Genetic Algorithms for Generation of Class Boundaries

Sankar K. Pal, *Fellow, IEEE,* and Sanghamitra Bandyopadhyay, and C. A. Murthy

***Abstract*— A method is described for finding decision boundaries, approximated by piecewise linear segments, for classifying patterns in $\Re^N, N \geq 2$, using an elitist model of genetic algorithms. It involves generation and placement of a set of hyperplanes (represented by strings) in the feature space that yields minimum misclassification. A scheme for the automatic deletion of redundant hyperplanes is also developed in case the algorithm starts with an initial conservative estimate of the number of hyperplanes required for modeling the decision boundary. The effectiveness of the classification methodology, along with the generalization ability of the decision boundary, is demonstrated for different parameter values on both artificial data and real life data sets having nonlinear/overlapping class boundaries. Results are compared extensively with those of the Bayes classifier, k-NN rule and multilayer perceptron.**

***Index Terms*— Evolutionary computation, hyperplane fitting, pattern recognition, variable mutation probability.**

I. Introduction

GENETIC algorithms (GA's) [1]–[5] are randomized search and optimization techniques guided by the principles of evolution and natural genetics, and have a large amount of implicit parallelism. GA's perform multimodal search in complex landscapes and provide near optimal solutions for objective or fitness function of an optimization problem. They have applications in fields as diverse as VLSI design, pattern recognition, image processing, neural networks, machine learning, jobshop scheduling, etc. [6]–[9].

In GA's, the parameters of the search space are encoded in the form of strings (chromosomes). A collection of such strings is called a population. Initially a random population is created, which represents different points in the search space. Based on the principle of survival of the fittest, a few among them are selected and each is assigned a number of copies that go into the mating pool. Biologically inspired operators like *mating, crossover,* and *mutation* are applied on these strings to yield a new generation of strings. The process of selection, crossover and mutation continues for a fixed number of generations or until a termination condition is satisfied. An excellent survey of GA's along with the programming structure used can be found in [5].

In this paper, an attempt is made to study the application of GA's for pattern classification in N-dimensional data space. Classification is a problem of generating decision boundaries that can successfully distinguish the various classes in the feature space. In real life problems, the boundaries between the different classes are usually nonlinear. In the present investigation, the characteristics of GA's are exploited in searching for a number of hyperplanes which can approximate the nonlinear boundaries while providing minimum misclassification of training sample points.

The feature space is generally unbounded and continuous in nature. However, if bounding information can be derived from the training patterns and the space is discretized to sufficiently small intervals in each dimension, then the requirements of GA's for formulating the classification problem are fulfilled.

A distinguishing feature of this approach is that the boundaries (approximated by piecewise linear segments) need to be generated explicitly for making decisions. This is unlike the conventional methods or the multilayered perceptron (MLP) based approaches, where the generation of boundaries is a consequence of the respective decision making processes.

Since the optimum number of hyperplanes, H, required for proper classification of a given data set is not known *a priori*, a conservative estimation (or overestimation) of H is normally done to initiate the algorithm. Consequently, some of the hyperplanes may be found to be redundant in the final output of the GA-based algorithm as far as their contribution toward the generation of boundary is concerned. A methodology for the automatic deletion of redundant hyperplanes is also developed. The process of elimination is performed as a postprocessing step. The effectiveness of the proposed recognition method in classifying both overlapping and nonoverlapping, nonconvex regions is extensively demonstrated on two sets of artificial data, Iris data and real life speech and satellite imagery data. The results are compared with those of Bayes classifier and k-NN classifier [10], and MLP [11]. The generalization ability of the decision boundary and the issue of removing redundancy is explained pictorially.

Section II describes the representation technique of the hyperplanes in the form of strings. In Section III, a description of the classification methodology is presented. Section IV deals with the selection of control parameters of the methodology. In Section V, the implementational aspects of the proposed methodology and the experimental results are presented. Finally, the discussion and conclusions are presented in Section VI.

II. Hyperplane Encoding

We consider a fixed number of hyperplanes (say, H) to denote a decision boundary in an N-dimensional feature

Manuscript received April 26, 1995; revised March 17, 1996 and July 5, 1997. This work was carried out when S. Bandyopadhyay held a fellowship awarded by the Department of Atomic Energy, Government of India.

The authors are with the Machine Intelligence Unit, Indian Statistical Institute, Calcutta 700035, India.

Publisher Item Identifier S 1083-4419(98)07293-8.

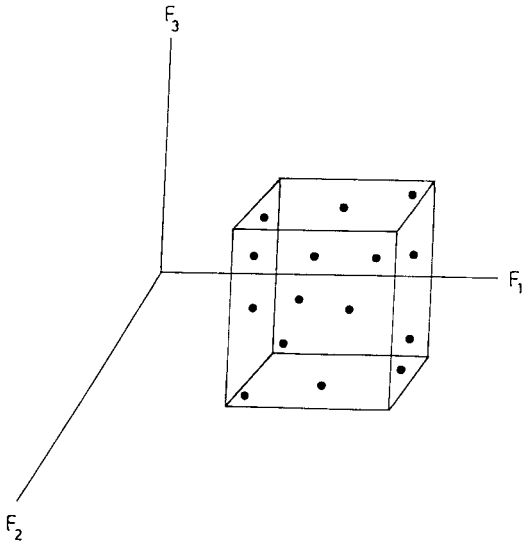

Fig. 1. Training samples and the enclosing hyper rectangle.

space $F_1, F_2, \cdots, F_N$. The value of H varies from problem to problem, depending on the number (say M) as well as the nature of the classes. These H hyperplanes need to be encoded as a single string. Note that each hyperplane provides two halfspaces—a positive halfspace and a negative halfspace, thereby yielding two regions. For H hyperplanes, the maximum number of such regions is 2^H. Thus $M \leq 2^H$ or $H \geq \log_2 M$.

A. Search Space for Hyperplanes

Let us assume that there are tr training patterns available. Then in the first step the maximum and minimum values of each of the features, $F_1, F_2, \cdots, F_N$, are computed. Let these be

$$(Max_1, Min_1), (Max_2, Min_2), \cdots, (Max_N, Min_N)$$

for the features $F_1, F_2, \cdots, F_N$, respectively. Then a hyper rectangle enclosing the sample points in the N-dimensional space is given by the vertices $(Min_1, Min_2, \cdots, Min_N)$, $(Max_1, Min_2, \cdots, Min_N)$, $(Min_1, Max_2, Min_3, \cdots, Min_N), \cdots, (Max_1, Max_2, Min_3, \cdots, Min_N), \cdots, (Max_1, Max_2, \cdots, Max_N)$.

Fig. 1 shows such an example for three-dimensional (3-D) space. The hyper rectangle represents the search space for the possible hyperplanes which may be considered as candidates for the formation of the decision boundary.

B. Hyperplane Generation

In this section, we describe a way of generating a finite number of parallel hyperplanes in a finite number of directions, so as to enable the separation of even the two closest points of the data set. For this purpose, a distance $dist$ is computed as follows.

Let $\mathcal{A}$ represent the training data set. Then we define

$$dist = \underset{\substack{x \in \mathcal{A} \\ y \in \mathcal{A} \\ x \neq y}}{Min}\ distance(x, y)$$

where $distance(x, y)$ denotes the Euclidean distance between points x and y. Consequently, the separation between two consecutive parallel hyperplanes, in any direction, is taken to be $dist/2$ to guarantee the above mentioned condition. Thus the maximum number of parallel hyperplanes, max_planes, which may have to be considered for searching in any particular direction within the search space is computed as

$$max\_planes = \left\lceil \frac{diag}{dist} * 2 \right\rceil$$

where $\lceil x \rceil$ gives the smallest integer greater than or equal to x and $diag$ is the length of the diagonal of the hyper rectangle enclosing the training points, i.e., see the bottom of page.

