Including a simplicity criterion in the selection of the best rule in a genetic fuzzy learning algorithm ☆

Luis Castillo, Antonio González *, Raúl Pérez

Departamento de Ciencias de la Computación e Inteligencia Artificial, E.T.S. de Ingeniería Informática Universidad de Granada, 18071-Granada, Spain

Received June 1998; received in revised form December 1998

Abstract

Learning algorithms can obtain very useful descriptions of several problems. Many different alternative descriptions can be generated. In many cases, a simple description is preferable since it has a higher possibility of being valid in unseen cases and also it is usually easier to understand by a human expert. Thus, the main idea of this paper is to propose simplicity criteria and to include them in a learning algorithm. In this case, the learning algorithm will reward the simplest descriptions. We study simplicity criteria in the selection of fuzzy rules in the genetic fuzzy learning algorithm called SLAVE. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Machine learning; Fuzzy rules; Genetic algorithms

1. Introduction

Genetic algorithms (GAs) have been used for developing learning algorithms [3,4,6,19,21–23,30,31]. Different approaches can be used for this purpose. The iterative approach [9,11] allows the easy development of learning algorithms. Moreover, the iterative approach has its roots in classical machine learning algorithms, but it uses a GA as a search tool.

Recently, an inductive learning algorithm based on the ideas of fuzzy logic (as knowledge representation) and genetic algorithms (as search algorithms) has been proposed [9,14,12]. This system, called Structural Learning Algorithm in Vague Environment (SLAVE), has proved to be very useful [13–15] in learning both, systems with discrete classes and systems with continuous classes.

The learning algorithm uses a genetic algorithm for obtaining both the structure of the rule (predictive variables) and the value of these variables. The search algorithm is guided by a heuristic function that is composed of a degree of completeness and a degree of consistency for each rule. However, different rules (with different structure and values) can have the same heuristic value and the learning algorithm selects one of them at random. In this case, it will be interesting to add additional criteria in order to obtain particular kinds of desirable rules. In this paper, we investigate the use of simplicity criteria. The idea is, on the one hand, to obtain simpler and more
understandable rules, and on the other hand, to obtain rules with a higher accuracy on unseen examples.

The methodology consists of defining a simplicity measure with respect to the variables and another simplicity measure with respect to the values. Both measures will be considered in the fitness function of the genetic algorithm. The main criteria is still the consistency and completeness of the rule, but in a tie situation, we use the simplicity criterion in variables and in a new tie situation we will use the simplicity criteria in values.

In the next section, we make a general description of SLAVE, Section 3 describes the genetic algorithm of SLAVE. Section 4 shows two simplicity criteria and includes them in the heuristic function of the genetic algorithm. Section 5 explores and analyzes different alternative optimizing criteria, and finally the last section shows the behaviour of SLAVE on different databases.

2. A general description of SLAVE

Structural Learning Algorithm in Vague Environment (SLAVE) [9,14,12] is an inductive learning algorithm based on the following iterative approach:
1. Use a GA to obtain ONE RULE for the system.
2. Incorporate the rule into the final set of rules.
3. Penalize this rule.
4. If the current set of rules is adequate to represent the examples in the training set, the system returns the set of rules as the solution. Otherwise go to step 1.

The main task of SLAVE consists in finding the best fuzzy rule (step 1) that describes the relationships between variables and classes. This task may be formalized using a search problem. There are several search algorithms but in SLAVE we have used genetic algorithms (GA) since they have shown to provide a powerful adaptive search technique in large and complex spaces. This problem is focused by fixing a class of the consequent variable and searching the best antecedent for this class. Once this rule has been learnt, a very easy way to penalize this rule (step 3), and thus be able to learn new rules, consists in eliminating all those examples that are covered to a certain degree by the rule from the training set.

The knowledge representation in SLAVE is a rule-based model in which each rule has the following structure:

\[ IF \; X_1 \; is \; A_1 \; and \; \ldots \; and \; X_n \; is \; A_n \; THEN \; Y \; is \; B, \]

where each variable \( X_i \) has a referential set \( U_i \) and takes values in a finite domain \( D_i \), for \( i \in \{1, \ldots, n\} \). The referential set for \( Y \) is \( V \) and its domain is \( F \). The value of the variable \( Y \) is \( B \), where \( B \in F \) and the value of the variable \( X_i \) is \( A_i \), where \( A_i \in P(D_i) \) and \( P(D_i) \) denotes the set of subsets of \( D_i \). We suppose that the domains for the variables, \( D_i \), are a set of fuzzy sets. Therefore, we are using linguistic variables [32] and fuzzy rules in our model, and we are learning a qualitative description of the original problem. This rule model allows us to both identify systems with a discrete consequent variable and to identify systems with a continuous consequent variable.

