ELSEVIER Fuzzy Sets and Systems 96 (1998) 37–51

FUZZY sets and systems

Completeness and consistency conditions for learning fuzzy rules[1]

Antonio Gonzalez *, Raul Perez

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

Received December 1995; revised August 1996

Abstract

The completeness and consistency conditions were introduced in order to achieve acceptable concept recognition rules. In real problems, we can handle noise-affected examples and it is not always possible to maintain both conditions. Moreover, when we use fuzzy information there is a partial matching between examples and rules, therefore the consistency condition becomes a matter of degree. In this paper, a learning algorithm based on soft consistency and completeness conditions is proposed. This learning algorithm combines in a single process rule and feature selection and it is tested on different databases. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Machine learning; Classification problems; Fuzzy logic; Fuzzy rules; Genetic algorithms

1. Introduction

Inductive learning has been successfully applied to concept classification problems. Usually, the knowledge is represented through rules meaning the relationships between the different problem variables. In this paper, we are interested in studying conditions that allow us to propose learning algorithms capable of learning concept classification rules. Moreover, we are interested in algorithms capable of handling fuzzy information [18] and capable of obtaining fuzzy rules. Several learning algorithms working in fuzzy environments have been proposed in the literature [3, 4, 9, 12, 15–17].

Two conditions that must be satisfied for a learning algorithm to obtain acceptable concept recognition rules were introduced in [11]. These conditions are the completeness and the consistency conditions. The completeness condition states that every example of some class must satisfy some rule from this class. 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, but in real cases both conditions are difficult to satisfy. In our case, we find two problems, first, on real problems we can handle noise-affected examples and therefore it is not always possible to keep the completeness and consistency conditions. Moreover, when we use fuzzy rules we have a second problem since the examples have a partial matching with the rules, and therefore the consistency condition becomes a matter of degree. Thus, we are interested in obtaining soft consistency and completeness conditions.

The basic idea of the proposed learning algorithm consists in, on the one hand, determining the best

* Corresponding author. Tel.: + 34.58.243199; fax: +34.58.243317; e-mail: a.gonzalez@decsai.ugr.es.

[1] This work has been supported by the CICYT under Project TIC95-0453.

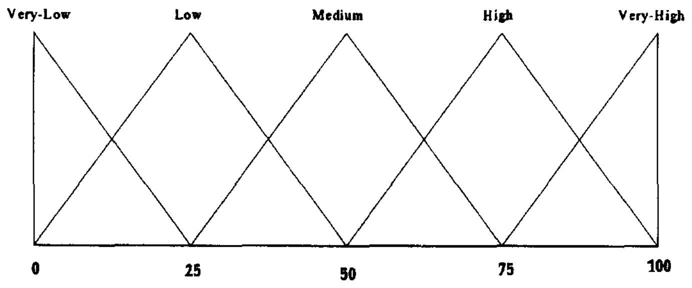

Fig. 1. The fuzzy domain of X_1 and X_2.

rules to represent a set of examples E (rule selection), on the other, obtaining for each rule the relevant variables in order to describe each particular concept (feature selection). Thus, the algorithm starts with all possible antecedent variables and a fixed consequent variable (representing the concept). We want to know which are the rules capable of identifying each value of the consequent variable, and for each rule, the relevant antecedent variables to describe the concept.

The rule selection is carried out by choosing among those rules that satisfy the soft consistency condition and cover the maximum number of positive examples. Thus, we fix a value of the consequent variable and next we select the rule that covers the maximum number of positive examples. Since it is possible that this value may be described for several different rules, we eliminate the examples covered by the previous rule and we repeat the process until the soft completeness condition is verified. These steps are carried out for each value of the consequent variable, and in the end, we have a set of rules that represent the original set of examples, E. In each step, we shall use a genetic algorithm [6] as a search method for the best rule.

The feature selection is based on the use of a particular model of rule that allow us to eliminate non-relevant variables. The rule model, that we use in this work, consists in the left-hand side (the antecedent part) of a conjunction of one or more variables whereas the right-hand side (the consequent part) indicates the value of the classification variable to be assigned to the examples that are matched by the left-hand side of the rule. Each member of the conjunction in the antecedent can have an internal disjunction, i.e., we accept that the value assigned to an antecedent variable in the rules can be a subset of its domain. For example, a rule might take the following expression:

$$\begin{aligned}&\text{If } (X_1 = \text{Very-High}) \text{ and}\\&(X_2 = \text{Very-Low or Low or Medium}) \text{ then}\\&(Y = \text{approximately equal to } 0)\end{aligned} \qquad (1)$$

where the domain of variables X_1 and X_2 has been represented in Fig. 1.

In order to learn the structure of a rule this model is fundamental in the proposed learning algorithm, since at the beginning it will start with all the possible antecedent variables and then it will decide which are the relevant variables for the fixed consequent. In this process, the algorithm uses the aforementioned model of rule to eliminate the non-relevant variables. When the best rule for a class contains a variable having a value made up of all possible values of its domain, then the algorithm has discovered that this variable is not relevant to this consequence and it can be eliminated from the rule. This model of rule has been used in the literature by several authors (see [2, 8, 11] for example). In the fuzzy learning literature, there are not many previous papers suggesting solutions to this difficult problem. The greater part of these papers achieve rules in which all the variables necessarily appear, and the algorithms are not able to eliminate non-relevant variables. Therefore, these algorithms need a prior reduction in the possible variable set. In [10], this task is carried out by an expert that proposes a pre-selection of "possible interesting" features. An exception is [15] in which an algorithm capable of determining the structure of the fuzzy rules is proposed.

One of the main advantages of the proposed learning algorithm is that combines in a single process the rule selection and the feature selection.

Moreover, the aforementioned model of rule is important since it allows us to change, in some cases, the granularity of the domain by combining different elements. For example, the rule shown in (1) assigns the value {Very-Low, Low, Medium} to the variable X_2. This value may be interpreted as a new label "less than or equal to medium". The proposed model of rule therefore allows us to include some structured descriptors [11] in an easy way.

This work deals with the problem of defining completeness and consistency conditions for learning fuzzy rules. The next section is devoted to introduce a soft consistency condition. The completeness condition and the learning algorithm are introduced in Section 3. We have implemented a system called SLAVE with the aforementioned features and in Section 4 we carry out different experimental tests in order to study its predictive performance. Finally, in the last section we propose a modification of the algorithm that attempts to simplify the practical choice of the parameters in SLAVE.

