RID3: An ID3-Like Algorithm for Real Data

N. R. PAL
SUKUMAR CHAKRABORTY

Machine Intelligence Unit, Indian Statistical Institute, Calcutta-700035, India

and

A. BAGCHI

Computer and Statistical Service Center, Indian Statistical Institute, Calcutta-700035, India

ABSTRACT

An ID3-like tree-based classifier named RID3 has been proposed. The classifier requires a ranking of the features according to their discriminability between classes. We propose a simple but effective scheme for feature ranking. RID3 first constructs a preliminary tree with a default threshold at each node. If the performance of the initial tree is not satisfactory, then the threshold at each node is tuned with genetic algorithms. RID3 is found to outperform nearest-neighbor classifier for all the data sets considered.

1. INTRODUCTION

In pattern recognition [1], an object is characterized by a set of features. The values of these features can be of three types:

(i) numerical,
(ii) categorical or nonnumerical,
(iii) fuzzy or ambiguous.

Numerical values of features have a proximity relation between them. For example, let us consider that a man is described by his HEIGHT, WEIGHT, etc. In this case, a man with height 6 feet is closely related to another person whose height is 5 feet 11 inches. So for numeric-valued features, we can extract some relationship between the data points by analyzing their distances.

Categorical values are nonnumeric. Each feature can have a number of values, called attribute values, and an object or a data point is characterized by values of these attributes (sometimes these values may be represented by presence or absence of these attributes). As an illustration, a man may be described by HAIR-COLOR, EYE-COLOR, etc. Let the different values of the attribute HAIR-COLOR be black, brown, and blond, and the values for EYE-COLOR be black, blue, and gray. So a man in this scheme can be described as HAIR-COLOR: brown, EYE-COLOR: blue.

We can denote these different attributes by numeric values also, such as HAIR-COLOR: $1 \rightarrow$ *black*, $2 \rightarrow$ *blond*, $3 \rightarrow$ *brown*. But we can never expect the proximity relation that we enjoyed in the case of numeric data, because "black is nearer to blond than to brown" carries no meaning (although 1 is nearer to 2 than 3). This makes categorical data unsuitable for proximity/distance-based analysis.

Finally, the fuzzy feature values are not well defined, so one cannot expect a precise value for a feature. For example, assume that a man is described by features like COMPLEXION and PHYSIQUE. COMPLEXION can have different values such as "Fair," "dark," etc., which are not precisely defined, but can be expressed in terms of membership functions.

We are accustomed to using linguistic values (not necessarily fuzzy) to represent categorical feature values, and human processing of such information creates no problem, but machine processing of such irreducibly nonnumeric data requires special attention to achieve a useful pattern-recognition system.

The ID3 [1] approach to classification consists of a procedure for synthesizing an efficient discrimination tree for classifying patterns that have nonnumeric values. The ID3 approach makes use of the labeled data, and determines how features might be examined in sequence until all the labeled examples have been classified properly. If the data set used to construct the tree is a representative of the much larger ensemble of patterns comprising the original body of the data, then we can expect a very large gain through use of ID3.

ID3 can be very effective under certain conditions, but should not be used beyond its scope of validity. Two serious limitations of the ID3 approach are:

1. it uses categorical data only;
2. it cannot deal with partial information.

In ID3, one way to handle numerical data is to convert the data to categorical data by dividing each feature into a number of groups and assigning one label for each group. Then each data point can be represented as a vector of labels depending on the attribute value ranges. The problem of this technique is that the number of categories is to be assumed and the finer details of the data set will be lost, because, for a range of values, the categorical value will be the same. In this article, we have proposed a classification algorithm which can deal with numerical data like IRIS, Mango_Leaf, Crude_Oil, etc. [2–4].

Our classifier is a decision-tree-based approach. Each node has a prototype vector and corresponds to a particular class. The tree creation process uses a ranking of the features, and the prototypical values of each class are computed as the class centroids. In order to find the class of an unlabeled data point, we start at the root of the tree. At each level, we test the distance of a particular feature value of the data point from some prototype (through the computation of membership values using the fuzzy c-means [5] formula), and then fit the data into the node with highest membership value. This is how we go down towards the leaf level until we get some node where the data point fits with a considerable amount of agreement. The decision-making process requires a threshold for each node which is learned using genetic algorithms [6].

We tried our proposed scheme on three data sets, and the performance is found to be better than the nearest-neighbor classifier [1].

2. RID3: THE PROPOSED SCHEME

We are given a p-dimensional data set $X=\{x_1,x_2,\ldots,x_n\}$, $x_i \in R^p$, with c classes $C=\{C_1,C_2,\ldots,C_c\}$. The ith component of any data point x_k represents the value of the ith feature f_i (we denote the feature set as $F=\{f_1,f_2,\ldots,f_p\}$). The problem is to construct a classification tree, whose each node is either a leaf-node or a decision-node. A leaf-node corresponds to a particular class, whereas a decision-node corresponds to a class as well as a test node with respect to some feature. There are three distinct parts of this algorithm:

(i) feature selection,
(ii) tree construction, and
(iii) tree tuning.

2.1. FEATURE RANKING

There are many techniques for feature ranking. Some of these techniques are based on interclass and intraclass distances, some are based on probabilistic, fuzzy models, while others are based on neural networks. Each approach has its advantages and drawbacks.