A basic component of the iterative approach of SLAVE is the selection of the best rule. The concept of the best rule is based on the notion of consistency and completeness. The consistency condition of a rule is associated with the examples from the training set that satisfy the antecedent of this rule and are correctly classified by the rule. On the other hand, the completeness of a rule is associated with the number of positive examples covered by the rule. In this sense, the best rule on a training set and a fixed class, is the one that is consistent and affects the highest number of examples.

In crisp models, the concepts of completeness and consistency are clear, however when we work with fuzzy examples or/and fuzzy rules, the concepts above must be reformulated in order to adapt them to the special characteristics of the linguistic labels. Thus, in [12] we proposed a adaptation of these concepts on fuzzy models called the degree of completeness and the degree of soft consistency for a rule, where both definitions use the concept of number of positive and negative examples as defined in [17,12].

**Definition 2.1.** The degree of completeness of a rule \( R \) (with a consequent \( Y = B \)) is defined as

\[ \Lambda(R) = \frac{n^+(R)}{n_B} \]
where \( n^+(R) \) is the number of positive examples of the rule \( R \) and \( n_B \) is the number of examples in the training set of the class \( B \).

The soft consistency degree [12] is based on the possibility of admitting some noise in the rules. The definition is based on two main ideas: to extend the crisp consistency definition to fuzzy rules and moreover to make gradual this measure. Thus, in order to define the soft consistency degree we use the following sets:

\[
\Delta^k = \{ R \mid n^-(R) < kn^+(R) \}
\]

with \( k \in (0,1] \) and

\[
\Delta^0 = \{ R \mid n^-(R) = 0 \}
\]

which represents in the general case the set of rules having a number of negative examples strictly less than a percentage (depending on \( k \)) of the positive examples and the complete consistent rules with \( k = 0 \).

**Definition 2.2.** The degree to which a rule \( R \) with \( n^+(R) > 0 \) satisfies the soft consistency condition is

\[
\Gamma_{k_1k_2}(R) = \begin{cases} 
1 & \text{if } R \in \Delta^{k_1}, \\
\frac{k_2n^+(R) - n^-(R)}{n^+(R)(k_2 - k_1)} & \text{if } R \notin \Delta^{k_1} \text{ and } R \in \Delta^{k_2}, \\
0 & \text{otherwise},
\end{cases}
\]

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

Thus, SLAVE fixes a \( B \) value for the consequent variable and taking into account the training set, selects the best antecedent \( A \) verifying simultaneously to a high degree, the completeness and the soft consistency condition. Therefore, the rule selection in SLAVE can be solved by the following optimization problem:

\[
\max_{A \in D} \{ A(R_B(A)) \times \Gamma_{k_1k_2}(R_B(A)) \},
\]

where \( D = P(D_1) \times P(D_2) \times \cdots \times P(D_n) \), \( \times \) represents the product operation and \( R_B(A) \) represents a rule with antecedent value \( A = (A_1, \ldots, A_n) \in D \) and consequent value \( B \in F \).

SLAVE uses a GA in this process. This genetic algorithm for a fixed class searches the best antecedent \( A \in D \) and maximizes the previous expression. The learning algorithm repeats this process until the complete set of rules has been obtained.

### 3. The genetic algorithm of SLAVE

In order to find the best antecedent for a particular class, the genetic algorithm of SLAVE [15,16,18] has been designed to work on two components: the first one represents the relevant variables of the rule and the second one represents the particular values of each variable. Thus, the chromosome of the genetic algorithm codifies information from two different sources: relevant variables and particular values. 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.

Now, in order to describe the genetic algorithm, we briefly explain its main components.

#### 3.1. Representation of an element of the population

Let us suppose we have \( n \) possible antecedent variables \( X_1, \ldots, X_n \) and each one of them has an associated fuzzy domain \( D_i \) with \( m_i \) components. In order to find the best rule, SLAVE fixes a class and then searches for the best antecedent for this class. Therefore, the genetic code must contain information on the possible antecedent variables and its values. We separate this code into two separate components: information on the variables present in the rule (relevant
variables of the rule) and information on the value of these variables:
- A variable chromosome.
- A value chromosome.

