Chapter 9 in *Fuzzy Information Engineering: A Guided Tour of Applications*,
ed. D. Dubois, H. Prade, and R. Yager, John Wiley & Sons, 1997.

Extracting Fuzzy Rules from Data for Function Approximation and Pattern Classification

Stephen L. Chiu
Rockwell Science Center

ABSTRACT

Extracting fuzzy rules from data allows relationships in the data to be modeled by "if-then" rules that are easy to understand, verify, and extend. This paper presents methods for extracting fuzzy rules for both function approximation and pattern classification. The rule extraction methods are based on estimating clusters in the data; each cluster obtained corresponds to a fuzzy rule that relates a region in the input space to an output region (or, in the case of pattern classification, to an output class). After the number of rules and initial rule parameters are obtained by cluster estimation, the rule parameters are optimized by gradient descent. Applications to a function approximation problem and to a pattern classification problem are also illustrated.

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/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.

This paper presents methods for extracting fuzzy rules from input/output data for both function approximation and pattern classification applications. For function approximation, where the output data correspond to a continuous-valued variable, the extracted rules express continuous-valued relationships (e.g., "if input is small then output is big"). For pattern classification, where the output data correspond to class assignments, the extracted rules have discrete-valued consequents (e.g., "if input is small then output is class 1"). In both cases, fuzzy rules provide a powerful framework for capturing and, perhaps more importantly, explaining the input/output data behavior.

The problem of extracting fuzzy rules from data for function approximation has been studied for some time [1]. Several methods for extracting fuzzy rules for function approximation have used clustering to determine the number of rules and initial rule parameters [2,3,4,5]. Each cluster essentially identifies a region in the data space that contains a sufficient mass of data to support the existence of a fuzzy input/output relationship. Because a rule is generated only where there is a cluster of data, the resultant rules are scattered in the input space rather than placed according to grid-like partitions in the input space. This fundamental feature of clustering-based rule extraction methods helps avoid combinatorial explosion of rules with increasing dimension of the input space. Also, because the clustering step provides good initial rule parameter values, the subsequent rule parameter optimization process usually converges quickly and to a good solution.

Early works on fuzzy pattern classification were focused not so much on rule extraction but on extending classical, crisp clustering algorithms by allowing an object to have partial membership in multiple clusters/classes [6,7,8,9]. Recently, with increasing awareness of the advantages of representing a classifier in the form of fuzzy rules, attention has turned to rule extraction. Extracting fuzzy rules for pattern classification can be viewed as the problem of partitioning the input space into appropriate fuzzy cells that separate the classes. Methods for extracting fuzzy rules for pattern classification often incorporate neural network concepts for adaptive learning. For example, an extension of Kohonen's Self-Organizing Map method was used in [10] to learn the membership function centers; the backpropagation technique was used in [11] to optimize membership functions that define grid-like input partitions as well as optimize parameterized fuzzy operators. These methods typically require the user to prespecify the structure of the rulebase (i.e., number of rules per class [10] or number of membership functions per input feature [11]) along with initial values for the adjustable parameters. A method that can quickly determine the rulebase structure and good initial membership functions in one pass was proposed in [12]. This method employs nested fuzzy cells, which results in nested rules such as "if input is in region A and not in region A', then output is class 1", where region A' is an exception region nested *inside* region A; other rules would handle region A', which may contain its own nested exception regions. Rule extraction involves generating deeper nested rules for increasingly smaller subregions until all errors are eliminated. The nested fuzzy cell method is fast and produces compact rules that do not grow with the dimension of the data space; however, it is sensitive to noisy data.

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 clustering method called *subtractive clustering* [5] 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.

In the following sections, we describe the rule extraction methods in detail. We also illustrate their use in a function approximation example and in a pattern classification example.