2. The k-consistency condition

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. We propose a weaker definition based on the non-fulfillment of this condition in several cases.

Let us suppose E is a set of examples containing every example

X_1	X_2	X_3	$\dots$	X_n	Y
e_1	e_2	e_3	$\dots$	e_n	$c(e)$

n antecedent variables $X_1, X_2, \dots, X_n$ and a consequent variable Y.

The referential set for each antecedent variable X_i is U_i and the referential set for the consequent variable is V. The domains for the rules will be D_i for the antecedent variables and F for the consequent variable. In this paper, we accept that the sets D_i $\forall i \in \{1, \dots, n\}$ and F are finite sets, and that some of them may be fuzzy domains, i.e., set made up by fuzzy sets in the respective referential sets.

Let e be a crisp example

$$e = (e_1, e_2, \dots, e_n, c(e)),$$

where $e_i \in U_i$ are the values of the antecedent variables X_i and $c(e)$ is the class associated to e. Let Δ be the rule set

$$\Delta = P(D_1) \times P(D_2) \times \cdots \times P(D_n) \times F,$$

where $P(X)$ denotes the set of subsets of X.

Let R be a crisp rule with an antecedent part $ant(R)$ and a consequent part $con(R)$. We can say that the example e adapts to the rule R iff

(i) $(e_1, \dots, e_n) \subseteq ant(R)$,
(ii) $c(e) \subseteq con(R)$.

For example, the example (10, red, 1) adapts to the rule "If X is less than 20 and Y is any colour, then Z is an integer".

By using the adaptation between example and rule, we can classify the examples as negative or positive to a rule. An example is positive if it adapts to the rule, and an example is negative if (i) is a true condition then (ii) is a false condition.

Let E be an example set. By using the concept stated, we can define the following subsets of E:

$$E^+(R) = \{e \in E \mid e \text{ is a positive example to } R\}$$

$$E^-(R) = \{e \in E \mid e \text{ is a negative example to } R\}$$

and the cardinality of these subsets shall be denoted by

$$n_E^+(R) = |E^+(R)|,$$

$$n_E^-(R) = |E^-(R)|.$$

Let $H \subseteq \Delta$ be a set of rules, then the following equivalence is obviously verified

H satisfies the consistency condition

$$\Leftrightarrow n_E^-(R) = 0, \ \forall R \in H.$$

Therefore, the consistency condition implies that the rules selected by the learning algorithm must not have negative examples. In order to weaken this strong condition, the first idea consists in allowing it to have some negative examples, but not many. How many examples can we admit? We can fix a hard threshold, i.e., H could satisfy a new consistency condition if and only if, for some ε, $n_E^-(R) \leqslant \varepsilon\ \forall R \in H$. However, this definition does not take into account the number of positive examples. Perhaps we might be interested in admitting two negative examples for a rule with 20 positive examples, but we do not admit it if the rule only has two positive examples. Therefore we propose the following weak definition of consistency.

Definition 2.1. Let $R \in \Delta$ be a rule and $k \in [0,1]$ be a fixed parameter. We say that the rule R, such that $n_E^+(R) > 0$, satisfies the k-consistency condition if and only if

- $k = 0$ then $n_E^-(R) = 0$.
- $k \in (0,1]$ then $n_E^-(R) < k n_E^+(R)$.

The classical consistency condition is included in this definition as a 0-consistency condition and, in general, the parameter k allows us to weaken this condition. Parameter k may be interpreted as a *noise threshold* since a rule satisfies the k-consistency condition when a noise level less than the 100*k percent of the positive examples, appears in the example set. Parameter k will be called the *consistency parameter*. This definition uses the same noise threshold for all rules. The k-consistency condition allows us to weaken the hard consistency condition gradually through the growth of the parameter k, as the following proposition shows.

Proposition 2.1. *Let $R \in \Delta$ be a rule and $k_1 \leqslant k_2$ then*

$$R \text{ satisfies the } k_1\text{-consistency condition} \Rightarrow R \text{ satisfies the } k_2\text{-consistency condition.}$$

By using this proposition, and if we denote by $\Delta^k \subseteq \Delta$, the set of rules that satisfies the k-consistency condition is then

$$\Delta^{k_1} \subseteq \Delta^{k_2},$$

$\forall k_1, k_2 \in [0,1]$ such that $k_1 \leqslant k_2$.

Therefore, if we increase the value of parameter k, we also increase the size of the search space for the best rules (see Fig. 2). This process might be necessary, since in some cases it is not possible to find good rules in such a small space.

Once we have achieved a weak definition of consistency for crisp examples and rules, we extend this definition to fuzzy examples and rules. In order to obtain this extension, we have to define the set of negative and positive examples of a rule. The basic concept in these definitions is the concept of compatibility between two fuzzy sets.

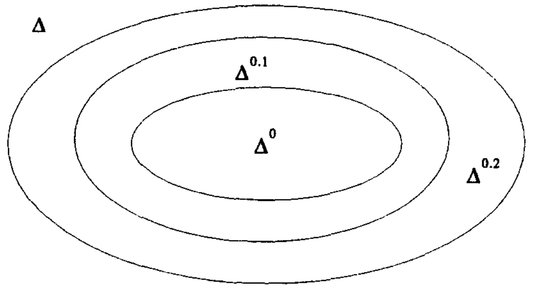

Fig. 2. The k-consistency spaces.

Definition 2.2. Let a and b be two fuzzy sets in a common referential set U, and $*$ a t-norm, we define the compatibility between a and b as the following function:

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

In this paper, we need to widen this definition to calculate the compatibility between two sets of fuzzy sets, and we propose the following definition.

Definition 2.3. Let Dom_1 and Dom_2 be two domains made up by fuzzy sets in a common referential set U, and $C_1 \subseteq Dom_1$ and $C_2 \subseteq Dom_2$ each be a set of fuzzy sets. We define the compatibility between both sets as

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

We use the same symbol σ for both definitions since the last definition is an extension of the former, and both coincide for unitary sets.

From these concepts, we define the adaptation between an example and the antecedent and consequent of a rule, respectively, through the extension to the cartesian product of a normalization of the said compatibility.

