Published in J. Advanced Computational Intelligence, Vol. 1, No. 1, 1997

An Efficient Method for Extracting Fuzzy Classification Rules from High Dimensional Data

Stephen L. Chiu

Rockwell Science Center
1049 Camino Dos Rios
Thousand Oaks, California 91360, USA
E-mail: slchiu@rsc.rockwell.com

We present an efficient method for extracting fuzzy classification rules from high dimensional data. A cluster estimation method called *subtractive clustering* is used to efficiently extract rules from a high dimensional feature space. A complementary search method can quickly identify the important input features from the resultant high dimensional fuzzy classifier, and thus provides the ability to quickly generate a simpler, more robust fuzzy classifier that uses a minimal number of input features. These methods are illustrated through the benchmark iris data and through two aerospace applications.

Keywords: Clustering, Classification, Fuzzy rule learning, Feature selection.

1. Introduction

One of the hallmarks of fuzzy logic is that it allows nonlinear input/output relationships to be expressed by a set of qualitative “if-then” rules. Nonlinear control or decision surfaces, process models, and pattern classifiers may all be expressed in the form of fuzzy rules. Most fuzzy systems are hand-crafted by a human expert to capture some desired input/output relationships that the expert has in mind. However, often an expert cannot express his or her knowledge explicitly; and, for many applications, an expert may not even exist. Hence, there is considerable interest in being able to automatically extract fuzzy rules from experimental input/output data. The key motivation for capturing data behavior in the form of fuzzy rules instead of, say, polynomials and neural networks, is that the fuzzy rules are easy to understand, verify, and extend. A system designer can check the automatically extracted rules against intuition, check the rules for completeness, and easily fine-tune or extend the system by editing the rulebase.

While the problem of extracting fuzzy rules from data for function approximation has been studied for some time [1,2], research into extracting fuzzy rules for pattern classification has become active only within the past 5 years. An early approach to extracting fuzzy rules for pattern classification involves partitioning the feature space into grid-like fuzzy cells that correspond to possible combinations of input feature values; the output class associated with each cell is obtained from the data samples that fall into the cell. Extensions to this approach include combining several rule bases, each with different cell partitioning resolution [3], and dynamically expanding and splitting fuzzy cells [4]. A related method that employs nested fuzzy cells (resulting in nested rules) can extract good fuzzy cells in one pass [5].

Recently, methods for extracting fuzzy rules for pattern classification have incorporated neural network concepts for adaptive learning [6,7]. These methods require the user to prespecify the structure of the rulebase, i.e., number of rules per class [6] or number of membership functions per input feature [7], along with initial values for the adjustable parameters.

Our work has been guided by the objective of developing a practical, easy-to-use software for extracting fuzzy rules from data for real-world, high-dimensional problems. Efficiency and robustness of the algorithm in dealing with high-dimensional and noisy data are primary factors driving our approach. Simplicity of the method from a user’s perspective is also important (i.e., minimize the number of parameters the user must specify and minimize the need for trial-and-error).

A cluster estimation method called *subtractive clustering* [8] forms the basis of our approach. Subtractive clustering is a fast and robust method for estimating the number and location of cluster centers present in a collection of data points. Initial fuzzy rules with rough estimates of membership functions are obtained from the cluster centers; the membership functions and other rule parameters are then optimized with respect to some output error criterion. This approach can be applied to extract rules for both function approximation and pattern classification, with some small differences in the detailed methods for these two types of applications (see [9]). Using subtractive clustering to extract rules for function approximation is described in [8,9], and the algorithm for this purpose is readily available in the commercial Fuzzy Logic Toolbox software for MATLAB. Here we focus on extracting rules for pattern classification, building upon the work described in [10]. We will first review the basic rule extraction method as presented in [10], and then present an efficient method for input feature selection that works jointly with the basic rule extraction method. We will illustrate these methods through application to the benchmark iris data. We will also describe two aerospace applications: one for detecting leakage in the space shuttle's auxiliary power unit, the other for modeling space shuttle pilot's control action during docking maneuvers.