2. Cluster Estimation

Clustering of numerical data forms the basis of many modeling and pattern classification algorithms. The purpose of clustering is to find natural groupings of data in a large data set, thus revealing patterns in the data that can provide a concise representation of the data behavior. Clustering algorithms typically require the user to prespecify the number of cluster centers and their initial locations; the locations of the cluster centers are then adapted in a way such that the cluster centers can better represent a set of archetypical data points covering the range of data behavior. The Fuzzy C-Means algorithm [13] and Kohonen's Self-Organizing Map [14] method are well-known examples of such clustering algorithms. For these algorithms, the quality of the solution, like that of most nonlinear optimization problems, depends strongly on the choice of initial values (i.e., the number of cluster centers and their initial locations).

Yager and Filev [4] proposed a simple and effective algorithm, called the mountain method, for estimating the number and initial location of cluster centers. Their method is based on gridding the data space and computing a potential value for each grid point based on its distances to the actual data points; a grid point with many data points nearby will have a high potential value. The grid point with the highest potential value is chosen as the first cluster center. The key idea in their method is that once the first cluster center is chosen, the potential of all grid points is reduced according to their distance from the cluster center. Grid points near the first cluster center will have greatly reduced potential. The next cluster center is then placed at the grid point with the highest remaining potential value. This procedure of acquiring new cluster center and reducing the potential of surrounding grid points repeats until the potential of all grid points falls below a threshold. Although this method is simple and effective, the computation grows exponentially with the dimension of the problem. For example, a clustering problem with 4 variables and each dimension having a resolution of 10 grid lines would result in 10^4 grid points that must be evaluated.

Chiu [5] proposed an extension of Yager and Filev's mountain method, called subtractive clustering, in which each data point, not a grid point, is considered as a potential cluster center. Using this method, the number of effective "grid points" to be evaluated is simply equal to the number of data points, independent of the dimension of the problem. Another advantage of this method is that it eliminates the need to specify a grid resolution, in which trade-offs between accuracy and computational complexity must be considered. The subtractive clustering method also extends the mountain method's criterion for accepting and rejecting cluster centers.

The subtractive clustering method works as follows. Consider a collection of n data points $\{x_1,x_2,...,x_n\}$ in an M dimensional space. Without loss of generality, we assume that the data points have been normalized in each dimension so that they are bounded by a unit hypercube. We consider each data point as a potential cluster center and define a measure of the potential of data point x_i as

$$P_i = \sum_{j=1}^{n} e^{-\alpha \| x_i - x_j \|^2} \qquad (1)$$

where $\alpha = 4/r_a^2$, (2)

$\|.\|$ denotes the Euclidean distance, and r_a is a positive constant. Thus, the measure of the potential for a data point is a function of its distances to all other data points. A data point with many neighboring data points will have a high potential value. The constant r_a is effectively the radius defining a neighborhood; data points outside this radius have little influence on the potential.

After the potential of every data point 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 by the formula

$$P_i \Leftarrow P_i - P_1^* e^{-\beta \| x_i - x_1^* \|^2} \qquad (3)$$

where $\beta = 4/r_b^2$

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. 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 set r_b to be somewhat greater than r_a ; a good choice is $r_b = 1.25\, r_a$.

When the potential of all data points has been revised 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, the potential of each data point is revised by the formula

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

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 revising potentials repeats until the remaining potential of all data points falls below some fraction of the potential of the first cluster center P_1^* . In addition to this criterion for ending the clustering process are criteria for accepting and rejecting cluster centers that help avoid marginal cluster centers. The following criteria are used:

if $P_k^* > \overline{e}\ P_1^*$

Accept x_k^* as a cluster center and continue.

else if $P_k^* < \underline{e}\ P_1^*$

Reject x_k^* and end the clustering process.

else

Let d_{min} = shortest of the distances between x_k^* and all previously found cluster centers.

if $\frac{d_{min}}{r_a} + \frac{P_k^*}{P_1^*} \geq 1$

Accept x_k^* as a cluster center and continue.

else

Reject x_k^* and set the potential at x_k^* to 0.

Do not revise the potential of other data points.

