PATTERN CLASSIFICATION USING GENETIC ALGORITHMS

Sanghamitra Bandyopadhyay
Machine Intelligence Unit
Indian Statistical Institute
Calcutta 700035, India

A thesis submitted to the *Indian Statistical Institute*
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
1998

ACKNOWLEDGEMENTS

Words seem insufficient to express my gratitude and indebtedness to *Prof. Sankar K. Pal*, who not only supervised my dissertation work, but also acted as a friend and philosopher in my time of need. I owe a lot to him for providing constant encouragement and affection throughout the last four years.

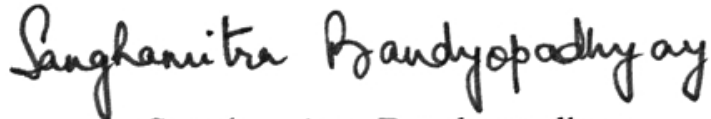
Heartfelt thanks are due to *Dr. C. A. Murthy* without whose help, guidance and support this work could not have progressed smoothly. I consider this thesis as a kind of joint venture between myself, *Prof. Sankar K. Pal* and *Dr. C. A. Murthy*, which is amply evident from the list of articles. Thanks are due to them for their kind permission to include the joint research work in this thesis.

I owe a lot to Dr. Ujjwal Maulik, who has helped me immensely in my thesis writing process. His encouragement, constant support and active help are gratefully acknowledged. Special note of thanks to Dr. M. K. Kundu, Dr. N. R. Pal, Dr. D. P. Mandal, Dr. S. Mitra, Dr. J. Basak, Dr. A. Ghosh, Mr S. N. Biswas, Mr. B. Umashankar, Dr. K. Pal, Dr. S. N. Sarbadhikari and my research colleagues Mousumi Acharyya, Susmita De, Rajat De, Suman Mitra and Sitabhra Sinha. I also acknowledge the office staff of Machine Intelligence Unit for providing a friendly environment, the workers in the reprography unit for careful photocopying and Sri. S. Chakraborty for drawing some of the diagrams.

I shall forever remain indebted to my parents for helping me in all phases of my life. It is their constant encouragement, enthusiasm and support that helped me face the challenges of life. Thanks are due to my in-laws, who supported me in my work, specially during the tough times of the thesis writing process.

I take this opportunity to express heartfelt gratitude to my teachers and friends at Indian Institute of Technology, Kharagpur, India, and Rajabazar Science College, Calcutta, India. I would also like to acknowledge Dr. H. Kargupta, with whom I had the wonderful opportunity to work at Los Alamos National Laboratory, New Mexico, USA.

Finally, I express my sincere thanks to the authorities of ISI for the facilities extended to carry out my research work, and to the Department of Atomic Energy, Govt. of India, for providing me fellowship to support my research.



ISI, Calcutta
February, 1998.

Sanghamitra Bandyopadhyay

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Scope of the Thesis

1.1 Introduction

Pattern recognition and machine learning form a major area of research and development activity that encompasses the processing of pictorial and other non-numerical information obtained from the interaction between science, technology and society. A motivation for the spurt of activity in this field is the need for people to communicate with the computing machines in their natural mode of communication. Another important motivation is that the scientists are also concerned with the idea of designing and making intelligent machines that can carry out certain tasks that we human beings do. The most salient outcome of these is the concept of future generation computing systems.

Machine recognition of patterns can be viewed as a two fold task, comprising learning the invariant properties of a set of samples characterizing a class, and of deciding that a new sample is a possible member of the class by noting that it has properties common to those of the set of samples. The tasks required for developing and implementing a decision rule can be described as a transformation from the measurement space M to the feature space F and finally to the decision space D, i.e.,

$$M \rightarrow F \rightarrow D.$$

Here the mapping $\delta : F \rightarrow D$ is the decision function, and the elements $d \in D$ are termed as decisions.