Features should be ranked according to their ability to discriminate different classes. The discriminating ability of a feature is dependent on the type of classifier we use to evaluate it. For example, the most important feature for training a multilayer perceptron (MLP) may be different from the most important feature for a nearest prototype classifier.

The ability to classify patterns by machine relies on an implicit assumption that classes occupy distinct regions in the feature space. Intuitively, the greater the distance between classes, the better is the chance of successful recognition. One approach could, therefore, be to select those features for which the classes are maximally separated. We present a feature-selection scheme [7] based on this principle, below.

As denoted earlier, let $X=\{x_i|x_i \in R^p,\ i=1,2,\ldots,n\}$ be the data set. There are c classes $C_1, C_2, \ldots, C_c$, with a priori class probability P_i for a class C_i, such that $\bigcup_{i=1}^{c} C_i = X$, $C_i \cap C_j = \phi\ \forall i \neq j$, and $|C_i| = n_i$. Let $Y=\{y_i|y_i \in R^{p'},\ p' \leq p,\ i=1,2,\ldots,n\}$ be a data set generated from X by some feature-selection technique, where the kth component of y_i, is equal to some lth component of x_i. In other words, Y is generated by deleting some $(p-p')$ rows of X, if X and Y are represented as matrices of orders $p \times n$ and $p' \times n$, respectively.

Now the matrices, within-cluster-scatter S_w and the between-cluster-scatter S_b, can be defined as

$$S_w = \sum_{i=1}^{c} P_i \frac{1}{n_i} \sum_{k=1}^{n_i} (y_{ik} - m_i)(y_{ik} - m_i)^t \tag{1}$$

and

$$S_b = \sum_{i=1}^{c} P_i (m_i - m)(m_i - m)^t. \tag{2}$$

Here m_i is the sample mean vector of the ith class, i.e.,

$$m_i = \left(\sum_{k=1}^{n_i} y_{ik} \right) n_i, \tag{3}$$

and m is the mixture sample mean, i.e.,

$$m = \sum_{i=1}^{c} P_i m_i. \tag{4}$$

Intuitively, for the feature-ranking task we like to maximize $\mathrm{tr}(S_b)$ and at the same time minimize $\mathrm{tr}(S_w)$. To achieve this, we can maximize

$$J(Y) = \frac{\mathrm{tr}(S_b)}{\mathrm{tr}(S_w)}, \tag{5}$$

where $\mathrm{tr}(A_{n\times n}) = \Sigma_{i=1}^{n} a_{ii}$, $A = [a_{ij}]_{n\times n}$.

The main drawback of the above criterion function ((5)) is, if for a particular feature subset (S), a class (C_i) is well scattered and a portion of (C_i) is overlapped with another class (C_j) such that their centroids (with respect to features in S) are far away, then $J(Y)$ (for features in S) may be greater than that for another feature subset, $S' \neq S$, which separates the two classes in such a fashion that a single hyperplane may pass between them but the centroids (with respect to the features in S') are not so apart. Intuitively, the second feature set is better than the first one although the criterion function may indicate the reverse. For the feature-ranking task, we take $p' = 1$, i.e., we compute $J(Y)$ for each feature (Y is one-dimensional vector), and rank the feature according to $J(Y)$.

In this investigation, we have proposed a simple yet effective method of ranking features. We have used the ranks obtained by both $J(Y)$ and the proposed scheme for constructing the classification tree as well as to classify the data points. In our tree-based approach, we go on classifying a data point with respect to different features, one by one, until we are able to make a satisfactory decision about its class.

2.1.1. A New Feature-Ranking Scheme

In pattern recognition, we call a feature good if it can discriminate between different classes. Thus, a good feature should not show much variation within a class but should have significantly different values for different classes. To get the feature evaluation index, we compute three indices: nondistinguishability (ND), intercluster separation (ICS), and class dispersion (CD), as follows.

First, we compute the natural prototypes of each class, where by "natural prototype" we mean the prototype that can be obtained by a clustering algorithm (ignoring the class labels of the data points). One can use the "hard c-means" or the "fuzzy c-means" (FCM) or some other clustering algorithm on the data set X to get c clusters. In the present case, we have used the FCM algorithm which, for the sake of completeness, is briefly discussed next.

Fuzzy c-means algorithm. The FCM algorithm [5] can be used to construct a fuzzy c-partition of a given data set. The fuzzy c-means formulates the clustering problem as a weighted least-square optimization problem.

Let the *prototype* vectors for the c classes be $V=\{v_1,v_2,\ldots,v_c\}$, $v_i \in R^p$, and u_{ik} be the membership of x_k $(x_k \in X)$ to the ith class. Then the FCM problem can be written as

$$\textit{minimize } J_m(U,V\colon X) = \sum_{k=1}^{n} \sum_{i=1}^{c} (u_{ik})^m (d_{ik})^2 \tag{6}$$

subject to

(i) $0 \leqslant u_{ik} \leqslant 1$, $\forall i,k$,
(ii) $\Sigma_i u_{ik} = 1$, $\forall k$, and
(iii) $0 < \Sigma_k u_{ik} < n$, $\forall i$.

Here U is the membership matrix, $m>1$ is a fuzzifier which controls the fuzziness in the resultant partition matrix U, $d_{ik}=\|x_k - v_i\|=$ the Euclidean distance between x_k and v_i. However $\|\ \|$ can be any other inner product induced norm also.