C. Hyperplane Specification

From elementary geometry [10], [12], we know that the equation of a hyperplane in N-dimensional space can be represented as

$$l_1 x_1 + l_2 x_2 + \cdots + l_N x_N = d \tag{1}$$

where $l_1 = \cos\alpha_1, l_2 = \cos\alpha_2, \cdots, l_N = \cos\alpha_N$. Here, $\alpha_1, \alpha_2, \cdots, \alpha_N$ are the angles that the unit normal, to the hyperplane, make with the N axes $F_1, F_2, \cdots, F_N$, respectively; d is the perpendicular distance of the hyperplane from the origin. It also follows that

$$l_1^2 + l_2^2 + \cdots + l_N^2 = 1. \tag{2}$$

Hence, N variables are needed to specify a hyperplane viz. $N-1$ angle variables (say, $\alpha_1, \alpha_2, \cdots, \alpha_{N-1}$) and d. Angle α_N is implicitly specified through (2). ♣

- Angle (Direction) Specification: The entire feature space will be spanned if the angles are allowed to vary in the range from 0 to π rad. The angle space is discretized to sufficiently small intervals as

$$0, \delta\pi, 2\delta\pi, \cdots, n * \delta\pi.$$

  The number of discrete angles considered is then equal to $n+1$, and $\delta = (1/n)$. An angle $\alpha_k, k = 1, 2, \cdots, N-1$ can thus be specified by a number $angle_k$ in the range $[0, n]$ such that $\alpha_k = angle_k * \delta\pi$. The remaining angle α_N is implicitly contained in the constraint equation (2) as $l_N^2 = 1 - \Sigma_{k=1}^{N-1} l_k^2$ and $\alpha_N = \cos^{-1} l_N$.
- Perpendicular Distance Specification: Once all the angles $\alpha_1, \alpha_2, \cdots, \alpha_N$ are fixed, the orientation of the hyperplane becomes fixed. For a given orientation, the perpendicular distances from the origin to the hyperplanes

$$diag = \sqrt{(Max_1 - Min_1)^2 + (Max_2 - Min_2)^2 + \cdots + (Max_N - Min_N)^2}$$

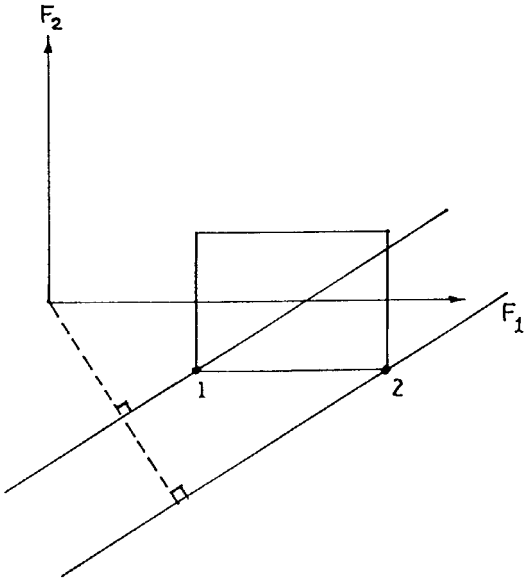

Fig. 2. Fixing the base hyperplane.

passing through the corner points in the base of the enclosing hyper rectangle (i.e., corner points with Min_N as the Nth coordinate) are computed from (1). (The perpendicular distance, d, assumes a negative value if it lies in the negative halfspace of the F_N axis.) Among these, the one with the minimum value, d_{min}, is selected as the base hyperplane. This is demonstrated in Fig. 2, for two-dimensional (2-D) feature space where the hyperplane (line) through point 2 is the base hyperplane. The search space for hyperplanes with this orientation is restricted at one end by the base hyperplane. In other words, all hyperplanes with $d < d_{\min}$, are automatically discarded from the search space. At the other end is a parallel hyperplane at a perpendicular distance of $(d_{\min} + max\_planes \times (dist/2))$ from the origin.

Therefore any hyperplane at a distance *offset* from the base hyperplane can be specified by a number p in the range $[0, max\_planes]$ such that $offset = p * (dist/2)$. The perpendicular distance d of that hyperplane from the origin is therefore

$$d = d_{\min} + offset.$$

From the above discussion it is clear that one needs the variables $angle_1, angle_2, \cdots, angle_{N-1}$, and p for specifying a hyperplane $hpln$. If each angle variable is represented by b_ang number of bits and perpendicular distance by b_perp bits, then the number of bits required for the binary encoding of a hyperplane is

$$(N-1) \times b\_ang + b\_perp.$$

III. Description of the Methodology

As was stated earlier, each string is composed of a fixed number, H, of hyperplanes. Each hyperplane is encoded in terms of $(N-1)$ angle variables and a perpendicular distance variable. The basic steps of the methodology are as shown in Fig. 3.

```
generate initial population G(0)
compute fitness G(0)
t = 0
while (termination condition not attained)
begin
    compute fitness (G(t))
    t = t + 1
    G(t) = select(G(t − 1))
    crossover(G(t))
    mutate(G(t))
end.
```

Fig. 3. Basic steps in GA's.

The operations of generating initial population, fitness computation, selection, crossover, and mutation are described in detail in the following subsections. The terminology used is stated here for ease of understanding.

tr	Number of training patterns.
Pop	Number of strings in a generation.
H	Number of hyperplanes encoded in a string.
N	Feature space dimension.
str_i	ith string in a generation.
$sign_i^k$	Sign string generated for string i of a population and training pattern k.

A. String Representation and Population Initialization

Each string is composed of a fixed number H of hyperplanes. Each hyperplane is encoded in terms of $N-1$ angle variables and a perpendicular distance variable (both assume integer values). If each angle variable is represented by b_ang number of bits and perpendicular distance by b_perp bits, then the length of each string, str_len equals

$$str\_len = ((N-1) \times b\_ang + b\_perp) \times H.$$

The GA generally works with a fixed population size of Pop. Initially, each of the Pop binary strings of length str_len, is generated by randomly selecting $N-1$ angle variables, $angle_k, k = 1, 2, \cdots, (N-1)$, and p from the intervals $[0, n]$ and $[0, max\_planes]$, respectively, for each of the H hyperplanes encoded in the string.

Example 1: Let $n = 4, max\_planes = 7, b\_ang = 3, b\_perp = 3, N = 3$ and $H = 1$. Then a string

$$\overbrace{000}^{angle_1}\ \overbrace{010}^{angle_2}\ \overbrace{100}^{p}$$

indicates a plane whose normal makes angles 0 and $\pi/2$ with the F_1 and F_2 axis, respectively. Its perpendicular distance d from the origin is $d_{\min} + 4 * (dist/2)$, for some $d_{\min}$ and $dist$. The equation of the plane is

$$x_1 = d_{\min} + 4 * \frac{dist}{2}.$$

B. Region Identification and Fitness Computation

The computation of the fitness is done for each string in the population. The fitness of a string is characterized by the number of points it misclassifies. If the number of samples misclassified by a string is denoted by $miss$ then the fitness of the string is computed as $(tr - miss)$, where tr is the number of training sample points. A string with the lowest misclassification is therefore considered to be the fittest among the population of strings. Note that every string $str_i, i = 1, 2, \cdots, Pop$ represents H hyperplanes denoted by $hpln_j^i, j = 1, 2, \cdots, H$.

For each $hpln_j^i$, the parameters $l_1^{ij}, l_2^{ij}, \cdots, l_N^{ij}$ and d^{ij} are retrieved. For each training pattern point $(x_1^k, x_2^k, \cdots, x_N^k), k = 1, 2, \cdots, tr$, the sign with respect to the hyperplane $hpln_j^i$, i.e., the sign of the expression

$$l_1^{ij} x_1^k + l_2^{ij} x_2^k + \cdots + l_N^{ij} x_N^k - d^{ij} \tag{3}$$

is found. The sign is digitized as 1 (0) if the point lies on the positive (negative) side of the hyperplane $hpln_j^i$. The process is repeated for each of the hyperplanes, at the end of which we have a string $sign_i^k$, subsequently to be referred to as the sign string. This string, of length H corresponds to the classification yielded by the string str_i of the population, for the kth training pattern. The class information of the kth training point is stored along with the sign string $sign_i^k$ for str_i in a linked list. This procedure is repeated for all the tr pattern points.

It is to be noted that although $sign_i^k$ can take on at most 2^H possible values (since H hyperplanes will yield 2^H possible classifications), all of them may not occur in practice. These sign strings, in fact, represent different regions of the search space. With each such sign string, a linked list is maintained. Each element of the list is an ordered pair indicating a class and its cardinality. The cardinality of a class denotes the number of training samples of that class which have been identified to fall into the region represented by the sign string.

The maximum class cardinality in the list for each sign string is found next. Then the region corresponding to that sign string is considered to provide the demarcation for the class possessing the maximum cardinality. All the points belonging to other classes which have been included in the same list, i.e., which lie in the same region, are considered to be misclassified. The number of misclassifications corresponding to all possible sign strings are summed up to give the resulting misclassification for the entire classifier string. It may so happen that the maximum cardinalities for two (or more) different sign strings may correspond to the same class. In that case, all these strings (correspondingly, union of all the different regions) are considered to provide the region for the class. A tie is resolved arbitrarily. The example stated below will clarify this method.