Select the data point with the next highest potential as the new x_k^* and re-test.

end if

end if

Here $\overline{e}$ specifies a threshold for the potential above which we will definitely accept the data point as a cluster center; $\underline{e}$ specifies a threshold below which we will definitely reject the data point. Good default values are $\overline{e}$ = 0.5 and $\underline{e}$ = 0.15 . If the potential falls in the gray region, we check if the data point offers a good trade-off between having a sufficient potential and being sufficiently far from existing cluster centers.

3. Extracting Rules for Function Approximation

For function approximation applications, clustering is performed in the combined input/output space, i.e., each data point x_i is a vector that contains both input and output values. Each cluster center found is in essence a prototypical data point that exemplifies a characteristic input/output behavior of the system.

Consider a set of m cluster centers $\{x_1^*, x_2^*, ..., x_m^*\}$ found in an M dimensional space. Let the first N dimensions correspond to input variables and the last M-N dimensions correspond to output variables. We decompose each vector x_i^* into two component vectors y_i^* and z_i^* , where y_i^* contains the first N elements of x_i^* (i.e., the coordinates of the cluster center in input space) and z_i^* contains the last M-N elements (i.e., the coordinates of the cluster center in output space).

We consider each cluster center x_i^* as a fuzzy rule that describes the system behavior. Intuitively, each cluster center represents the rule:

Rule i: If {input is near y_i^*} then output is near z_i^* .

Given an input vector y, the degree of fulfillment of rule i is defined as

$$m_i = e^{-\alpha \| y - y_i^* \|^2} \qquad (4)$$

where a is the constant defined by Eq. (2). We compute the output vector z via

$$z = \left[\sum_{i=1}^{m} m_i z_i^* \right] \div \left[\sum_{i=1}^{m} m_i \right] . \qquad (5)$$

We can view this computational model in terms of a fuzzy inference system employing traditional fuzzy if-then rules. Each rule has the following form:

if Y_1 is A_{i1} & Y_2 is A_{i2} & ... then Z_1 is B_{i1} & Z_2 is B_{i2} ...

where Y_j is the j'th input variable and Z_j is the j'th output variable; A_{ij} is an exponential membership function in the i'th rule associated with the j'th input and B_{ij} is a membership function in the i'th rule associated with the j'th output. For the i'th rule, which is represented by cluster center x_i^*, A_{ij} is given by

$$A_{ij}(Y_j) = \exp\left\{ -\frac{1}{2} \left(\frac{Y_j - y_{ij}^*}{s_{ij}} \right)^2 \right\} \qquad (6)$$

and B_{ij} can be any symmetric membership function centered around z_{ij}^*, where y_{ij}^* is the j'th element of y_i^*, z_{ij}^* is the j'th element of z_i^*, and $s_{ij}^2 = 1/(2a)$. Our computational scheme is equivalent to an inference method that uses multiplication as the AND operator, weights the consequent of each rule by the rule's degree of fulfillment, and computes the final output value as a weighted average of all the consequents.

There are several approaches to optimize the rules. One approach is to apply a gradient descent method to optimize the parameters z_{ij}^*, y_{ij}^*, and s_{ij} in Eqs. (5) and (6) to reduce the root-mean-square (RMS) output error with respect to the training data. Gradient descent formulas to perform this optimization can be found in [15]. This is the approach adopted by Yager and Filev [4] to optimize the initial rules obtained from the mountain method.

Another approach is to let the consequent parameter z_{ij}^* be a linear function of the input variables, instead of a simple constant. That is, we let

$$z_{ij}^* = G_{ij}\, y + h_{ij}$$

where G_{ij} is an N-element vector of coefficients and h_{ij} is a scalar constant. The if-then rules then become the Takagi-Sugeno type. As shown by Takagi and Sugeno [1], given a set of rules with fixed premise membership functions, optimizing G_{ij} and h_{ij} in all consequent equations is a simple linear least-squares estimation problem. Chiu [5] adopted this approach to optimize the rules obtained from the subtractive clustering method; optimizing only the coefficients in the consequent equations allows a significant degree of model optimization to be performed without adding much computational complexity.