The necessary conditions for optimality of J_m, which are given below, can be derived using Langrangian method:

$$v_i = \sum_{k=1}^{n} (u_{ik})^m x_k \Big/ \sum_{k=1}^{n} (u_{ik})^m, \qquad 1 \leqslant i \leqslant c \tag{7}$$

and

$$u_{ik} = \left(\sum_{j=1}^{c} (d_{ik}/d_{jk})^{2/(m-1)} \right)^{-1}, \qquad 1 \leqslant k \leqslant n, \quad 1 \leqslant i \leqslant c. \tag{8}$$

The algorithm can be started initializing either on U or on V. In our implementation, we initialized on U. Using the value of U, we compute the centroids in (7). Then the new centroids are used to compute U using (8).

The process is iterated in this way between (8) and (7) until either $\|U_t - U_{t-1}\| < \epsilon$ or $\|V_t - V_{t-1}\| < \epsilon$ or both are satisfied.

Here U_t and V_t represent the membership and the centroid matrices at the tth iteration and ϵ is a small positive number. We calculated the natural centroids $V = \{v_1, v_2, \ldots, v_c\}$, with $\epsilon = 0.0001$. For FCM, the choice of an optimal m is still an open question. Theoretically, it is shown [5] that as $m \to \infty$, $u_{ik} \to 1/c$, $\forall i = 1, 2, \ldots, c$; i.e., with higher values of m, the fuzzy clusters become indistinguishable. On the other hand, as $m \to 1^+$, u_{ik} becomes crisp; $u_{ik} \to 1$ for the closest prototype and $u_{ik} \to 0$ for the rest. Thus, for low values of m, the fuzziness in the partition is lost. These facts suggest that neither a high nor a low value of m is desirable. Empirically, it is found that FCM works better in the neighborhood of $m = 2$. In our experiment, we tried different values of m around $m = 2$, and the performance of the system is found to be consistent. We report the result for $m = 2.5$ just as an illustrative case.

Now we compute the set of actual centroids $m = \{m_1, m_2, \ldots, m_c\}$, $m_i \in R^p$, considering the actual labels of the data points. In other words, we compute for each feature i, $i = 1, 2, \ldots, p$,

$$m_{ij} = \frac{1}{|C_j|} \sum_{x_k \in X_j} x_{ik}, \qquad j = 1, 2, \ldots, c. \tag{9}$$

So m_{ij} is the mean of the ith feature values of the data points belonging to class j. m_{ij} will be called the actual prototype of the feature i corresponding to the class j.

Note that, if the data have well-separated clusters and we use a good clustering algorithm, then the natural centroids and the actual centroids will be close. Moreover, for a good feature (a feature with better discriminating power), the corresponding component of the natural centroid and the actual centroid will be more close. Using m, V, X, we calculate three measures for determining the feature ordering:

(a) *NonDistinguishability*$_i$ $(ND_i) = \sum_{j=1}^{c}(v_{ij} - m_{ij})^2$,
(b) *InterClusterSeparation*$_i$ $(ICS_i) = \sum_{j=1}^{c}(\sum_{k=1}^{c}(m_{ij} - m_{ik})^2)$,
(c) *ClassDispersion*$_i$ $(CD_i) = \sum_{j=1}^{c}(\sum_{x_k \in X_j}(x_{ik} - m_{ij})^2)$.

ND Measures the distance between a component (feature) of the natural centroids (computed without considering the class labels), and the actual centroids (computed using the labels of the data points). Thus, if the clusters are distinguishable with respect to a feature, the ND for that feature will be low, i.e., for a good feature, ND should be low.

ICS Measures the distances between the cluster centroids with respect to a particular feature. Intuitively, for well-separated clusters, this distance should be high.

CD Measures the total distances of each data point (using only one feature) from its cluster centroid. Clearly, for compact class structures, CD should be low.

For a good feature, we expect that ND should be low, whereas ICS should be high and CD should be low.

Thus, the overall feature evaluation index (FEI) for the ith feature can be defined as

$$FEI_i = (ND_i * CD_i)/ICS_i. \tag{10}$$

A problem with (10) is that when, for a feature, all the data points have the same value, clearly ICS will become zero. To get around this problem, we modify (10) as

$$FEI_i = (ND_i * CD_i)/(1 + ICS_i). \tag{11}$$

Since ICS can never be negative, (11) can be used safely. The features are ranked according to this index and are used in the classification tree according to this ranking. The lower the value of FEI for a feature, the higher should be its rank.

Suppose the data set is such that with respect to feature i, the centroids are well separated but the classes have overlap. In this case, the actual centroids based on feature i may be well separated, but the natural centroids are likely to be away from the actual centroids, and they may not be well separated because of the overlap between the classes. Consequently, ND_i will be high, resulting in a high value of FEI_i. Thus, FEI_i is not expected to have the drawback of $J(Y)$ discussed earlier.

2.1.2. Results on Feature Ranking

We have used three data sets: Anderson's IRIS [2], Mango_Leaf [3], and Crude_Oil [4].

IRIS is a four-dimensional data set consisting of 150 points divided into three classes of equal size 50. The four features $\{f_1, f_2, f_3, f_4\}$ are {Sepal Length, Sepal Width, Petal Length, and Petal Width}. IRIS has been used

TABLE 1
Feature Ranking for IRIS

Feature no.	Rank by $J(Y)$	Rank by FEI
$f1$	3	3
$f2$	4	4
$f3$	1	1
$f4$	2	2

in many research investigations related to pattern recognition and has become a sort of benchmark-data.