2. Extracting Fuzzy Rules

2.1 The Subtractive Clustering Method

To extract rules from data, we first separate the training data into groups according to their respective classes. Consider a group of n data points $\{x_1,x_2,...,x_n\}$ for a specific class, where x_i is a vector in the input feature space. Without loss of generality, we assume that the feature space is normalized so that all data are bounded by a unit hypercube. We consider each data point as a potential cluster center for the group and define a measure of the potential of data point x_i to serve as a cluster center as

$$P_i = \sum_{j=1}^{n} e^{- a \, \| x_i - x_j \|^2} \tag{1}$$

where

$$a = \frac{4}{r_a^2} \tag{2}$$

$\|.\|$ denotes the Euclidean distance, and r_a is a positive constant. Thus, the measure of the potential of a data point is a function of its distances to all other data points in its group. A data point with many neighboring data points will have a high potential value. The constant r_a is effectively a normalized radius defining a neighborhood; data points outside this radius have little influence on the potential. Note that because the data space is normalized, $r_a = 1.0$ is equal to the length of one side of the data space.

After the potential of every data point in the group has been computed, we select the data point with the highest potential as the first cluster center. Let x_1^* be the location of the first cluster center and P_1^* be its potential value. We then revise the potential of each data point x_i in the group by the formula

$$P_i \Leftarrow P_i - P_1^* \, e^{- b \, \| x_i - x_1^* \|^2} \tag{3}$$

where

$$b = \frac{4}{r_b^2}$$

and r_b is a positive constant. Thus, we subtract an amount of potential from each data point as a function of its distance from the first cluster center. The data points near the first cluster center will have greatly reduced potential, and therefore will unlikely be selected as the next cluster center for the group. The constant r_b is effectively the radius defining the neighborhood which will have measurable reductions in potential. To avoid obtaining closely spaced cluster centers, we typically choose $r_b = 1.25 \, r_a$.

When the potential of all data points in the group has been reduced according to Eq. (3), we select the data point with the highest remaining potential as the second cluster center. We then further reduce the potential of each data point according to their distance to the second cluster center. In general, after the k'th cluster center has been obtained, we revise the potential of each data point by the formula

$$P_i \Leftarrow P_i - P_k^* \, e^{- b \, \| x_i - x_k^* \|^2}$$

where x_k^* is the location of the k'th cluster center and P_k^* is its potential value.

The process of acquiring new cluster center and reducing potential repeats until the remaining potential of all data points in the group is below some fraction of the potential of the first cluster center P_1^* ; we typically use $P_k^* < 0.15 P_1^*$ as the stopping criterion. Some additional criteria for accepting and rejecting cluster centers are detailed in [8]. This cluster estimation method, called *subtractive clustering*, is an extension of the grid-based *mountain clustering method* introduced by Yager and Filev [11]. The subtractive clustering method is designed for high dimension problems with moderate number of data points, because its computation grows linearly with the data dimension and as the square of the number of data points.

2.2. Converting Clusters to Initial Rules

Each cluster center found in the training data of a given class identifies a region in the feature space that is well-populated by members of that class. Thus, we can translate each cluster center into a fuzzy rule for identifying the class. Suppose cluster center x_i^* was found in the group of data for class c1; this cluster center provides the rule:

Rule i: If $\{x$ is near $x_i^*\}$ then class is c1.

The degree of fulfillment of $\{x$ is near $x_i^*\}$ is defined as

$$m_i = e^{- a \, \| x - x_i^* \|^2} \tag{4}$$

where a is the constant defined by Eq. (2). We can also write this rule in the more familiar form:

Rule i: If X_1 is A_{i1} & X_2 is A_{i2} &... then class is c1.

where X_j is the j'th input feature and A_{ij} is the membership function in the i'th rule associated with the j'th input feature. The membership function A_{ij} is given by

$$A_{ij}(X_j) = \exp\left\{ -\frac{1}{2} \left(\frac{X_j - x_{ij}^*}{s_{ij}} \right)^2 \right\} \tag{5}$$

where x_{ij}^* is the j'th element of x_i^* , and $s_{ij}^2 = 1/(2a)$. The degree of fulfillment of each rule is computed by using multiplication as the AND operator.

By applying subtractive clustering to each class of data individually, we thus obtain a set of rules for identifying each class. The individual sets of rules can then be combined to form the rulebase of the classifier. For example, suppose we found 2 cluster centers in class c1 data, and 5 cluster centers in class c2 data, then the rulebase will contain 2 rules that identify class c1 members and 5 rules that identify class c2 members. When performing classification, the output class of the classifier is simply determined by the rule with the highest degree of fulfillment.

