VGA-Classifier: Design and Applications

Sanghamitra Bandyopadhyay, C. A. Murthy, and Sankar K. Pal

***Abstract*—A method for pattern classification using genetic algorithms (GAs) has been recently described in [1], where the class boundaries of a data set are approximated by a fixed number H of hyperplanes. As a consequence of fixing H *a priori*, the classifier suffered from the limitation of overfitting (or underfitting) the training data with an associated loss of its generalization capability. In this paper, we propose a scheme for evolving the value of H automatically using the concept of variable length strings/chromosomes. The crossover and mutation operators are newly defined in order to handle variable string lengths. The fitness function ensures primarily the minimization of the number of misclassified samples, and also the reduction of the number of hyperplanes. Based on an analogy between the classification principles of the genetic classifier and multilayer perceptron (with hard limiting neurons), a method for automatically determining the architecture and the connection weights of the latter is described.**

***Index Terms*—Genetic algorithms, multilayer perceptron, pattern recognition, variable length chromosomes.**

I. Introduction

Genetic Algorithms (GAs) [2] are randomized search and optimization techniques guided by the principles of evolution and natural selection. They are efficient, adaptive, and robust search processes having a large amount of implicit parallelism, well suited for complex problems that are multimodal and/or not well characterized. In the area of pattern recognition, there are many tasks involved in the process of analyzing/identifying a pattern which need appropriate parameter selection and efficient search in complex spaces in order to obtain optimum solutions. Therefore, the application of GAs for solving certain problems of pattern recognition (which need optimization of computation requirements, and robust, fast and close approximate solution) appears to be appropriate and natural [3].

Recently, an application of GAs has been reported in the area of (supervised) pattern classification in $\Re^N$ [1] for designing a *GA-classifier*. It attempts to approximate the class boundaries of a given data set with a fixed number (say H) of hyperplanes in such a manner that the associated misclassification of data points is minimized during training. Note that estimation of a proper value of H is crucial for the algorithm to perform well. Since this is difficult to achieve, one may frequently use a conservative value of H (i.e., a value larger than necessary) while designing the classifier. This leads to the problem of an overdependence of the algorithm on the training data, especially for small sample sizes. In other words, since a large number of hyperplanes can readily and closely fit the classes, this may provide good performance during training but a poor generalization capability [1]. Moreover, a large value of H unnecessarily increases the computational effort, and may lead to the presence of redundant hyperplanes in the final decision boundary. (A hyperplane is termed redundant if its removal has no effect on the classification capability of the *GA-classifier*.)

The present paper deals with the issues of overcoming these limitations along with subsequent developments, and has two parts. In the first part, a method has been described to automatically evolve the value of H as a parameter of the classifier design problem. For this purpose, the concept of variable length strings in GAs (VGAs) [4] has been adopted. Unlike the conventional GAs, here the length of a string is not fixed. New crossover and mutation operators are accordingly defined. A factor has been incorporated into the fitness function that rewards a string with smaller number of misclassified samples as well as smaller number of hyperplanes. Let the classifier so designed utilizing the concept of variable string lengths be called *VGA-classifier*. A comparison of the performance of the *VGA-classifier* with those of its fixed length version, k-NN rule, Bayes maximum likelihood classifier and Multilayered Perceptron (MLP) is demonstrated extensively on speech data, cancer detection data, and an artificially generated data set with the number of dimensions ranging from two to nine and complexity ranging from simple (*Cancer*) to complicated overlapping, linearly inseparable classes (*Vowel*). A significant point about this nonparametric classifier is that it is able to approximate any type of decision boundary through the automatic evolution of H.

In the second part, an analogy between the classification principles of the *VGA-classifier* and MLP is established. Based on the analogy, an algorithm for determining automatically the architecture and the connection weights of a multilayer perceptron is described. The effectiveness of the MLP, thus derived using VGA, and its comparison with conventional MLP (having several architectures), are provided for the above mentioned three data sets. In this context we mention other research described in the literature for determining the architecture of neural networks using GAs [5], [6]. In both these approaches, the weights and/or the connectivity information are encoded in the chromosomes, and their appropriate values are evolved by the GA. On the other hand, the proposed algorithm determines the appropriate hyperplanes using GAs, and then designs the MLP directly so as to generate these hyperplanes. Since it has been proved that under limiting conditions the number of hyperplanes will be minimum, the resulting MLP architecture will also be optimum under such conditions.