Mango_Leaf, on the other hand, 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 such as 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, uppermidrib/lowermidrib, upper Pe/lower Pe, and so on. The terms upper and lower are used with respect to maximum breadth position.

Gerrid and Lantz [4] chemically analyzed crude oil samples from three zones of sandstone. It is a five-dimensional data set with 56 data points and three classes, Wilhelm, Sub-Muilinia, and Upper (Mulinia, second subscales, first subscales).

The features are vanadium (in percent ash), iron (in percent ash), beryllium (in percent ash), saturated hydrocarbons (in percent area), and aromatic hydrocarbons (in percent area).

The ranking obtained by $J(Y)$ and FEI for different data sets are presented in Tables 1–3. From these tables, we find that the ranks obtained for IRIS using both schemes are the same, while for the other two data sets, the obtained ranks are quite different. Consequently, as we will see later, the decision trees for Mango_leaf and Crude_oil created based on ranks obtained by $J(Y)$ and FEI will be different, and their performance will also be different.

2.2. *CREATION OF THE TREE*

The basic classifier in our algorithm is a tree which is made once for all, using the training data set. After building up of the tree, we are able to predict the class of any data point (unlabeled).

TABLE 2
Feature Ranking for Mango_Leaf

Feature no.	Rank by $J(Y)$	Rank by FEI
$f1$	11	7
$f2$	5	13
$f3$	13	18
$f4$	6	15
$f5$	3	9
$f6$	16	17
$f7$	18	11
$f8$	17	12
$f9$	1	2
$f10$	8	16
$f11$	14	6
$f12$	9	4
$f13$	15	3
$f14$	2	8
$f15$	7	14
$f16$	4	10
$f17$	12	5
$f18$	10	1

Each node in the decision tree is either a leaf node (a decision node) or an internal node (a testing node). Each leaf node solely represents a class (unique). If any data point reaches this node, after traversing from the root of the tree (the traversing mechanism is explained later in this section), we will conclude that class of the data point is that represented by the leaf node. On the other hand, each internal node represents a test with respect to a feature, as well as it can also represent a class. For internal node, if

TABLE 3
Feature Ranking for Crude_Oil

Feature no.	Rank by $J(Y)$	Rank by FEI
$f1$	3	2
$f2$	2	1
$f3$	5	5
$f4$	1	4
$f5$	4	3

the computed membership value is greater than the threshold, then we assign the data point to the class represented by that node.

For construction of the tree, we need two things:

1. an ordering of the features of the given data set; let the ordering be $f_{i_1}, f_{i_2}, \ldots, f_{i_p}$, as done in feature-ranking section, and
2. for each feature f_i, the prototypes in each class, i.e., m_{ij} for $i = 1,2,\ldots,p$, $j = 1,2,\ldots,c$ (as calculated in (9)).

Now the tree is constructed as follows:

(i) Each internal node has exactly c children, each child corresponds to a class, and there are p levels in the tree.

(ii) Let us denote the jth child of the ith node of level $l-1$, $i = 1,2,\ldots,c^{l-1}$ *and* $j = 1,2,\ldots,c$, by N_{ij}^l, i.e., N_{ij}^l is a node at level l. The structure of the node N_{ij}^l is that it has a prototype vector m_{ij}^l of dimension l, of which the first $l-1$ components are inherited from its parent, and the lth component is the value of the prototype of the feature f_{i_l} for class j.

(iii) Each node N_{ij}^l has a threshold value θ_{ij}^l which is initially set to 0.5 for all nodes. These thresholds are updated later to get an improved performance of the classifier.

As an example, let us consider the IRIS data set. The ranking of the features is

$$\mathbf{f}_3, \mathbf{f}_4, \mathbf{f}_1, \mathbf{f}_2,$$

and the constructed tree is shown in Figure 1.

Since f_3 is the most important feature for IRIS, in level 1, the classification is done with respect to f_3 (feature 3). This is depicted by the three children of the root node at level 1. In the next level, the tree is then grown using f_4 (feature 4), the second important feature for IRIS. Thus, for each of the three nodes at level 1, we get three children for level 2. Figure 1 shows only the subtrees for classes 1 and 3. The process is continued until we reach level 4.

2.3. TRAVERSING MECHANISM ALONG THE TREE

The depth of the tree is equal to the number of features "p." Each node contains the level of itself and a prototype vector whose dimension is equal to the level. Each node has a class label and has c children, if it is not a leaf node.