2.3 Optimizing Membership Functions

After the initial rulebase of the classifier has been obtained by subtractive clustering, we use a gradient descent algorithm to tune the individual x_{ij}^* and s_{ij} parameters in the membership functions (cf. Eq. 5) to minimize a classification error measure.

We define the following classification error measure for a data sample that belongs to some class c:

$$E = \frac{1}{2}\left(1 - m_{\text{c,max}} + m_{\neg\text{c,max}}\right)^2 \qquad (6)$$

where $m_{\text{c,max}}$ is the highest degree of fulfillment among all rules that infer class c and $m_{\neg\text{c,max}}$ is the highest degree of fulfillment among all rules that do not infer class c. Note that this error measure is zero only if a rule that would correctly classify the sample has degree of fulfillment of 1 and all rules that would misclassify the sample have degree of fulfillment of 0. An important attribute of this error measure is that optimization involves adjusting only the rules responsible for $m_{\text{c,max}}$ and $m_{\neg\text{c,max}}$, since all other rules do not affect the error measure.

The membership function parameters are updated according to the following gradient descent formulae [10]:

$$x_{ij}^* \Leftarrow x_{ij}^* - l\,\frac{\partial E}{\partial x_{ij}^*} \quad \text{and} \quad s_{ij} \Leftarrow s_{ij} - l\,\frac{\partial E}{\partial s_{ij}}$$

where l is a positive learning rate. After carrying out the partial differentiation, we obtain

$$x_{ij}^* \Leftarrow x_{ij}^* \pm l\, m_i \frac{\left(1 - m_{\text{c,max}} + m_{\neg\text{c,max}}\right)\left(X_j - x_{ij}^*\right)}{s_{ij}^2} \qquad (7)$$

$$s_{ij} \Leftarrow s_{ij} \pm l\, m_i \frac{\left(1 - m_{\text{c,max}} + m_{\neg\text{c,max}}\right)\left(X_j - x_{ij}^*\right)^2}{s_{ij}^3} \qquad (8)$$

where these update formulae are applied to only two rules: the rule that provided $m_{\text{c,max}}$ and the rule that provided $m_{\neg\text{c,max}}$; the '+' sign is used for the rule that provided $m_{\text{c,max}}$ and the '-' sign is used for the rule that provided $m_{\neg\text{c,max}}$. Here X_j is the j'th input feature value of the data sample and m_i is the degree of fulfillment of rule i.

This gradient descent algorithm can be viewed as a type of competitive learning algorithm: a winner in the "good rule" category is reinforced and a winner in the "bad rule" category is punished. Because only two rules are updated each time, the algorithm is highly efficient.

We have found that a classifier can achieve higher accuracy and the resultant classification rules are easier to understand if the membership functions are allowed to be "two-sided" Gaussian functions. A two-sided Gaussian function may have a flat plateau region and different standard deviations on the left and right sides (see Fig. 1). For two-sided Gaussian functions, there are four adjustable parameters per membership function: the left and right side peak positions (left and right x_{ij}^*), and the left and right side standard deviations (left and right s_{ij}). The update formulae in Eqs. (7) and (8) are mathematically valid for two-sided Gaussian functions when the feature value X_j is outside the plateau region; these equations apply to the left side parameters if X_j is on the left side of the plateau and apply to the right side parameters if it is on the right side of the plateau. However, the update formulae are no longer valid when X_j is inside the plateau region, because the error gradient there is zero. We use the following heuristic procedure to handle this situation: if the rule is a "good" rule, then do not modify the membership function (since it already provides the maximum membership grade); if the rule is a "bad" rule, then use Eq. (7) with '+' sign and Eq. (8) with '-' sign to adjust the parameters. The left side parameters are adjusted if X_j is closer to the left peak and the right side parameters are adjusted if X_j is closer to the right peak. In this way, the membership function of a bad rule is still moved away from X_j in a consistent manner, although the move is not based on the true gradients.

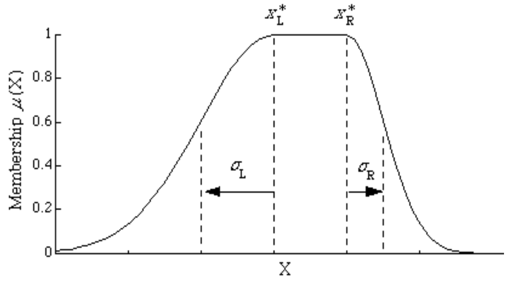

Fig. 1. A "two-sided" Gaussian function.