II. Description of *VGA-Classifier*

A. Criteria for VGA-Classifier

As mentioned in Section I, to overcome the limitations of fixing H *a priori* in *GA-classifier*, the use of variable length strings, representing variable number of hyperplanes for optimally modeling the decision boundary, seems natural and appropriate. In the process, if we aim at reducing the number of misclassified points only, as was the case for fixed length strings, then the algorithm may try to fit as many hyperplanes as possible for this purpose. This, in turn, would obviously be awkward with respect to the generalization capability of the classifier. Thus, the fitness function should be defined in such a way, so as to minimize the number of misclassified samples as well as the requisite number of hyperplanes. Note that in order to be able to tackle the presence of different string lengths in the same population in VGA, the operators need to be changed appropriately, although their sequence of execution may be the same as in conventional GAs [2].

Therefore, the objective of the *VGA-classifier* is to place an appropriate number of hyperplanes in the feature space such that it minimizes the number of misclassified samples and also reduces the number of hyperplanes. Use of variable length strings enables one to check, automatically and efficiently, various decision boundaries consisting of different number of hyperplanes in order to attain the said criterion.

B. Chromosome Representation and Population Initialization

The chromosomes are represented by strings of 1, 0, and # (don't care), encoding the parameters of variable number of hyperplanes. In $\Re^N$, N parameters are required for representing one hyperplane. These are $N-1$ angle variables, $\alpha_1^i, \cdots, \alpha_{N-1}^i$, indicating the orientation of

Manuscript received November 7, 1998; revised October 29, 1999 and June 23, 2000. This paper was recommended by Associate Editor R. Popp.

The authors are with the Machine Intelligence Unit, Indian Statistical Institute, Calcutta 700 035, India.

hyperplane i ($i = 1, 2, \cdots, H$ when H hyperplanes are encoded in the chromosome) with respect to the N coordinate axes, and one perpendicular distance variable, d^i indicating its perpendicular distance from the origin. Let $H_{\max}$ represent the maximum number of hyperplanes that may be required to model the decision boundary of a given data set. It is specified *a priori*. Let the angle and perpendicular distance variables be represented by b_1 and b_2 bits, respectively. Then l_H, the number of bits required to represent a hyperplane, and $l_{\max}$, the maximum length that a string can have, are as follows:

$$l_H = (N-1) * b_1 + b_2 \tag{1}$$

$$l_{\max} = H_{\max} * l_H. \tag{2}$$

Let string i represent H_i hyperplanes. Then its length is $l_i = H_i * l_H$. Initial population is created in such a way that the first and the second strings encode the parameters of $H_{\max}$ and 1 hyperplanes, respectively, to ensure sufficient diversity in the population. For the remaining strings, the number of hyperplanes H_i is generated randomly in the range [1, $H_{\max}$], and the l_i bits are initialized randomly to 1s and 0s.

C. Fitness Computation

As mentioned in Section II, the fitness function (which is maximized) is defined in such a way that

- a string with smaller value of misclassifications is considered to be fitter than a string with a larger value, irrespective of the number of hyperplanes i.e., it primarily minimizes the number of misclassified points, and then
- among two strings providing the same number of misclassifications, the one with the smaller number of hyperplanes is considered to be fitter.

The number of misclassified points for a string i encoding H_i hyperplanes is found as follows : Let the H_i hyperplanes provide M_i distinct regions which contain at least one training data point. (Note that although $M_i \leq 2^{H_i}$, in reality it is bounded by the size of the training data set.) For each such region and from the training data points that lie in this region, the class of the majority is determined, and the region is considered to represent (or be labeled by) the said class. Points of other classes that lie in this region are considered to be misclassified. The sum of the misclassifications for all the M_i regions constitutes the total misclassification $miss_i$ associated with the string. Accordingly, the fitness of string i may be defined as

$$fit_i = (n - miss_i) - \alpha H_i \quad 1 \leq H_i \leq H_{\max} \tag{3}$$

$$= 0, \qquad \text{otherwise} \tag{4}$$

where n = size of the training data set and $\alpha = (1/H_{\max})$.

Let us now explain how the first criterion is satisfied. Let two strings i and j have number of misclassifications $miss_i$ and $miss_j$, respectively, and the number of hyperplanes encoded in them be H_i and H_j, respectively. Let $miss_i < miss_j$ and $H_i > H_j$. (Note that since the number of misclassified points can only be integers, $miss_j \geq miss_i + 1$.) Then,

$$fit_i = (n - miss_i) - \alpha H_i,$$

$$fit_j = (n - miss_j) - \alpha H_j.$$

The aim now is to prove that $fit_i > fit_j$, or that $fit_i - fit_j > 0$. From the above equations, $fit_i - fit_j = miss_j - miss_i - \alpha(H_i - H_j)$. If $H_j = 0$, then $fit_j = 0$ [from (4)] and therefore $fit_i > fit_j$. When $1 \leq H_j \leq H_{\max}$, we have $\alpha(H_i - H_j) < 1$ since $(H_i - H_j) < H_{\max}$. Obviously, $miss_j - miss_i \geq 1$. Therefore, $fit_i - fit_j > 0$, or, $fit_i > fit_j$. The second criterion is also fulfilled since $fit_i < fit_j$ when $miss_i = miss_j$ and $H_i > H_j$.