The variable chromosome (VAR) codifies the relevant/irrelevant variables for the particular rule. We use a real code with \( n + 1 \) components, in which the \( i \)th element of the \( j \)th chromosome \( \tau_C(X^j_i) \) \((i = 1, \ldots, n)\) contains a value between 0 and 1 representing the relevance degree of the \( i \)th variable with respect to the particular class, 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 value between 0 and 1 representing a threshold of activation associated to the \( j \)th chromosome. A variable \( X_i \) will be considered as a component of the antecedent of the rule for a particular class if \( \tau_C(X^j_i) \geq T_j \). Otherwise, the variable will be considered as irrelevant for the rule. The next section explains how these values are obtained for the first population. The genetic algorithm will change these initial values in order to obtain a better estimation of them.

The value chromosome (VAL) codifies any elements of \( P(D_1) \times \cdots \times P(D_n) \) and we use a vector of \( m_1 + \cdots + m_n \) zero-one components (active or non-active values), such that,

\[
\text{component}(m_1 + \cdots + m_{r-1} + s) = \begin{cases} 
1 & \text{if the } s \text{th element in the domain } D_r \\
0 & \text{otherwise,}
\end{cases}
\]

where \( m_r \) are the number of components of the domain \( D_r \) of the variable \( X_r \).

**Example 1.** 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 Fig. 1. In this case, \( m_1 = 3, m_2 = 5, m_3 = 2 \). Let us suppose the following values for the component vector:

\[
\begin{align*}
\text{component}(1) &= 1, & \text{component}(6) &= 1, \\
\text{component}(2) &= 1, & \text{component}(7) &= 0, \\
\text{component}(3) &= 0, & \text{component}(8) &= 0, \\
\text{component}(4) &= 1, & \text{component}(9) &= 0, \\
\text{component}(5) &= 1, & \text{component}(10) &= 1,
\end{align*}
\]

and let us suppose that the relevance degrees for the \( C \) class of an individual of the population are

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

and by taking the value 0.6 as the threshold then the code

\[
\text{VAR}(0.5, 0.7, 0.1) T_{\text{threshold}} 0.6 \\
\text{VAL}((110)(11100)(01))
\]

represents the following antecedent:

\[
X_2 = \{A_{21}, A_{22}, A_{23}\}
\]

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 \( (0.5 \leq 0.6) \),
- \( (11100) \) represents \( \{A_{21}, A_{22}, A_{23}\} \) and is included in the antecedent since \( X_2 \) is activated in the variable chromosome \( (0.7 \geq 0.6) \),
- \( (01) \) represents \( \{A_{32}\} \) but \( X_3 \) is not included in the antecedent since it is not activated in the variable chromosome \( (0.1 \leq 0.6) \).

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

### 3.2. Generating the first population

In the genetic algorithm the value chromosome is obtained by selecting examples at random from the class that must be learned and assigning the most specific antecedent that best covers it. This antecedent is
made up of only one label for each antecedent variable. The label for each antecedent variable is the one that gives the highest degree of membership for each component in the example. For example, by using the domains of Fig. 1, the example (r_1, r_2, r_3) of Fig. 2 generates the chromosome (001)(00100)(10) corresponding to the antecedent $X^*_1$ is $A_{13}$ and $X^*_2$ is $A_{23}$ and $X^*_3$ is $A_{31}$.

In the generation of the initial population the variable chromosome is built up by using an information function of each variable with respect to the fixed class on the training examples. The value $\tau_C(X)$ is calculated using the following expression:

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

with

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

where $C$ is a fixed class of the consequent variable $Y$ and

$$H(X, Y) = \sum_x \sum_y p(x, y) \log_2 p(x, y)$$

is the entropy of Shannon, with $p()$ being a probability measure. In these calculations we use the formulation proposed in [10] in order to obtain the probability on fuzzy events. The value of $\tau_C(X'_j)$ is the same for each element of the population and the genetic algorithm will change this initial value in a different way for each element. Initially, the threshold of activation is randomly defined for each element of the population. $T_j$ takes a value in the interval

$$\left[ \min_i \tau_C(X'_j), \max_i \tau_C(X'_j) \right].$$

Both $\tau_C(X'_j)$ and $T_j$ are affected by the genetic operators during the evolution of the genetic algorithm and a different threshold is considered in each chromosome. Therefore, the initial relevance degree, calculated from previous formulas, will be modified during the evolution process until it reaches an appropriate value.

