosting algorithms". *IEEE Transactions on Fuzzy Systems*, **12 (3)**, 296–308 (2004).
16. Ishibuchi, H., Yamamoto, T., "Comparison of heuristic criteria for fuzzy rule selection in classification problems", *Fuzzy Optimization and Decision Making*, **3 (2)**, 119–139 (2004).
17. Gabryel, M., Rutkowski, L., "Evolutionary learning of mamdani-type neuro-fuzzy systems", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 4029 LNAI*, 354–359 (2006).
18. Hühn, J.C., Hüllermeier, E., "FR3: A fuzzy rule learner for inducing reliable classifiers", *IEEE Transactions on Fuzzy Systems*, **17 (1)**, 138–149 (2009).
19. Stavrakoudis, D.G., Galidaki, G.N., Gitas, I.Z., Theocharis, J.B., "Enhancing the interpretability of Genetic Fuzzy classifiers in land cover classification from hyperspectral satellite imagery", *2010 IEEE World Congress on Computational Intelligence, WCCI 2010, art. no. 5584718* (2010).
20. Alcalá-Fdez, J., Fernández, A., Luego, J. Derrac, J. and García S., "Keel data-mining software tool: Data set repository, integration and algorithms and experimental analysis framework", *Multiple-Valued Logic and Soft Computing*, **17 (2-3)**, 255–287 (2011).
21. González, A., Pérez, R., "Completeness and consistency conditions for learning fuzzy rules ", *Fuzzy Set and Systems*, **96**, 37–51 (1998).
22. Michalski, R. S., Mozetic, I., Hong, J., Lavrac, N., "The multi-purpose incremental learning system AQ15 and its testing application to three medical domains", *Proceedings of AAA-86 Fifth National Conference on Artificial Intelligence*, 1041–1045, Morgan Kaufmann (1986).
23. González, A., Pérez, R., "Selection of relevant features in a fuzzy genetic learning algorithm", *IEEE Trans. on Systems, Man and Cybernetics-Part. B Cybernetics*, **31 (3)**, 417–425 (2001).
24. Castillo L., González A., Pérez R., "Including a simplicity criterion in the selection of the best rule in a genetic fuzzy learning algorithm", *Fuzzy Sets and Systems*, **120 (2)**, 309–321 (2001).
25. González, A., Pérez, R., Valenzuela, A. "Diagnosis of myocardial infarction through fuzzy learning techniques", *Proc. IFSA'95, Sao Paulo*, **1**, 273–276 (1995).
26. González, A., Pérez, R., "A two level genetic fuzzy learning algorithm for solving complex problem", *Proc. IFSA'97, Prague*, 192–197 (1997).
27. González, A., "Improving the genetic algorithm of SLAVE", *Mathware & Soft Computing*, 59–70 (2009).
28. González, A., Pérez, R., Caises,Y., and Leyva E., "An Efficient Inductive Genetic Learning Algorithm for Fuzzy Relational Rules", *International Journal of Computational Intelligence Systems*, 212–230 (2012).
29. O. Dunn,"Multiple Comparisons among means". *Journal of the American Statistical Association*, **56 (293)**, 52-64 (1961).
30. García, D. González, A., Pérez, R., "A filter proposal for including feature construction in a genetic learning algorithm", *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, **20**, 31–49 (2012).

31. García, D. González, A., Pérez, R., "A Feature Construction Approach for Genetic Iterative Rule Learning Algorithm ", *Journal of Computers and Systems Sciences*, **80 (1)**, 101–117 (2014).
32. García, D. González, A., Pérez, R., "A New Iterative Model to Simplify the Knowledge Extracted on a Fuzzy Rule-Based Learning Algorithm", *2013 IEEE International Conference on Fuzzy Systems, Hyderabad July 7-10* (2013).
33. Otero, J., Sánchez, L., "Induction of descriptive fuzzy classifiers with the Logitboost algorithm", *Soft Computing*, **10 (9)**, 825–835 (2006).
34. Quinlan J.R., "C4.5: Programs for Machine Learning", *Morgan Kauffman Publishers* (1993).
35. Domingos P., Pazzani M., "On the optimality of the simple Bayesian classifier under zero-one loss", *Machine Learning*, **29**, 103–137 (1997).
36. Mansoori, E.G., Zolghadri, M.J., Katebi, S.D., "SGERD: A Steady-State Genetic Algorithm for Extracting Fuzzy Classification Rules From Data", *Transactions on Fuzzy Systems*, **16 (4)**, 1061–1071 (2008).