Fuzzy Rule Extraction From ID3-Type Decision Trees for Real Data

Nikhil R. Pal, *Senior Member, IEEE,* and Sukumar Chakraborty

***Abstract*—This paper proposes a method to construct a fuzzy rule-based classifier system from an ID3-type decision tree (DT) for real data. The three major steps are rule extraction, gradient descent tuning of the rule-base, and performance-based pruning of the rule-base. Pruning removes all rules which cannot meet a certain level of performance. To test our scheme, we have used the DT generated by RID3, an ID3-type classifier for real data. In this process, we made some improvements of RID3 to get a tree with less redundancy and hence a smaller rule-base. The rule-base is tested on several data sets and is found to demonstrate an excellent performance. Results obtained by the proposed scheme are consistently better than C4.5 across several data sets.**

***Index Terms*—Decision tree (DT), decision tree pruning, fuzzy rule extraction, rule-base pruning, rule-base tuning.**

I. Introduction

THERE are various approaches to fuzzy rule extraction from numerical data for classification [11]–[21]. For example, Nozaki *et al.* [15] proposed an adaptive method for fuzzy rule extraction for classification. Their method consists of two learning procedures: an error correction-based learning scheme and an additional learning procedure. The first learning procedure adjusts the grade of certainty of each rule based on its classification performance. This is followed by an additional learning procedure to realize better classification boundaries between classes. Finally, Nozaki *et al.* [15] proposed a rule pruning scheme. Exploratory data analysis, such as clustering, is also used to facilitate rule extraction. For example, Chiu [28] used a subtractive clustering method to find clusters in the training data; each cluster is then translated into a fuzzy rule.

Another commonly used approach for rule extraction is to build a decision tree (DT) from the training samples and extract rules from it. This approach is used because semantically a path of a DT and a rule are almost the same. There are different types of DTs [6], [7] available, each having its advantages and disadvantages [8], but probably the most popular one is Quinlan's ID3 [1]. The main concern about ID3 is that it can deal with categorical data only and the DT generated is a crisp one. Different variations of ID3 are now available where authors have tried to overcome these limitations of ID3 and extract fuzzy rules from the DT [16]–[24]. Quinlan also came up with a new algorithm named C4.5 [4]. Some of these algorithms are discussed below.

Manuscript received September 22, 2000; revised May 31, 2001. The work of N. R. Pal was supported in part by the Department of Science and Technology, Government of India. This paper was recommended by Associate Editor T. Sudkamp.

The fuzzy DT of Chang and Pavlidis [17] used a fuzzy decision function (not the node splitting function) at each internal node. If an internal node V has k branches, for example, then the fuzzy decision function produces a vector in $[0, 1]^k$. Each component of the decision function can be thought of as an edge weight of a branch coming out of the node V. Each leaf node of the DT has a crisp label associated with it. For an unknown $\mathbf{x} \in R^p$, the firing strength of each path from root to every leaf is computed using either product or minimum of the decision function values on the path. If minimum is used, they call it a *fuzzy decision tree*; for product, it is called a *probabilistic decision tree*. $\mathbf{x}$ is assigned the crisp label of the leaf having the highest firing strength. Chang and Pavlidis spent a majority of their paper on theoretical results about search algorithms to find the max-firing strength solution, but did not give any method for construction of the tree.

Maher and St. Clair [16] proposed an algorithm, UR-ID3, which combined uncertain reasoning with the rule set produced by ID3 to deal with uncertain and noisy training and test data. In their approach, they start with a DT made by ID3, then the attribute values associated with each branch of the tree is considered to be approximate. Triangularly shaped membership functions are used to model the approximate values. For each attribute value of a data point, a support interval is calculated using the approximate value associated with the corresponding branches of the tree. The classification of a test sample is done by considering the set of support intervals for different possible classifications. Each path from the root to a leaf is then interpreted as a rule. One problem of this algorithm is that for continuous valued data, one has to assume some quantization of the features and the performance of the algorithm may greatly depend on the optimality of this choice.

Following the idea of UR-ID3, Chi and Yan [19] proposed an algorithm to generate fuzzy rules from a similar DT. To fuzzify the crisp rules, in the case of continuous data, an interval of values is associated with each node of the tree; while in the case of categorical data, for each feature they have used a relational matrix of size $L \times L$ (where L is the number of different attributes for that feature). Then the output is calculated as a weighted sum of the firing strengths generated by all rules. The weights are learned by a two-layer perceptron network. To get the final output, they combine the outputs of the perceptron with the outputs of an optimized nearest-neighbor (NN) classifier proposed by Yan [26] using a three-layer perceptron. Apart from the problem of quantization for continuous data, for cate-

gorical data the relational matrix for each feature is another area, which needs either a subjective judgment or some guidelines.

Yuan and Shaw [22] described an algorithm to construct a fuzzy DT and then extract a rule-base from it. While constructing the DT, they used two constants, *significant level threshold* (α) and *truth level threshold* (β). β is used to determine whether a node is a leaf or not and α is used to determine the label of a data point. However, no guideline for choosing these constants has been given. For real data, they quantify each feature into C levels (C is the number of classes) using Kohonen's feature map.

Many variants of fuzzy ID3 have also been proposed. The basic idea behind fuzzy ID3 is to build a DT of ID3-type where, at each node, the best feature is selected depending on the gain in entropy. Based on some stopping criterion, a decision is made whether a node will be a leaf node or we will have further branching from it. One such criterion is that if the gain is less than some predetermined threshold, then that node becomes a leaf node. Some popular measures of gain can be found in Jun *et al.* [43]. Tsang *et al.* [23] developed one such algorithm and then they associated each branch with a weight called *local weight (LW)*. Each leaf node is also associated with a weight called *global weight (GW)*. After the tree construction, they tune these weights by a hybrid neural network. For real data, they fuzzified each feature into three linguistic attributes. Obviously, this may not give the best results for all data sets. Another similar approach is found in Ichihashi *et al.* [24] where, after the fuzzy ID3 DT is constructed, the membership functions associated with different nodes are approximated with B-Splines (for better modeling capability of complex functions). The parameters of the B-Splines are then learned using a three-layered neural network.