Let e be an example composed of different features

$$e = (e_1, e_2, \ldots, e_n, c(e)),$$

where e_i is now a fuzzy set in the referential set U_i, and $c(e)$ is the class assigned to the example e.

Usually, the features of the examples are crisp more than fuzzy, but without difficulty we can extend the kind of examples that the learning algorithm can use. In general, to allow fuzzy examples is useful in order to include some possible imprecision in the data. For example, fuzzy inputs can appear when we have imprecise rules in which we have no complete confidence and we can decide to include them as new examples to be considered. Moreover, fuzzy inputs can appear when we have missing values and we decide to replace them by its complete domain (a set of fuzzy sets) representing that any value is possible.

Let $R_B(A)$ be a rule with antecedent part $A = (A_1, \ldots, A_n)$ $A_i \subseteq D_i$ and consequent part $B \in F$. For each component of the antecedent obviously the following quotient is a measure of possibility

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

and its dual is a measure of necessity

$$Nec(A_i|e_i) = 1 - \frac{\sigma(e_i, \bar{A}_i)}{\sigma(e_i, D_i)},$$

where the negation is considered in this work as the complementary of the set A_i, that is,

$$\bar{A}_i = \{a \in D_i \mid a \notin A_i\}.$$

Thus, we can define the adaptation between an example and the antecedent of a rule combining through a t-norm the different measures of possibility or necessity of each A_i given the evidence supported by the example. Since we have two possible measures, we obtain two adaptation concepts.

- Upper adaptation between example and antecedent of $R_B(A)$:

$$U(e,A) = Poss(A_1|e_1) * Poss(A_2|e_2) * \cdots * Poss(A_n|e_n).$$

- Lower adaptation between example and antecedent of $R_B(A)$:

$$L(e,A) = Nec(A_1|e_1) * Nec(A_2|e_2) * \cdots * Nec(A_n|e_n).$$

In the same way, we define the adaptation between the example and the consequent of the rule as

- Upper adaptation between example and consequent of $R_B(A)$:

$$U(e,B) = \frac{\sigma(c(e), B)}{\sigma(c(e), F)}.$$

- Lower adaptation between example and consequent of $R_B(A)$:

$$L(e,B) = 1 - \frac{\sigma(c(e), \bar{B})}{\sigma(c(e), F)}.$$

Obviously, the following inequalities are verified:

$$L(e,A) \leqslant U(e,A) \quad \forall e \in E,$$

and

$$L(e,B) \leqslant U(e,B) \quad \forall e \in E.$$

These inequalities show that we have two possible adaptation concepts, the first one is a stricter definition based on a measure of necessity and the second one is a wider definition based on a measure of possibility.

From these concepts we can define the positive and negative example set given a rule.

Definition 2.4. The lower and upper set of positive examples for the rule $R_B(A)$ are the following fuzzy subsets of E,

$$E_L^+(R_B(A)) = \{(e, L(e,A) * L(e,B)) \mid e \in E\},$$

$$E_U^+(R_B(A)) = \{(e, U(e,A) * U(e,B)) \mid e \in E\},$$

where $A = (A_1, \ldots, A_n)$, and $*$ is a t-norm.

Definition 2.5. The lower and upper set of negative examples for the rule $R_B(A)$ are the following fuzzy subsets of E,

$$E_L^-(R_B(A)) = \{(e, L(e,A) * L(e,\bar{B})) \mid e \in E\},$$

$$E_U^-(R_B(A)) = \{(e, U(e,A) * U(e,\bar{B})) \mid e \in E\},$$

where $A = (A_1, \ldots, A_n)$, and $*$ is a t-norm.

Obviously,

$$E_L^+(R_B(A)) \subseteq E_U^+(R_B(A)),$$

$$E_L^-(R_B(A)) \subseteq E_U^-(R_B(A)).$$

Usually, the domain of the consequent variable in concept classification problems, consists in nominal descriptors, i.e., the domain is formed by independent symbols or names, and therefore no structure is assumed to relate the values in the domain. In this case, the said definition may be simplified, since $L(e,B) = U(e,B) = 1$ if $c(e) = B$ and 0 otherwise, and, for example, the upper set of positive examples becomes

$$E_U^+(R_B(A)) = \{(e, U(e,A)) \mid e \in E \text{ and } c(e) = B\}.$$

Once we have definitions for positive and negative examples of a rule, we can use a counter to determine the number of positive and negative examples for a rule, which are the concepts needed in the k-consistency condition. Thus, let us denote $|A| = \sum_{u \in U} \mu_A(u)$, as the cardinality of a fuzzy subset. By using this, we obtain the following definitions:

- Number of positive examples on $R_B(A)$
  - lower value $n_L^+(R_B(A)) = |E_L^+(R_B(A))|$,
  - upper value $n_U^+(R_B(A)) = |E_U^+(R_B(A))|$.

  Obviously,

$$n_L^+(R_B(A)) \leqslant n_U^+(R_B(A)).$$

- Number of negative examples on $R_B(A)$
  - lower value $n_L^-(R_B(A)) = |E_L^-(R_B(A))|$,
  - upper value $n_U^-(R_B(A)) = |E_U^-(R_B(A))|$.

  Obviously,

$$n_L^-(R_B(A)) \leqslant n_U^-(R_B(A)).$$

Now, Definition 2.1 can be also applied to fuzzy rules. However, since we have two concepts for positive and negative examples of a rule, we also have two different consistency definitions for fuzzy rules: a *lower consistency condition* (using n_L^+ and n_L^-) and an *upper consistency condition* (using n_U^+ and n_U^-). We denote by Δ_L^k and Δ_U^k, the set of rules that satisfies the lower and upper k-consistency conditions, respectively. These definitions are the basis for the learning algorithm that we propose in the next section.

3. Learning algorithm and completeness condition

As we said in the introduction we are interested in determining the best rules to represent the set of examples E. The complete space of antecedents for a fixed consequent is the set Δ. But now, we shall choose among those rules that satisfy the k-consistency condition, with $k \in [0,1]$ being a fixed parameter. Therefore, the search space is $\Delta^k \subseteq \Delta$. Thus, first we fix a value of the consequent variable and next we select the rule of Δ^k that covers the maximum number of positive examples. We eliminate the examples covered by the previous rule and we repeat the process until we have examples of this class. These steps are carried out for each value of the consequent variable, and finally, we obtain a set of rules that represents the original set of examples E.