D. Genetic Operators

Among the operations of selection, crossover and mutation, the selection operation used here may be one of those used in conventional GAs, while crossover and mutation need to be newly defined for VGA. These are now described in detail.

1) Crossover: Two strings, i and j, having lengths l_i and l_j, respectively, are selected from the mating pool. Let $l_i \leq l_j$. Then string i is padded with #s so as to make the two lengths equal. Conventional crossover like single point crossover, two point crossover [2] is now performed over these two strings with probability μ_c. The following two cases may now arise:

- All the hyperplanes in the offspring are complete. (A hyperplane in a string is called *complete* if all the bits corresponding to it are either defined (i.e., 0s and 1s) or #s. Otherwise it is incomplete.)
- Some hyperplanes are incomplete.

In the second case, let u = number of defined bits (either 0 or 1) and t = total number of bits per hyperplane = $(N-1) * b_1 + b_2$ [from (1)]. Then, for each incomplete hyperplane, all the #s are set to defined bits (either 0 or 1 randomly) with probability u/t. In case this is not permitted, all the defined bits are set to #. Thus each hyperplane in the string becomes complete. Subsequently, the string is rearranged so that all the #s are pushed to the end, or, in other words, all the hyperplanes are transposed to the beginning of the strings. The information about the number of hyperplanes in the strings is updated accordingly.

2) Mutation: In order to introduce greater flexibility in the method, the mutation operator is defined in such a way that it can both increase and decrease the string length. For this, the strings are padded with #s such that the resultant length becomes equal to $l_{\max}$. Now for each defined bit position, it is determined whether conventional mutation [2] can be applied or not with probability μ_m. Otherwise, the position is set to # with probability μ_{m_1}. Each undefined position is set to a defined bit (randomly chosen) according to another mutation probability μ_{m_2}. These are described in Fig. 1.

Note that mutation may result in some incomplete hyperplanes, and these are handled in a manner similarly as was done for the crossover operator. For example, the operation on the defined bits, i.e., when $k \leq l_i$ in Fig. 1, may result in a decrease in the string length, while the operation on #s, i.e., when $k > l_i$ in the figure, may result in an increase in the string length. Also, mutation may yield strings having all #s indicating that no hyperplanes are encoded in it. Consequently, this string will have fitness = 0 and will be automatically eliminated during selection.

As in conventional GAs, the operations of selection, crossover and mutation are performed here over a number of generations till a user specified termination condition is attained. Elitism is incorporated such that the best string seen up to the current generation is preserved in the population. The best string of the last generation, thus obtained, along with its associated labeling of regions provides the classification boundary of the n training samples. After the design is complete, the task of the classifier is to check, for an unknown pattern, the region in which it lies, and to put the label accordingly.

III. Implementation and Results

The effectiveness of VGA in automatically determining the value of H of the classifier is demonstrated here for a variety of data sets which range in dimensions from two to nine, and in complexity from simple, disjoint classes to complicated, overlapping classes. The recognition scores of the *VGA-classifier* are also compared with those of the fixed length *GA-classifier*, and Bayes maximum likelihood classifier, k-NN rule and MLP. Three data sets are utilized for performing these experiments. These comprise a two-dimensional artificial data sets, *ADS 1*

[1], and two real life data sets, namely, three-dimensional *Vowel* [1], [7] and and nine-dimensional *Cancer* [8].

For the experiments, the population size and crossover probability are kept fixed at 20 and 0.8, respectively. μ_{m_1} and μ_{m_2} are fixed at 0.1. The values of b_1 and b_2 are chosen to be 8 and 16, respectively. We have chosen the mutation probability value μ_m to vary approximately in the range of [0.015, 0.333]. The range is divided into eight equispaced values. Initially μ_m has a high value (= 0.333), which is slowly decreased in steps to 0.015 and then increased in steps to 0.333. The *GA-classifier* and *VGA-classifier* are implemented with 100 and 200 iterations executed with each value of μ_m, respectively, thereby giving a maximum of 1500 and 3000 iterations, respectively. (Since the search space for VGAs, comprising all combinations of one to $H_{\max}$ hyperplanes, is larger than that of GAs, the former is executed longer.) 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, it becomes 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 best string obtained so far has a large Hamming distance from the actual optimal string. This may very well happen for deceptive problems [2]. Thus, if we continue with the small mutation probability value, it may be too long before the best string is found. So to avoid being stuck at a local optima, the value is again gradually increased. Even if the best string had been found earlier, we lose 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.