A third approach also involves using the Takagi-Sugeno rule format, but it combines optimizing the premise membership functions by gradient descent with optimizing the consequent equations by linear least squares estimation. This is the ANFIS (Adaptive Network-Based Fuzzy Inference System) methodology developed by Jang [16]. When the ANFIS approach is used to optimize the same rules, the resultant model is generally more accurate than that obtained from either of the two preceding approaches. This is the preferred approach that we adopt here.

Although the number of clusters (or rules) is automatically determined by our method, we should note that the user specified parameter r_a (i.e., the radius of influence of a cluster center) strongly affects the number of clusters that will be generated. A large r_a generally results in fewer clusters and hence a coarser model, while a small r_a can produce excessive number of clusters and a model that does not generalize well (i.e., by over-fitting the training data). Therefore, we may regard r_a as an approximate specification of the desired resolution of the model, which can be adjusted based on the resultant complexity and generalization ability of the model.

4. Extracting Rules for Pattern Classification

For pattern classification applications, we first separate the data into groups according to their respective classes; subtractive clustering is then applied to the input space of each group of data individually to extract the rules for identifying each class of data.

The clusters found in the data of a given group identify regions in the input space that map into the associated class. Hence, we can translate each cluster center into a fuzzy rule for identifying the class. For example, suppose subtractive clustering was applied to the group of data for class c1 and cluster center x_i^* was found, then this cluster center provides the rule:

Rule *i*: If {input is near x_i^* } then class is c1.

Given an input vector *x*, the degree of fulfillment of rule *i* is defined as

$$m_i = e^{-\alpha \| x - x_i^* \|^2} \qquad (7)$$

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

$$A_{ij}(X_j) = \exp\left\{ -\frac{1}{2}\left(\frac{X_j - x_{ij}^*}{s_{ij}} \right)^2 \right\} \qquad (8)$$

where x_{ij}^* is the *j*'th element of x_i^*, and $s_{ij}^2 = 1/(2\alpha)$. The degree of fulfillment of each rule is computed by using multiplication as the AND operator, and, because the fuzzy system performs classification, we simply select the consequent of the rule with the highest degree of fulfillment to be the output of the fuzzy system. It would be erroneous to interpolate the output value from the different rule consequents because the class values usually have no numerical meaning, e.g., a class 2 object is not necessarily "between" class 1 and class 3 in any sense.

After the initial rules have been obtained by subtractive clustering, we use gradient descent to tune the individual x_{ij}^* and s_{ij} parameters in the membership functions (cf. Eq. 8) to minimize a classification error measure.

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

$$E = \frac{1}{2}\left(1 - m_{c,\max}(x) + m_{\neg c,\max}(x)\right)^2 \qquad (9)$$

where $m_{c,\max}(x)$ is the highest degree of fulfillment among all rules that assign *x* to class c and $m_{\neg c,\max}(x)$ is the highest degree of fulfillment among all rules that do not assign *x* to class c. Note that this error measure is zero only if a rule that would correctly classify the sample has a degree of fulfillment of 1 and all rules that would misclassify the sample have a degree of fulfillment of 0. Gradient descent formulas to optimize the membership function parameters with respect to this error measure can be found in [17]. An important feature of this optimization process is that only the rules responsible for $m_{c,\max}$ and $m_{\neg c,\max}$ are updated, since all other rules do not affect the error measure. 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). The

two-sided Gaussian function is better suited than the standard Gaussian function for capturing the relations in pattern classification data.

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 gradient descent equations derived for standard Gaussian functions are mathematically valid for two-sided Gaussian functions when the input value X_j is outside the plateau region, but are no longer valid when X_j is inside the plateau region because the error gradient there is zero. A method for handling this special situation is also presented in [17].

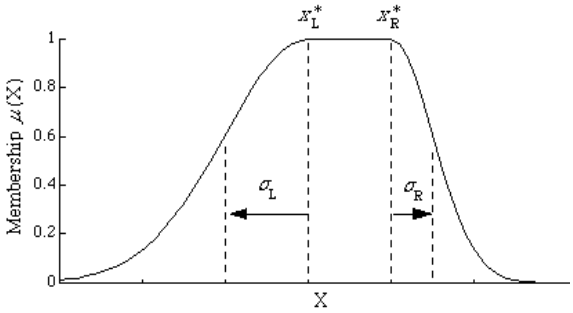

Fig. 1. A "two-sided" Gaussian function.