Recently, Wang and Yeung [25] proposed a new algorithm for generating a fuzzy DT and then extracting a rule-base from it. They first compute the relative importance of each linguistic term for each feature. Then the importance of a feature is calculated as a weighted average of the relative importance of its linguistic terms and the feature having the maximum importance is selected first. They also use the same *truth level threshold* (β) of Yuan and Shaw [22] to determine whether or not a node is a leaf. The rules extracted from the tree are of the form $IF\,(x_1\ is\ A_1\ AND\ x_2\ is\ A_2\ AND\ \cdots\ x_l\ is\ A_l)\ THEN\ y\ is\ B_k;\ CF(Lw_1, Lw_2, \ldots, Lw_l)$. Here Lw_i is the certainty factor associated with the atomic clause "x_i *is* A_i" and represents the relative importance calculated for that linguistic term of the corresponding feature. In their paper [25], Wang and Yeung compare various aspects of their method with those of [22] and fuzzy ID3. While the computational complexity of their algorithm is high, the comprehensibility of the extracted rule set are better than that of fuzzy ID3 and less than Yuan and Shaw's method.

Earlier, we proposed a DT called RID3 [2], which is an ID3-like DT that deals with real valued data. In RID3, each node except the root has a prototype vector and it corresponds to a particular class. The tree creation process uses a ranking of features, and the prototypical values of each class are computed as class centroids. To find the label of a data point, we traverse through the tree starting at the root, and at each level we test the similarity of a set of features with the prototype of the node in terms of a membership value calculated using the fuzzy c-means formula (FCM) [9] and fit the data point to the best matching node. We continue going down this way until we find some node where either the data point has a membership higher than the threshold of that node, or it is a leaf node. The thresholds are tuned using genetic algorithms (GAs) [10].

This paper is broadly divided in two parts. In the first part, we propose an improved tuning scheme for RID3 by which, along with improving the classification performance, we achieve improved computational efficiency by restricting the average classification depth. Then we propose a pruning scheme to reduce the size of the tree. The improved version of RID3 is termed "IRID3." Pruning of DTs is itself a topic of research, and there are a number of pruning methods available in the literature. A good survey paper by Esposito *et al.* [40] may be consulted for more details. Our method is conceptually closest to the *critical value pruning (CVP)* method described in [40]. Although, as pointed out by the authors, the CVP method can fail in some special cases, in our case this will not happen.

In the second part, we propose an algorithm to build a fuzzy rule-based system to classify real data points. To extract rules from the DT, we select some specific paths of the tree and convert each of them to a rule. The feature value examined at each level is converted to a clause in the antecedent part of the rule, and the terminal label of the path becomes the consequent or the outcome of the rule. The rules are fuzzy and the membership functions are of Gaussian nature, although other choices can be used. For illustration purpose, we have used the DT made by IRID3, but no special feature of IRID3 has been used. The basic assumption is that the structure of the DT should be such that, at each level, a new feature is accounted in the classification process in addition to the features used in the previous levels. The initial rule-base is tuned using gradient search, then a pruning phase removes redundant rules, if any.

We have tested our scheme on various data sets commonly used in the literature and found quite satisfactory and consistent results across all data sets. We have also compared our result with Quinlan's C4.5 [4]. The results on training data and test data are found to be consistently better than C4.5.

The rest of the paper is as follows. Section II briefly describes the RID3 algorithm. In Section III, we discuss the improved tuning and pruning of RID3. Section IV describes rule extraction, rule-base tuning, and rule-base pruning techniques. We conclude in Section V.

II. A Brief Description of RID3

We are given a p-dimensional data set $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}_i \subset R^p$ with c classes $C = \{c_1, c_2, \ldots, c_c\}$. The ith component of any data point represents the value of the ith feature f_i (we denote the feature set as $F = \{f_1, f_2, \ldots, f_p\}$).

RID3 involves three steps:

1) feature ranking;
2) tree construction;
3) threshold tuning.

The feature ranking step is *optional* as we can use any arbitrary order of the features, but it is a *desirable* step because it can help to reduce the size of the tree. With a good feature ranking, important features will be considered in the higher levels of the tree and most data points will be classified in the top levels without going down to deeper levels.

A. Feature Ranking

Although any feature ranking scheme [27] can be used, we discuss here a simple scheme for the sake of completeness.

For the ith feature we calculate a feature evaluation index (FEI_i)

$$FEI_i = ND_i * CD_i * (1 + ICS_i)^{-1} \tag{1}$$

where

$$ND_i = \sum_{j=1}^{c} (v_{ij} - m_{ij})^2 \tag{2}$$

$$CD_i = \sum_{j=1}^{c} \left(\sum_{\mathbf{x}_k \subset X_j} (x_{ik} - m_{ij})^2 \right) \tag{3}$$

$$ICS_i = \sum_{j=1}^{c} \left(\sum_{k=1}^{c} (m_{ij} - m_{ik})^2 \right). \tag{4}$$

ND stands for nondistinguishability, *CD* stands for cluster-dispersion, and *ICS* stands for inter-cluster-separation. Here m_{ij} is called the *actual prototype* of the ith feature corresponding to class j and is calculated using the formula $m_{ij} = (1/|C_j|) \sum_{\mathbf{x}_k \subset X_j} x_{ik}$, $j = 1, 2, \ldots, c$; $i = 1, 2, \ldots, p$. v_{ij} is the *natural prototype* of the ith feature corresponding to class j calculated using the FCM algorithm. X_j is the set of data points corresponding to class j. Lower FEI corresponds to better ranking. For a detailed discussion of FEI, readers may refer to [2].

B. Tree Construction

Let the ordering of the features done by a feature ranking scheme be $f_{R_1}, f_{R_2}, \ldots, f_{R_p}$. Let m_{ij}, $i = 1, 2, \ldots, p$; $j = 1, 2, \cdots, c$ be the ith component of the jth actual prototype. The tree is constructed as follows.