The process is terminated if the maximum number of iterations has been executed or a string with zero misclassification is attained. Note that the probability values are taken to be somewhat high since, due to space constraints, the population size had to be kept low. It is known in the literature [9] that, in order to get good performance, the probability values should be high if the population size is low.

A. Performance of the VGA-Classifier

Tables I and II show the number of hyperplanes H_{VGA} as determined automatically by the *VGA-classifier* for modeling the class boundaries of *ADS 1, Vowel* and *Cancer* data sets when the classifier is trained with 10% and 50% samples, respectively. Two different values of $H_{\max}$, viz., 6 and 10, are used for this purpose. The overall recognition scores obtained during testing of the *VGA-classifier* along with their comparison with those of the corresponding fixed length version (i.e., *GA-classifier* with $H = 6$ and 10) are also shown. The scores provided here are the average values obtained over five different runs of the algorithms. The purpose of this exercise is to compare the performance of the *VGA-classifier* and *GA-classifier* starting with the same number of hyperplanes, i.e., $H_{\max}$ for *VGA-classifier* and H for *GA-classifier*.

The results demonstrate that in all the cases, the *VGA-classifier* is able to evolve an appropriate value of H_{VGA} from $H_{\max}$. In addition, its recognition score on the test data set is found, on an average, to be higher than that of the *GA-classifier*. There is only one exception to this for the *Vowel* data when 10% of the samples is used for training (Table II). In this case, $H_{\max} = 6$ does not appear to be a high enough value for modeling the decision boundaries of overlapping *Vowel* classes with *VGA-classifier*. This is also reflected in both the tables, where the scores for *VGA-classifier* with $H_{\max} = 6$ are less than those with $H_{\max} = 10$.

In all the cases where the number of hyperplanes for modeling the class boundaries is less than 6, the scores of *VGA-classifier* with $H_{\max} = 6$ are found to be superior to those with $H_{\max} = 10$. This is so because with $H_{\max} = 10$, the search space is larger than that for $H_{\max} = 6$, which makes it difficult for the classifier to arrive at the optimum arrangement quickly or within the maximum number of iterations considered here. (Note that it may have been possible to further improve the scores and also reduce the number of hyperplanes, if more iterations of VGA were executed.)

In general, the scores of the *GA-classifier* (fixed length version) with $H = 10$ are seen to be lower than those with $H = 6$ because of two reasons; overfitting of the training data and difficulty of searching a larger space. The only exception is with *Vowel* for training with 50% data where the score for $H = 10$ is larger than that for $H = 6$. This is

```
Begin
    l_i = length of string i
    Pad string i with # so that its length becomes l_max
    for k = 1 to l_max do
        Generate rnd, rnd1 and rnd2 randomly in [0,1]
        if k ≤ l_i do /* defined bits */
            if rnd < μ_m do /* Conventional mutation */
                flip bit k of string i
            else /* try changing to # */
                if rnd1 < μ_m1 do
                    Set bit k of string i to #
                endif
            endif
        else /* k > l_i i.e., # */
            if rnd2 < μ_m2 do /* Set to defined */
                Position k of string i set to 0 or 1 randomly
            endif
        endif
    endfor
End
```

Fig. 1. Mutation operation for string i.

TABLE I
H_{VGA} AND THE COMPARATIVE OVERALL RECOGNITION SCORES (%) DURING TESTING (WHEN 10% OF THE DATA SET IS USED FOR TRAINING AND THE REMAINING 90% FOR TESTING)

Data set	*VGA-classifier* $H_{max} = 10$		Score for *GA-classifier* $H = 10$	*VGA-classifier* $H_{max} = 6$		Score for *GA-classifier* $H = 6$
	H_{VGA}	Score		H_{VGA}	Score	
ADS 1	3	95.62	84.26	4	96.21	93.22
Vowel	6	73.66	69.21	6	71.19	71.99
Cancer	2	93.34	77.27	1	95.45	93.01

TABLE II
H_{VGA} AND THE COMPARATIVE OVERALL RECOGNITION SCORES (%) DURING TESTING (WHEN 50% OF THE DATA SET IS USED FOR TRAINING AND THE REMAINING 50% FOR TESTING)

Data set	*VGA-classifier* $H_{max} = 10$		Score for *GA-classifier* $H = 10$	*VGA-classifier* $H_{max} = 6$		Score for *GA-classifier* $H = 6$
	H_{VGA}	Score		H_{VGA}	Score	
ADS 1	4	96.41	95.92	4	96.83	96.05
Vowel	6	78.26	77.77	6	77.11	76.68
Cancer	2	93.69	92.73	2	97.66	93.67

expected, in view of the overlapping classes of the data set and the significantly large size of the training data. One must note in this context that the detrimental effect of overfitting on the generalization performance increases with decrease in the size of the training data.