Example 2: Let there be 8 training patterns belonging to two classes, 1 and 2, in a 2-D feature space $F_1 - F_2$. The decision boundaries in the 2-D space will be lines. Let us assume H to be 3, i.e., three lines will be used to classify the points. Let the training set and a set of three lines be as shown in Fig. 4. Each point $i_j, i = 1, 2, \cdots, 8$, and $j = 1, 2$ indicates that it is the ith training point and that it belongs to class j. Let the positive and the negative sides of the lines be as shown in Fig. 4. Then, point 1_1 yields a sign string 111 since it lies on the positive side of all the three lines Line_1, Line_2, and Line_3. The corresponding linked list formed for all the eight points is shown in Fig. 5. It is to be noted that one region (denoted by sign string 110) is accidentally empty, while two regions (100 and 101) do not exist. The number of misclassifications for the example is found to be $1 + 1 = 2$, one each for sign strings 001 and 111. Note that in this example both the strings 000 and 001 are providing the regions for class 2 (assuming that the tie for region 111 is resolved in favor of class 1). ♣

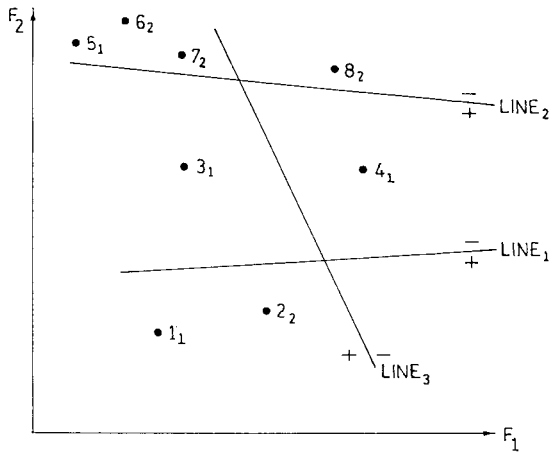

Fig. 4. Region identification for $H = 3$ and $tr = 8$.

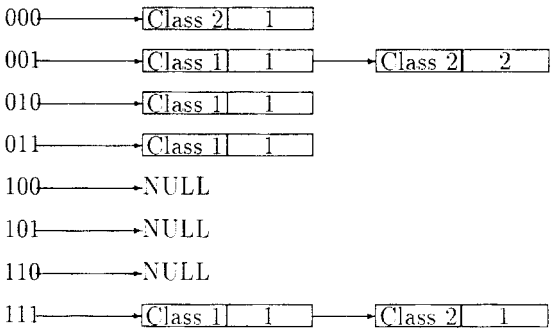

Fig. 5. Linked list for the example in Fig. 4.

In a similar fashion, the number of misclassified samples for all the strings in the population is computed. The best string of each generation or iteration is the one which has the fewest misclassifications. This string is stored after each iteration. If the best string of the previous generation is found to be better than the best string of the current generation, then the previous best string replaces the worst string of the current generation. This implements the *elitist strategy*, where the best string seen up to the current generation is propagated to the next generation.

C. Selection

The *roulette wheel* selection procedure has been adopted here to implement a *proportional selection* strategy. Each string is allocated a slot of the roulette wheel subtending an angle, proportional to its fitness, at the center of the wheel. A

random number in the range of 0 to 2π is generated. A copy of a string goes into the mating pool if the random number falls in the slot corresponding to the string. For a fixed population size Pop, this process is repeated Pop times, at the end of which as many strings go into the mating pool for further operations.

D. Crossover

A pair of strings is picked up at random and the single point crossover operator is applied according to a fixed crossover probability. For this operation, a random number cr_pt in the range of 0 to str_len is generated. This is called the crossover point. The portion of the strings lying to the right of the crossover point are exchanged to yield two new strings. There is a catch to this apparently simple method in this case. The strings resulting from the crossover operator may not be valid in the sense that they may not model the hyperplane equation (1). A hyperplane may be generated where the constraint equation (2) may not hold. In that case the original pair of strings is recalled, and another crossover point is generated. The process may be repeated a certain number of times until a crossover point which effectively exchanges the entire strings is generated.

Note that this is one way of tackling the problem of invalid child generation in constrained optimization problems. Other methods in this respect are available in [4].

Example 3: Let the two parents, for the parameters given in Example 1, be as follows:

$$\begin{array}{ccccc|ccccc} 0 & 0 & 0 & & 0 & 1 & 0 & & 1 & 0 & 0 \\ 0 & 1 & 0 & & 0 & 0 & 0 & & 0 & 0 & 1. \end{array}$$

Let the crossover site be as shown. Then the offspring are

$$\text{child } 1 = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1$$
$$\text{child } 2 = 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0.$$

Obviously child 1 is not valid since $angle_1 = angle_2 = 0$. Hence, $l_1 = l_2 = 1$ which violates the constraint equation. Thus a new crossover point is selected. Let it be as follows:

$$\begin{array}{ccccccc|ccccc} 0 & 0 & 0 & & 0 & & 1 & 0 & & 1 & 0 & 0 \\ 0 & 1 & 0 & & 0 & & 0 & 0 & & 0 & 0 & 1. \end{array}$$

Then the offspring are

$$\text{child } 1 = 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1$$
$$\text{child } 2 = 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0$$

which are allowable.

E. Mutation

Mutation is done on a bit by bit basis (for binary strings) [1], [5] according to some mutation probability value mut_prob. Thus, more than one bit may be complemented in a string if mut_prob so permits. As all the bits in a string are subjected to the mutation operator, a record of the bits being complemented is kept. Finally, the resulting string is checked to find out if an invalid hyperplane has resulted from the mutation operator, i.e., one not conforming to the constraint equation (2). In that case, only those bits which have been complemented are flipped back one after the other. After each reflipping, the resultant hyperplane is checked for validity. If at any step a valid hyperplane is formed, then the process is discontinued. The process of reflipping effectively means that the corresponding bits were not mutated at all. Note that this could as well have been the case, as mutation is a random operation.

Example 4: From Example 3, after crossover we had child $1 = 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1$. Let mutation in bits 5 and 8 (from left) result in the string 0 0 0 0 0 0 0 1 1, which is invalid as explained in Example 3. Reflipping bit 5, we get 0 0 0 0 1 0 0 1 1 which is allowable.

F. Termination

The process of fitness computation, selection, crossover and mutation continues for a fixed number of iterations max_itrns or until the termination condition (a string with misclassification number reduced to zero) is achieved. Let the string obtained on termination be 0 0 0 0 1 0 0 0 0. This indicates a plane passing through the origin and lying in the $F_2 - F_3$ space. The equation of the plane is

$$x_1 = 0.$$

G. Postprocessing (Deletion of Redundant Hyperplanes)

As mentioned in Section II, for an M class problem we have the number of hyperplanes $H \geq \log_2 M$. In order to tackle the intricacy of certain data sets, consideration of $H = \log_2 M$ may not be sufficient for proper classification. For this reason, we try to make an overestimation of H (i.e., H sufficiently greater than $\log_2 M$) for constituting the decision boundary. As a result, in the final output of the algorithm, some hyperplanes may become redundant in the sense that their removal does not change the recognition capability of the classifier. Elimination of these redundant hyperplanes is a natural extension of this work. A flowchart for this process in given in Fig. 6. It is described as follows.

Let A represent the set of correctly classified sample points and S represent a set of hyperplanes (initially equal to the set of hyperplanes obtained on termination of the algorithm). From S, a set $S1$ is formed such that all the points in A lie on only one side of each hyperplane in $S1$. Another set $S2$ $(S2 = \{h_1, h_2, \cdots, h_p\}, p \leq H)$ is also formed such that one side (subsequently referred to as *unique*) of each hyperplane in the set has sample points in A of only one class. Finally, a set S' is formed from the remaining hyperplanes (i.e., $S' = S - S1 - S2$).

Hyperplanes represented by $S1$ are obviously redundant, and are put into the set of redundant hyperplanes R. Now, two cases for $S2$ may arise.

$S2$ is empty: In this case, all the hyperplanes in S' are considered to be nonredundant and are put in a list of nonredundant hyperplanes NR. A further redundancy check, $check\_red$, is done on NR and then the process terminates. A description of $check\_red$ is given subsequently.