1) Each internal node, including the root, has exactly c children, each child corresponds to a class, and there are p levels in the tree.
2) If the jth child of the ith node at level $l - 1$ is denoted as N_{ij}^l, $i = 1, 2, \ldots, c^{l-1}$; $j = 1, 2, \ldots, c$, then the node is characterized by the following:
   a) It has got a prototype vector of dimension l.
   b) The first $l - 1$ components of the prototype are exactly the same as that of its parent.
   c) The lth component is the prototypical value of feature f_{R_l} corresponding to class j, i.e., $m_{R_l j}$.
3) Each node, except the root, has a threshold associated with it (denoted by θ_{ij}^l). Initially all thresholds are set to some α where $\alpha \in (0, 1)$.

C. Tuning Using Genetic Algorithms

To determine the label of an unknown data point, we start at the root node. The value of the most important feature (i.e., f_{R_1}) is tested with all children of the root node and passed on to the node where the membership (representing the similarity of the data point with the prototypical value of the node, calculated using the FCM formula) is the maximum. If the membership is greater than the threshold of that node, then the data point is labeled with the label of that node, or else it goes further down in the same manner till either a leaf node is encountered or the membership value exceeds the threshold.

Clearly the performance of the tree depends on the thresholds. We use GA to tune the threshold values. We use binary coding for the thresholds to form strings of 0s and 1s as chromosomes. For each set of thresholds, we get one such chromosome, which is evaluated using the percentage of misclassification (M) as the fitness function E, i.e., $E = M$. Chromosomes are copied to the mating pool with the probability proportional to their fitness (better performance, more number of copies). Then we do the usual crossover and mutation on these chromosomes to get a mating pool of new chromosomes. In this way, the process evolves and generates new sets of thresholds giving better performance [2]. We run the GA for $T_{G_{max}}$ number of iterations and pick up the best set of thresholds for the final DT.

III. IRID3: An Improved Version of RID3

For a p-dimensional data distributed in c classes, the number of nodes in the tree will be $(c^{p+1} - 1)/(c - 1)$. Such a tree may have too many redundant nodes, and rule-base generated from RID3 may (and usually does) have many redundant rules. Moreover, RID3 does not pay any attention to the depth at which classification decisions are made. A DT should be optimized so that decisions are made at lower depth (nearer to the root)—this can reduce the cost of measurement of features, which is often very important for applications relating to medical diagnosis. Therefore, we make the following two modifications.

1) The tuning process is improved so that classification can be done using fewer number of nodes (features).
2) A pruning process is introduced to remove nodes which are not useful.

A. Improved Tuning

We want the samples to get classified at lower depths (i.e., nearer to the root), so that nodes at higher depths of the tree become redundant. In RID3 the objective function of GA was $E = M$, where M is the percentage misclassification. This is changed to $E = M + \eta L$, where L is the average depth of classification and η is the "Penalty for Depth." While tuning the thresholds, initially we set η to a value such that ηL is greater than M. This means that reduction of classification depth receives a higher priority over minimization of misclassification. Thus the thresholds are so tuned that the data points get classified at lower depths (i.e., near the root). As the GA evolves, we gradually continue to decrease the value of η so that minimization of missclassification starts dominating. We start with a value of η such that $M < \eta L$ and then reduce the value of η to zero in k equal steps. In other words, after $T_{G_{max}}/k$ iterations

of GA, η is reduced once. Note that $k = T_{G_{\max}}$ corresponds to linear reduction of η after every iteration of GA. Moreover, the last $T_{G_{\max}}/k$ iterations of GA are evolved with $\eta = 0$. This refines the tree to minimize only the misclassification. In the reported results, we have used $k = 10$. With this modification, we are able to classify the data points with low average depth without losing performance.

B. Pruning

The pruning step tailors the DT to keep only the required nodes. After tuning is over, we classify the training samples once and mark the nodes that are taking part in this process. Then we remove all nodes that are NOT marked. Since our tuning is such that classification is done at lower depths (nearer to the root), we get a large number of unmarked nodes. Therefore, after pruning we get a tree of drastically reduced size. Consequently, this will prevent test data points from traversing any part of the original tree which is not supported by the training data. In addition to this, because of pruning of the DT, in the rule generation phase we get an efficient (both in terms of size and quality) rule-base. After tuning and pruning, the thresholds of the reduced DT are retuned by GA, again for a fixed number of iterations.

C. Complexity Analysis of IRID3

The different steps of IRID3 are analyzed one by one.

As stated earlier, the feature ranking step is an optional one, and any method of feature ranking can be used, so we do not consider this step for the time complexity analysis.

The complexity of the tree construction is $\bigcirc(|T|)$ where $|T|$ is the number of nodes in the tree. The initial tree being a c-ary complete tree of maximum depth p, $|T| = \sum_{i=0}^{p} c^i = (c^{p+1} - 1)/(c - 1)$. Therefore, the complexity of the tree construction is $\bigcirc(c^p)$.

There are many expressions available for the time complexity of GA in the literature, e.g., Ankenbradt showed [41] using a recurrence relation and the theory of induction that the average and worst case time complexity of GA is $(evaluate.m \log m)/\log r$, where m is the population size, r is the fitness ratio, and $evaluate$ is the time complexity of the domain dependent evaluation function. In [41], Ankenbradt assumed that there would be no mutation, only crossover will be used to generate the new mating pool after selection is over. Droste *et al.* [42] showed that the expected run time of the $(1 + 1)$ evolutionary algorithm with mutation probability $1/K$ is $\bigcirc(K \log K)$ where K is the number of variables. This expression holds for linearly separable functions where only mutation and no crossover is used. However, in the present case of threshold tuning, the function is not linearly separable, as well as we use both crossover and mutation for generation of new chromosomes, so neither of the above two expressions can be used to represent the time complexity of our case.

However, we run the GA for a maximum number of iterations $T_{G_{\max}}$, so the time for convergence is not of much importance here. Thus, for our case, the time complexity is simply $\bigcirc(evaluate)$. In evaluating once, the following computations are involved: generation of the chromosomes $(\bigcirc(|T| = c^p))$, getting the thresholds from the chromosomes $(\bigcirc(|T| = c^p))$, and evaluation of the training samples $(\bigcirc(npc))$. Therefore, the time complexity of tuning is $\bigcirc(npc + c^p)$.

The pruning is a very simple step where we only evaluate the training samples once and remove the unused nodes from the tree, which involves visiting every node once. Therefore, the computation involved is $\bigcirc(npc + c^p)$.