As an illustration, the decision boundary obtained by the *VGA-classifier* for *ADS 1*, when 10% of the data set is chosen for training, is shown in Fig. 2.

B. Comparison with Existing Methods

For the purpose of comparing the performance of the *VGA-classifier* with those of Bayes maximum likelihood 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), we have provided Table III when 10% data is used for training. For MLP we considered two hidden layers with three different values (5, 10 and 20) of the number of nodes in each layer. Bayes maximum likelihood classifier is implemented assuming normal distribution of classes, with unequal dispersion matrices and *a priori* probabilities. For k-NN rule we assumed k = $\sqrt{n}$, n being the total number of training samples. (It is known that as the number of training patterns n goes to infinity, if the values of k and k/n can be made to approach infinity and 0, respectively, then the k-NN classifier approaches the optimal Bayes classifier [10]. One such value of k for which the limiting conditions are satisfied is $\sqrt{n}$.)

Comparing Tables I and III, the *VGA-classifier* is found to provide the best performance for *ADS 1*. In the case of *Cancer*, its performance is found to be comparable to that of k-NN classifier, the one providing the best result for the data set. For *Vowel*, the Bayes classifier is found to provide the best recognition score (which conforms to earlier findings made in [7]), followed by the score of the *VGA-classifier*.

Besides this, the *VGA-classifier* is seen to be able to automatically approximate different kinds of class boundaries through the evolution of H. This makes it applicable to various types of data sets.

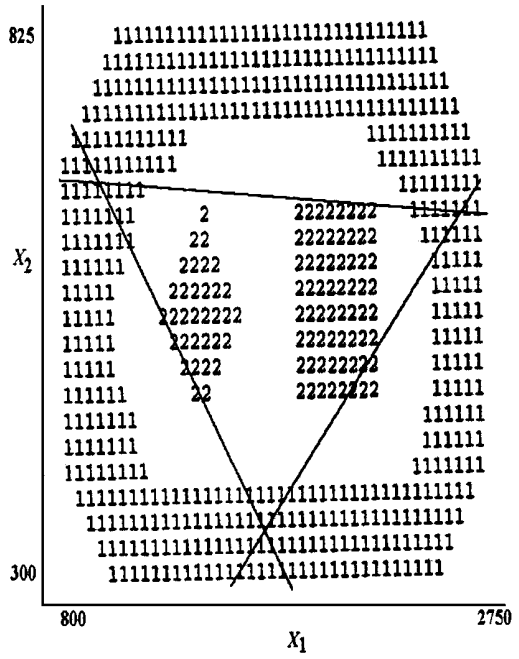

Fig. 2. *ADS 1* along with the *VGA* boundary for $H_{\max} = 10$.

TABLE III
COMPARATIVE RECOGNITION SCORES (%) WHEN 10% DATA IS USED FOR TRAINING

Data Set	Bayes	k-NN	MLP with Nodes in Hidden Layers		
			5:5	10:10	20:20
ADS 1	85.65	90.23	82.47	82.47	82.47
Vowel	77.73	70.35	65.26	67.55	68.48
Cancer	91.39	95.77	94.97	94.97	94.97

IV. RELATION BETWEEN *VGA-Classifier* AND MLP: DETERMINATION OF NETWORK ARCHITECTURE

It is known in the literature that MLP with hard limiting nonlinearities approximates the decision boundary by piecewise linear surfaces. The parameters of these surfaces are encoded in the connection weights and threshold biases of the network. Similarly, the *VGA-classifier* also generates decision boundaries by appropriately fitting a number of hyperplanes in the feature space. The parameters are encoded in the chromosomes. Thus a clear analogy exists between these two models.

Both the methods start from an initial randomly generated state (the set of initial random weights in MLP). They also iterate over a number of generations while attempting to decrease the classification error in the process. The obvious advantage of the GA based method over that of the MLP is that the *GA-classifier* performs concurrent search for a number of sets of hyperplanes, each representing a different classification in the feature space. On the other hand, the MLP deals with only one such set. Thus it has a greater chance of getting stuck at a local optimum, which the *GA-classifier* can overcome. Moreover, *VGA-classifier* does not assume any fixed value of the number of hyperplanes, while MLP assumes a fixed number of hidden nodes and layers. This results in the problem of over fitting with an associated loss of generalization capability for MLP.