$S2$ is non-empty: In this case, for each element, $h_i, 1 \leq i \leq p$, of $S2$, all the points of A which lie on its *unique* side, i.e., which can be classified by h_i alone, are put into a set A'. A is now set to be equal to the difference of A and A', i.e., $A = A - A'$. At the same time, h_i is also removed from

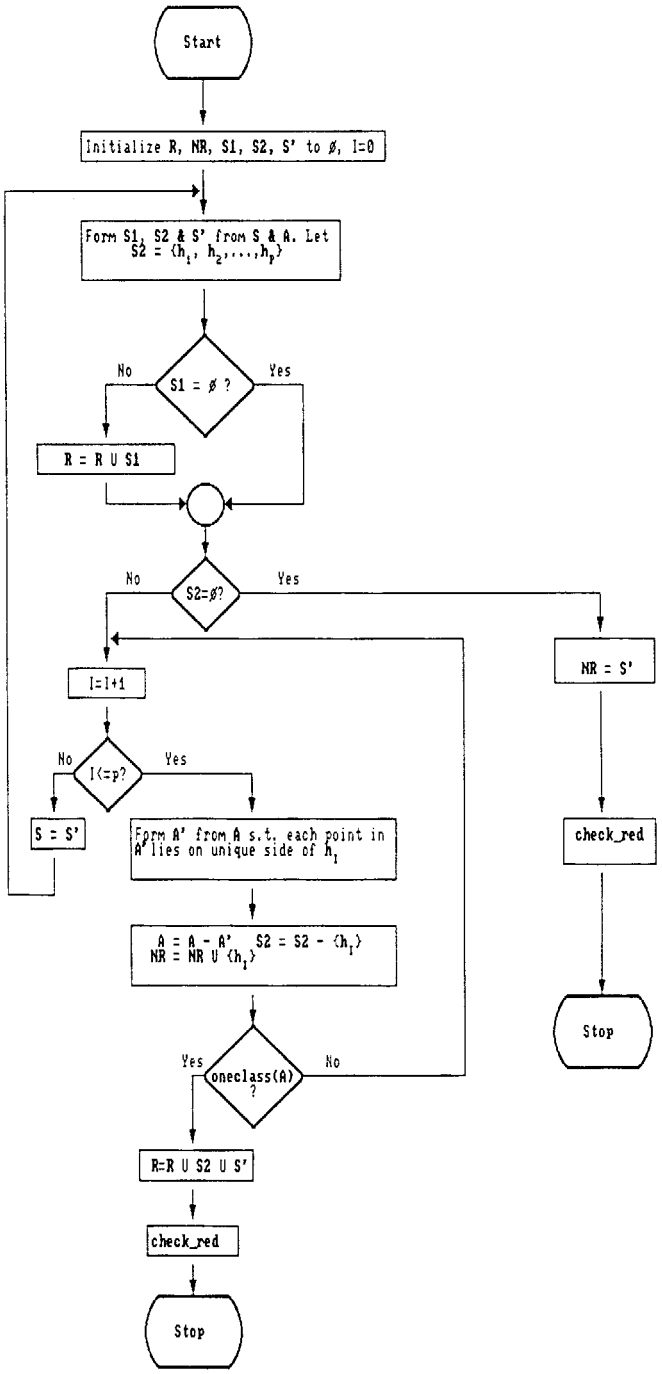

Fig. 6. Flowchart for the process of elimination of redundant hyperplanes.

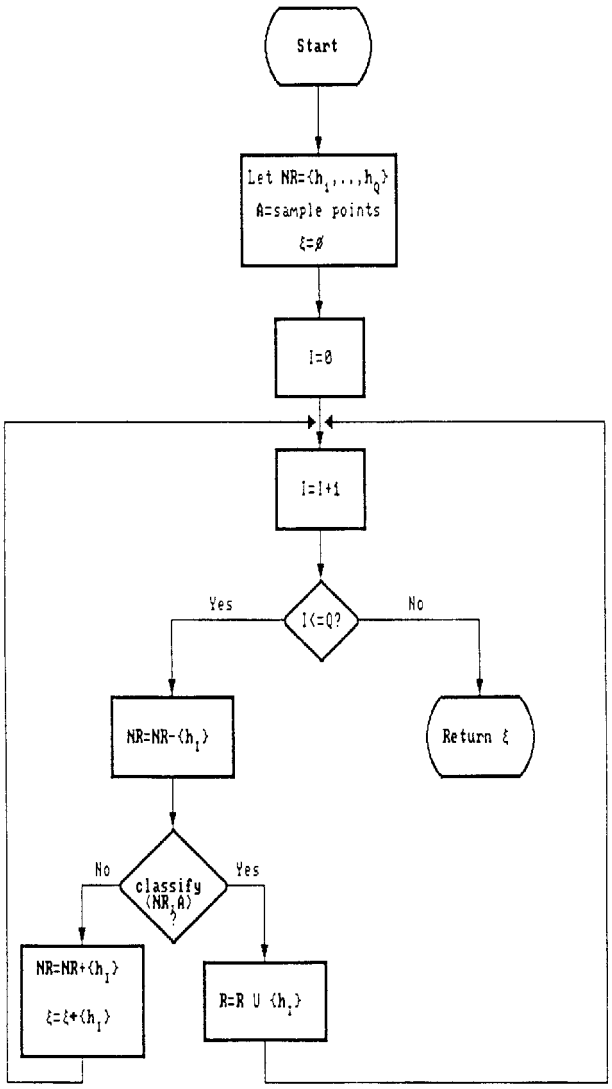

Fig. 7. Flowchart for the process $check\_red$.

$S2$ and put into NR. The remaining points of A are checked to see if all of them belong to the same class. (The function $oneclass(A)$ in Fig. 6 performs this check. It returns $True$ if all the points still in A belong to only one class. Otherwise, it returns $False$.) Thus two further cases may arise.

- If $oneclass(A)$ returns $True$, then the remaining hyperplanes of $S2$ and those in S' are declared to be redundant and these are put into R. The process then calls $check\_red$ with NR.
- If $oneclass(A)$ returns $False$ then the next element of $S2$ is considered. If $oneclass(A)$ returns $False$ for all the elements in $S2$ then the entire process is repeated while considering the sample points that are still unclassified (i.e., those which are still present in A) as A and the hyperplanes in S' as the set S.

The processing block (Fig. 6) of $check\_red$ results in a set $\mathcal{E}$, the set of hyperplanes essential for the complete classification of all the elements in A (the correctly classified points in the training data set), from the set NR formed in the process described before. The detailed flowchart of $check\_red$ is presented in Fig. 7. It eliminates those hyperplanes in NR which are not essential for classifying the samples points. Each element, say h_i, of NR is considered at a time. It is first removed from NR, and then it is checked if the remaining hyperplanes in NR can successfully classify all the points in A. (This is performed by the function $classify(NR, A)$ which returns $True$ if the set of hyperplanes in NR correctly classifies all the points in A. Otherwise, it returns $False$. The operation of $classify(NR, A)$ is similar to the method of region identification described in Section III-B. To return $True$, the linked list formed for NR must have at most a

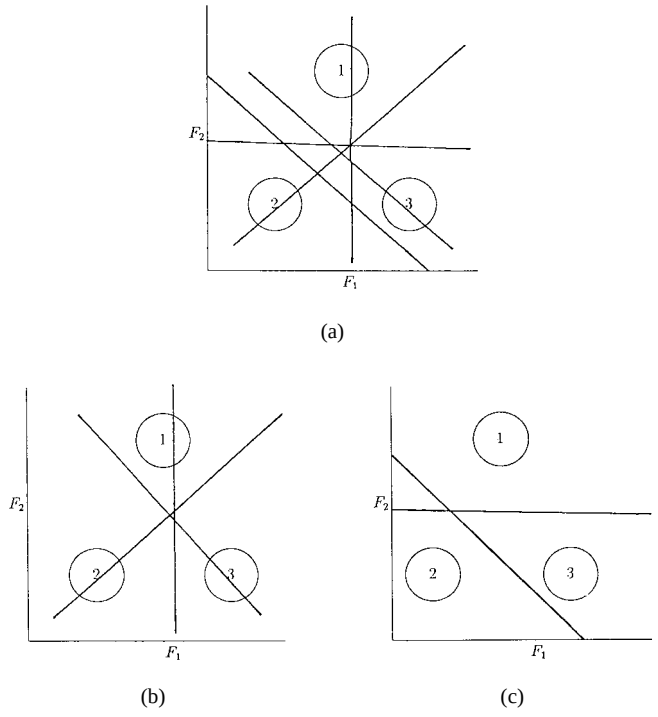

Fig. 8. (a) Three classes and a set S of five lines for partitioning them. (b) Subset of S comprising three lines which can successfully partition the three classes in Fig. 8(a). (c) Subset of S comprising two lines which can successfully partition the three classes in Fig. 8(a).

single entry for each $sign$ string.) Thus the following two cases may arise.

$classify(NR, A)$ *returns* $True$: In this case, h_i is considered to be redundant, and it is put into R. The process is then repeated for other elements in NR.