Therefore, the overall complexity of IRID3 $\bigcirc(npc + c^p)$.

D. Results on RID3 and IRID3

We tested RID3 with a large number of data sets and the results obtained are quite good and fairly consistent across the data sets. Before presenting the results, we provide brief descriptions of the data sets.

1) Description of the Data Sets Used:

IRIS [29]: This is a four-dimensional data consisting of 150 points divided into three classes of equal size 50. The four features are: sepal length, sepal width, petal length, and petal width. IRIS has been used in many research investigations related to pattern recognition and has become a sort of benchmark data.

Mango_leaf [30]: This is a data set with number of features $p = 18$, (i.e., 18-dimensional data) with 166 data points. It has three classes representing three kinds of mango. The feature set consists of measurements like area (A), perimeter (Pe), maximum length (L), maximum breadth (B), petiole (P), length+petiole $(L + P)$, length/petiole (L/P), length/maximum breadth (L/B), $(L + P)/B$, A/L, A/B, A/Pe, upper_midrib/lower_midrib, upper Pe/Lower Pe, and so on. The terms "upper" and "lower" are used with respect to the maximum breadth position.

Crude_oil [31], [32]: Gerrid and Lantz chemically analyzed Crude_oil samples from three zones of sandstone. It is a five-dimensional data with 56 points and three classes named wilhelm, submuilinia, and upper (mulinia, second subscales, first subscales). The features of Crude_oil are vanadium (in percent ash), iron (in percent ash), beryllium (in percent ash), saturated hydrocarbons (in percent area), and aromatic hydrocarbons (in percent area).

Norm4 [33]: The data set Norm4 is a sample of 800 points consisting of 200 points each from the four components of a mixture of four class 4-variate normals.

Vowel [34]: Telugu is one of the Indian languages spoken in the southern part of the country. The Vowel data set consists of 871 samples of discrete phonetically balanced speech samples for the Telugu vowels in consonant-vowel nucleus-consonant (CNC) form. These samples are generated from three male informants in the age group of 25 to 30 years. The first three formant frequencies are taken as the features. Vowel has six substantially overlapped classes.

Glass [35]: The Glass Identification data set has six classes with nine continuously valued attributes: refractive index, sodium, magnesium, aluminum, silicon, potassium, calcium, barium, and iron. The unit of measurement of all attributes but refractive index is weight percent in corresponding oxide. The six classes are named as building_windows_float_processed, building_windows_non_float_processed, vehicle_windows_float_ processed, containers, tableware, and headlamps.

Myo_electric [39]: This is a four-dimensional data set consisting of 72 points divided into two classes; class 1 has 38 points while class 2 has the remaining 34 points.

2) Results: Table I shows the comparison of the RID3 with its improved version IRID3. In column 3, the numbers in () list the percentage reduction of number of nodes after tuning and pruning. Except for Myo_electric data, in all other cases the reduction is by more than **75**% and in some cases it is even more than **95**%. For Myo_electric data, the reduction is by 51%.

Columns 4 and 5 give the average depth of decision making (classification) with the training data. In IRID3 the "improved tuning" is expected to reduce the average depth of classification than that of RID3. Columns 4 and 5 depict that this is indeed the case for all data sets except Mango_leaf and Norm4. The average depth of classification for Mango_leaf and Norm4 is little more than that of RID3. This can happen due to the following reason: pruning may result in some degradation of performance of the DT because at each node, the total membership of unity is redistributed among its remaining children. Therefore, after pruning, we retuned the DT for a fixed number of iterations using GA to minimize only the misclassification (i.e., without the "penalty for depth" term). This additional tuning phase may increase the average depth of classification to reduce the number of misclassification. Comparing columns 7 and 8, we find that with such a massive reduction of tree size, the performance of IRID3 remains almost the same as that of RID3.

TABLE I
COMPARATIVE STUDY OF RID3 AND IRID3

Data Set Used	No. of nodes in the decision tree		Average depth of classification		Percentage of correct classification	
	RID3	IRID3	RID3	IRID3	RID3	IRID3
IRIS	121	6 (95)	1.35	1.35	98	98
Mango_leaf	364	67 (81.4)	1.63	2.66	70.5	69
Crude_oil	364	18 (95)	2.79	1.93	91.1	91.1
Myo_electric	31	15 (51.6)	3	2.22	93.1	93.1
Norm4	341	81 (76.2)	2	2.4	83	81
Vowel	259	58 (77.6)	2.1	1.2	59	60.7
Glass(FEI)	9331	253(97.3)	3	2.86	41.1	35.5

IV. GENERATION OF FUZZY RULES FROM THE DECISION TREE

The final DT obtained after pruning is used for generation of fuzzy rules. The three major steps involved in rule generation are

Step 1) extraction of the initial rule-base;
Step 2) tuning of the rule-base parameters;
Step 3) pruning of the rule-base.

A. Extraction of the Initial Rule-Base

With the final DT, we classify the training sample once and the following steps are executed.

Step 1) Whenever a data point is classified at a particular node, we mark that node as a *classifier node*.
Step 2) In every node we maintain the mean (μ) and standard deviation (σ) of the data points that pass through it.

The rules are then constructed as follows.

1) Each path starting from the root up to a *classifier node* is converted to a rule.
2) The number of clauses in the antecedent part of the rule is the same as the number of levels encountered in the corresponding path.
3) Each node encountered in the path (except the root node) is converted to a clause and is associated with an initial membership function of Gaussian-type with *position* $= \mu$ and *width* $= \sigma$, i.e., membership function is given by $m(x) = e^{-(x-\mu)^2/2\sigma^2}$, where x is the corresponding feature value of the data point.
4) The consequent of the rule, i.e., the class assigned by the rule, is the same as the class label assigned by the corresponding path.

It should be noted that the antecedent parts of different rules generated by the above scheme generally have different number of clauses. A path involving two levels will be converted to a rule having two clauses in the antecedent part, whereas a path involving five levels will be converted to a rule having five antecedent clauses. To calculate the final output of a rule, we use product as the intersection operator. For example, S_{ir}, the firing strength of the rth rule on the ith data point is calculated as $S_{ir} = \prod_{j=1}^{l} m_{ir}^{j}$, where m_{ir}^{j} is the membership of the R_jth feature value (R_j is the feature with rank j) of the ith data point in the fuzzy set associated with the jth clause of the rth rule and l is the total number of clauses in the rth rule. The initial μ and σ of the Gaussian membership function associated with a node are taken as the mean and standard deviation of the feature values of all training data points passing through that node. Note that, every level of the tree corresponds to a unique input feature.