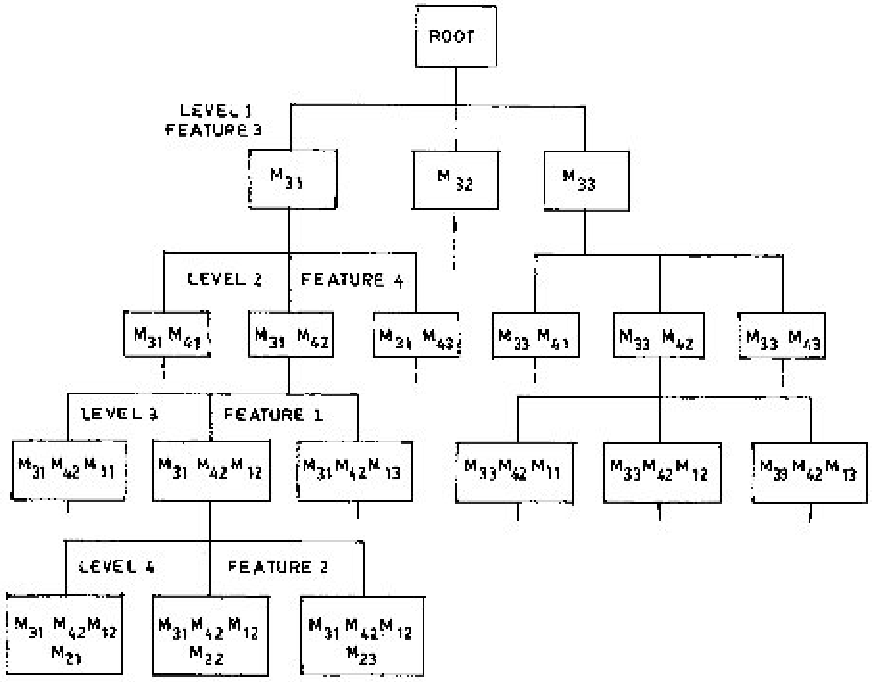

Fig. 1. The classifier tree constructed by RID3 for IRIS data.

Given a data point $x=\{x_1, x_2, \ldots, x_p\}$, we proceed as follows for labeling it.

Step 1: Start from the root, i.e., N_{00}^0.

Step 2: If we are in the ith node of level $l-1$, then its c children are the nodes $N_{i1}^l, N_{i2}^l, \ldots, N_{ic}^l$ whose prototypes are $m_{i1}^l, m_{i2}^l, \ldots, m_{ic}^l$.

Step 3: We calculate the membership of the data point x to all these nodes using the formula given in (8). The membership to the kth child is given by $\mu_k=(\sum_{j=1}^{c}(d_k/d_j)^{2/(m-1)})^{-1}$, $1 \leqslant k \leqslant c$. Here $d_k^2=\|x^l-m_{ik}^l\|^2$, where $x^l=(x_{l1}, x_{l2}, \ldots, x_{ll})$ is an l-dimensional vector and $l_1, l_2, \ldots, l_c$ is the ranking of the features. In other words, x^l is obtained considering the most important l components of x in order.

Step 4: Let $\mu_m=\max(\mu_1, \mu_2, \ldots, \mu_c)$. If $\mu_m \geqslant \theta_{im}^l$, we conclude that x belongs to the class N_{im}^l (θ_{ij}^l is the threshold of the node N_{ij}^l), else we set our current node of operation as N_{im}^{l}, and repeat steps 2 to 4 until either we reach a node where the computed membership value is greater than the threshold of that node, or we reach the leaf level. If, at the leaf level, we do not get a membership greater than the threshold, then we either conclude that the data point is ambiguous, or we assign x to the class represented by the leaf node where we have reached.

2.4. *THE IMPORTANCE OF THRESHOLD*

The threshold plays an important role in the proposed classification scheme. Suppose that our data set has three classes, and that class 1 is well separated from classes 2 and 3, which have a good amount of overlapping. Now an observation from class 1 comes for testing, and its membership values to the three different children at the very first level (computed based on the most important feature) are something like 0.9, 0.05, 0.05. For such a case, even if we fix the threshold at 0.8, the system assigns the data point to the right class, *class* 1.

Now consider a data point which actually belongs to class 2 but, as far as the most important feature is concerned, it lies midway between the corresponding prototypes of class 2 and class 3. Suppose the computed membership values are, say, 0.1, 0.5, 0.4. Since our threshold is 0.8, we cannot make any decision at this level. So we have to go down along the tree, and it may happen that the membership is never able to achieve such a high value as 0.8, because some other feature can drag the data point to some other class. This can happen, as all features are not always useful.

The previous discussion makes it clear that the choice of the threshold values is a decisive factor for good classification performance. Moreover, it also suggests that some nodes should have high threshold (the nodes corresponding to well-separated classes), whereas some others may need to have low thresholds to avoid unnecessary traversals and to reduce the use of less important features.

In our algorithm, we learned the thresholds using genetic algorithms, which are explained in the next section.

3. TREE TUNING USING GENETIC ALGORITHMS

Genetic algorithms (GAs) are probabilistic heuristic search processes based on natural genetic systems. They are highly parallel and believed to be robust in searching global optimal solution of complex optimization problems. They recombine structural information to locate new points in the search space with expected improved performance.

GAs are capable of solving a wide range of complex optimization problems using three simple genetic operations (reproduction/selection, crossover, and mutation) on coded solutions (chromosomes/strings) for the parameter set, not the parameters themselves, in an iterative fashion. GAs consider several points in the search space simultaneously, which reduces the chance to converge to a local optimum. They use the payoff or penalty (i.e., objective) function called the fitness function and do not need any other auxiliary information.

GAs exploit historical information to speculate on new search points by the crossover operation on a pair of potential strings selected by the reproduction/selection operation. Mutation, on the other hand, is a secondary genetic operation of genetic algorithms. It occasionally alters a random bit position of a randomly selected string. It has a great importance to sustain the genetic diversity, which reduces the chance of getting stuck to a local optimum. Mutation also helps the algorithm to recover information, which is sometimes essential to obtain a good solution, lost in the earlier generations.