3.3. Evaluation function (Fitness)

The aim of the genetic algorithm is to find the best rule. The best rule is defined here as the one that simultaneously verifies the completeness and consistency degree to the highest degree. Using the previous definitions for a rule $R_B(A)$ and a set of training examples $E$, we can define the evaluation function $eval_{k_1k_2}$ as

$$eval_{k_1k_2}(R_B(A)) = A(R_B(A)) \times \Gamma_{k_1k_2}(R_B(A)),$$

where $\times$ is the product operator.

3.4. Genetic operators

In the genetic algorithm, the calculation of the selection probability follows a linear ranking [1], with $\eta_{min} = 1.5$, and the sampling algorithm is the roulette wheel selection [8].

The elitist strategy [5] is considered as well, in order to ensure the best performing chromosome always survives intact from one generation to the next one. This is necessary since the best chromosome may disappear, due to crossover or mutation.

Since the code of the genetic algorithm has two components, one with a real code (variable chromosome) and the other one with a binary code (value chromosome), we need to use different genetic operators for each structure.

With respect to variable chromosome (real code), we use the non-uniform mutation [7] that alters a gene with a certain probability and the $BLX_{z}$ crossover operator [24], that permits the combination of two chromosomes for generating a new chromosome. Finally,
the value of \( \tau_C \) is recalculated whenever the algorithm obtains a rule since due to the architecture of SLAVE, when this happens, the examples covered by the rules are eliminated and the training set is changed.

With respect to the value chromosome (binary code), we use the following genetic operators:
- Crossover operator over two points.
- Mutation operator.
- AND and OR operators.
- Rotation operator.
- Generalization operator.

The two first operators are well-known standard operators. The AND and OR operator exchange genetic information in a similar way to the crossover operator but using the AND and OR operators. The rotation operator [14] is a modified version of the traditional inversion operator that we propose in order to include a higher diversity in the searching problem. This operator takes a cutoff point in an element of the population and changes the position of the two segments. Finally, the generalization operator [14] is a specific operator that attempts to make the rules returned by the learning process clearer and more understandable.

3.5. Termination condition

The genetic algorithm tries to search the best antecedent for a fixed class. Thus, the idea is to finish when there is strong reason to believe that we have really obtained the best rule. Thus, in the implementation of the termination condition we distinguish if we are extracting rules for a class in which we have at least one learned rule or we are learning rules for a class in which we have no rules. We make a more exhaustive search when we wish 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 from the last population if one of the next 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 for, at least, a fixed number of iterations and rules with this consequent having already been obtained.
- There are no rules with the value of the consequent which have been obtained before, the fitness function does not increase the value for a fixed number of iterations and the current best rule can eliminate at least one example from the training set.

4. The simplicity criteria

A combination of two criteria based on the extension and the adaptation of the two classical conditions is used for the evaluation of the rules in the iterative method proposed in the SLAVE learning algorithm: the consistency and completeness conditions. In SLAVE, the consistency degree determines the level to which the examples of the training set, that are covered by the antecedent of a rule, satisfy this rule. On the other hand, the completeness degree determines the strength of a rule, that is, how many examples of the class that is being learnt support the validity of the antecedent of the rule. The combination of these two measures by a product operator permits us to obtain rules that are both consistent and complete simultaneously. The use of the previous criterion has proved to be very useful for learning when used on different kinds of problems [13–15]. However, with this heuristic criterion many different rules can have the same evaluation function value. The original SLAVE genetic algorithm selects at random one of these rules. In this work, we study the behaviour of SLAVE with the inclusion of new criteria in order to discriminate among these rules instead of the random selection. The general criteria say: we prefer, among the best rules, those which are simpler and more understandable.

With the inclusion of these new criteria, we wish to obtain different goals:
(i) to improve the comprehension of the acquired knowledge, and
(ii) to obtain a set of rules which have a higher degree of accuracy on unseen examples.

Now, we need to introduce new concepts to formalize these ideas.