3. Selecting Input Features

One of the more difficult challenges in function approximation and pattern classification is selecting the important input variables from all possible input variables. Incorporating only the important variables into a model provides a simpler, more useful, and more reliable model. Understanding the relative importance of variables also allows the system designer to focus his or her efforts on the variables that matter, eliminating the time and cost involved in measuring and processing unimportant variables.

Complementing our basic rule extraction method, we have also developed an efficient method for identifying the important input variables (i.e., features). This method is based on generating an initial fuzzy classifier that employs all possible input features, and then systematically removing particular antecedent clauses in the fuzzy rules of this initial classifier to test the sensitivity of the classifier with respect to each input feature. This approach avoids the computational bottleneck encountered by most feature selection methods: the need to repeatedly generate new classifiers to test each combination of features.

For example, suppose that the initial classifier has three inputs, with rules of the form:

If X_1 is A_{i1} & X_2 is A_{i2} & X_3 is A_{i3} then class is c.

We can test the importance of the X_2 input in the classifier by temporarily removing the antecedent clauses that involve X_2, thus truncating the rules to the form:

If X_1 is A_{i1} & X_3 is A_{i3} then class is c.

If the resultant classifier's performance does not degrade with respect to some performance measure (e.g., by testing the classifier with respect to an independent set of checking data), then we can eliminate X_2 from the list of possibly important inputs. In practice, we need not actually remove antecedent clauses from the rules, but simply let the associated clauses (e.g., “X_2 is A_{i2}”) be assigned a truth value of 1.0 to achieve the effect.

The systematic procedure for input feature selection is as follows:

1. Evaluate the performance of the initial classifier with all candidate inputs present.
2. For each remaining input, evaluate the performance of the classifier with this input temporarily removed.
3. Permanently remove the input associated with the best classifier performance obtained in step 2. Record the resultant reduced input set and the associated classifier performance.
4. If there are still inputs remaining in the classifier, go back to step 2 to eliminate another input. Otherwise go to step 5.
5. Choose the best input set from among the sets recorded in step 3.

This is essentially a backward selection procedure that starts with all possible inputs and reduces the number of inputs by one at each stage. For example, starting with 4 input features, this procedure first selectively removes one input to arrive at the best 3 input set; from among the 3 remaining inputs, it then selectively removes one more input to arrive at the best 2 input set, and so on; among these “best” input sets, the set that provided the best overall performance is chosen at the end, after we have reached a classifier with no input. The input selection process for a four-input initial classifier is shown in Fig. 2. Because testing the different input combinations requires only truncating the rules in an initial full-input classifier and not generating any new classifiers, the input selection process is highly efficient.

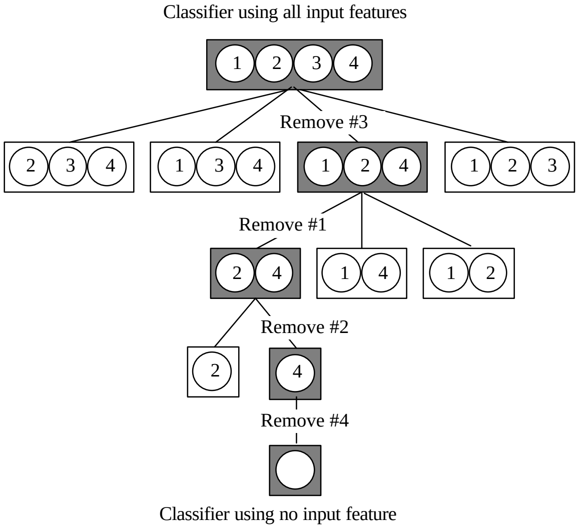

Fig. 2. Input selection process for a four-input initial classifier. The different subsets of inputs considered at each stage are shown; the shaded subsets are the ones that provided the best performance at each stage. After the input removal process is carried to the end, we select from among the shaded subsets the subset that provides the best performance/complexity trade-off.

While a typical backward selection process stops when the classifier's performance becomes unacceptable at some stage, we found it more useful to carry the process to the end until the performance of a classifier with no input is evaluated. It is important to let the human designer examine the classifier performance as a function of the number of inputs removed and determine how to trade off accuracy for simplicity.