Let us consider the problem of optimizing a complex function having n parameters $x_1, x_2, \ldots, x_n$. To solve such a problem, genetic algorithms start with a set of initial strings/chromosomes $P = S_i$: $i = 1, 2, \ldots, M$, as the initial approximations of the parameter set. In GA literature, P is called a population. Each string S_i represents a coded version of an approximate solution set $a_i = (a_{i1}, a_{i2}, \ldots, a_{in})$. Usually a binary string of length $L = n.l$ is taken as a string or chromosomal representation of an approximation. A substring comprising bits $(i-1)*l$ through $i.l$, $i = 1, 2, \ldots, n$, represents an approximation of the ith parameter. The population of the strings then undergoes a sequence of three genetic operations to produce usually an improved population. These operations are selection, crossover, and mutation. This process is repeated until some stopping criterion is reached.

Figure 2 gives a schematic description of the basic structure of genetic algorithms.

3.1. GA FOR FINDING OPTIMAL SET OF THRESHOLDS

For our classification tree, we have to find out an optimal threshold for each node. This means that the number of parameters is equal to the number of nodes in the tree. For a data set with n features and c classes, the number of nodes is equal to $N = c^n$. Each string/chromosome comprises the binary representation of all the thresholds. We use seven-bit

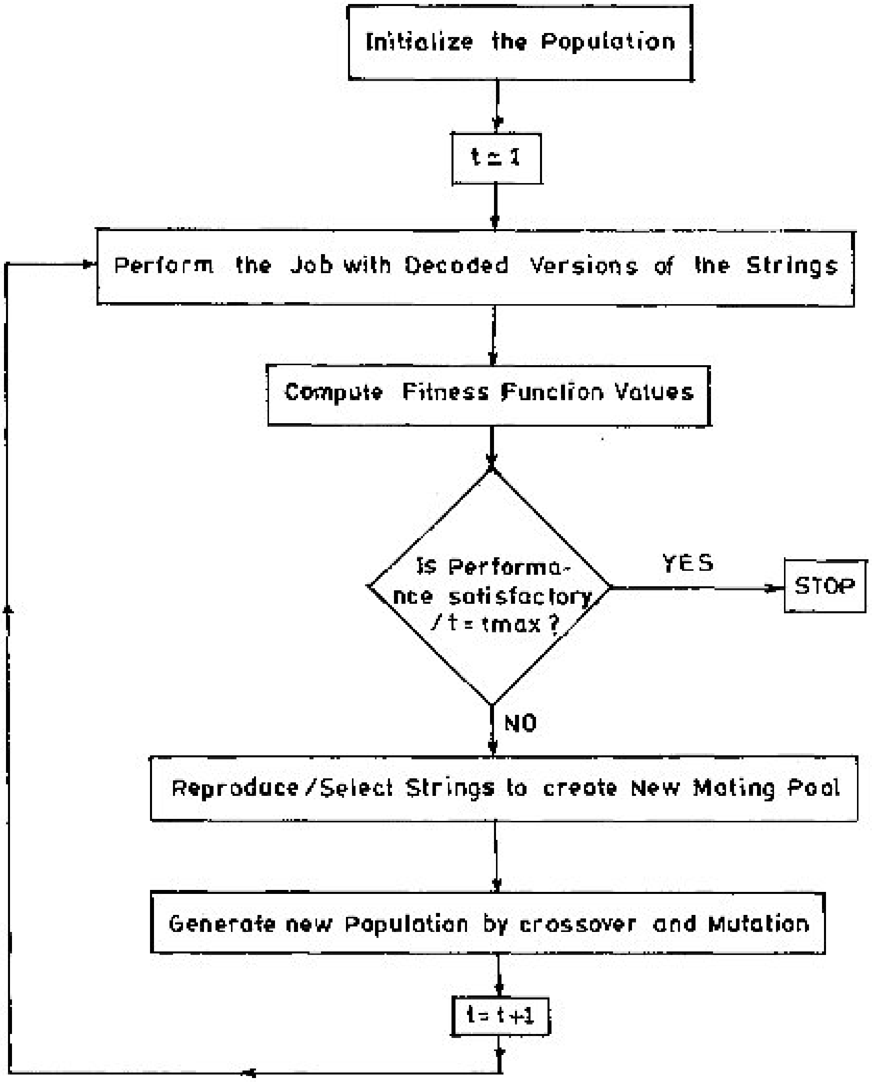
Initialize the Population
t = 1
Perform the Job with Decoded Versions of the Strings
Compute Fitness Function Values
Is Performance satisfactory / t = tmax ?
YES
STOP
NO
Reproduce / Select Strings to create New Mating Pool
Generate new Population by crossover and Mutation
t = t+1

Fig. 2. The basic steps of genetic algorithms.

representation for the individual threshold. The fitness function or the objective function is to minimize the number of wrong classifications by the classifier. The different genetic operations are performed on these strings.

Selection is a process in which potential strings (strings with higher fitness value) of the population P are copied into a mating pool, $T = \{S_i: S_i \in P\}$, depending on their fitness function values. More specifically, selection takes P as input and produces an output T. The string $S_i \in P$ having fitness value f_i is replicated $p_i = f_i / \Sigma_i f_i * |T|$ times, $|T|$ is the size of the mating pool.

Using the set of thresholds corresponding to each string, we test the classifier with the training set and evaluating the fitness function, i.e., compute the number of correct classifications. Then, depending on the fitness values corresponding to different sets of thresholds, the strings are copied into the mating pool proportionately.

Crossover operation produces offspring by exchanging information between two potential strings selected randomly from the mating pool generated by the selection process. For example, consider two strings $a = 0101010101$ and $b = 1010101010$ of length 10 selected randomly from the mating pool. A random position 7 is selected for crossing over. After crossover, the two strings are $a = 0101010010$ and $b = 1010101101$.