5. Application Examples

To illustrate the rule extraction methods, we now apply them to a function approximation problem and to a pattern classification problem.

5.1. Automobile Trip Prediction

We consider a function approximation problem in which we wish to predict the number of automobile trips generated from an area based on the area's demographics. The data used for rule extraction consists of demographic and trip data for 100 traffic analysis zones in New Castle County, Delaware. The original data set provided in [18] contains five input variables: population, number of dwelling units, number of car ownerships, median household income, and

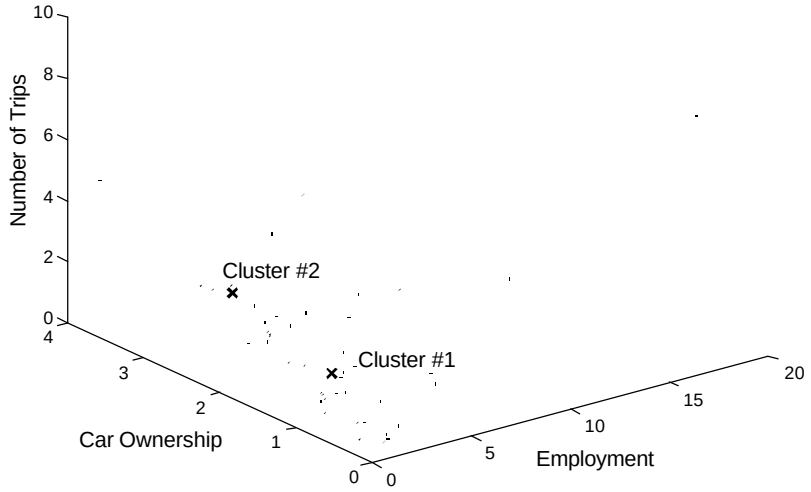

Fig. 2. Training data for the automobile trip prediction problem; all numbers are expressed in units of a thousand. The two cluster centers found are marked by X's.

total employment. However, to better graphically illustrate the rule extraction method, here we consider only two input variables: car ownership and employment. A discussion on how to select the most important input variables is beyond the scope of this paper; the interested reader is referred to [19].

Of the 100 data points, we randomly selected 75 points as training data for generating the rules and 25 points as checking data for validating the rules. Using cluster radius r_a = 0.5, two cluster centers were found in the training data. Recall that the cluster radius is expressed relative to the normalized data space, hence r_a = 0.5 corresponds to half the width of the data space. The training data and the cluster centers are shown in Fig. 2, where the cluster centers are marked by Xs. The two cluster centers were translated into two rules of the Takagi-Sugeno type, with initial premise membership functions obtained from the cluster center positions via Eq. (6). The ANFIS approach was then applied to optimize the model; that is, the rules' premise membership functions were optimized by gradient descent while the consequent equations were optimized by linear least squares estimation. The optimization process stops when the improvement in RMS error after a pass through the data becomes less than 0.1% of the previous RMS error. The resultant rules are shown in Figure 3. A comparison of the actual output versus the output predicted by the fuzzy model is shown in Figure 4. The fuzzy model has an RMS output error of 0.582 with respect to the training data and an error of 0.586 with respect to the checking data, indicating the model generalizes well. The computation time for generating this model was 2 seconds on a Macintosh computer with 68040 processor running at 25 MHz.