**Definition 4.1.** Let \( R_\beta(A) \) be a rule in which \( A = (A_1, \ldots, A_n) \) and \( A_i \in P(D_i) \). We say that the variable \( X_i, i = 1, \ldots, n \) is irrelevant in this rule if \( A_i = D_i \).

The number of irrelevant variables of a rule will be denoted as \( i(R_\beta(A)) \).
This definition is based on the treatment of rules made in SLAVE where the disjunction of adjacent values is taken as the convex hull of the fuzzy labels (see [14,12] for details).

By using this definition, we can propose the concept of simplicity of a rule. A rule is simpler than an other one if it has a lower number of relevant variables. Therefore, we propose the following definition.

**Definition 4.2.** Let \( R_b(A) \) be a rule, the simplicity degree in variables of this rule is

\[
\text{svar}(R_b(A)) = \frac{i(R_b(A))}{n},
\]

where \( n \) is the number of possible antecedent variables.

The second concept is presented through the following example.

**Example 2.** Let \( X_2 \) be a variable with an associated ordered domain \( D_2 \) (see Fig. 1). 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[12] whereas the second one does not have a similar interpretation. If both values are equally appropriate for describing the value of a variable, we prefer the first one since it is easier to understand. The tie situation is generated by a lack of examples covered by the label \( A_{24} \). The preference of the second antecedent is directly related to the generalization principle applied when we have a lack of information.

Thus, we propose a new concept, the stable value.

**Definition 4.3.** Given a particular value \( A_i \in P(D_i) \) for a variable \( X_i \), we say that \( A_i \) is stable if and only if \( A_i \) is composed of an unique sequence of adjacent values of \( D_i \).

**Definition 4.4.** 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 so that \( D_i \) is an ordered domain and \( A_i \) or \( \bar{A}_i \) is a stable value.

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

**Definition 4.5.** Let \( R_b(A) \) be a rule, we define the simplicity degree in values of a rule as

\[
\text{sval}(R_b(A)) = \frac{1 + e(R_b(A))}{1 + p},
\]

where \( p \leq n \) is the number of variables with an ordered associated domain.

Now, we propose a new fitness function composed of three components:

\[
\text{fitness}(R_b(A)) = (\text{eval}_{k_1k_2}(R_b(A)), \text{svar}(R_b(A)), \text{sval}(R_b(A)))
\]

and we use a lexicographical order to select the best rule, that is, we maintain the initial criterion (consistency and completeness), and in a tie situation, we use the simplicity criterion in variables and in a new tie situation we will use the simplicity criterion in values. Thus, the lexicographical order uses

\[
(\max, \max, \max)
\]

as optimizing criteria, that is, to maximize in the first component, in a tie situation, to maximize in the second component and in a new tie situation also to maximize in the third component.

**Example 3.** Let us suppose three variables \( X_1, X_2 \) and \( X_3 \) with domain \( D_1, D_2 \) and \( D_3 \), respectively (see Fig. 1). Let us suppose a fixed consequent \( B \) and three possible antecedents

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

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

and

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

with the same value of consistency \( \times \) completeness, that is

\[
\text{eval}_{k_1k_2}(R_b(A)) = \text{eval}_{k_1k_2}(R_b(A'))
\]

\[
= \text{eval}_{k_1k_2}(R_b(A''))
\]
in this case, the fitness function uses the previous concepts to decide the best antecedent.

- The antecedent $A$ corresponds to
  
  \[
  X_1 = \{A_{11}, A_{13}\} \\
  \text{and } X_2 = \{A_{23}, A_{25}\} \\
  \text{and } X_3 = \{A_{31}\}
  \]
  
  with values
  \[
  svar(R_B(A)) = 0 \quad \text{and} \quad sval(R_B(A)) = \frac{3}{4},
  \]
  since $\{A_{11}, A_{13}\}$ is equivalent to NOT $A_{12}$.

- The antecedent $A'$ corresponds to
  
  \[
  X_1 = \{A_{11}, A_{12}, A_{13}\} \\
  \text{and } X_2 = \{A_{23}, A_{24}, A_{25}\} \\
  \text{and } X_3 = \{A_{31}\}
  \]
  
  with values
  \[
  svar(R_B(A)) = \frac{1}{3} \quad \text{and} \quad sval(R_B(A)) = \frac{1}{2}.
  \]

Then

\[
\text{fitness}(A) \leq \text{fitness}(A'') \leq \text{fitness}(A')
\]

and the best choice is the antecedent $A'$.