When we get a data point with unknown class, we calculate the firing strength of all rules in the rule-base and select the rule with the maximum firing strength. Then we label the data by the consequent of that rule. Let us illustrate the rule extraction procedure by an example. Consider IRIS data ($p = 4$, $c = 3$). The feature ranking according to the feature evaluation index, FEI, described in Section II-A is 4, 3, 2, 1. After tuning and pruning we get the tree shown in Fig. 1.

In Fig. 1, there are four leaf nodes and the number of extracted rules is also four. However, in general, the number of rules may be greater than the number of leaf nodes, because whenever any decision node (internal node) classifies some data points, then the path from the root up to that particular node also gets converted to a rule. The fuzzy rules extracted from this DT are the following:

R_1: **IF** x_4 **is** A_{41} **THEN Class 1**
R_2: **IF** x_4 **is** A_{42} **AND** x_3 **is** A_{32} **THEN Class 2**
R_3: **IF** x_4 **is** A_{12} **AND** x_3 **is** A_{33} **THEN Class 3**
R_4: **IF** x_4 **is** A_{43} **THEN Class 3**

Here x_4 is the fourth feature (having rank 1 in this case) value of a data point and A_{41} is the fuzzy set whose initial membership function is defined by the μ and σ associated with node 2 of the tree. For R_1, x_4 is the petal length and A_{41} is a Gaussian function with $\mu = 0.246$ and $\sigma = 0.104\,326$. Thus, semantically R_1 says: **if petal length is close to** 0.246, **then class is 1**. Considering the scatterplot [Fig. 2] of features 3 and 4 of IRIS, we see that R_1 conforms to the structure of the data. In Fig. 2,

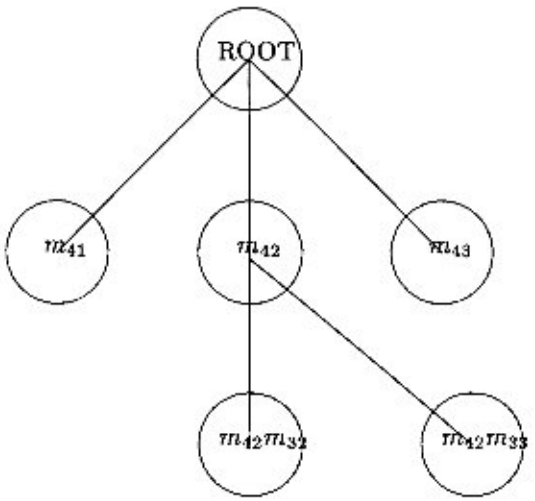

Fig. 1. Final DT obtained on IRIS data by IRID3.

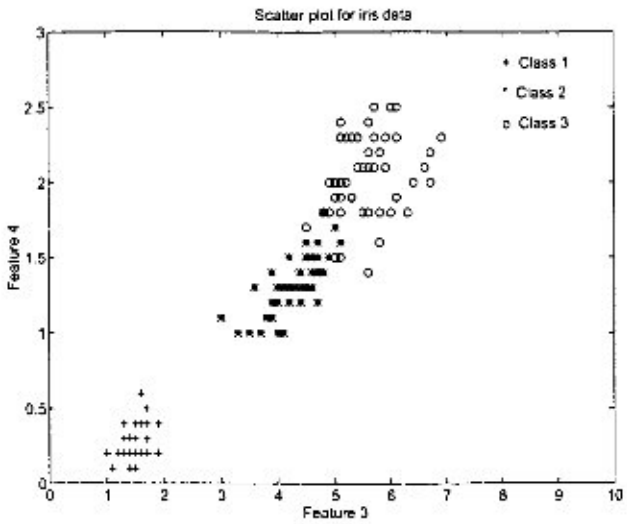

Fig. 2. Scatter plot of features 3 and 4 of IRIS.

the x-axis corresponds to feature 3 and the y-axis corresponds to feature 4. Points corresponding to different classes are represented by different symbols in Fig. 2.

B. Tuning of the Extracted Rules

Classification by IRID3 is threshold-based; consequently a node as well as its child *belonging to the same class* may both classify some data points and thus both may get marked as *classifier nodes*. As a result two rules may be generated. Clearly the antecedent clauses of these two rules will be almost similar. In fact, the antecedent part of the rule corresponding to the parent node is contained in the antecedent of the rule corresponding to the child (which has got one extra clause) and both of them classify for the same class. The chance that one of these two rules will become redundant is very high.

The μ and σ of the fuzzy sets associated with the rules are very important and proper choice of these will certainly increase the performance of the rule-base. In our case these parameters are initialized by the *mean* and *standard deviation* of the data points passing through that node. We do not consider how many of them are properly classified and how many are misclassified. Moreover, a lot of data points belonging to other classes may also pass through this node. Hence, with the μ and σ initialized by the *mean* and *standard deviation* of those data points may (usually will) not be the best possible choice. Thus, they should be tuned for a better performance. We have used gradient descent technique to tune the parameters of the rules which we discuss next.

1) Tuning of the Rules: Let α_r denote the firing strength of the rth rule on a data point $\mathbf{x}$. Let there be N rules with n_i rules representing class i. Let $\alpha_{i_{\max}}$ be the maximum firing strength of a rule that represents the ith class. Then, we can minimize $\sum_{\mathbf{x}} \sum_{i=1}^{c} (\alpha_{i_{\max}} - t_i)^2$, where $t_i = 1$ for the correct class and $t_i = 0$ for the rest. In this case with every data point parameters of at most c rules will be updated. If $(\alpha_{i_{\max}} - t_i) = 0$ for some group of rules, then no rule from that group will be updated.