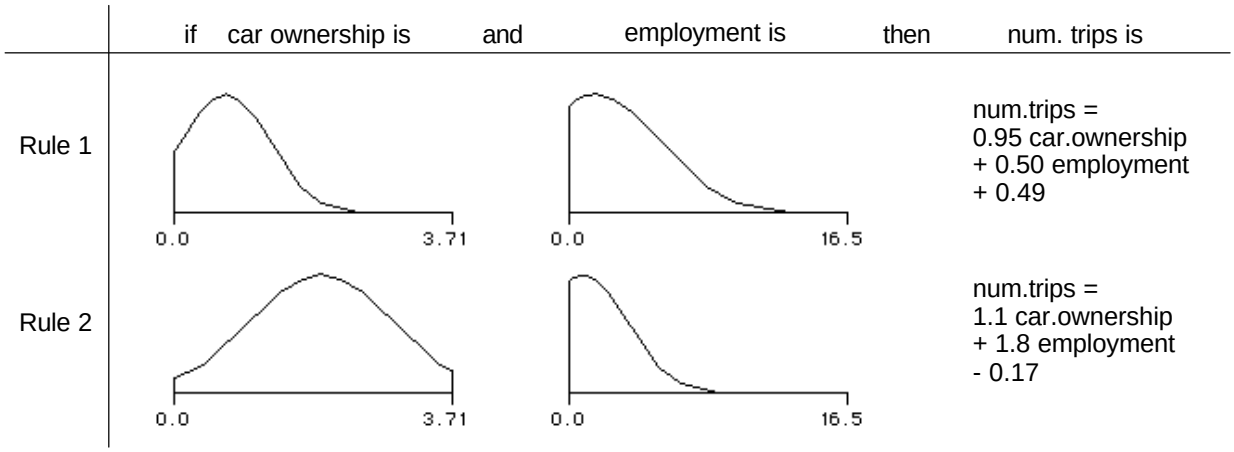

Fig. 3. Rules extracted for the trip prediction model.

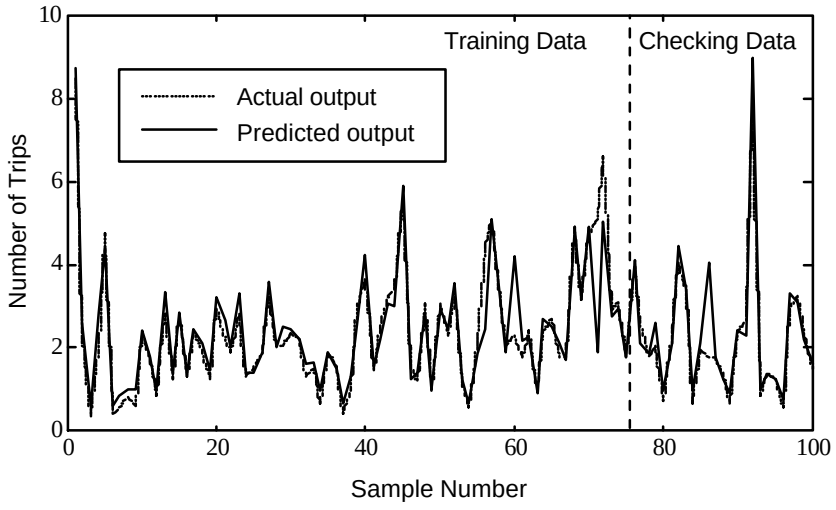

Fig. 4. Comparison of actual output values with the predicted output values for the trip prediction model. The number of trips are expressed in units of a thousand.

5.2. Iris Classification

We now consider a classical benchmark problem in pattern classification – Fisher's iris flower data [20]. 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 as training data (40 samples of each species) and 30 samples as checking data (10 samples of each species). To simplify the illustration, here we consider only two input variables: petal width and petal length (using an analysis method similar to that described in [19], we have determined the petal width and petal length to be the most important variables).

The training data were normalized and then separated into 3 groups according to their species. Using a cluster radius of 0.5, one cluster was found in each group of data, resulting in one rule for classifying each species. The rules' membership functions were allowed to be two-sided Gaussians and were optimized by gradient descent. The optimization process stops when the improvement in the error measure after a pass through the data becomes less than 0.1% of the previous error measure (the error measure is defined as the average of the individual error measures given by Eq. 9). The resultant rules are shown in Figure 5. The fuzzy classifier misclassified 3 samples out of 120 samples in the training data and classified all samples correctly in the checking data. The computation time for generating this classifier was 6 seconds on the Macintosh computer.