This input selection procedure can be used with various classification performance measures. The simplest performance measure is the misclassification rate with respect to an independent set of checking data. However, we found that using the average of the classification error measure defined in Eq. (6) provides finer discriminating ability in ranking the relative importance of inputs. Specifically, we use the following performance measure:

$$J = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(1 - m_{c,\max}(x_i) + m_{\neg c,\max}(x_i)\right)^2} \qquad (9)$$

where x_i is a sample in the checking data set and n is the number of checking data. Comparing this performance measure with Eq. 6 shows that this is basically our classification error measure averaged over the checking data set; the purpose of the square root operation is to make this performance measure similar to the standard RMS-type error measure. Note that when all input features (or antecedents) are removed from the rules, all rules will have a degree of fulfillment of 1.0 and the performance measure in Eq. 9 will be 1.0.

4. Application Examples

4.1 Classification of Fisher's Iris Data

We have applied the rule extraction method to the benchmark iris data [12]. The iris data consists of a set of 150 data samples that maps 4 input feature values (sepal length, sepal width, petal length, petal width) into one of 3 species of iris flower. There are 50 samples for each species. We used 120 samples (40 samples of each species) as training data and 30 samples (10 samples of each species) as checking data.

Using a cluster radius of r_a = 0.5 (the default in our rule extraction software), our method extracted a 4-input initial classifier consisting of 6 rules: 1 rule for class 1 (Iris setosa), 2 rules for class 2 (Iris versicolor), and 3 rules for class 3 (Iris virginica). The membership functions were optimized using Eqs. 7 and 8; the optimization stops when the improvement in the average error measure after a pass through the training data becomes less than 0.01 percent of the previous average error measure. (The average error measure is given by Eq. 9 evaluated over the training data.) The classification error on the training data was 2.5% and the error on the checking data was 0%. This result is similar to that reported in [13] for two neural networks. However, our method achieved this result the first time it is applied, and the computation time for rule extraction was only 25 seconds on a Macintosh computer with a 68040 processor running at 25MHz.

We then applied the input selection procedure to the 4-input initial classifier, using the performance measure of Eq. (9) evaluated over the checking data as the selection criterion. The performance measure as a function of the number of inputs remaining in the classifier is shown in Fig. 3, where the name of the input removed at each stage is indicated next to each performance datum. We see that petal width is the last input removed; hence it is probably the most important feature for classifying the flowers. The performance measure reached an optimum (minimum) after sepal width and sepal length are removed; this suggests that we should remove these two features and keep the remaining petal length and petal width features. The computation time for input selection was 2 seconds on the Macintosh computer. A new classifier based on only the petal length and petal width features was then generated from the training data, again using a cluster radius of 0.5. The resultant classifier has only 3 rules (one rule for identifying each class) and produced the same classification error with respect to the training and checking data as the previous 4-input initial classifier. The computation time for extracting the 2-input classifier was 4 seconds on the Macintosh computer. The rules for the 2-input classifier are shown in Fig. 4.

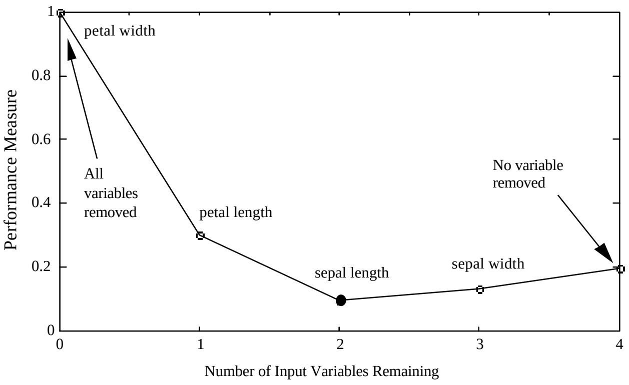

Fig. 3. Classification performance measure with respect to checking data as a function of number of inputs remaining in the initial classifier. The point of minimum error is shown as a black circle. The input removed at each stage is shown next to the resultant error point.