5. Exploring a different optimizing criterion

In the previous section we have taken a maximizing criteria related to the different components of the fitness function. The idea is to evaluate the good quality of a rule by using an ordered set of criteria. The most important criteria is the rules which are selected with a high degree of consistency and completeness. Next, in a tie situation the selection of simpler rules, that is, rules with the lowest number of variables, and finally, in a new tie situation we use the understandability criterion. In the first and third criteria, it seems very clear that the best option is to maximize each criteria, but with regard to the second one, we have two alternatives: to maximize or to minimize this component. Obviously, in these cases the behaviour of the system and therefore the learned set of rules will be completely different. We will now briefly analyze the behaviour for each choice:

- first option (max, max, max),
- second option (max, min, max).

The option chosen in the previous section has been to maximize the second component, that is, to select the rule with the least number of variables in the description of the rule. In this case, SLAVE attempts to find a reduced number of variables in such a way that, the system can discriminate among rules of different classes. Thus, the system has a clear tendency towards obtaining more discriminant rules.

In the other option, (max, min, max), SLAVE attempts to find rules which incorporate a higher number of different variables, therefore the system has a tendency towards learning more descriptive rules.

In order to demonstrate these two different behaviours, we apply SLAVE to a decimal digit LED display problem [27]. This is not a problem in which fuzzy knowledge can be included, however it is a problem in which it is very easy to demonstrate both of the different behaviours. In Fig. 3 a LED decimal digit display is shown.

The example set consists of 1000 randomly generated examples, where each example is composed of 24 binary attributes (the seven leds plus 17 irrelevant attributes) and 10 equiprobable classes that correspond to each one of the digits. Using the fitness function with a lexicographical order (max, max, max) the knowledge base obtained is shown in Table 1.

However, when we use the option (max, min, max), the knowledge base obtained is completely different, and can be seen in Table 2.
Table 1
Knowledge base obtained for option (max,max,max)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Antecedent</th>
<th>Consequent</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF Down-right is OFF THEN Class is TWO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Up-right is OFF AND Down-left is OFF THEN Class is FIVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Up-center is OFF AND Mid-center is OFF THEN Class is ONE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Mid-center is OFF AND Down-left is ON THEN Class is ZERO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Up-left is OFF AND Mid-center is ON THEN Class is THREE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Up-right is OFF THEN Class is SIX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Up-center is OFF THEN Class is FOUR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Mid-center is OFF THEN Class is SEVEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF Down-left is ON THEN Class is EIGHT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OTHERWISE Class is NINE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In both cases, SLAVE has eliminated the 17 irrelevant variables in the descriptions of the rules, but while in the first option, only the variables that are needed for distinguishing a class from the rest of the classes appear in each rule, in the other knowledge base, the seven relevant variables appear in the antecedent of the rule with the values that are needed for describing each class.

For example, if we look at the rules that represent the digit two in each rule set, we can see that the Down-right variable only appears in the first one. The number two is the unique digit that with this representation has the value OFF in this led, therefore this variable/value permits us to discriminate this digit from the rest of the digits. However, in Table 2, the antecedent of the rule for the digit two contains the seven relevant variables, the Down-right variable and the other six variables that correspond to the other six leds, perfectly describing the number two (see Fig. 4).

The accuracy obtained by SLAVE on the test sets in both cases is 100% since in this data set, noise has not been considered.

In this domain and with the knowledge bases of Tables 1 and 2, we can see how the rules obtained by the first option are simpler than the rules obtained by the second option. This difference has two interesting aspects: noise dependence and accuracy. With respect to the noise, since the more discriminant rules of Table 1 obtain a reduced number of variables, it is possible to correctly classify a noisy example if the value of the variable included in the rule is also correct. However, a simple change in a led produces a bad classification in the rule base of Table 2. Moreover, the accuracy in the second option is expected to be better than in the first option, since the system has less dependence on the overfitting problem (on a noise data set). Thus, although a descriptive set of rules is very useful for understanding a concept, a system or a process, frequently a discriminant set of rules is better for classification problems. Moreover, the rules in Table 2 are particular cases of rules in Table 1 as can be easily checked. In any case, SLAVE can select both types of rules and we only need to change the kind of optimizing criteria in the second component of the fitness function. The next section shows the behaviour of SLAVE using the first criterion on more complex data sets.