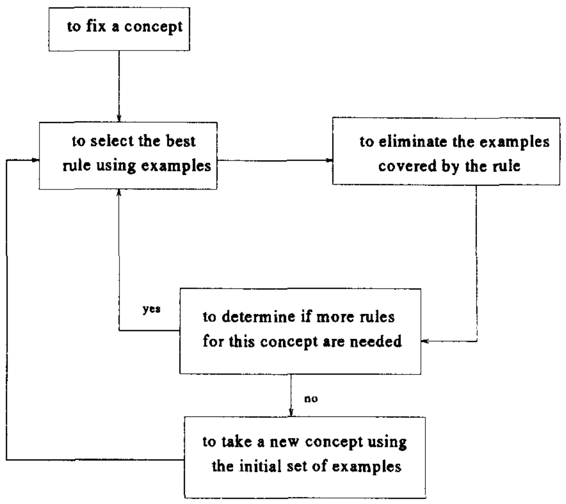
to fix a concept
to select the best rule using examples
to eliminate the examples covered by the rule
yes
to determine if more rules for this concept are needed
no
to take a new concept using the initial set of examples

Fig. 3. The rule selection.

The basic step in the algorithm that we propose consists in obtaining rules covering the maximum number of examples in the training set E, i.e., for each value $B \in F$, the main problem is

$$\max_{A \in D}\{n^+(R_B(A)) \mid R_B(A) \in \Delta^k\}, \qquad (2)$$

where in the definition of n^+ and n^-, we can use the lower or upper definitions from the previous section, and $D = P(D_1) \times P(D_2) \times \cdots \times P(D_n)$.

In order to solve the problem of obtaining the rules covering the maximum number of examples, we need to clarify the concept of the set of examples covered by a fuzzy rule. We know that an example is covered by a rule to a certain degree, therefore, in a sense, a rule covers all the examples, but we are interested in the examples covered with the higher degree. Therefore, we propose the following definition.

Definition 3.1. Let R be a rule and E a set of examples. The subset of E representing the examples λ-covered by R is the λ-cut of $E^+(R)$. λ will be called the covering parameter.

We said earlier that the algorithm must repeat the process of searching for the best rule and eliminating the examples covered by it while there are examples of the present class. Really, this condition is equivalent to the completeness condition given in the previous section and it is very difficult to satisfy when we work with noisy or fuzzy examples. Thus, we propose a weaker condition that we call weak completeness condition for a class. The latter attempts to determine when the number of examples covered for a set of rules on a fixed class is sufficient to represent such a class.

In order to propose the definition, let us suppose

$$T = \{R_1, R_2, \ldots, R_s, R_{s+1}\}$$

an ordered set of rules for a class C, being R_1 the first rule and R_{s+1} the last rule. Let E be the initial set of examples, and let $C_\lambda(R_i)$ be the set of examples λ-covered by the rule R_i. If we denote by

$$E^i = E - \bigcup_{j=1}^{i} C_\lambda(R_j),$$

we can say that the learning algorithm must obtain

- the rule R_1 using E,
- the rule R_2 using E^1,
- the rule R_3 using E^2,

and in general the rule R_i is obtained using E^{i-1}.

Definition 3.2. We say that the set of rules $T = \{R_1, R_2, \ldots, R_s, R_{s+1}\}$ of the class C verifies the weak completeness condition if and only if

$$R_{s+1} \notin \Delta^k$$

or

$$R_{s+1} \in \Delta^k \quad \text{and} \quad C_\lambda(R_{s+1}) = \emptyset.$$

Obviously, this definition is associated to an algorithm that generates an ordered set of rules, and the order corresponds to an optimality criterion, that is, R_i is a better rule (in some sense) that R_{i+1}.

The last rule R_{s+1} is really only used to conclude that the number of rules is enough to represent the class, but it is not included in the set of rules of this class, that is, the output of the learning algorithm is $\{R_1, R_2, \ldots, R_s\}$.

In order to give a detailed description of the learning algorithm, first we have to solve the associated optimization problem (2) by means of a procedure called *GENETIC*, which basically uses a genetic algorithm to solve the best rule search problem.

3.1. The genetic algorithm

Genetic algorithms (GAs) are theoretically and empirically proven to provide robust search capabilities in complex spaces, offering an interesting approach to problems requiring efficient and effective search [6].

In order to use a GA to solve problem (2), we need to define the main components of our problem in the common formulation of genetic algorithms. The final genetic algorithm adapted to solve (2) has been called *GENETIC*. First, we need a representation of the potential problem solution, i.e., the antecedent of the rules. In [7] we propose the following binary coding:

If the database contains n possible antecedent variables and each one of them has a fuzzy domain D_i associated with m_i components

$$X_i \to D_i = \{A_{i1}, \ldots, A_{im_i}\},$$

then we use the following method of coding any elements of $P(D_1) \times P(D_2) \times \cdots \times P(D_n)$, a vector of $m_1 + m_2 + \cdots + m_n$ zero-one components, such that,

$$component(m_1 + \cdots + m_{r-1} + s) = \begin{cases} 1, & \text{if the } s\text{th element of the domain } D_r \text{ is a value of the } X_r \text{ variable,} \\ 0, & \text{otherwise.} \end{cases}$$

Example 1. Let us suppose we have three variables X_1, X_2 and X_3, such that the fuzzy domain associated with each one is

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

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

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

In this case, the vector 1010010111 represents the following antecedent

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

and

X_3 is $\{A_{31}, A_{32}\}$.

Since X_3 takes all the elements in the domain D_3, the said antecedent is equivalent to

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

Another important component of the genetic algorithm is the method for creating the initial solution population. In *GENETIC* we use a procedure for obtaining antecedents with a high possibility to guide the search toward good solutions. The procedure consists in taking a subset of examples at random among those with the current consequent and that they have not been eliminated yet. For each one of these examples, we select the most specific antecedent having the highest adaptation with the example.

For example, let us suppose three variables X_1, X_2 and X_3 with the associated fuzzy domain described in Fig. 1, the same for all the variables. Thus, if we choose the example (26, 54, 100), the procedure will generate the antecedent corresponding to binary code 00100 00100 00001.