Mutation is an occasional alteration of a random bit. A random bit of a randomly selected string in the population is selected and the bit value is reversed. Mutation is necessary for the following three reasons:

(i) to regain the information lost during early generations,
(ii) to obtain a bit value which does not exist in the locus of that bit in any of the strings in the population,
(iii) to sustain the genetic diversity in order to reduce the chance of getting stuck to a local optimum.

In Figure 3, we summarize the entire process starting from clustering of the training data down to classification of unlabeled data. Figure 3 shows the steps as used in this investigation, but the underlying philosophy of RID3 is quite general, and some of the steps can be replaced by other alternatives. For example, the feature ranking can be done using any other method. Similarly, the threshold tuning algorithm can also be changed.

4. RESULTS ON CLASSIFICATION

To evaluate the performance of RID3, we have used the same three data sets—IRIS, Mango_leaf, and Crude_Oil. The results of our algo-

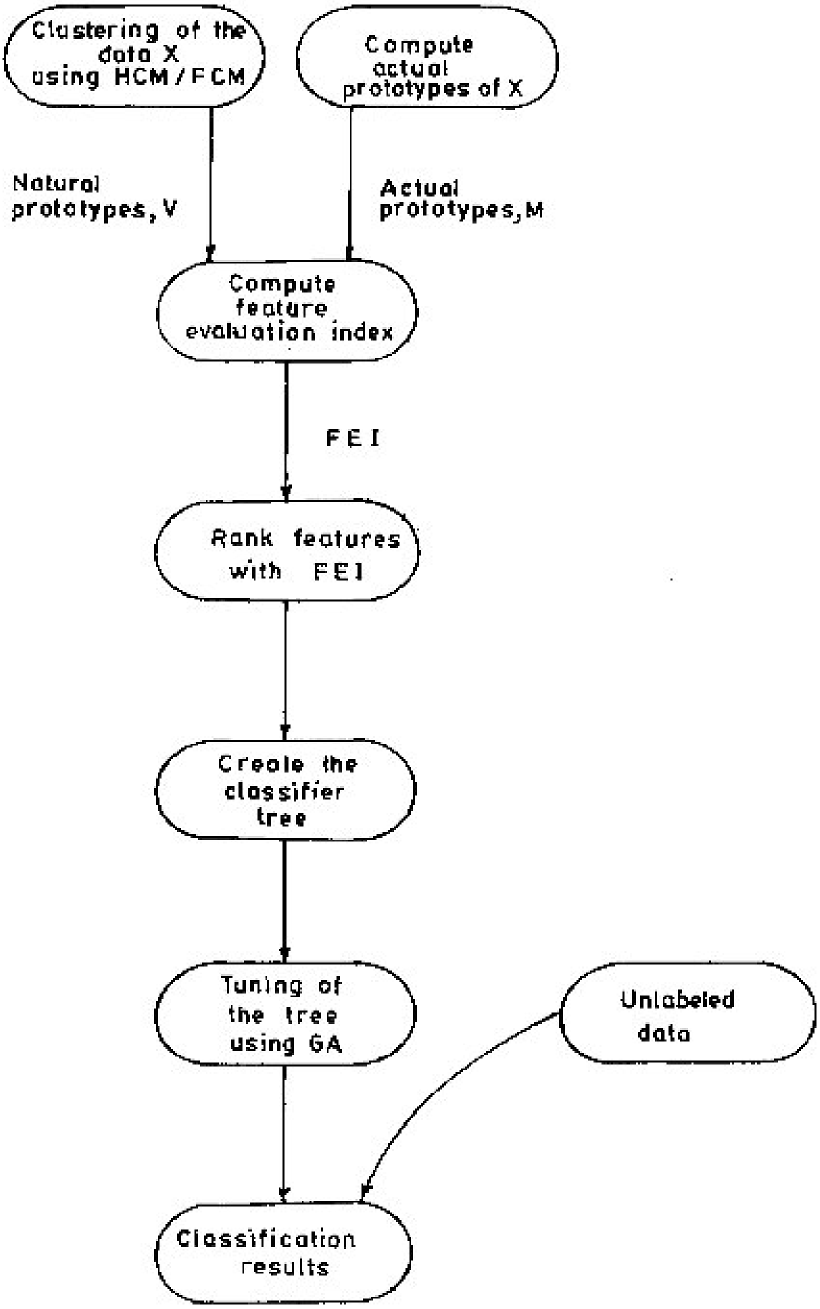
Clustering of the data X using HCM / FCM
Compute actual prototypes of X
Natural prototypes, V
Actual prototypes, M
Compute feature evaluation index
FEI
Rank features with FEI
Create the classifier tree
Tuning of the tree using GA
Unlabeled data
Classification results

Fig. 3. A schematic description of the overall system.

rithm are compared with those of the Nearest-Neighbor (NN) [1] classifier. Before describing the results, we present a brief description of the NN classifier for the sake of completeness.

NEAREST-NEIGHBOR ALGORITHM

(a) Store all the training instances.
(b) For each of the testing instances,
   i. Measure its distance from all the training instances.
   ii. If the nearest instance is "x," then the testing instance is assigned to the class of "x."