Another alternative error function, which we use, is as follows. Let $\alpha_{c\max}$ be the maximum firing strength of a rule giving correct classification, generated by the rule $R_{c\max}$, and $\alpha_{w\max}$ be the maximum firing strength of a rule giving wrong classification, generated by the rule $R_{w\max}$. We try to minimize the error $E = \sum (1 - \alpha_{c\max} + \alpha_{w\max})^2$ as in [28]. After each training sample is classified, we update the parameters of the pair of rules $\{R_{c\max}, R_{w\max}\}$ using the following:

$$\mu_{R_{c\max_i}} = \mu_{R_{c\max_i}} - \eta_\mu \frac{\delta E}{\delta \mu_{R_{c\max_i}}} \tag{5}$$

$$\sigma_{R_{c\max_i}} = \sigma_{R_{c\max_i}} + \eta_\sigma \frac{\delta E}{\delta \sigma_{R_{c\max_i}}} \tag{6}$$

$$\mu_{R_{w\max_i}} = \mu_{R_{w\max_i}} + \eta_\mu \frac{\delta E}{\delta \mu_{R_{w\max_i}}} \tag{7}$$

$$\sigma_{R_{w\max_i}} = \sigma_{R_{w\max_i}} + \eta_\sigma \frac{\delta E}{\delta \sigma_{R_{w\max_i}}}. \tag{8}$$

Here the index i refers to clause number in the corresponding rule, i.e., for the first antecedent clause $i = 1$, for the second clause $i = 2$, and so on. The tuning process is repeated until the rate of decrement in E becomes negligible or the iteration reaches a *maximum* limit.

Since a Gaussian membership function is extended to infinity, for any data point all rules will be fired to some extent. In our implementation, if the firing strength is less than a threshold, ϵ ($\approx$0.01), then the rule is not assumed to be fired. Thus, under this situation, the rule-base extracted by the system may not be complete with respect to the training data. In fact, this can also happen when we use membership functions with triangular or trapezoidal shapes. Next we provide a scheme for ensuring completeness of the rule-base with respect to the training data.

If the training data set contains a data point which does not fire any rule, then we convert that data point into a rule. We generate a rule having the maximum number of clauses (which is the same as the maximum height of the final DT) in the antecedent, the μs of the antecedent clauses are initialized by the corresponding feature values of the data point and σs of the clauses are initially set to some small predetermined value γ, $\gamma \in \langle 0, 1 \rangle$. The consequent of this rule is assigned by the actual label of the data point. This rule will classify the data point

under consideration and also other data points in its neighborhood, which otherwise might have remained unclassified, if this rule was not there in the rule-base. Since σ is small, this rule will not interfere with other rules.

C. Pruning the Rule-Base

The proposed rule extraction method, however, does not rule out the possibility of presence of redundant rules as well as bad rules, which should be removed from the rule-base for a better performance as well as efficiency, as discussed earlier. We propose a very simple but effective scheme for rule minimization. After tuning is over we classify the training sample once more with the rule-base and do the following.

1) With each rule, we maintain two variables, *Correct_Count* and *Incorrect_Count*.
2) Whenever any data point is classified correctly by a rule, we increase its *Correct_Count* by 1.
3) Whenever any data point is classified incorrectly by a rule, we increase its *Incorrect_Count* by 1.
4) After all training samples are exhausted, the rules that satisfy any one of the following two criteria are deleted.
   a) A rule whose *Correct_Count* is less than its *Incorrect_Count*.
   b) A rule whose *Correct_Count* is less than a certain percentage of the cardinality of the training set.

Criterion a) represents a case where a rule wrongly classifies more points than it correctly classifies; consequently, such a rule should be deleted. Criterion b) suggests a situation where a rule fails to represent an adequate number of data points and such rules should also be deleted to avoid bad generalization. After rule pruning is finished, we run the tuning process for some more iterations to reattenuate the parameters of the remaining rules in the changed rule-base environment. The entire rule generation process is schematized in Figs. 3 and 4.

D. Complexity Analysis of Rule-Base Extraction, Tuning, and Pruning

The three main steps involved are rule extraction, rule tuning, and rule-base pruning. We analyze the complexity of each step one by one.

It is obvious from the IRID3 algorithm that the maximum number of *classifier node* (i.e., which classifies at least one data point) is n (we assume that $\sigma^p \gg n$) and in the worst case all these nodes will be at the pth level, so complexity of the rule-extraction process in the worst case is $\bigcirc(pn)$ and the number of rules is n. If we use some other DT for rule generation, this number may change.

Since in the worst case, every rule will have p antecedents, and classification of a single data point involves finding the firing strength on every rule, the cost of classification of every data point is $\bigcirc(pn)$.

In the tuning phase, after classifying each data point we tune the parameters of only two rules, so it involves $\bigcirc((p+1)n+4p)$ computation [$\bigcirc(pn)$ for computation of firing strength, $\bigcirc(n)$ for finding the two rules to be updated and $\bigcirc(4p)$ for updating the parameters of the rules]. Therefore, the complexity of one iteration (a complete pass through the training data) is $\bigcirc((p+1)n^2+4pn)$. In our simulation, the training is continued for at most $I_{T_{\max}}$ iterations. Therefore, the complexity of the entire tuning process is $\bigcirc((p-1)n^2+4pn)$.