Genetic algorithms (GAs) [22, 35, 46, 51, 58, 61, 63, 64, 73, 81, 89, 126, 170, 206] are randomized search and optimization techniques guided by the principles of evolution and natural genetics. The term was first mentioned by Bagley in 1967 [8], when he devised a genetic algorithm based game playing program using some commonly used operators. He found that the GA was insensitive to the game non-linearity, and performed well over a range of environments. It was then with the pioneering work of Holland in 1975 [89] that GAs were firmly established as an effective search and optimization strategy.

GAs mimic some of the processes observed in natural evolution, which include operations like selection, crossover and mutation. They perform multimodal search in complex landscapes and provide near optimal solutions for objective or fitness function of an optimization problem. They are efficient, adaptive and robust search

processes, with a large amount of implicit parallelism [73, 89]. Genetic algorithms are a relatively recent development, and are gradually finding widespread applications in solving problems requiring efficient and effective search, in business, scientific and engineering circles [22, 58, 64, 68, 81, 153, 170].

Many tasks involved in the process of recognizing a pattern need appropriate parameter selection and efficient search in complex and large spaces in order to attain optimal solutions. This makes the process not only computationally intensive, but also leads to a possibility of losing the exact solution. Therefore, the application of GAs for solving certain problems of pattern recognition, that require optimization of computation requirements, and robust, fast and close approximate solution, seems appropriate and natural. Additionally, the existence of the proof of convergence of GAs to the global optimal solution as the number of iterations goes to infinity [28], further strengthens the theoretical basis of its use in search problems.

The present thesis provides some results of investigation, both theoretical and experimental, demonstrating the effectiveness of genetic algorithms in designing classifiers for pattern recognition in $\mathbb{R}^N$. The underlying philosophy of the methodology involves search for an appropriate arrangement of a number of linear as well as nonlinear surfaces in the feature space which can approximate the class boundaries of a data set. Before we describe the scope of the thesis, we provide an overview of some pattern classification techniques.

Section 1.2 presents a description of the basic concept, features and techniques of pattern recognition briefly, since it is a well studied subject. The basic principles and features of genetic algorithms are presented in Section 1.3 in details, along with its various theoretical and empirical aspects. As mentioned, GAs are gradually finding applications in diverse fields. Some of the attempts made so far in relation to pattern recognition problems are described in Section 1.3. Finally, Section 1.4 discusses the scope of this thesis.

1.2 Overview of Pattern Classification Techniques

A typical pattern recognition system consists of three phases as shown in Fig. 1.1. These are a *data acquisition, feature extraction* and *classification.* In the data

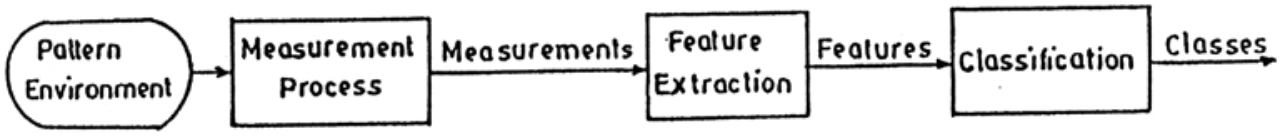


Figure 1.1: A typical pattern recognition system

acquisition phase, depending on the environment within which the objects are to be classified, data are gathered using a set of sensors. These are then passed on to the feature extraction phase, where the dimensionality of the data is reduced by measuring/retaining only some characteristic features or properties. In a broader perspective, this stage significantly influences the entire recognition process. Finally, in the classification phase, the extracted features are passed on to the classifier that evaluates the incoming information and makes a final decision. This phase basically establishes a transformation between the features and the classes. Different forms of transformation can be a Bayesian rule of computing *a posterior* class probability, nearest neighbor rule, linear discriminant functions, perceptron rule, nearest prototype rule etc. [2, 3, 55, 56, 65, 67, 69, 80, 155, 209].

Pattern recognition, by its nature, admits many approaches, sometimes complementary, sometimes competing, to the approximate solution of a given problem. These include decision theoretic approach (both deterministic and probabilistic), syntactic approach, connectionist approach, and fuzzy set theoretic approach.