TABLE IV
COMPARATIVE RESULTS OF *VGA-Classifier*, MLP DERIVED USING NCA AND SOME MLPS USING CONVENTIONAL BACK-PROPAGATION (* INDICATES THE ARCHITECTURE OBTAINED USING NCA)

Data Set	*VGA-classifier*			MLP with SIGMOID (BP)	
	H_{VGA}	r	Overall Recog. Score (%)	Arch.	Overall Recog. Score (%)
ADS 1	3	5	95.62	2:5:5:2	82.47
				2:10:10:2	82.47
				2:20:20:2	82.47
				2:3:5:2*	82.47
Vowel	6	7	73.66	3:5:5:6	65.26
				3:10:10:6	67.55
				3:20:20:6	68.48
				3:6:7:6*	56.36
Cancer	2	4	93.34	9:5:5:2	94.97
				9:10:10:2	94.97
				9:20:20:2	94.97
				9:2:4:2*	95.70

A. Network Construction Algorithm

Since our aim is to model the equation of hyperplanes, we use the hard limiting function

$$f(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

in the neurons of the MLP. In this connection, let w_{ij}^k denote the connection weight on the link between the jth neuron in layer $(k-1)$ and the ith neuron in layer k. Also, let θ_i^k denote the input bias to the ith neuron in layer k.

Let us assume that the *VGA-classifier* provides H_{VGA} hyperplanes, designated by $\{Hyp_1, Hyp_2, \cdots, Hyp_{H_{VGA}}\}$, r regions, designated by $\{R_1, R_2, \cdots, R_r\}$, and k classes, designated by $\{C_1, C_2, \cdots, C_k\}$. (Note that more than one region may be labeled with a particular class, indicating that $r \geq k$.) Let the parameters of the H_{VGA} hyperplanes be $(\alpha_1^h, \alpha_2^h, \cdots, \alpha_{N-1}^h, d^h), h = 1, 2, \cdots, H_{VGA}$. Let R^i $(i = 1, 2, \cdots, k)$ be the region representing class C_i, and let it be a union of r_i regions given by $R^i = R_{j_1^i} \cup R_{j_2^i} \cup \cdots \cup R_{j_{r_i}^i}, 1 \leq j_1^i, j_2^i, \cdots, j_{r_i}^i \leq r$. Note that each R^i is disjoint.

The network construction algorithm (NCA) is a four step process where the number of neurons, their connection weights and the threshold values are determined. The steps are described below.

Step 1: Allocate N neurons in the input *layer* 0, where N is the dimensionality of the input vector. The neurons in this layer simply transmit the value in the input links to all the output links.

Step 2: Allocate H_{VGA} neurons in *layer 1*. Each neuron is connected to the N neurons of *layer 0*. Let the equation of the ith hyperplane $(i = 1, 2, \cdots, H_{VGA})$ be $c_1^i x_1 + c_2^i x_2 + \cdots + c_N^i x_N - d = 0$, where $c_{N-p}^i = \cos\alpha_{N-(p+1)}^i \sin\alpha_{N-p}^i \sin\alpha_{N-(p-1)}^i \cdots \sin\alpha_{N-1}^i$, $p = 0, 1, \cdots, (N-1)$. Then the corresponding weights on the links to the ith neuron in *layer* 1 from those in *layer 0* are $w_{ij}^1 = c_j^i, j = 1, 2, \cdots, N$, and the bias or threshold is $\theta_i^1 = -d^i$, since it is added to the weighted sum of the inputs to the neurons.

Step 3: Allocate r neurons in *layer 2* corresponding to the r regions. If the ith region R_i $(i = 1, 2, \cdots, r)$ lies on the positive side of the jth hyperplane Hyp_j $(j = 1, 2, \cdots, H_{VGA})$, then $w_{ij}^2 = +1$, otherwise $w_{ij}^2 = -1$, and $\theta_i^2 = -(H_{VGA} - 0.5)$.

Step 4: Allocate k neurons in output layer, *layer 3*, corresponding to the k classes. The task of these neurons is to combine all the distinct regions that actually correspond to a single class. Let the ith class $(i = 1, 2, \cdots, k)$ be a combination of r_i regions. That is, $R^i = R_{j_1^i} \cup R_{j_2^i} \cup \cdots \cup R_{j_{r_i}^i}$. Then the ith neuron of *layer 3*, $(i = 1, 2, \cdots, k)$, is connected to neurons $j_1^i, j_2^i \cdots j_{r_i}^i$ of *layer 2* and, $w_{ij}^3 = 1, \forall j \in \{j_1^i, j_2^i \cdots j_{r_i}^i\}$ whereas $w_{ij}^3 = 0, \forall j \notin \{j_1^i, j_2^i \cdots j_{r_i}^i\}$ and $\theta_i^3 = r_i - 0.5$.