The *GENETIC* procedure uses the following fitness function that measures the power of the rule (positive examples) whenever the rule satisfies the k-consistency condition.

$$fitness(R_B(A)) = \begin{cases} n^+(R_B(A)) & \text{if } R_B(A) \in \Delta^k, \\ 0 & \text{otherwise,} \end{cases}$$

where the number of positive examples and the definition of Δ^k have been achieved with the same adaptation function. Thus, in fact, we have two different fitness functions, one based on the lower consistency condition and the other one based on the upper consistency condition. From now on, we assume that the choice of the consistency type is an additional parameter of the genetic algorithm and we do not indicate the specific type selected.

The *GENETIC* procedure uses a set of genetic operators: the ranking linear selection, the mutation operator, the two point crossover operator and a self-crossover operator. This last operator is a modification of the crossover operator that uses only one element in the population and it allows us to explore new space zones. Moreover, we have considered an elitist model.

Finally, the genetic algorithm ends, and gives as output the best rule founded in the last population, if at least one of the following sentences is true:

- the number of iterations is greater than a fixed limit.
- the fitness function of the best rule of the population does not increase its value during at least a fixed number of iterations and previously rules with this consequent already have being obtained.
- there are no rules with this value of the consequent yet, 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, with λ being the covering parameter.

The input of the *GENETIC* procedure is a set of examples E and a value of the consequent variable $B \in F$, and the output is a single rule $R_B(A) \in \Delta^k$ representing the k-consistent rule with consequent B with the largest number of positive examples, and corresponding to the best rule of the last population of the genetic algorithm.

3.2. The learning algorithm

Now, by using the consistency and completeness conditions, the *GENETIC* optimization procedure and the previous general description, we can state the following learning algorithm:

Learning Algorithm

1. Let $RULES$ be the set of rules, at the beginning $RULES$ is empty, and let E be the training set of examples.
2. Repeat for each $B \in F$
   - 2.1 Assign the set E to $EXAMPLES$.
   - 2.2 Run the *GENETIC* procedure, with the input, $EXAMPLES$ and B. Let R be the rule obtained as the output of this procedure.
   - 2.3 While the set of rules of class B does not satisfy the weak completeness condition, add R to $RULES$, eliminate the examples λ-covered by R from $EXAMPLES$ and go to step 2.2. Otherwise, take another value of the consequent variable and go on to step 2.
3. Give the set $RULES$ as the output of the algorithm.

This learning algorithm has as input a set of examples, a set of parameters and a basic structure (the consequent variable and a set of all possible antecedent variables), and generates as output a set of rules that satisfies the k-consistency and the weak completeness condition.

It would be worthwhile studying some properties of this algorithm, which would demonstrate its overall behaviour.

3.3. Properties

Firstly, we attempt to discover whether the algorithm can generate contradictory rules, i.e., rules which have the same antecedent but different consequents. The answer is negative.

Lemma 3.1. *Let RULES be the set of rules generated by the learning algorithm and $R_{B_1}(A), R_{B_2}(A) \in RULES$, with $B_1 \neq B_2$ then*

$$n^-(R_{B_1}(A)) \geqslant n^+(R_{B_2}(A)).$$

Proof. The positive and negative example sets were defined using a lower or an upper measure, therefore this result must be proved for each measure.

- By using the upper case:

$$U(e,\bar{B}_1) = \frac{\sup_{b \in \bar{B}_1} \sigma(c(e),b)}{\sigma(c(e),F)} \geqslant \frac{\sigma(c(e),B_2)}{\sigma(c(e),F)} = U(e,B_2).$$

Therefore,

$$n_U^-(R_{B_1}(A)) = \sum_{e \in E} U(e,A) * U(e,\bar{B}_1) \geqslant \sum_{e \in E} U(e,A) * U(e,B_2) = n_U^+(R_{B_2}(A)).$$

- By using the lower case: In a similar way it is very easy to show that $U(e,\bar{B}_2) \geqslant U(e,B_1)$, therefore $L(e,\bar{B}_1) = 1 - U(e,B_1) \geqslant 1 - U(e,\bar{B}_2) = L(e,B_2)$ and then $n_L^-(R_{B_1}(A)) = \sum_{e \in E} L(e,A) * L(e,\bar{B}_1) \geqslant \sum_{e \in E} L(e,A) * L(e,B_2) = n_L^+(R_{B_1}(A))$.

Proposition 3.1. *If $R_{B_1}(A) \in RULES$ then $R_{B_2}(A) \notin RULES$ $\forall B_2 \in F$ such that $B_1 \neq B_2$.*

Proof. Let us suppose that both $R_{B_1}(A), R_{B_2}(A) \in RULES$. In this case both rules must verify the k-consistency condition, i.e.,

$$n^-(R_{B_1}(A)) < kn^+(R_{B_1}(A)),$$

$$n^-(R_{B_2}(A)) < kn^+(R_{B_2}(A)).$$

By using these expressions and the previous lemma

$$n^+(R_{B_2}(A)) \leqslant n^-(R_{B_1}(A)) < kn^+(R_{B_1}(A)),$$

$$n^+(R_{B_1}(A)) \leqslant n^-(R_{B_2}(A)) < kn^+(R_{B_2}(A)).$$

Therefore,

$$n^+(R_{B_2}(A)) < kn^+(R_{B_1}(A)) < k^2 n^+(R_{B_2}(A)),$$

and this expression implies that

$$k^2 > 1$$

since $n^+(R_{B_2}(A)) > 0$, and this is a contradictory result with the range of the parameter $k \in [0,1]$.

This first result shows that the algorithm cannot produce contradictory rules, since both cannot verify the k-consistency condition simultaneously. The second result, which we are going to prove, shows the basic algorithm's behaviour.

Proposition 3.2. *Let $R_B(A_1), R_B(A_2) \in \Delta^k$ such that $A_1 \subseteq A_2$ then*

$$\mathit{fitness}(R_B(A_1)) \leqslant \mathit{fitness}(R_B(A_2)).$$

Proof. Let $A_1 = (A_{11}, \ldots, A_{1n})$ and $A_2 = (A_{21}, \ldots, A_{2n})$. Since $A_1 \subseteq A_2$ then $A_{1j} \subseteq A_{2j}$ $\forall j = 1, \ldots, n$. For each example e we have

$$Poss(A_{1j}|e_j) \leqslant Poss(A_{2j}|e_j),$$

and

$$Nec(A_{1j}|e_j) \leqslant Nec(A_{2j}|e_j).$$

Therefore,

$$U(E, A_1) \leqslant U(e, A_2),$$

and

$$L(E, A_1) \leqslant L(e, A_2).$$

Thus, the result is obvious.