In the decision theoretic approach [55, 56, 67, 209], once a pattern is transformed, through feature selection, to a vector in the feature space, its characteristics are expressed only by a set of numerical values. Classification can be done using deterministic or probabilistic technique. On the other hand, when a pattern is rich in structural information (e.g., picture recognition, character recognition, scene analysis) i.e., the structural information plays an important role in describing and recognizing the patterns, syntactic approaches, which deal with representation of structures via sentences, grammars and automata, are usually used. In the syntactic method [65, 209], the ability of selecting and classifying the simple pattern primitives and

their relationships represented by the composition operations is the vital criterion of making a system effective. Since the techniques of composition of primitives into patterns are usually governed by the formal language theory, the approach is often referred to as linguistic approach. An introduction to a variety of approaches based on this idea can be found in [65].

For any pattern recognition system, one desires to achieve robustness with respect to random noise and failure of components. Another important requirement is to have output in real time. It is also desirable for the system to be adaptive to changes in the environment. Moreover, a system can be made artificially intelligent if it is able to emulate some aspects of the human processing system. Connectionist (neural network based) approaches [108, 183, 184] to pattern recognition are attempts to achieve these goals. Neural networks, can be formally defined as *massively parallel interconnections of simple (usually adaptive) processing elements that interact with objects of the real world in a manner similar to biological systems.* Its origin can be traced to the work of Hebb [86], where a local learning rule is proposed. The benefit of neural nets lies in the high computation rate provided by their inherent massive parallelism. This allows real-time processing of huge data sets with proper hardware backing. All information is stored distributed among the various connection weights. The redundancy of interconnections produces a high degree of robustness resulting in a *graceful degradation* of performance in the case of noise or damage to a few nodes/links. Neural network models have been studied for many years with the hope of achieving human like performance (artificially), particularly in the field of pattern recognition, by capturing the key ingredients responsible for the remarkable capabilities of the human nervous system. Note that these models are extreme simplifications of the actual human nervous system.

Methods and technologies developed under the above mentioned categories may, again, be fuzzy set theoretic in order to handle uncertainties, arising from vague, incomplete, linguistic, overlapping patterns, at various stages of pattern recognition systems. This approach is developed based on the realization that a pattern may belong to more than one class, with varying degree of class membership. Accordingly, fuzzy decision theoretic, fuzzy syntactic, fuzzy neural approaches are developed [25, 27, 100, 101, 149, 157].

The methods developed in this thesis (using genetic algorithms) belong to decision

theoretic approach. In the following section we describe the tasks of data acquisition and feature selection, and a few classification techniques.

1.2.1 Data Acquisition

Pattern recognition techniques are applicable in a wide domain, where the data may be qualitative, quantitative, or both; they may be numerical, linguistic, pictorial, or any combination thereof. Generally, the data structures that are used in pattern recognition systems are of two types : *object data vectors* and *relational data.* Object data, sets of numerical vectors of Q features, are represented in the sequel as $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_t\}$, a set of t feature vectors in the Q dimensional measurement space Ω_Y. The ith object, $i = 1, 2, \ldots, t$, observed in the process has vector Y_i as its numerical representation; y_{ij} is the jth, $(j = 1, 2, \ldots, Q)$ feature associated with the object i.

Relational data is a set of t^2 numerical relationships, say $\{r_{ii'}\}$, between pairs of objects. In other words, $r_{ii'}$ represents the extent to which object i and i' are related in the sense of some binary relationship ρ. If the objects that are pairwise related by ρ are called $O = \{o_1, o_2, \ldots, o_t\}$, then $\rho : O \times O \rightarrow I\!R$.

Sometimes, data acquisition phase includes some preprocessing tasks like noise reduction, filtering, encoding, enhancement for extracting pattern vectors. For example, if the input pattern is an image, these preprocessing operations play a crucial role for extracting salient features for its recognition.