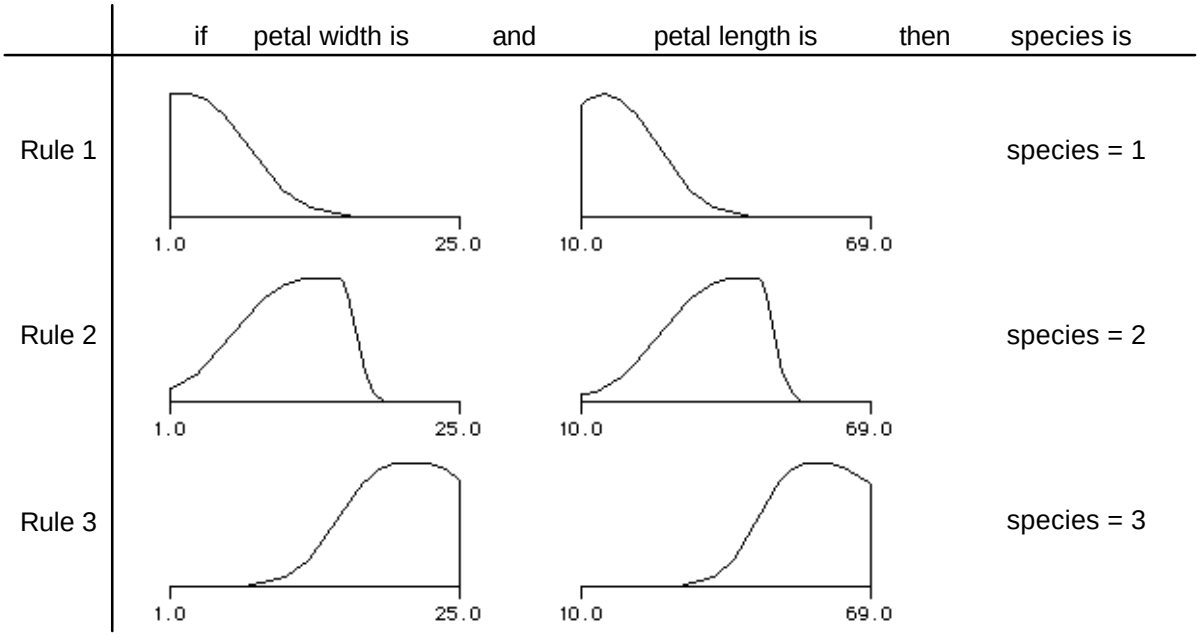

Fig. 5. Rules extracted for the iris classifier.

Looking at the rules in Figure 5, the advantages of capturing data behavior in the form of fuzzy rules become clear. We can see that flowers in species 1 generally have small petals (short petal width and short petal length); flowers in species 2 generally have medium petals; and flowers in species 3 generally have large petals. Furthermore, we see that the petal shapes generally follow the same proportion, i.e., there are no strange combinations such as a short petal width with a long petal length. Our ability to easily interpret fuzzy rules makes fuzzy rule-based models and classifiers easy to understand and debug.

6. Conclusion

We presented efficient methods for extracting fuzzy rules from data based on cluster estimation. Rules can be generated to solve both function approximation and pattern classification problems. These methods have been successfully applied to extract fuzzy rules for a variety of applications, including the modeling of human operators,

software quality prediction, selecting the best robot arm posture for performing a task [21], numeric character recognition [17], blood cell classification, underwater mine detection, and power generator leakage detection. Where comparison with neural network is available, the performance of the fuzzy rule-based model or classifier has been comparable to those based on neural network [5,17]. However, our rule extraction method is generally more efficient and easier to use than neural network, typically producing good results without any trial and error. In addition, fuzzy rule-based models and classifiers are easy to understand, verify, and extend.

7. References

[1] Takagi, T. and Sugeno, M. Fuzzy identification of systems and its application to modeling and control. *IEEE Trans. on Systems, Man & Cybernetics 15*, 116-132, 1985.