4.2 Shuttle Auxiliary Power Unit Leakage Detection

The rule extraction method was applied to detect leakage in the space shuttle's auxiliary power unit (APU). The input to the classifier is a sequence of APU chamber pressure values as a function of time; the output is an indication of whether there is a leakage. Figure 5 shows some time responses of the APU chamber pressure in the presence and absence of leakage. There are 103 pressure response curves in the data set; each time response consists of pressure measurements at 57 consecutive time points (i.e., 57 input features). Because of the limited number of data samples, we used the entire 103

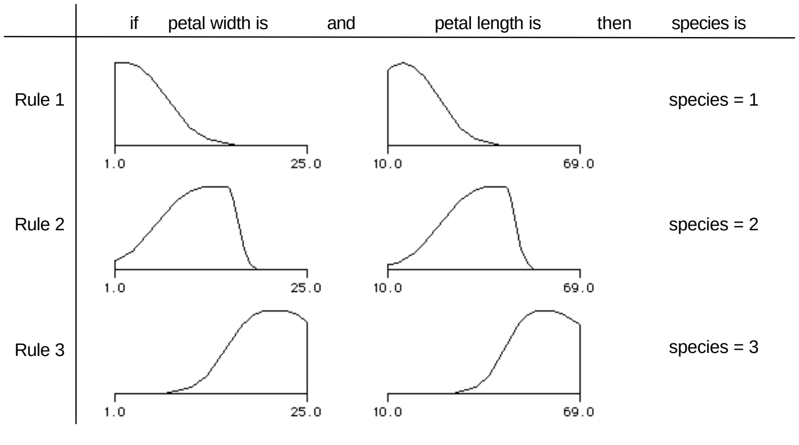

Fig. 4. Rules extracted for the 2-input iris classifier.

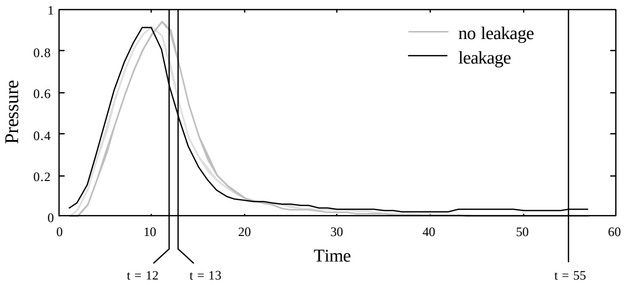

Fig. 5. Time responses of APU chamber pressure.

samples as both training data in the rule extraction phase and as checking data in the input selection phase.

Using r_a = 0.5, we extracted an initial 57-input classifier consisting of 2 rules (one rule for identifying leakage cases and one rule for identifying non-leakage cases). This initial classifier produced no error on the data set. We then applied the input selection procedure to the initial classifier; pressure values at only 3 time points were found to be important for distinguishing the two cases (pressure at times t=12, t=13, and t=55, as shown in Fig. 5). Finally, a new classifier based on the three selected inputs was generated. The total computation time for extracting the initial classifier, selecting inputs, and extracting the final classifier was 3.5 minutes on the Macintosh computer. The rules for the final classifier, which also produced no error on the data set, are shown in Fig 6.

4.3 Shuttle Pilot Docking Emulation

The rule extraction method was applied to mimic the space shuttle pilot's control actions during docking maneuvers. The inputs to the system are the shuttle's distance, azimuth, elevation, and their rates of change with respect to the docking target (total of 6 inputs); the outputs are the shuttle's jet firing commands along the three (x-y-z) translational axes (total of 3 outputs). The jet firing command for each translational axis can be either +1 (fire in positive direction), -1 (fire in negative direction), or 0 (do not fire), resulting in three possible classes for each output.

Of the 3600 data samples obtained from simulation, we used half as training data and half as checking data. Using r_a = 0.5, we extracted an initial 6-input classifier to determine the x-axis jet firing command. This initial classifier consisted of 18 rules and produced 1.6% error on the training data and 2.0% error on the checking data. After applying the input selection procedure to the initial classifier, 4 of the inputs were identified as important. A new classifier based on the four selected inputs was then generated by using r_a = 0.5. The resultant 4-input classifier has 12 rules, and produced 2.9% error on the training data and 2.4% error on the checking data. The total computation time for extracting the initial classifier, selecting inputs, and extracting the final classifier was 19 minutes on the Macintosh.

5. Summary

We presented an efficient method for extracting fuzzy classification rules from high dimensional data. This method incorporates several novel, computationally efficient techniques: the use of subtractive clustering to obtain the number of rules and initial membership functions; an efficient gradient descent algorithm for optimizing membership functions; and a method for selecting input features without generating new classifiers to test feature combinations. Applications of this method have shown that it is computationally fast and robust. A good classifier can usually be extracted the first time the method is applied, without any trial-and-error adjustment of parameter settings. This method was illustrated through the benchmark iris data and through two aerospace applications.