$classify(NR, A)$ *returns* $False$: In this case, h_i is considered to be essential, and it is put back into NR as well as in $\mathcal{E}$. The process is then repeated for other elements in NR. ♣

Apparently one may think that the consideration of $check\_red$ alone will be sufficient for the task of deletion of redundant hyperplanes. But this will not be capable of dealing with the situations where two different subsets of S can individually classify the patterns. For example, consider Fig. 8(a) in 2-D feature space, where S consists of the five lines which are considered to provide the decision boundary for the three classes shown inside the respective circles. The two different subsets of three and two lines are shown in Fig. 8(b) and (c), respectively. Individually, each of the two subsets can successfully partition the classes. If only $check\_red$ is used in this situation, then depending on the order of scanning of the output lines, the two lines shown in Fig. 8(c) may be deleted. $check\_red$ would then declare the three lines shown in Fig. 8(b) to be the essential ones. Obviously, this is not the minimal set; actually the two deleted lines constitute the minimal set. The postprocessing task preceding $check\_red$ takes care of such situations. In spite of this, the resulting set of essential hyperplanes may not be minimal since the process of deletion of redundant hyperplanes is dependent on the sequence of the hyperplanes in S, and it does not take their interrelationships into account.

IV. Selection of Control Parameters

As already stated, there are various methods of fixing population size, crossover probability and mutation probability values. Each of the parameters can be kept fixed or variable. In this work the population size as well as the crossover probability is kept fixed.

The other parameter to be chosen is the mutation probability value, the importance of which has been stressed in [13] and [14]. In [15], it is shown that the mutation probability must be chosen in the range of $(0, (a-1/a)]$ where a is the cardinality of the set of alphabets for the strings ($cardinality = 2$ for binary strings). In [14], it is proved that the the mutation probability value must be within the range of (c/str_len) to 0.5, where c is the number of bit positions of a string in a generation that must be changed for arriving at the optimal solution for binary strings. Such *a priori* knowledge of c is almost impossible to acquire. Thus although a theoretical guideline for selecting the mutation probability value has been suggested, it is of little practical significance.

In this paper we have chosen the mutation probability value to vary approximately in the range of $1/str\_len$ to 0.5. Initially it has a high value, which slowly decreases with generations, and then increases again after attaining the minimum specified value. Given that the number of bits in a string is str_len, the different possible strings are 2^{str_len}. Our aim is to search this space and arrive at the optimal string as fast as possible. The initial high value of the mutation probability ensures sufficient diversity in the population which is desired, as at this stage, the algorithm knows very little about the nature of the search space. As generations pass, the algorithm slowly moves toward the optimal string. It is therefore necessary that the space be searched in detail without abrupt changes in population. Consequently, we decrease the mutation probability value gradually until it is sufficiently small. It may so happen that in spite of this, the optimal string obtained so far has a large Hamming distance from the actual optimal string. This may very well happen for deceptive problems [1], [16]. Thus, if we continue with the small mutation probability value, it may be too long before the optimal string is found. So to avoid being stuck at a local optima, the value is again gradually increased. Even if the optimal string had been found earlier, we loose nothing since the best string is always preserved in subsequent generation of strings. Ideally the process of decreasing and then increasing the mutation probability value should continue, but here we have restricted the cycle to just one due to practical limitations.

V. Implementation and Results

A. Data Sets

Both artificial and real life data sets of different dimensions (2, 3, and 4) have been used to test and compare the results of the algorithm proposed in this paper. A description of the data sets is given here.

Artificial Data: Two artificial data sets $ADS1$ and $ADS2$ shown in Figs. 9 and 10, respectively, with two classes 1 and 2, were generated. The first one consists of 557 data points

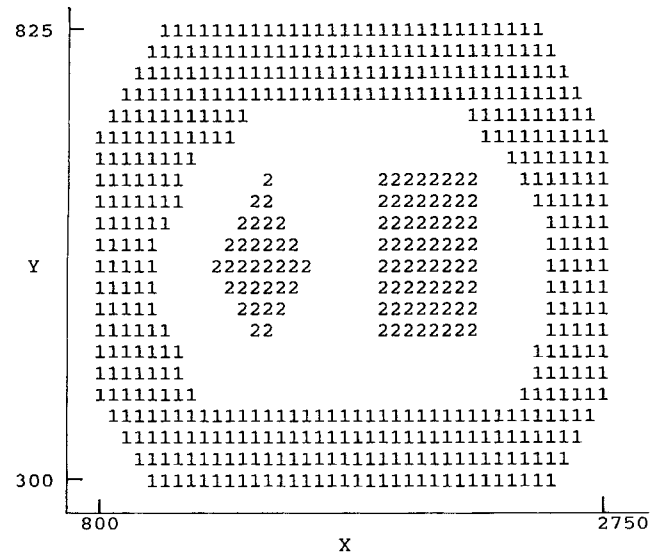

Fig. 9. ADS1 data.

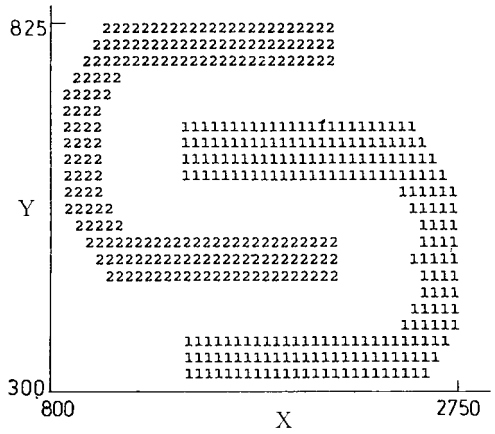

Fig. 10. ADS2 data.

while the second consists of 417 data points. The boundaries for both the data sets is seen to be highly nonlinear, although the classes are separable.

Vowel Data: This data consists of 871 Indian Telugu vowel sounds [17], [20]. These were uttered in a consonant-vowel-consonant context by three male speakers in the age group of 30–35 years. The data set has three features F_1, F_2, and F_3, corresponding to the first, second, and third vowel formant frequencies, and six classes $\{\delta, a, i, u, e, o\}$. The distribution of the six classes in the $F_1 - F_2$ plane is available in [17] and [20]. (It is known [17] that these two features are more important in characterizing the classes than F_3.) Note that the boundaries of the classes are very ill-defined and overlapping.

Landsat Data: This data set, demonstrated in Fig. 11, shows the satellite imagery data of rocks, vegetation and soil. It has 795 samples with five classes as described [18]

Class	description
1	Manda Granite
2	Romapahari Granite
3	Vegetation
4	Black Phillite
5	Alluvium.

Note that the original data has four components per sample for the four bands (red, green, blue, and infrared). The experiments conducted here have been done using the two principal components as the two features (Fig. 11).

Iris Data: This data represents different categories of irises. The four feature values per sample represent the sepal length, sepal width, petal length, and the petal width in centimeters [19]. It has three classes 1, 2, and 3 with 50 samples per class.

B. Genetic Parameters

For our experiment, a fixed population size of ten is chosen. The crossover probability is fixed at 0.8. A variable value of mut_prob is selected from the range [0.015, 0.333]. Initially it assumes a high value, gradually decreasing at first, and then increasing again in the later stages of the algorithm. 100 iterations are performed with each mutation probability value. The process is executed for a maximum of 1500 iterations in case it does not attain zero misclassification. H is fixed at six, although a study of the variation of the recognition score with the value of H is also demonstrated. The experimental results are described below taking different percentages, $perc$, of the data sets as the training set.

Remarks: In the binary implementation, each element of the chromosome vector (1 or 0) is represented by one character (usually 8 bits) in memory. Consequently for str_len elements, the space required is str_len characters. This programming practice, detailed in [5], allows simplified implementation of genetic operators like crossover and mutation at the cost of increased complexity of GA in terms of space and time requirements.

C. Comparison with Existing Methods

The performance of this methodology is compared with the performance of MLP, Bayes classifier and k-NN classifier. For MLP, learning rate η is initially fixed at 2.0. This is decreased by a factor of 2, up to a prespecified minimum value, if the mean squared error starts oscillating. In case the error decreases very slowly, then the learning rate is doubled, the reason being that most likely the algorithm has confronted a plateau in the error surface.

k-NN algorithm is executed taking k equal to $\sqrt{tr}$, where tr is the number of training samples. For the Bayes classifier, unequal dispersion matrices and unequal *a priori* probabilities ($= (tr_i/tr)$ for tr_i patterns from class i), are considered. In each case, we assume a multivariate normal distribution of the samples. Comparison with these three methods is performed for all the data sets. Although for some data sets, application of Bayes classifier with the assumption of normal distribution may not be meaningful, the results are included for keeping parity with others.