6. Experimental studies

In this section, we study the performance of SLAVE with the inclusion of the new criteria in the goal function of the genetic algorithm. We consider empirical studies that show the improvement in comparison to the original SLAVE, and with other learning algorithms such as CN2 [2], LVQ [20] or C4.5 [28].

The databases used have been obtained from the UCI Repository of Machine Learning Databases and Domain Theories (ftp.ics.uci.edu) [27] and they are the following:

- The IONOSPHERE data [29]. This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high frequency antennas with a total transmitted power to the order of 6.4 kW. The targets were free electrons in the ionosphere. The database is composed of 34 continuous attributes plus the class variable, using 351 examples.

- SOYBEAN database [26]. These data correspond to the information used to develop an expert system for soybean disease diagnosis. There are 19 classes and 35 categorical attributes, some nominal and some ordered. The number of instances is 307 and there are unknown values.
Table 2
Knowledge base obtained for option (max, min, max)

<table>
<thead>
<tr>
<th>Up-center</th>
<th>Up-left</th>
<th>Up-right</th>
<th>Mid-center</th>
<th>Down-left</th>
<th>Down-right</th>
<th>Down-center</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ZERO</td>
</tr>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>ONE</td>
</tr>
<tr>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>TWO</td>
</tr>
<tr>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>THREE</td>
</tr>
<tr>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>FOUR</td>
</tr>
<tr>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>FIVE</td>
</tr>
<tr>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>SIX</td>
</tr>
<tr>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>SEVEN</td>
</tr>
<tr>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>EIGHT</td>
</tr>
</tbody>
</table>

Fig. 4. Description of number two in Table 2.

- WINE recognition data. These data are the results of a chemical analysis of wines from the same region but with different types of grapes, using 13 continuous variables and 178 examples.

For the different databases (except for the SOYBEAN database in which we have used the partition found in the machine learning database repository), we have produced five partitions in training and test sets (70% and 30%, respectively). Furthermore, SLAVE needs to discretize the referential of the continuous variables in order to carry out the learning process. In this experiment, we have used fuzzy domains on these variables, with each fuzzy domain being composed of five fuzzy labels, uniformly distributed on its definition range.

For each database we have run the different versions of SLAVE using the parameters described in Table 3.

Table 3
SLAVE’s parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of iterations</td>
<td>500</td>
</tr>
<tr>
<td>Population size</td>
<td>20</td>
</tr>
<tr>
<td>Mutation probability (binary coded genes)</td>
<td>0.01</td>
</tr>
<tr>
<td>Non-uniform mutation probability (real coded genes)</td>
<td>0.001</td>
</tr>
<tr>
<td>Crossover probability</td>
<td>0.6</td>
</tr>
<tr>
<td>( BLX_\zeta ) crossover probability</td>
<td>0.6</td>
</tr>
<tr>
<td>Value of ( \alpha ) parameter</td>
<td>0.15</td>
</tr>
<tr>
<td>AND operator probability</td>
<td>0.01</td>
</tr>
<tr>
<td>OR operator probability</td>
<td>0.1</td>
</tr>
<tr>
<td>Rotation operator probability</td>
<td>0.01</td>
</tr>
<tr>
<td>Generalization operator probability</td>
<td>0.05</td>
</tr>
<tr>
<td>( t )-norm ( k_1 )</td>
<td>0</td>
</tr>
<tr>
<td>( k_2 )</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4
Comparative results obtained from the initial and modified versions of SLAVE

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Test</td>
<td>Rules</td>
</tr>
<tr>
<td>WINE</td>
<td>89.50</td>
<td>7.6</td>
</tr>
<tr>
<td>SOYBEAN</td>
<td>99.01</td>
<td>27</td>
</tr>
<tr>
<td>IONOSPHERE</td>
<td>89.50</td>
<td>13.8</td>
</tr>
</tbody>
</table>

Section 3 (SLAVE[1]) and with the new version of Section 4, in which we have included the optimizing criteria (max, max, max) (SLAVE[2]). In order to compare these versions, we have selected two mea-
sures: the prediction capacity of the knowledge bases on the test sets and the number of the rules in these knowledge bases. The results represent the average on five executions (each execution has been repeated on each one of the five partitions in the case of the IONOSPHERE and WINE databases) on each one of the previous measures.