For any given point, at most one output neuron, corresponding to its class, will be high. Also, none of the output neurons will be high if an unknown pattern, lying in a region with unknown classification (i.e., there were no training points in the region) becomes an input to the network.

The NCA guarantees that the total number of hidden layers (excluding the input and output layers) will be at most two. In this context, Kolmogorov's Mapping Neural Network Existence Theorem must be mentioned. The theorem states that any continuous function can be implemented exactly by a three layer, including input and output layers, feedforward neural network. The proof can be found in [11]. However, nothing has been stated about the selection of connection weights and the neuronal functions.

1) Post Processing Step: The structure of the network obtained from the application of NCA may be further pruned. Any neuron in *layer 3* (output layer) that has an input connection from only one neuron in layer 2 may be eliminated. Consequently, this step may result in a neuron in layer i being connected to a neuron in layer $i+2$. In the extreme case, when all the neurons in the output layer (*layer 3*) get their inputs from exactly one neuron in *layer 2*, the output layer can be totally eliminated, and *layer 2* becomes the output layer. This reduces the number of layers from three to two. This situation will arise when $r = k$, i.e., a class is associated with exactly one region formed by the H_{VGA} hyperplanes.

B. Implementation and Results

The performance of the *VGA-classifier* (and consequently that of the MLP derived using NCA) with $H_{\max} = 10$ is compared with that of a conventional MLP having the same architecture as provided by NCA, but trained using the back propagation (BP) algorithm with the neurons executing the sigmoid function. For the purpose of comparison, we have also considered here three more typical architectures for the conventional MLP having two hidden layers with 5, 10, and 20 nodes in each layer, respectively. As earlier, the learning rate and momentum factor were kept fixed at 0.8 and 0.2, respectively. Table IV summarizes the results obtained for *ADS 1, Vowel* and *Cancer* data. The number of hyperplanes (H_{VGA}) and regions (r) obtained by the *VGA-classifier* starting from $H_{\max}$ = 10 are mentioned in columns 2–3. "Arch." in the table denotes the MLP architecture (i.e., the number of nodes in each of the four layers). Note that the number of nodes in the input layer is the same as the number of input features of the data set, and similarly, the number of nodes in the output layer is the same as the number of classes present in the data set. Values of H_{VGA} and r correspond to (or determine) the number of nodes in the first and second hidden layers of the MLP obtained by NCA. These are shown in the last row of column 5 for each data (marked with asterisk). Note that we have not included the recognition scores for MLPs using hard limiters (obtained by the application of NCA) since it will be identical to those of the *VGA-classifier*.

From Table IV it is found that for *ADS 1* and *Vowel*, the *VGA-classifier*, and hence the MLP derived using NCA, provide significantly better performance than the conventional MLPs considered here. For *Cancer*, the MLP whose architecture is derived using NCA, but trained using BP (i.e., entry in the last row of column 5 for *Cancer*) provides the best result. The same version of the MLP is also found to provide the best performance among the other conventional MLPs for all the data sets (except *Vowel*). For the *Vowel* data, it is evident from the table that as the network size becomes larger, the overall recognition score of the MLP using sigmoid function improves. Thus the reason of poor performance of the MLP whose architecture is derived using NCA may be that it is not sufficiently large to be able to model the complex overlapping class boundaries of *Vowel*.

The Arch. values of the MLPs mentioned in the last row of column 5 for each data set are the ones obtained without the application of the post processing step. For example in the case of *ADS 1* data, the number of hyperplanes and regions is 3 and 5 (columns 2–3), respectively. Keeping analogy with this, the Arch. value is mentioned to be 2:3:5:2. In practice, after post processing, the said values became 2:3:4:2. Similarly, for *Vowel* and *Cancer* data, the values after post processing were found to be 3:6:2:6 and 9:2:3:2, respectively.

V. Conclusions

The concept of variable string lengths has been utilized in GAs for developing a pattern classifier (called *VGA-classifier*) that can determine the required number of hyperplanes automatically in order to approximate the class boundaries of any data set in $\Re^N$. New genetic operators are defined for handling variable length chromosomes. The fitness function takes care of minimization of the number of misclassified samples as well as the required number of hyperplanes.

Experimental results on different kinds of data (with nonlinear, overlapping class boundaries and dimensions ranging from two to nine) indicate that given a value of $H_{\max}$, the *VGA-classifier* is able not only to automatically evolve an appropriate value of H for a given data set, but also to result in improved performance as compared to its fixed length version. The performance of the classifier is also found to be comparable to, sometimes better than, those of the k-NN rule, Bayes maximum likelihood classifier and MLP.