References

1. T. Takagi and M. Sugeno, "Fuzzy identification of systems and its application to modeling and control," IEEE Trans. on Systems, Man & Cybernetics, vol. 15, pp. 116-132, 1985.
2. R. Yager and D. Filev, Essentials of Fuzzy Modeling and Control, John Wiley, New York, 1994.
3. H. Ishibuchi, K. Nozaki, H. Tanaka, "Pattern classification by distributed representation of fuzzy rules," Proc. 1st IEEE Int'l Conf. on Fuzzy Systems, pp. 643-650, San Diego, USA, 1992.

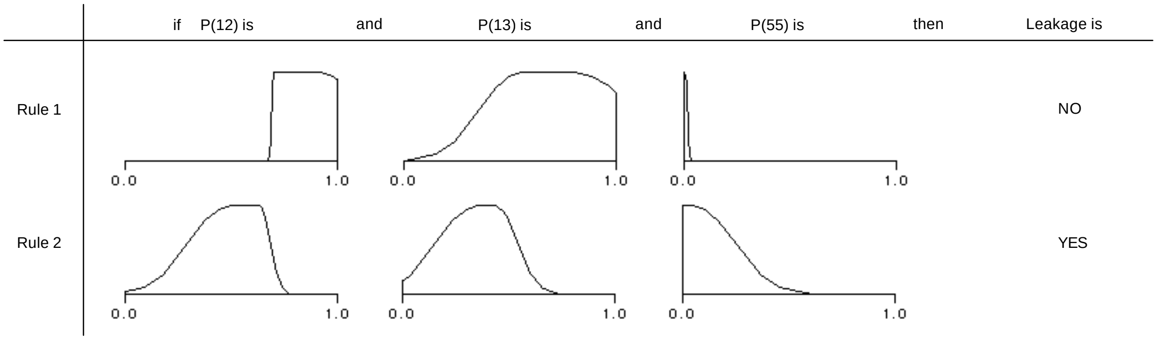

Fig. 6. Rules extracted for detecting APU leakage.

4. P.K. Simpson, “Fuzzy min-max neural networks,” Proc. IJCNN-91, Singapore, pp. 1658-1669, 1991.

5. S. Abe and M.S. Lan, “A classifier using fuzzy rules extracted directly from numerical data,” Proc. 2nd IEEE Int'l Conf. on Fuzzy Systems, pp. 1191-1198, San Francisco, USA, 1993.

6. F.L. Chung and T. Lee, “A fuzzy learning method for membership function estimation and pattern classification,” Proc. 3rd IEEE Int'l Conf. on Fuzzy Systems, pp. 426-431, Orlando, USA, 1994.

7. C.T. Sun and J.S. Jang, “A neuro-fuzzy classifier and its applications,” Proc. 2nd IEEE Int'l Conf. on Fuzzy Systems, pp. 94-98, San Francisco, USA, 1993.

8. S. Chiu, “Fuzzy model identification based on cluster estimation,” J. of Intelligent and Fuzzy Systems, vol. 2, no. 3, pp. 267-278, 1994.

9. S. Chiu, “Extracting fuzzy rules from data for function approximation and pattern classification,” to appear as Chapter 9 in *Fuzzy Set Methods in Information Engineering: A Guided Tour of Applications,* ed. D. Dubois, H. Prade, and R. Yager, John Wiley, 1997.

10. S. Chiu, “Extracting fuzzy rules for pattern classification by cluster estimation,” *Proc. 6th Int. Fuzzy Systems Association Congress* (IFSA ‘95), vol. 2, pp. 273-276, Sao Paulo, Brazil, July 1995.

11. R. Yager and D. Filev, “Approximate clustering via the mountain method,” IEEE Trans. Syst. Man, & Cybernetics, vol. 24, pp. 1279-1284, 1994.

12. R.A. Fisher, “The use of multiple measurements in taxonomic problems,” Annals of Eugenics, 7, pp. 179-188, 1936.

13. C.H. Chen and H. Lai, “A comparison study of the gradient descent and the conjugate gradient backpropagation neural networks,” Proc. World Congress on Neural Networks, vol. 3, pp. 401-405, Portland, USA, 1993.