D. Experimental Results

The proposed GA based algorithm is tested on the data sets described in Section V-A. Tables I–VI present the results corresponding to artificially generated data sets (ADS1 and ADS2), Vowel data, Landsat data and Iris data, for different percentages of the training samples (e.g., $perc = 5, 10, 50$) and

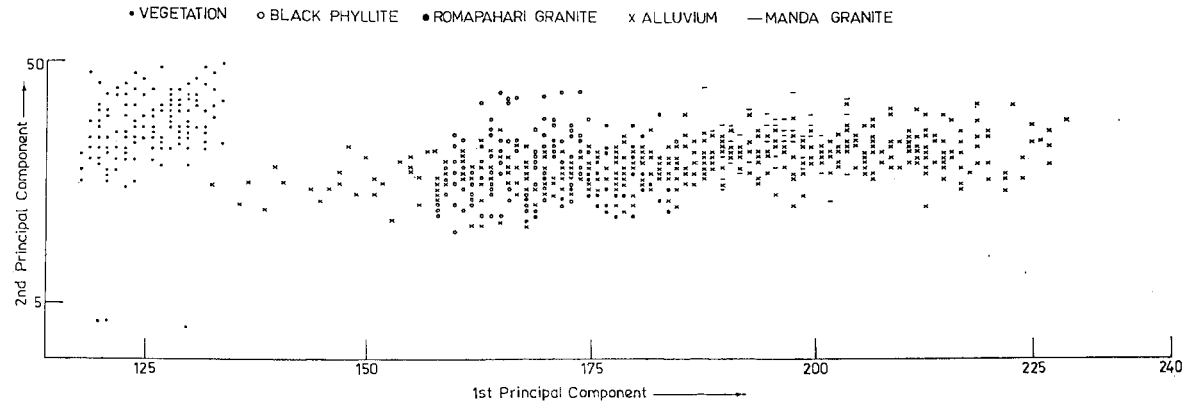

Fig. 11. Landsat data.

TABLE I
RECOGNITION SCORES (%) FOR ADS1 FOR $H = 6$

Class	$perc$=50	$perc = 10$	$perc = 5$
1	98.70	96.44	85.24
2	91.96	82.10	94.63
Overall	97.37	93.93	86.89

TABLE II
RECOGNITION SCORES (%) FOR ADS2 FOR $H = 6$

Class	$perc$=50	$perc = 10$	$perc = 5$
1	95.79	79.23	75.28
2	86.38	82.57	78.97
Overall	92.86	80.78	76.62

TABLE III
RECOGNITION SCORES (%) FOR VOWEL DATA ($F_1 - F_2$) FOR $H = 6$

Class	$perc$=50	$perc = 10$	$perc = 5$
δ	31.32	44.20	19.32
a	74.85	82.51	65.36
i	87.46	82.73	74.12
u	78.56	90.76	67.49
ϵ	85.38	72.72	95.04
o	90.76	75.31	95.66
Overall	78.25	76.62	73.18

TABLE IV
RECOGNITION SCORES (%) FOR LANDSAT DATA FOR $H = 6$

Class	$perc$=50	$perc = 10$	$perc = 5$
1	92.86	85.71	93.98
2	25.00	41.67	65.79
3	98.15	100.00	94.15
4	100.00	93.81	86.83
5	83.02	66.32	45.68
Overall	84.71	82.56	82.38

$H = 6$. The results shown are the average values computed over several runs of the algorithm from different initial points.

It is shown in Table I that for $perc = 50$ and 10, the recognition ability of the classifier is considerably high for class 1 of the data set ADS1. Class 2, on the other hand, is recognized relatively poorly. This disparity is expected since the region for class 2 is of a relatively smaller size compared to the region of class 1, and it is totally surrounded by region 1. Table II shows the classwise and overall recognition scores for ADS2, which are again seen to be considerably high. Note that both the artificial data sets have nonoverlapping, nonlinear class boundaries.

TABLE V
RECOGNITION SCORES (%) FOR VOWEL DATA ($F_1 - F_2 - F_3$) FOR $H = 6$

Class	$perc$=50	$perc = 10$	$perc = 5$
δ	57.63	39.36	26.47
a	84.44	83.24	75.00
i	69.77	74.98	65.03
u	82.89	71.23	89.51
ϵ	85.62	83.96	94.39
o	87.32	91.68	54.12
Overall	77.64	75.04	71.84

TABLE VI
RECOGNITION SCORES (%) FOR IRIS DATA FOR $H = 6$

Class	$perc$=50	$perc = 10$	$perc = 5$
1	100.00	97.78	100.00
2	96.00	86.67	91.49
3	92.00	100.00	78.72
Overall	96.00	94.81	90.07

Table III presents the results on the overlapping classes of vowel data with F_1 and F_2 as features. It is seen from the table that class δ yields a poor recognition score for all values of $perc$. This conforms to earlier findings [17], [20], when Bayes and Fuzzy set theoretic classifiers, and MLP were used for vowel classification problem. Since F_3 is a more characteristic feature for speakers than vowels, its inclusion as an additional feature may not necessarily increase the score over that obtained with $F_1 - F_2$ combination [17]. This is evident in Table V where instead of improving the performance, F_3 is seen to increase the confusion in recognition of the system; thereby resulting in a small decrease in performance (over those for $F_1 - F_2$ plane).

TABLE VII
VARIATION OF RECOGNITION (%) WITH H FOR ADS1

Class	Surfaces						
	8	7	6	5	4	3	2
1	94.93	96.44	96.44	94.75	94.81	95.29	85.99
2	79.55	82.10	82.10	69.04	85.51	48.02	48.30
Overall	92.23	93.93	93.93	90.24	93.18	87.00	79.38

TABLE VIII
VARIATION OF RECOGNITION (%) WITH H FOR VOWEL DATA ($F_1 - F_2$)

Class	Surfaces						
	8	7	6	5	4	3	2
δ	39.06	26.56	44.20	29.69	0.00	0.00	0.00
a	86.25	78.89	82.51	82.50	75.00	93.75	0.00
i	85.71	87.66	82.73	87.66	76.62	86.36	87.66
u	71.85	61.74	90.76	87.41	97.04	66.67	0.00
ϵ	72.04	91.93	72.72	68.28	69.89	88.71	0.00
o	83.85	83.85	75.31	78.26	67.70	79.50	100.00
Overall	75.90	76.22	76.62	75.77	70.25	75.77	37.95

Table IV shows the performance of the algorithm on the Landsat data shown in Fig. 11. Class 3 is seen to be amenable to a consistently good recognition score since this class has almost no overlap with the other classes and lies at one extreme end of the class distributions. Recognition of class 2 is seen to be poor since this is totally overlapped with other classes, especially, with classes 4 and 5. It is interesting to note that the recognition score for class 2 increases at the cost of recognition score for class 5 when $perc = 5$. A good performance of the proposed algorithm is also observed for the four-dimensional Iris data (Table VI), which is known to have a very small overlap [19]. As expected, the overall recognition score for Vowel data, Landsat data and Iris data shows a gradual decrease with the decrease in the value of $perc$.

In order to demonstrate the variation of recognition score with H, we have considered ADS1 and Vowel data in $F_1 - F_2$ plane only. Tables VII and VIII show the results for $H = 8$, 7, 6, 5, 4, 3, 2 and $perc = 10$. For both the data sets it is observed that $H = 6$ provides the best result. Increasing the number of surfaces (i.e., H), increases the recognition capability of the algorithm for a specific range of H. Beyond this, the performance gets degraded, since further increase in the number of surfaces makes the resulting decision boundary greatly dependent on the training data set. In other words, when a large number of surfaces is given for constituting the decision boundary, the algorithm can easily place them to approximate the distribution of the training set and hence the boundary closely. This may not necessarily be beneficial (in the sense of generalization) for the test data set, as the results for $H = 8$ (Table VII) and $H = 7$ and 8 (Table VIII) demonstrate.

Although for $H = 2$, the resultant classification system yields a good overall performance for ADS1 data (Table VII), it is not a good choice since the recognition of class 2 (which is embedded in class 1) falls drastically. For the Vowel data, since we have $M = 6$, the minimum number of hyperplanes required (as mentioned in Section II) for its proper classification is 3 ($3 \geq \log_2 6$). However, because of the complexity of the data set, even $H = 4$ is not sufficient for proper classification. This is evident from Table VIII where the classifier fails to recognize some of the classes for $H = 4$, 3, and 2.