1.2.2 Feature Selection

Feature selection is one of the major tasks in any automatic pattern recognition system. The main objective of *feature selection* [21, 23, 38, 55], is to retain the optimum salient characteristics necessary for the recognition process and to reduce the dimensionality of the measurement space Ω_Y so that effective and easily computable algorithms can be devised for efficient classification. The problem of feature selection has two aspects. The formulation of a suitable criterion to evaluate the goodness of a feature and the selection of optimal subset from the available features.

The major mathematical measures so far devised for the estimation of feature quality are mostly statistical in nature, and can be broadly classified into two categories.

- Feature selection in the measurement space.
- Feature extraction in the transformed space.

The techniques in the first category generally reduce the dimensionality of the feature set by discarding redundant information carrying features. On the other hand, those in the second category utilize all the information contained in the pattern vectors, and map higher dimensional pattern vector to a lower dimensional one.

Feature selection is the process of selecting a map of the form $X = f(Y)$, by which a sample Y $(=y_1, y_2, \ldots, y_Q)$ in a Q dimensional measurement space Ω_Y is transformed into a point X $(=x_1, x_2, \ldots, x_N)$ in an N dimensional feature space Ω_X, where $N < Q$.

The pioneering research on feature selection mostly deals with statistical tools. Later, the thrust of the research shifted to the development of various other approaches to feature selection, including fuzzy and neural approaches [99, 142, 149, 179].

1.2.3 Classification

The problem of classification is basically one of partitioning the feature space into regions, one region for each category of input. Thus it attempts to assign every data point in the entire feature space to one of the possible (say k) classes. Classifiers are usually, but not always, designed with labeled data, in which case these problems are sometimes referred to as *supervised classification* (where the parameters of a classifier function D are learned). Some common examples of the *supervised* pattern classification techniques are the nearest neighbor rule, Bayes maximum likelihood classifier, perceptron rule. Many clustering algorithms (e.g., K-means, Isodata) are used as precursors to the design of a classifier when the only data available are unlabeled data. In such cases, the problems are sometimes referred to as *unsupervised classification*. Some of the commonly used classification (supervised) methods, with which our GA based classifiers are compared, are now described here.

Decision Theoretic Approach

Let $C_1, C_2, \ldots, C_i, \ldots, C_k$ be the k possible classes in an N dimensional feature space. Let $\mathbf{x} = [x_1, x_2, \ldots, x_j, \ldots, x_N]'$ be an unknown pattern vector. In deterministic classification approach, it is assumed that there exists only one unambiguous pattern class corresponding to each of the unknown pattern vectors.

If the pattern $\mathbf{x}$ is a member of class C_i, the *discriminant (decision) function*, $D_i(\mathbf{x})$ associated with the class C_i, $i = 1, 2, \ldots, k$ must then possess the largest value. In other words, classificatory decision would be as follows :

$$\text{Decide} \quad \mathbf{x} \in C_i, \qquad if \quad D_i(\mathbf{x}) > D_l(\mathbf{x}) \tag{1.1}$$

$(\forall\, i, l)$ in $(1, \ldots, k)$ and $i \neq l$. Ties are resolved arbitrarily. The decision boundary in the feature space between regions associated with the classes C_i and C_l would be governed by the expression

$$D_i(\mathbf{x}) - D_l(\mathbf{x}) = 0. \tag{1.2}$$

Many different forms satisfying Eq. (1.2) can be selected for $D_i(\mathbf{x})$. The functions that are often used are linear discriminant functions, quadratic discriminant functions, polynomial discriminant functions. One commonly used piecewise linear discriminant function which is used in nearest neighbor (NN) classifier involves distance as a similarity measure for classification. This is described below.

NN rule [56, 67, 209] :

Let us consider a set of n pattern points of known classification $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, where it is assumed that each pattern belongs to one of the classes $C_1, C_2, \ldots, C_i, \ldots, C_k$. The NN classification rule then assigns a pattern $\mathbf{x}$ of unknown classification to the class of its nearest neighbor, where $\mathbf{x}_i \in \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ is defined to be the nearest neighbor of $\mathbf{x}$ if

$$D(\mathbf{x}_i, \mathbf{x}) = \min_l \{D(\mathbf{x}_l, \mathbf{x})\}, \qquad l = 1, 2, \ldots, n \tag{1.3}$$

where D is any distance measure definable over the pattern space.