This result shows that if a rule $R_B(A_2)$ is more general than another rule $R_B(A_1)$, i.e., $A_1 \subseteq A_2$, then the first rule has more positive and negative examples. The algorithm always chooses the rule with more positive examples, therefore if the more general rule continues verifying the k-consistency condition then the algorithm prefers this more general description. Finally, we can say that the learning algorithm chooses the most general description for a rule that satisfies the k-consistency condition.

4. Experimental studies

We have implemented SLAVE (Structural Learning Algorithms in Vague Environments) a graphical environment for testing and experimenting with the proposed learning algorithm. SLAVE is a C written program working in an OpenWindows environment. The algorithm uses several parameters in the learning process. Here we want to study the predictive performance of SLAVE for different consistency type and parameters. One of the greatest difficulties associated with SLAVE will be the choice of these parameters. In this section, we shall propose a modified fitness function that make this selection easy.

4.1. The domains

We have tested the learning algorithm on six databases: five correspond to artificial domains and one to a natural domain. Two from the artificial domains, ART1 and ART2, have been generated from a decision tree and the other three were proposed by Quinlan in [13] and they are called MONK1, MONK2 and MONK3, respectively. These domains correspond to single-class learning problems. The sixth is the well known database *Iris Plants Database*, IRIS, created by Fisher [5] that correspond a multi-class learning problem. In the following subsections we explain each domain.

4.1.1. ART1 and ART2 databases

These artificial domains have been designed to reveal the predictive performance of SLAVE when noise increases in the different database sets. For these databases, we use six-feature world in which four (X_1,X_2,X_3,X_5) have been considered as continuous ones and two-feature (X_4,X_6) are nominals. The ART1 database has 200 examples (144 of the 0-class and 56 of the 1-class) and it was generated from the decision tree of Fig. 4.

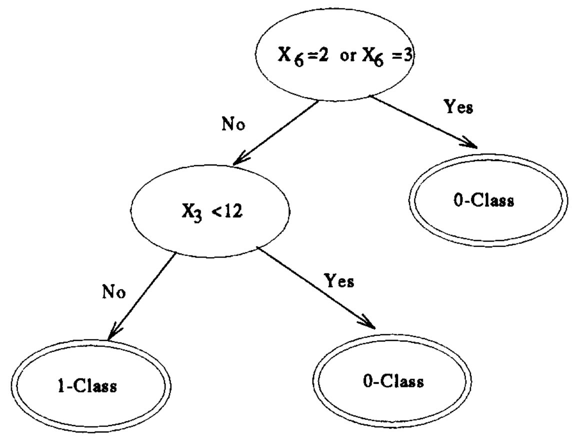

Fig. 4. A decision tree.

The ART2 database contains 200 examples (140 of the 0-class and 60 of the 1-class), and it was generated from the same decision tree but adding a 10% of noise.

In order to use SLAVE, the continuous features have been considered as fuzzy ones and the associated domain corresponds to those shown in Fig. 1.

4.1.2. MONK's databases

The MONK's problems are a collection of three binary classification problems over a six-attribute discrete domain. Each training/test data is of the form

(value1) (value2) (value3) (value4) (value5) (value6) $\rightarrow$ (class).

where (value n) represents the value of attribute #n, and (class) is either 0 or 1, depending on the class this example belongs to. The attributes may take the following values:

attribute #1: $\{1,2,3\}$
attribute #2: $\{1,2,3\}$
attribute #3: $\{1,2\}$
attribute #4: $\{1,2,3\}$
attribute #5: $\{1,2,3,4\}$
attribute #6: $\{1,2\}$

Thus, the six attributes span a space of $432 = 3 \times 3 \times 2 \times 3 \times 4 \times 2$ examples.

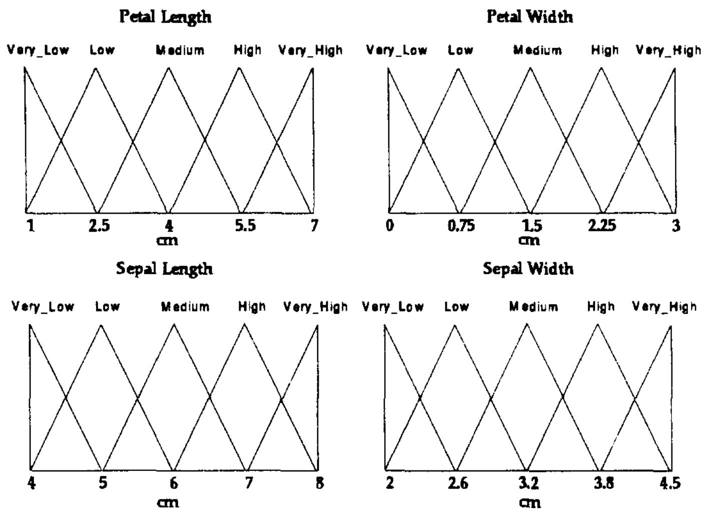

Fig. 5. Domains for the iris problem.

The "true" concepts underlying each MONK's problem are given by:

- MONK1: (attribute #1 = attribute #2) or (attribute #5 = 1)
- MONK2: (attribute #n = 1) for EXACTLY TWO choices of n (in $1, 2, \ldots, 6$)
- MONK3: (attribute #5 = 3 and attribute #4 = 1) or (attribute #5 $\neq$ 4 and attribute #2 $\neq$ 3). MONK3 has 5% additional noise (misclassifications) in the training set.

In these three databases, all the possible combinations are considered in its test set, that is, the test set contains 432 examples.

4.1.3. *Iris databases*

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

For classifying each plant four continuous attributes are used. The domain of each continuous variable of this database has been produced without any specific information on the problem, and we used a simple method that generates a fixed number of uniformly distributed linguistic labels. These domains for each variable are shown in Fig. 5.