[2] Sugeno, M. and Yasukawa, T. A fuzzy-logic-based approach to qualitative modeling. *IEEE Trans. on Fuzzy Systems 1*, 7-31, 1993.

[3] Wang, L.X. Training of fuzzy logic systems using nearest neighborhood clustering. *Proc. 2nd IEEE Int'l Conf. on Fuzzy Systems (FUZZ-IEEE '93)*, San Francisco, CA, March 28-April 1, 13-17, 1993.

[4] Yager, R. and Filev, D. Generation of fuzzy rules by mountain clustering. *J. of Intelligent and Fuzzy Systems 2*, 209-219, 1994.

[5] Chiu, S. Fuzzy model identification based on cluster estimation. *J. of Intelligent and Fuzzy Systems 2*, 267-278, 1994.

[6] Ruspini, E.H. Numerical methods for fuzzy clustering. *Inform. Sci. 2*, 319-350, 1970.

[7] Dunn, J. A fuzzy relative of the ISODATA process and its use in detecting compact, well separated clusters. *J. of Cybernetics 3*, 32-57, 1974.

[8] Bezdek, J. Cluster validity with fuzzy sets. *J. of Cybernetics 3*, 58-71, 1974.

[9] Keller, J., Gray, M. and Givens Jr., J. A fuzzy k-nearest neighbor algorithm. *IEEE Trans. Syst., Man, Cybern., 15*, 580-585, 1985.

[10] Chung, F.L. and Lee, T. A fuzzy learning method for membership function estimation and pattern classification. *Proc. 3rd IEEE Int'l Conf. on Fuzzy Systems (FUZZ-IEEE '94)*, Orlando, FL, June 26-29, 426-431, 1994.

[11] Sun, C.T. and Jang, J.S.R. A neuro-fuzzy classifier and its applications. *Proc. 2nd IEEE Int'l Conf. on Fuzzy Systems (FUZZ-IEEE '93)*, San Francisco, CA, March 28-April 1, 94-98, 1993.

[12] Abe, S. and Lan, M.S. A classifier using fuzzy rules extracted directly from numerical data. *Proc. 2nd IEEE Int'l Conf. on Fuzzy Systems (FUZZ-IEEE '93)*, San Francisco, CA, March 28-April 1, 1191-1198, 1993.

[13] Bezdek, J. *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, New York, 1981.

[14] Kohonen, T. The self-organizing map. *Proceedings of IEEE 78*, 1464-1480, 1990.

[15] Wang, L.X. and Mendel, J.M. Back-propagation fuzzy system as nonlinear dynamic system identifiers. *Proc. 1st IEEE Int'l Conf. on Fuzzy Systems (FUZZ-IEEE '92)*, San Diego, CA, March 8-12, 1409-1418, 1992.

[16] Jang, J.S.R. ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Trans. Syst., Man, Cybern. 23*, 665-685, 1993.

[17] Chiu, S. Extracting fuzzy rules for pattern classification by cluster estimation. *Proc. 6'th Int. Fuzzy Systems Association Congress (IFSA '95)*, Sao Paulo, Brazil, July 22-28, vol. 2, 273-276, 1995.

[18] Kikuchi, S., Nanda, R., and Perincherry, V. Estimation of trip generation using the fuzzy regression method. *Proc. 1994 Annual Meeting of Transportation Research Board*, Washington DC, January, 1994.

[19] Chiu, S. Selecting input variables for fuzzy models. To appear in *J. of Intelligent & Fuzzy Systems*, 1996.

[20] Fisher, R.A. The use of multiple measurements in taxonomic problems. *Annals of Eugenics 7*, 179-188, 1936.

[21] Chiu, S. and Cheng, J. Automatic generation of fuzzy rulebase for robot arm posture selection. *Proc. 1st Joint Conf. of North American Fuzzy Information Processing Society (NAFIPS '94), Industrial Fuzzy Control & Intelligent Systems Conf., and NASA Joint Technology Workshop on Neural Networks & Fuzzy Logic*, San Antonio, TX, December 18-21, 436-440, 1994.