This table shows that SLAVE[2] obtains the best results on all databases in prediction capacity and the number of the rules. Furthermore, we can see, this improvement is higher on the WINE and IONOSPHERE databases, in the first an important increase is produced on the prediction capacity (from 89.50 to 96.76) and on the second an important reduction is produced in the number of rules (from 13.8 to 4).

As regards to the SOYBEAN database, we can see that the number of rules in both versions are the same, however, the SLAVE[2] prediction capacity is higher. This is caused by the simplicity criteria included in the fitness function, in this case, the simpler rules obtained by this algorithm have a slightly better behaviour than the rules proposed by SLAVE[1].

In order to compare the quality of the results obtained by SLAVE[2], we have used three well-known learning algorithms (C4.5, CN2 and LVQ). These algorithms represent three different learning methodologies:

- **C4.5**: This is an implementation of the well-known C4.5 classification algorithm based on classification trees and it is included in [28].
- **CN2**: This algorithm inductively learns a set of propositional rules from an example set. The algorithm was proposed in [2] and it is based on the methodology of the learning algorithm of the AQ family [25], and it attempts to improve the behaviour when the example set is affected by noise. The implementation used in this work was developed by Boswell in 1990 and it can be obtained from [http://www.cs.utexas.edu/users/pclark/software.html](http://www.cs.utexas.edu/users/pclark/software.html).
- **LVQ**: This software contains all the necessary programs for the application of the learning vector quantization (LVQ) algorithm in statistical classification or pattern recognition. LVQ is an adaptive learning method based on the Kohonen self-organizing maps [20]. The implementation used in this work is version 3.1 of the LVQ-PAK, available at [ftp://cochlea.hut.fi/pub/lq-pak](ftp://cochlea.hut.fi/pub/lq-pak).

Table 5 shows the accuracy of the five test sets for these learning algorithms. For CN2 and C4.5 we have used the default parameters of each algorithm, and for LVQ we have used as codebook number a third of the number of training examples, $\alpha = 0.03$ and the number of iteration equal to 100 times the number of codebook.

The results obtained by SLAVE[2] are the best of the three proposed databases when these are compared with the results obtained by the previous three learning algorithms. Among them, we can remark that the accuracy of SLAVE[2] on the IONOSPHERE database (92.88) in comparison with the result obtained by LVQ on this database (87.97), is the best one obtained by the other learning algorithms.

Furthermore, when we compare the difference in the number of rules between SLAVE[2] and CN2, which is a learning algorithm which represents the knowledge acquired using rules, we can see, that SLAVE[2] obtains a lower number than CN2 for all databases. A noteworthy result is the IONOSPHERE database where SLAVE obtained four rules and CN2 obtained 16.4 rules.

Therefore, the previous results show that the inclusion of the new criteria in the goal function of the SLAVE learning algorithm leads to an improvement in its prediction capacity and an important reduction in the number of rules of the knowledge base obtained by this system. Furthermore, these knowledge bases behave better on the tested databases than other well-known learning algorithms.

### 7. Concluding remarks

The main component of the SLAVE learning algorithm is a genetic algorithm. In previous works
we have shown that the use of this genetic algorithm permits the learning system to improve the capacity to detect the relevant variables involved in the learning problems.

In this work, we have presented the inclusion of new criteria in the fitness function of the genetic algorithm in order to increase the simplicity and comprehension of the knowledge bases obtained by SLAVE. These criteria have been defined so that decisions can be taken about the rules which have the same value in terms of the consistency and completeness measures established in the same learning algorithm. In these situations, the new criteria requires the system to take the simplicity of a rule into account when making a decision. In order to evaluate the simplicity, we have considered two parameters: the simplicity of the variables, this determines the simplicity of a rule by counting the number of relevant variables that are involved in the antecedent of the rule, and the simplicity of the values, this is determined by evaluating the distribution of the values assigned to the relevant variables.

The use of simplicity criteria in variables and values in the learning algorithm allows SLAVE to obtain different types of rule sets: more descriptive or more discriminant.

Finally, the use of both simplicity criteria allows us to improve the comprehension of the final set of rules and to obtain rule sets with a higher degree of accuracy on unseen examples. Therefore, the addition of a simplicity heuristic in the learning algorithm SLAVE has proved to be very useful for learning fuzzy rule sets.

References