4.2. *Learning algorithms*

We tried to investigate the predictive accuracy of SLAVE in the aforementioned databases. So, we carried out different trials considering different consistency types (lower and upper) and different consistency parameters. The common parameters used in all the trials were:

Covering parameter λ	0.8
Size population	20
Maximum iteration number	500
t-norm	minimum

In order to compare the behaviour of the learning algorithm we also used the well-known learning algorithms, CART [1], C4.5 [13] and backpropagation (BP) neural networks [14]. One of the main reasons to select these algorithms is that they can use continuous valued attributes and we do not need to use a discretization technique for them.

4.3. *Inference model*

In order to study SLAVE's predictive performance, we included an inference model on fuzzy rules in this system. Since all the examples considered here are crisp and they have a crisp consequent. We used the inference method which takes the consequent corresponding to the rule with maximum adaptation with the example, i.e., let RULES be a set of rules

$$RULES = \{R_i \mid i = 1, \ldots, n_r\},$$

with

$$antecedent(R_i) = (A_{i1}, A_{i2}, \ldots, A_{in}),$$

$$consequent(R_i) = B_i,$$

and the example

$$e = (e_1, e_2, \ldots, e_n).$$

Let $\Pi(R_i|e) = \min_j\{\max_{a\in A_{ij}}\{\mu_a(e_j)\}\}$ be an adaptation measure between each rule and the example. Then, the inference process involves selecting the consequent B_s such that

$$\max_i\{\Pi(R_i|e)\} = \Pi(R_s|e).$$

However, there is a decision problem in several cases. The controversial situation appears for example when we have B_{s_1} and B_{s_2} such that

$$\max_i\{\Pi(R_i|e)\} = \Pi(R_{s_1}|e) = \Pi(R_{s_2}|e)$$

and

$$B_{s_1} \neq B_{s_2}.$$

We say that in this case we have a conflict in the rule set, and we need to decide what is the best consequent.

Initially, the set of rules is ordered according to the number of examples that each rule eliminated in the learning process. When a conflict appears, SLAVE predictive process selects the consequent of the rule with lower order between the rules of conflict.

Example 2. Let *RULES* be a set of rules learned with *SLAVE* and e an example that we want to classify. Let us suppose that for classifying the e example a conflict between three rules in *RULES*, R_1, R_2 and R_3 is produced, where each rule eliminated d_1, d_2 and d_3 examples in the training set, respectively. Let us suppose that $d_1 < d_2 < d_3$, then *SLAVE* for solving the conflict selects the consequent of R_1 rule.

The draw cases are solved by maintaining the order in which SLAVE obtained the rules.

This criterion for solving conflicts is based on a probabilistic prediction. SLAVE selects the consequent of the one rule that has the most probability of success using its knowledge about the training set. Thus, when the initial set of rules has been ordered by the previous criterion, SLAVE checks the adaptation between the example and each rule and gives the consequent corresponding to the first best value.

4.4. Results

We have taken three different partitions from each training and test set in the different database sets except in the MONK's problem that we have only used the partitions proposed in [13]. For ART1, ART2 and IRIS the examples were randomly divided into a training set (75%) and a test set (25%). The following tables show the accuracy rates for each database on different consistency parameters and the lower and upper consistency conditions, respectively. The accuracy of Table 1 corresponds to the average accuracy of the different partitions for each test set.

In order to compare the results obtained in the different learning databases Table 2 shows the average accuracy for BP[2], CART and C4.5 in the different partitions of each example set.

SLAVE improves the accuracy of the reference algorithms (BP, CART and C4.5) for all databases except for the monk's problem where BP (in MONK1 and MONK2) and C4.5 (in MONK3) obtain the best results, being in both cases SLAVE the second better result. The monk's problem is composed by nominal variables, where the fuzzyness of SLAVE seems to be less useful (more appropriate for discretizing continuous antecedent variables), even in this case SLAVE obtains a better result than other more classic learning approaches.

We can see in the tables that on ART1 and MONK1 databases the lower case obtains the higher accuracy and these are the less noisy databases considered, while the upper case obtains its better result for databases with a higher noise level (the best for IRIS and ART2 and the second better result for MONK2 and MONK3). After this analysis, we can conclude

[2] The experimental tests have been made using the shareware Aspirin/MIGRAINES system copyright of Russell Leighton and the MITRE corporation.

Table 1

k	ART1	ART2	IRIS	MONK1	MONK2	MONK3
Lower case						
0	100	94.42	88.80	97.22	48.85	78.12
0.01	100	93.33	89.51	77.78	60.47	78.12
0.1	99.23	90.45	90.7	91.67	60.47	90.62
Upper case						
0.1	97.24	93.33	79.98	91.67	65.12	93.75
0.2	98.68	94.12	93.74	75.00	48.85	96.88
0.3	98.68	94.61	94.45	83.33	48.85	96.88

Table 2

Learning Algorithm	ART1	ART2	IRIS	MONK1	MONK2	MONK3
CART	99.23	89.32	92.96	83.27	60.30	92.32
C4.5	99.23	93.85	91.13	75.70	65.00	97.20
BP	99.23	87.89	91.56	100	100	93.75

that the upper case has better behaviour for databases with high noise level. The lower case obtain a better result in databases with low level of noise.

From the tables, we can see that the accuracy of SLAVE is strongly associated to the value of the consistency parameter, and the best value for this parameter is different for each database.

With these results, we can conclude that SLAVE is capable of learning with reasonable rate accuracy, but the final result is highly dependent on the consistency parameter chosen, and it is not easy to know a priori what this parameter is. The following section suggests an alternative that makes selection of the consistency parameter easier and that even improves the rates of accuracy.

5. A modified fitness function: The two parameter model

The change between feasible and non-feasible solutions in the fitness function in the *GENETIC* procedure is radical. With the genetic algorithms this situation can be avoided so that the process evolves from a near but non-feasible solution to the best one. In order to solve this problem and to avoid the risky choice of only one consistency parameter we propose a new fitness function that smoothes the change between k-consistency and non k-consistency rules. This new fitness function uses two parameters, a maximum and a minimum consistency parameter. In this case, the learning algorithm uses the following fitness function:

$$\mathit{fitness}(R_B(A)) = \begin{cases} n^+(R_B(A)) & \text{if } R_B(A) \in \Delta^{k_1}, \\ \dfrac{k_2 n^+(R_B(A)) - n^-(R_B(A))}{k_2 - k_1} & \text{if } R_B(A) \in \Delta^{k_2} \text{ and } R_B(A) \notin \Delta^{k_1}, \\ 0 & \text{otherwise} \end{cases}$$

with $k_2 > k_1$.