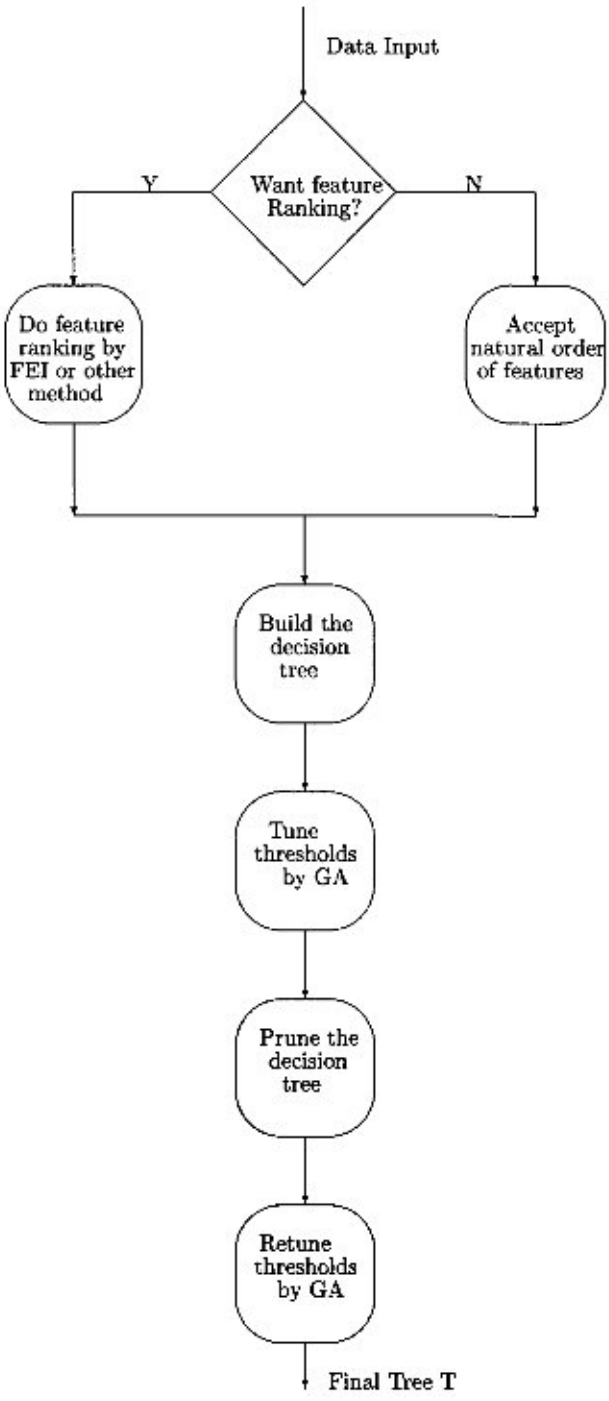

Fig. 3. Steps involved in building the DT from training data.

In pruning phase we evaluate the training data once, which is an $\bigcirc(pn^2)$ process.

Therefore, the overall complexity in the worst case is also $\bigcirc(pn^2)$.

E. Results

The performance of the rule-base extracted from IRID3 is shown in Table II. Comparing columns 2 and 3 of Table II, we

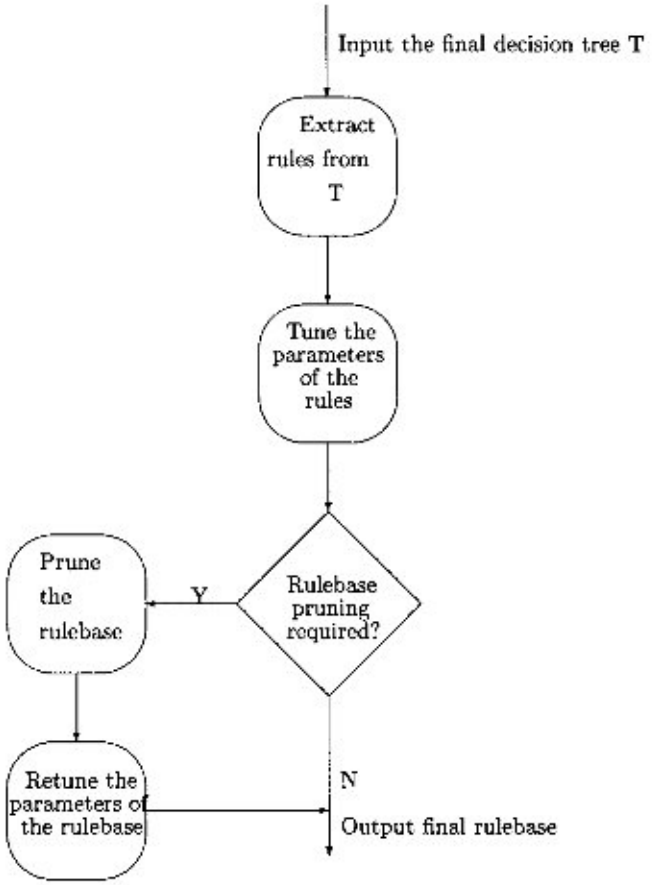

Fig. 4. Steps involved in the rule generation process.

find that tuning of the rule-base improves the performance of the rule-base a lot, which implies that the initialization of the parameters of the rules was not the optimum, as discussed earlier. For all data sets (with the exception of Glass), we used the ranks suggested by the simple index FEI; while for Glass, we consider two different sets of five features, one of which is as per the FEI ranks.

Comparing columns 3 and 4, we find that although for IRIS, Crude_oil, and Vowel data, the performance remains the same before and after pruning, for Mango_leaf, there is a little degradation in performance (less than 1%). This has happened possibly because the pruning process has deleted some rules as per Criterion b), which were correctly classifying a few points and the residual rule-base is not able to correctly classify those points. For Myo_electric, Norm4 and Glass data, the performance is improved by pruning. This means that there were some bad rules in the original rule-base, which were incorrectly classifying some data points.

From columns 5 and 6 we find that except for IRIS, the number of rules in the rule-base is decreased for all data sets. Sometimes, more than **70%** rules of the original rule-base are deleted, which shows the effectiveness of our pruning scheme.

Comparing Table II with Table I, we find that except IRIS, for all data sets, there is a significant improvement in the recognition score achieved by the rule-base; for IRIS the performance remains the same. For Glass data, as we mentioned earlier, use of a different set of five features (using a different ranking), the recognition score by IRID3 is improved by 6%. For this case, the percentage of correct classification by the rule-base becomes 79.4%, i.e., an improvement of about **38%.**

TABLE II
PERFORMANCE OF THE RULEBASE ON DIFFERENT DATA SETS

Data Set Used	Percentage of correct classification			Number of Rules	
	Not Tuned	Tuned	Pruned	Not Pruned	Pruned
IRIS	96	98	98	4	4
Mango_leaf	50	83.1	82.5	31	14
Crude_oil	85.7	91.1	91.1	11	7
Myo	80.6	95.8	98.6	8	5
Norm4	65.3	93.8	94.8	41	12
Vowel	56.8	81.5	81.5	15	7
Glass	36.4	77.1	79.4	88	49
Glass with FEI	21	61.7	65.4	54	30