For our purpose, we divided the data set randomly into two equal halves and then treated one set as the test set and the other as the design (training) set and vice versa. To elaborate further, we have $X = X_1 \cup X_2 \cup \cdots \cup X_c$. We randomly partitioned X into X^D and X^T such that,

$$X = X^D \cup X^T, \qquad X^D \cap X^T = \phi,$$

$$X^D = X_1^D \cup X_2^D \cup \cdots \cup X_c^D,$$

$$X^T = X_1^T \cup X_2^T \cup \cdots \cup X_c^T,$$

$$X_i = X_i^D \cup X_i^T \quad \text{and} \quad X_i^D \cap X_i^T = \phi,$$

$$\text{If } |X_i| = n_i, \text{ then } |X_i^D| = \left\lfloor \frac{n_i}{2} \right\rfloor \text{ and } |X_i^T| = \left\lceil \frac{n_i}{2} \right\rceil.$$

The results obtained by RID3 and the nearest-neighbor [1] algorithms are as shown in Table 4. The percentage of correct classification reported in the table is the average score obtained by switching the training and test sets. We have tested the algorithm with several random partitions, and Table 4 reports a typical result. Table 4 clearly reveals that RID3 outperforms NN classifier for all the three data sets. The importance of proper threshold at each node and the success of GA in finding the same are also reflected by Table 4. In each of the cases, we find a significant improvement in performance of RID3 after tuning. For Mango_Leaf and Crude_Oil, tuning improves the performance by 17% and 9%, respectively. If the classes are well separated, then we can use a fixed high threshold at each node, and the performance of the system is not expected to change with small changes in the value of the threshold. Each of the data sets used in the present investigation has substantial overlap between some of the classes and hence, as explained in Section 2.4, proper choice of thresholds can improve the performance of the system. Our experiment shows that tuning indeed finds optimal or near-optimal thresholds for each node and thereby improves the performance of the classifier.

TABLE 4
Comparison of RID3 with Nearest Neighbor

	Percentage of correct classification				
		Ranked by (FEI)		Ranked by $J(Y)$	
Data set	N.N.	RID3 without tuning	RID3 with tuning	RID3 without tuning	RID3 with tuning
IRIS data Population = 150 Dimension = 4 Classes = 3	94	94	98	94	98
Mango_Leaf data Population = 166 Dimension = 18 Classes = 3	56	50	67	66.67	68
Crude_Oil data Population = 56 Dimension = 5 Classes = 3	75	75	84	75	88

We want to emphasize that success of the proposed scheme is dependent on the order in which features are used, i.e., the ranking of the features. But this dependence is not severe. To illustrate this fact, we have reported in Table 4 the results obtained by RID3 with the ranks produced by $J(Y)$. Although the rankings obtained by $J(Y)$ and FEI are quite different for Crude-Oil and Mango-Leaf, the final classifier performances are almost the same.

5. CONCLUSION AND DISCUSSION

The most vital drawback of the ID3-type algorithm is that it cannot handle numerical data, it deals with categorical data only. We can convert a real data set to a categorical one, by quantizing each feature and assigning different labels for different slots. As mentioned earlier, a problem associated with this is that we do not know the optimal number of categories for a particular feature and we have to assume it beforehand. The choice of the number of categories will have significant impact on the performance of the classifier. Moreover, for such schemes the finer details of the data will be lost.

In this paper, we proposed an algorithm RID3 which works on real data. We implicitly assumed that each feature has some amount of clustering corresponding to each class. Based on this, we have ranked the features using their clustering tendency.

In the next phase, we construct a tree in which every node has a prototype of features corresponding to a class, along with a threshold based on which we can determine whether a data point coming into that node belongs to this class or not. The preliminary tree is constructed with threshold = 0.5 at all nodes, which is further tuned using GAs.

Due to this tree structure, we can enter into this tree at any level and we can skip any level as well. Using this, we can develop a scheme for handling data with missing features which will be the theme of our future work.

However, as with any other algorithm, there is some scope for improvements:

(a) The algorithm is nonincremental, so an incremental version of the algorithm will be helpful for dealing with data sets which are not completely known beforehand.

(b) For "c" number of classes and "p" number of features, the total number of nodes is c^{p+1}. So for a large number of features, the number of nodes will be very large. A possible remedy to this problem may be to build the tree up to a predetermined level, and store the prototypes of the other features in an array. While testing, if for any data point we require more levels than that, we build the required subtree dynamically. Another possibility may be to prune the tree after it is constructed.

REFERENCES

1. Y. H. Pao, *Adaptive Pattern-Recognition and Neural Network*, Addison-Wesley, Reading, MA, 1989.
2. R. A. Fisher, The use of multiple measurements in taxonomic problems, *Ann. Eugen.* 7:179–188 (1936).
3. A. Bhattacharjee, Some aspects of mango (*Mangifora Indica L*) leaf growth features in varietal recognition, M.S. Thesis, Calcutta University, Calcutta, India, 1986.
4. P. M. Gerrid and R. J. Lantz, Chemical analysis of 75 crude oil samples from pliocene sand units, elk oil fields, California, U.S. Geological Survey Open-File Report, 1969.
5. J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum, New York, 1981.
6. D. Bhandari, N. R. Pal, and S. K. Pal, Directed mutation in genetic algorithms, *Inform. Sci.* 79:251–270 (1994).
7. J. T. Tou and R. Gonzalez, *Pattern Recognition Principles*, Addison-Wesley, Reading, MA, 1974.

Received 10 November 1995; revised 20 March 1996