TABLE IX
COMPARATIVE RECOGNITION SCORES (%) FOR ADS1 FOR $perc = 10$

Class	Bayes	k-NN	MLP	GA
1	100.00	93.57	94.06	96.44
2	22.73	62.09	76.01	82.10
Overall	86.45	89.28	92.40	93.93

TABLE X
COMPARATIVE RECOGNITION SCORES (%) FOR ADS2 FOR $perc = 10$

Class	Bayes	k-NN	MLP	GA
1	74.63	74.00	75.00	79.23
2	74.57	77.59	75.86	82.57
Overall	74.60	75.66	75.40	80.78

TABLE XI
COMPARATIVE RECOGNITION SCORES (%) FOR VOWEL DATA ($F_1 - F_2$) FOR $perc = 10$

Class	Bayes	k-NN	MLP	GA
δ	46.15	13.85	36.92	44.20
a	81.48	87.65	7.41	82.51
i	84.52	85.16	84.67	82.73
u	94.12	85.29	82.35	90.76
ϵ	71.12	71.66	68.98	72.72
o	75.31	83.33	62.79	75.31
Overall	77.61	75.95	64.45	76.62

A comparison of the performance of the GA based algorithm is made with that of the Bayes classifier, k-NN classifier (for $k = \sqrt{tr}$) and the MLP. The results are presented in Tables IX–XII for only the 2-D data sets when $H = 6$ and $perc = 10$. For both the artificial data sets, ADS1 (Table IX) and ADS2 (Table X), the GA based algorithm provides the best result. This is followed by the scores for MLP and k-NN rule for the two data sets respectively. Note that k-NN classifier is reputed to partition well this type of nonoverlapping, nonlinear regions. As expected, the Bayes classifier performs poorly for both these data sets, since the assumption of multivariate normal distribution is not valid here. Table XI shows the result for the vowel data (in $F_1 - F_2$ plane) where Bayes classifier is seen to provide the best score (77.61%), closely followed by the result of the proposed algorithm (76.62%). The Bayes classifier is known to perform well for this data set [17], assuming multivariate normal densities for the classes. For Landsat data (Table XII) also, Bayes classifier performs best (84.24% overall score) followed by the score of the proposed GA based algorithm (82.56%). As expected, all the four classifiers yield high recognition scores for class 3. MLP is seen to perform poorly for both the Vowel data set (Table XI) and Landsat data (Table XII).

TABLE XII
COMPARATIVE RECOGNITION SCORES (%) FOR LANDSAT DATA FOR $perc = 10$

Class	Bayes	k-NN	MLP	GA
1	93.65	88.10	84.92	85.71
2	62.96	58.33	77.78	41.67
3	100.00	100.00	100.00	100.00
4	86.08	91.24	85.57	93.81
5	60.00	37.89	0.00	66.32
Overall	84.24	81.03	76.85	82.56

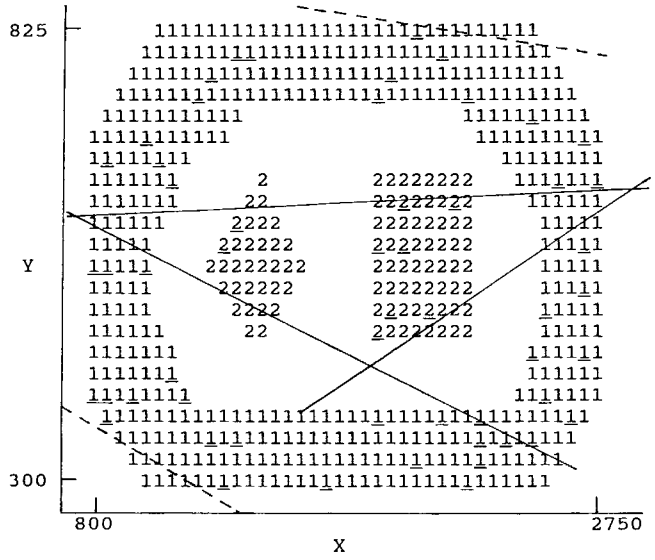

Fig. 12. Classification of 90% test data set using the final boundary generated with 10% training data set for $H = 5$. Training patterns are underscored. Dotted lines indicate redundant lines.

In order to demonstrate the generalization capability of the class boundaries obtained from the training data set, and the role of the postprocessing (for deletion of redundant lines), let us consider Figs. 12 and 13. These figures show the lines generated after the termination of the GA based algorithm for 10% training data of ADS1 with $H = 5$ and 6, respectively, and their ability in classifying the remaining test data. The training patterns are underscored. These lines were obtained on termination of training when the number of misclassified samples becomes zero. After postprocessing for the deletion of redundant lines is performed, two lines are declared to be redundant in both the cases. These are shown by dotted lines in Figs. 12 and 13. Therefore, it is basically three lines (for Fig. 12) and four lines (for Fig. 13) which are contributing to the generation of actual boundary for proper classification, although the algorithm started with $H = 5$ and 6, respectively.

Note that, although both three lines (Fig. 12) and four lines (Fig. 13) are seen to provide zero misclassification of the training data set, from the point of generalization capability over the test data set, four lines are found to provide better performance (viz., Fig. 13, and Table VII for $H = 6$). Note further that if we start the algorithm with $H = 3$ and 4, the resulting boundary and hence the recognition score may not be similar to those obtained after deletion of redundant lines from $H = 5$ and 6, respectively. One of the reasons is that

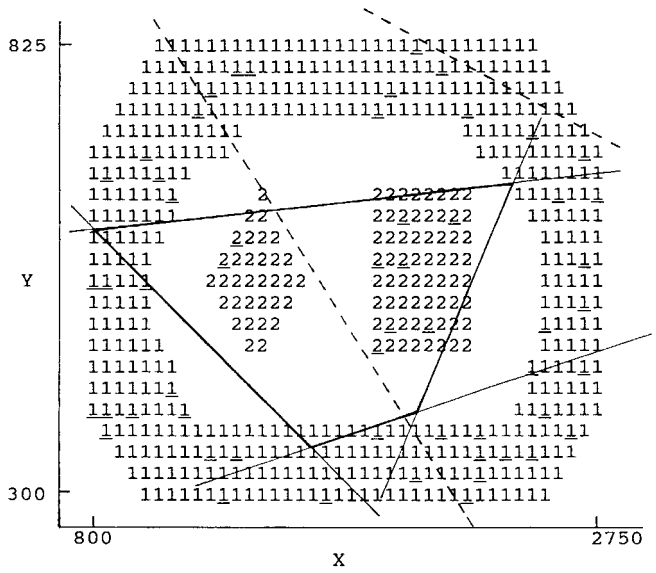

Fig. 13. Classification of 90% test data set using the final boundary generated with 10% training data set for $H = 6$. Training patterns are underscored. Dotted lines indicate redundant lines.

the performance of the algorithm is dependent on the initial seed point. This is evident from the results corresponding to $H = 3$ (Table VII) where the recognition score is seen to be comparatively lower than that for $H = 5$ (with deletion of 2 redundant lines). However, for $H = 4$, the results are seen to be comparable to those for $H = 6$ (with deletion of two redundant lines).

VI. DISCUSSION AND CONCLUSIONS

A method of generating class boundaries in $\Re^N, N \geq 2$, using GA's has been described along with its demonstration on both artificial and real life data having overlapping, concave regions. Since an exact value of the number of hyperplanes required for modeling the decision boundary of a given data set is very difficult to find *a priori*, the method includes a scheme for the automatic deletion of redundant hyperplanes resulting from its conservative estimate. An extensive comparison of the methodology with other classifiers, namely the Bayes classifier (which is well known for discriminating overlapping classes), k-NN classifier and MLP (which are well known for discriminating nonoverlapping, nonlinear regions by generating piecewise linear boundaries) is also presented. The results of the proposed algorithm are seen to be comparable to, sometimes better than, them in discriminating both overlapping and nonoverlapping, nonconvex regions.

The generalization capability of the decision boundary is demonstrated diagrammatically for different values of H. It is observed that an increase in the value of H does not necessarily result in an increase in the generalization capability of the classifier. The reason is that a large number of hyperplanes can quickly approximate the boundary of the training data, which may not be beneficial for the overall classification of the test data.

The method of classification described in this article is sensitive to rotation of the data sets due to the way of choosing

the enclosing hyper rectangle around the data points. It is also evident from the method of specifying a hyperplane (Section II) that translation, dilation or contraction of the data sets would produce a similar change of the decision boundary.

Note that since the deletion algorithm is dependent on the sequence of input hyperplanes taken from S and it does not take the mutual relationships of the hyperplanes into account, the resulting $\mathcal{E}$ containing the essential hyperplanes may not always be minimal or unique. Again, since the deletion process does not help in improving the recognition score for a given H, and a very large value of H leads to a degradation in the recognition capability of the classifier, an appropriate selection of H is necessary.