In this fitness function we assign the complete power of the rule $(n^+(R_B(A)))$ to k_1-consistency rules, zero to non-k_2-consistency rules, and the rest of the rules $(\bar{\Delta}^{k_1} \cap \Delta^{k_2})$ receive a continuous linear decreasing function between both values.

5.1. Results

Table 3 shows some results using the two parameter model.

The results with the learning algorithm using this new fitness function are most regular, that is, the

Table 3

k_1–k_2	ART1	ART2	IRIS	MONK1	MONK2	MONK3
The k_1–k_2 model: Lower case						
0.0/0.2	100	93.33	92.66	88.89	55.81	90.60
0.0/0.4	100	94.61	92.66	100	65.12	96.88
0.1/0.4	98.32	93.86	85.53	83.33	53.49	96.88
0.1/0.5	97.30	86.67	85.53	77.78	48.84	96.88
The k_1–k_2 model: Upper case						
0.0/0.2	93.60	94.12	81.67	100	48.84	93.75
0.0/0.4	97.92	94.61	94.72	91.67	62.79	96.88
0.1/0.4	98.68	94.61	95.43	80.56	60.47	96.88
0.1/0.5	98.68	93.33	93.03	77.78	65.12	96.88

accuracy has not a strong variation moving in close consistency values. Moreover, in some cases we even achieve greater accuracy, for example, MONK1 obtains the maximum possible accuracy. SLAVE keeps its better results for ART1, ART2 and IRIS.

In general, a good result is obtained in the interval 0.0/0.4 in the lower case, and in the interval 0.1/0.4 in the upper case. Here, it seems that the solution involves selecting an interval that includes a theoretical true consistency value.

In any case, SLAVE has shown to be an useful tool to learn fuzzy rules for classification problems.

6. Concluding remarks

In this paper we described a learning algorithm capable of identifying the fuzzy rules that describe a particular system. This algorithm

- is based on theoretical extensions of classical learning conditions (consistency and completeness) having good general properties
- can handle fuzzy information (examples or rules);
- combines in a single process rule and feature selection;
- is a useful tool for solving multi-class learning problems;
- allows change of the granularity of the elements of the domains in some cases;
- generates a small number of fuzzy rules that are very easy to understand by an expert;
- obtains a level of accuracy that improves those values obtained using well-known learning algorithms.

However, the research must continue, at least, in two important aspects: On the one hand, to enrich the language of the rules that SLAVE can use, and on the other hand, to provide SLAVE with the capacity to automatically determine some important parameters of the learning algorithm such as the consistency threshold or the consistency type.

References

[1] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, *Classification and Regression Trees* (Monterey, Wadsworth, CA, 1984).

[2] K. De Jong, W.M. Spears and D.F. Gordon, Using genetic algorithms for concept learning, *Machine Learning* **13** (1993) 161–188.

[3] R. De Mori and L. Saitta, Automatic learning of fuzzy naming relations over finite languages, *Inform. Sci.* **21** (1980) 93–139.

[4] M. Delgado and A. González, An inductive learning procedure to identify fuzzy systems, *Internat. J. Fuzzy Sets and Systems* **55** (1993) 121–132.

[5] R.A. Fisher, The use of multiple measurements in taxonomic problems, *Ann. Eugenics* **7**, Part II (1936) 179–188; also in: *Contributions to Mathematical Statistics* (Wiley, New York, 1950).

[6] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley, New York, 1989).

[7] A. González, R. Pérez and J.L. Verdegay, Learning the structure of a fuzzy rule: a genetic approach, *Proc. 1st Eur. Congr. on Fuzzy and Intelligent Technologies*, Vol. 2 (1993) 814–819.

[8] A. González, A learning methodology in uncertain and imprecise environments, *Internat. J. Intelligent Systems* **19** (1995) 357–371.

[9] H. Ishibuchi, K. Nozaki, N. Yamamoto and H. Tanaka, Construction of fuzzy classification systems with rectangular

fuzzy rules using genetic algorithms, *Fuzzy Sets and Systems* **65** (1994) 237–253.

[10] L. Lesmo, L. Saitta and P. Torasso, Fuzzy production rules: a learning methodology, in: P.P. Wang, Ed., *Advances in Fuzzy Sets Theory and Applications* (Plenum Press, New York, 1983) 181–198.

[11] R.S. Michalski, A theory and methodology of inductive reasoning, in: R.S. Michalski, J. Carbonell and T. Mitchell, Eds., *Machine Learning: An Artificial Intelligence Approach*, Vol. 1 (Morgan Kaufmann, San Mateo, CA).

[12] X.T. Peng and P. Wang, On generating linguistic rules for fuzzy rules, in: B. Bouchon, L. Saitta, R.R. Yager, Eds., *Uncertainty and Intelligent Systems*, Lectures Notes in Computer Science, Vol. 313 (1988) 185–192.

[13] J.R. Quinlan, C4.5 Programs for Machine Learning (Morgan Kaufmann, Los Altos, 1993).

[14] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning interior representation by error propagation, in: D.E. Rumelhart and J.L. McClelland, Eds., *Parallel Distributed Processing*, Vol. 1, Ch. 8 (MIT Press, Cambridge, MA, 1986).

[15] M. Sugeno and K. Tanaka, Successive identification of a fuzzy model and its applications to prediction of a complex system, *Fuzzy Sets and Systems* **42** (1991) 315–334.

[16] R.M. Tong, Synthesis of fuzzy models for industrial processes, *Internat. J. Gen. Systems* **4** (1978) 143–162.

[17] L.X. Wang and J.M. Mendel, Generating fuzzy rules by learning from examples, *IEEE Trans. Systems Man Cybernet.* **22** (1992) 1414–1427.

[18] L.A. Zadeh, The concept of a linguistic variable and its applications to approximate reasoning, Part I, *Inform. Sci.* **8** (1975) 199–249; Part II, *Inform. Sci.* **8**, 301–357; Part III, *Inform. Sci.* **9**, 43–80.