TABLE III
PERFORMANCE COMPARISON OF OUR RULEBASE AND RULEBASE FROM C4.5

Data Set Used	Our Rulebase			C4.5 Rulebase		
	No. of Rules	% Misclassification Trng data	% Misclassification Tst data	No.of Rules	% Misclassification Trng data	% Misclassification Tst data
IRIS	4	1.3	4	3	2.7	6.7
Mango_leaf(1)	4	31.0	41.5	5	9.5	32.9
Mango_leaf(2)	21	10.7	32.9			
Crude_oil	11	6.9	33.3	4	0.0	44.4
Myo	4	0	8.3	3	0	22.2
Norm4	11	5.5	4.8	9	4	6.75
Glass	15	21.5	47.3	10	12.5	59.5

So far, we have not evaluated the generalization ability of the rules extracted by our scheme. Next, we do so and also compare our results with the outputs of C4.5. For this we partition the data set X into training (X_{Tr}) and test (X_{Te}) sets such that $X = X_{Tr} \cup X_{Te}$; $X_{Tr} \cap X_{Te} = \phi$; and $|X_{Tr}| = |X_{Te}| = |X|/2$.

Table III compares the performance of the proposed rule-based system with the rules from C4.5 in terms of generalization capability. For Crude_oil, although C4.5 achieves a training error of 0%, the test error percentage is 44.5, which is 11% more than that achieved by the IRID3 rule-base. Similarly, for Glass data, the training error is 9% lower for C4.5 but the test error is 12% more than that by our rule-base. In the case of IRIS, of course, our rule-base achieves lower error rate for both training and test data. For Mango data C4.5 performs better than the rule-base both for the training and the test data (the first row for Mango data). This poor performance may be attributed to the severe pruning of the rule-base. If we do not prune the rule-base, the performance of the proposed system is comparable to that of C4.5 for both training and test data. This corresponds to the second row for Mango_leaf. For Myo and Norm4, the performance of both classifiers on the training data is comparable but for the test data, our rule-base outperforms C4.5. To summarize, the fuzzy rule-base extracted by our system is found to have a better generalization than C4.5 and except for Glass and Crude oil, the number of rules extracted by both systems are comparable.

In order to establish further the effectiveness of our scheme we refer to some results reported in [16], which compared four classifiers using 75% of IRIS for training and the remaining 25% for testing. The experiment was repeated for three such

[35] R. C. Holte, "Very simple classification rules perform well on most commonly used data set," *Mach. Learn.*, vol. 11, pp. 63–91, 1993.
[36] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Trans. Neural Networks*, vol. 3, pp. 683–697, May 1992.
[37] H. Ishibuchi, T. Nakashima, and T. Murata, "Performance evaluation of fuzzy classifier systems for multi-dimensional pattern classification problems," *IEEE Trans. Syst., Man, Cybern. B*, vol. 29, pp. 601–618, Oct. 1999.
[38] H. Ishibuchi and M. Nii, "Minimizing the measurement cost in the classification of new samples by neural network based classifiers," in *Proc. IIZUKA'98*, vol. 2, pp. 634–637.
[39] K. Mehrotra, C. K. Mohan, and S. Ranka, *Elements of Artificial Neural Networks*. Cambridge, MA: MIT Press, 1996.
[40] F. Esposito, D. Malerba, and G. Semeraro, "A comparative analysis of methods for pruning decision trees," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 19, pp. 476–491, May 1997.
[41] C. A. Ankenbradt, "Theory of convergence and a proof of the time complexity of genetic algorithms," in *Foundations of Genetic Algorithms*, G. J. Rawlins, Ed., pp. 53–68.
[42] S. Droste, T. Jansen, and I. Wagner. (1997) A rigorous complexity analysis of the $(1 + 1)$ evolution strategy for separable functions with Boolean inputs. [Online] Tech. Rep. SFB 531, ISSN 1433-3325. Available: http://sfbci.informatik.unidortmund.de/reihaci.html
[43] B. H. Jun, C. S. Kim, and J. Kim, "A new criterion in selection and discretization of attributes for the generation of decision trees," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 19, pp. 1371–1375, Dec. 1997.

Nikhil R. Pal (M'91–SM'00) received the B.Sc. degree (with honors) in physics and the M.S. degree in business management from the University of Calcutta, Calcutta, India, in 1979 and 1982, respectively. He received the M.Tech. and Ph.D. degrees in computer science from the Indian Statistical Institute, Calcutta, in 1984 and 1991, respectively.

Currently, he is a Professor in the Electronics and Communication Sciences Unit of the Indian Statistical Institute. From September 1991 to February 1993, July 1994 to December 1994, October 1996 to December 1996, and January 2000 to July 2000, he visited the Computer Science Department of the University of West Florida, Pensacola. He was also a Guest Faculty of the University of Calcutta. His research interest includes image processing, pattern recognition, fuzzy sets theory, measures of uncertainty, neural networks, genetic algorithms, and fuzzy logic controllers. He has coauthored a book entitled *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing* (Norwell, MA: Kluwer, 1999), coedited a volume of *Advances in Pattern Recognition and Digital Techniques, ICAPRDT'99*, and edited the book *Pattern Recognition in Soft Computing Paradigm* (Singapore: World Scientific, 2001). He is an Associate Editor of the *International Journal of Fuzzy Systems* and the *International Journal of Approximate Reasoning*

Dr. Pal is an Associate Editor of IEEE TRANSACTIONS ON FUZZY SYSTEMS and IEEE TRANSACTIONS ON SYSTEMS MAN AND CYBERNETICS—PART B (electronic version).

Sukumar Chakraborty received the B.E. degree in electrical engineering from Jadavpur University, Calcutta, India, in 1993, and the M.Tech. degree in computer science from the Indian Statistical Institute, Calcutta, in 1995.

From August 1995 to July 1996, he was with Wipro Systems as a Senior Software Engineer. From July 1996 to July 1999, he was with the R&D Center of VEDIKA Software Pvt. Ltd. as Senior Design Engineer. Currently, he is with Interra Information Technologies (India) Pvt. Ltd., Salt Lake, Calcutta. His research interests include pattern recognition, neural networks, fuzzy systems, and machine learning.