The *VGA-classifier* is found to determine automatically the architecture and the associated connection weights of MLP. The method guarantees that the architecture will involve at most two layers (excluding the input and output layers), with the neurons in the first and second hidden layers being responsible for hyperplane and region generation, and those in the output for providing a combination of regions for the classes. This investigation will augment the application domain of the *VGA-classifier* to those areas where MLP has widespread use.

Since the principle of *VGA-classifier* is used for developing NCA, it becomes mandatory to consider hard limiting neurons in the derived MLP. Although this makes the network rigid and susceptible to noise and corruption in the data, one may use NCA for providing a possible appropriate structure of conventional MLPs.

References

[1] S. K. Pal, S. Bandyopadhyay, and C. A. Murthy, "Genetic algorithms for generation of class boundaries," *IEEE Trans. Syst., Man, Cybern.*, vol. 28, no. 6, pp. 816–828, 1998.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[3] S. K. Pal and P. P. Wang, *Genetic Algorithms for Pattern Recognition*. Boca Raton, FL: CRC, 1996.

[4] D. E. Goldberg, K. Deb, and B. Korb, "Do not worry, be messy," in *Proc. 4th Int. Conf. Genetic Algorithms*, R. K. Belew and J. B. Booker, Eds. San Mateo, CA, 1991, pp. 24–30.

[5] S. K. Pal and D. Bhandari, "Selection of optimal set of weights in a layered network using genetic algorithms," *Inform. Sci.*, vol. 80, pp. 213–234, 1994.

[6] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Trans. Neural Networks*, vol. 5, no. 1, pp. 39–53, 1994.

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

[8] O. L. Mangasarin, R. Setiono, and W. H. Wolberg, "Pattern recognition via linear programming: Theory and application to medical diagnosis," in *Large-Scale Numerical Optimization*, T. F. Coleman and Y. Li, Eds. Philadelphia, PA: SIAM, 1990, pp. 22–30.

[9] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, pp. 122–128, 1986.

[10] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. New York: Academic, 1972.

[11] R. Hecht-Nielsen, "Kolmogorov's mapping neural network existence theorem," in *Proc. 1st IEEE Int. Conf. Neural Networks*, vol. 3, San Diego, CA, 1987, pp. 11–14.

Using Cluster Skeleton as Prototype for Data Labeling

Yuhui Yao, Lihui Chen, and Yan Qiu Chen

***Abstract*—A new approach, designed for clustering data whose underlying distribution shapes are arbitrary, is presented. This study is concerned with the use of the skeleton of a cluster as its prototype, which can represent the cluster more closely than that of using a single data point. The given data set is then partitioned into those skeleton-represented clusters without any prior knowledge nor assumptions of hidden structures. A novel function called *cluster characteristic function* (CCF) has been constructed and the associated theorems have been proposed and proved that the proper number of clusters can be determined with the approach.**

***Index Terms*—cluster characteristic function, clustering, fuzzy c-means, skeleton clustering, unsupervised learning.**

I. Introduction

Clustering refers to the process of grouping samples so that the samples are similar within each group and different between any two groups. It attempts to discover the inherent structure in a data set. It is unsupervised learning and no *priori* information on the data distribution can be assumed. In recent years, many schemes have been proposed for solving this intrinsically difficult problem. These methods can be sorted into four major categories:

2) prototype-based central point clustering;
3) learning-network-based central point clustering;
4) shell-clustering;
5) graph clustering.

Prototype-based central point clustering attempts to use the central point of a cluster as the prototype to represent the cluster and then labels all samples with those prototypes. The first published method is hard c-means proposed by Duda and Hart [1]. Later modifications and improvements include:

2) *Fuzzy c-means*, proposed by Bezdek [2]. Although fuzzy c-means can find a partition of data for a fixed number c of clusters, it needs a good cluster index [18], [19] to automatically determine the optimal number of clusters. It is computationally expensive since it needs to perform the clustering for a range of c values. Furthermore, reliable validity measures are quite difficult to attain, and the number of clusters that optimizes a particular validity may not always be "correct" [4].
3) *Possibilistic c-means*, a mode-seeking algorithm, proposed by Krishnapuram and Keller [3].
4) Statistical models such as *probabilistic mixtures*, proposed by Titterington [5].
5) Vector quantization approaches such as *the generalized Lloyd algorithm* [6].
6) Maxim Clustering Entropy Principle models such as *least biased fuzzy clustering method*, proposed by Gerardo Beni and Xiaomin Liu [7].

Learning-network-based central point clustering uses learning networks to achieve data labeling. During the process of training, the network updates all the vector weights and lets them to be

Manuscript received October 29, 1999; revised June 23, 2000. This paper was recommended by Associate Editor W. Pedrycz.

The authors are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798.

Publisher Item Identifier S 1083-4419(00)08805-1.