It is known in the literature [15] that as the number of iterations goes toward infinity, the Elitist model of GA will certainly result in the optimal string. Thus, for the problem under consideration, for infinitely many iterations, any value of H should provide the minimal misclassification for that H. This further strengthens the necessity for a proper selection of H. In this regard, the concept of variable string length in GA's [21] may be adopted where the value of H could be kept variable and can be evolved as an outcome of the GA process. Such an investigation is in progress in our laboratory. A preliminary study in $\Re^2$ showed that 4 lines were required for approximating the decision boundary of data set $ADS1$ for 50% training sample. The overall recognition score was 98.28%. Similarly for $ADS2$ with 50% training data, the decision boundary was found to comprise five lines, of which one was redundant, and 95% overall recognition score was obtained. Alternatively, a meta level GA may also be used for determining H.

In the present investigation we assumed binary representation, because it is well studied in the literature. However, in many real life problems, binary representation may not be the natural one. In that case, one may use some other forms, e.g., floating point representation [4]. Also, the method of binary implementation adopted here may not be efficient in terms of space and time requirements and one may use some other forms of implementation of GA. The purpose of this investigation is to determine whether the characteristics of GA can be exploited for designing an efficient classifier for pattern recognition. Greater importance is therefore given to the issue of improving the recognition score as compared to other standard methods, rather than the implementation aspect of GA.

The GA based classifier is designed for points in $\Re^N$. For dealing with complex data structures like trees, digraphs etc., the representation scheme needs to be modified accordingly. This would mean redefining all the genetic operations and tasks including "Postprocessing" so as to work directly on such structures.

Proper selection of genetic parameters for an application of GA is still an open issue. These parameters are usually selected heuristically. There are no guidelines on the exact strategies to be adopted for different problems. In this work we have taken a fixed population size and crossover probability. mut_prob is kept variable, having a high initial value, then decreasing and finally increasing again. Ideally, this cycle of increasing and decreasing mut_prob should continue for a number of times. We have terminated it after just one cycle due to practical limitations. Directed mutation [22] could also have been used which combines the merits of both genetic search and gradient descent search for accelerating convergence. Investigation is therefore necessary to determine these controlling parameters properly in order to improve the performance of the proposed method.

Again, discretization of the feature space, which is unbounded and continuous, poses a problem in digital pattern recognition with respect to the performance of the systems. In our investigation, a hyper rectangle has been constructed around the set of data points which makes the search space bounded. The possible orientations (angle values) of the hyperplanes are considered to be fixed, and parallel hyperplanes in any direction are kept separated by a small distance $dist/2$. As the discretization is made finer, the performance of the classifier usually improves, but the size of the search space increases; thereby increasing the number of iterations required for GA. An automatic selection of these parameters is therefore necessary as in the case of genetic operators, stated above.

REFERENCES

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

[2] L. Davis, *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.

[3] K. De Jong, "Learning with genetic algorithms: An overview," *Machine Learning 3*. Norwell, MA: Kluwer, 1988, pp. 121–138.

[4] Z. Michalewicz, *Genetic Algorithms + Data Structure = Evolution Programs*. Berlin, Germany: Springer-Verlag, 1992.

[5] J. L. R. Filho and P. C. Treleavan, "Genetic algorithm programming environments," *IEEE Comput.*, pp. 28–43, June 1994.

[6] S. K. Pal and D. Bhandari, "Selection of optimum set of weights in a layered network using genetic algorithm," *Inf. Sci.*, vol. 80, pp. 213–234, 1994.

[7] S. K. Pal, D. Bhandari, and M. K. Kundu, "Genetic algorithms for optimal image enhancement," *Pattern Recognit. Lett.*, vol. 15, pp. 261–271, Mar. 1994.

[8] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Comput.*, vol. 14, pp. 347–361, 1990.

[9] *Proc. 4th Int. Conf. Genetic Algorithms*, Univ. Calif., San Diego, July 1991.

[10] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*. Reading, MA: Addison-Wesley, 1974.

[11] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, pp. 4–22, Apr. 1987.

[12] M. R. Spiegel, *Theory and Problems of Vector Analysis*. Singapore: McGraw-Hill, 1981.

[13] J. D. Schaffer, R. Caruana, L. J. Eshelman, and R. Das, "A study of control parameters affecting online performance of genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann, 1989, pp. 51–60.

[14] C. A. Murthy, D. Bhandari, and S. K. Pal, "Optimal stopping time for genetic algorithms," *Fundam. Inform.*, to be published.

[15] D. Bhandari, C. A. Murthy, and S. K. Pal, "Genetic algorithm with elitist model and its convergence," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 10, pp. 731–747, 1996.

[16] R. Das and D. Whitley, "The only challenging problems are deceptive: Global search by solving order-1 hyperplane," in *Proc. 4th Int. Conf. Genetic Algorithms*, Univ. Calif., San Diego, July 1991, pp. 166–173.

[17] S. K. Pal and D. D. Majumdar, "Fuzzy sets and decision making approaches in vowel and speaker recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-7, pp. 625–629, 1977.

[18] A. Pal, "Some applications of GGA for automatic learning of class parameters in the presence of wrong samples," *Inf. Sci.*, vol. 67, no. 3, pp. 189–208, 1993.

[19] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Ann. Eugenics*, vol. 3, pp. 179–188, 1936.

[20] S. K. Pal and S. Mitra, "Multilayer perception, fuzzy sets and classification," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 683–697, 1992.
[21] D. E. Goldberg, K. Deb, and B. Korb, "Don't worry, be messy," in *Proc. 4th Int. Conf. Genetic Algorithms*, Univ. Calif., San Diego, July 1991, pp. 24–30.
[22] D. Bhandari, N. R. Pal, and S. K. Pal, "Directed mutations in genetic algorithms," *Inf. Sci.*, vol. 79, pp. 251–270, 1994.

Sankar K. Pal (M'81–SM'84–F'93) received the M.Tech. and Ph.D. degrees in radiophysics and electronics in 1974 and 1979, respectively, from the University of Calcutta, Calcutta, India. In 1982, he received the Ph.D. degree in electrical engineering from Imperial College, University of London, London, U.K.

He is a Distinguished Scientist and Founding Head of Machine Intelligence Unit, Indian Statistical Institute, Calcutta. His research interests mainly include pattern recognition, image processing, neural nets, genetic algorithms, and fuzzy sets and systems. He is a co-author of the book *Fuzzy Mathematical Approach to Pattern Recognition* (New York: Wiley, 1986), and a co-editor of two books *Fuzzy Models for Pattern Recognition* (New York: IEEE Press, 1992), and *Genetic Algorithms for Pattern Recognition* (Boca Raton, FL: CRC, 1996).

Dr. Pal was awarded the Fulbright Post-Doctoral Visiting Fellowship in 1986 to work at the University of California, Berkeley, and the University of Maryland, College Park. In 1989, he received an NRC-NASA Senior Research Award to work at the NASA Johnson Space Center, Houston, TX. He received the 1990 Shanti Swarup Bhatnagar Prize in Engineering Sciences, in 1993, he received the Jawaharlal Nehru Fellowship, the Vikram Sarabhai Research Award, and the NASA Tech Brief Award. In 1994, he received the IEEE TRANSACTIONS ON NEURAL NETWORKS Outstanding Paper Award and in 1995, the NASA Patent Application Award. He is a Fellow of the Indian National Science Academy, Indian Academy of Sciences, National Academy of Sciences, India, and Indian National Academy of Engineering, and a Member of the Executive Advisory Editorial Board, IEEE TRANSACTIONS ON FUZZY SYSTEMS and the *International Journal of Approximate Reasoning* and an Associate Editor of the IEEE Transactions on Neural Networks, *Pattern Recognition Letters*, *Neurocomputing*, *Applied Intelligence*, *Information Sciences*, *International Journal of Knowledge Based Intelligent Engineering Systems*, and the *Far-East Journal of Mathematical Sciences*.

Sanghamitra Bandyopadhyay recived the Bachelors degrees in physics and computer science from Calcutta University, Calcutta, India, in 1988 and 1991, respectively, and the Master's degree in computer science from the Indian Institute of Technology, Kharagpur.

She joined the Indian Statistical Institute, Calcutta, as a Senior Research Fellow of the Department of Atomic Energy, Government of India, in April 1994. Until September 1997, she was on a six-month project at Los Alamos National Laboratory, Los Alamos, NM. Her current research interests include genetic algorithms, pattern recognition, neural networks, and image processing.

Ms. Bandyopadhyay received the A. K. Chowdhury Memorial Award for being judged best student in computer science in 1991. She was the recipient of the first Shankar Dayal Sharma Gold Medal and the Institute Silver Medal in 1993 from the Indian Institute of Technology.

C. A. Murthy was born in Ongole, India, in 1958. He received the Master's degree and Ph.D. degrees from the Indian Statistical Institute (ISI), Calcutta.

He is an Associate Professor with ISI. Currently, he is visiting the Pennsylvania State University, University Park. His fields of research include pattern recognition, image processing, neural networks, fractals, genetic algorithms, and fuzzy sets.