Since the aforesaid scheme employs the class label of only the nearest neighbor to $\mathbf{x}$, this is known as the 1-NN rule. If k neighbors are considered for classification, then the scheme is termed as the k-NN rule. The k-NN rule assigns a pattern $\mathbf{x}$ of

unknown classification to class C_i if the majority of the k nearest neighbors belongs to class C_i. The details on the k-NN rule along with the probability of error is available in [56, 67, 209]. Since the k-NN classifier generates piecewise linear boundaries, like the classifiers based on genetic algorithms to be described in the later chapters, we have demonstrated its performance as a comparison. ♣

In most of the practical problems, the features are usually noisy and the classes in the feature space are overlapping. In order to model such systems, the feature values $x_1, x_2, \ldots, x_j, \ldots, x_N$ are considered as random values in the probabilistic approach. The most commonly used classifier in such probabilistic systems is the *Bayes maximum likelihood classifier*, which is now described.

Bayes Maximum Likelihood Classifier [2, 209] :

Let P_i denote the *a priori* probability and $p_i(\mathbf{x})$ denote the class conditional density corresponding to the class C_i $(i = 1, 2, \ldots, k)$. If the classifier decides $\mathbf{x}$ to be from the class C_i, when it actually comes from C_l, it incurs a loss equal to L_{li}. The expected loss (also called the conditional average loss or risk) incurred in assigning an observation $\mathbf{x}$ to the class C_i is given by

$$r_i(\mathbf{x}) = \sum_{l=1}^{k} L_{li}\ p(C_l/\mathbf{x}), \tag{1.4}$$

where $p(C_l/\mathbf{x})$ represents the probability that $\mathbf{x}$ is from C_l. Using Bayes formula, Eq (1.4) can be written as,

$$r_i(\mathbf{x}) = \frac{1}{p(\mathbf{x})} \sum_{l=1}^{k} L_{li}\ p_l(\mathbf{x}) P_l, \tag{1.5}$$

where

$$p(\mathbf{x}) = \sum_{l=1}^{k} p_l(\mathbf{x}) P_l.$$

The pattern $\mathbf{x}$ is assigned to the class with the smallest expected loss. The classifier which minimizes the total expected loss is called the *Bayes classifier* .

Let us assume that the loss (L_{li}) is zero for correct decision and greater than zero but same for all erroneous decisions. In such situations, the expected loss, Eq. (1.5), becomes

$$r_i(\mathbf{x}) = 1 - \frac{P_i p_i(\mathbf{x})}{p(\mathbf{x})}. \tag{1.6}$$

Since $p(\mathbf{x})$ is not dependent upon the class, the Bayes decision rule is nothing but the implementation of the decision functions

$$D_i(\mathbf{x}) = P_i p_i(\mathbf{x}), \qquad i = 1, 2, \ldots, k, \tag{1.7}$$

where a pattern $\mathbf{x}$ is assigned to class C_i if $D_i(\mathbf{x}) > D_l(\mathbf{x})$, $\forall l \neq i$. This decision rule provides minimum probability of error. It is to be noted that if the *a priori* probabilities and the class conditional densities are estimated from a given data set, and the Bayes decision rule is implemented using these estimated values (which may be different from the actual values), then the resulting classifier is called the *Bayes maximum likelihood classifier.*

Assuming normal (Gaussian) distribution of patterns, with mean vector μ_i and covariance matrix $\sum_i$, the Gaussian density $p_i(\mathbf{x})$ may be written as

$$p_i(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{N}{2}} |\sum_i|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mu_i)' \sum_i{}^{-1} (\mathbf{x} - \mu_i)\right], \quad i = 1, 2, \ldots, k. \tag{1.8}$$

Then, $D_i(\mathbf{x})$ becomes (taking log)

$$D_i(\mathbf{x}) = \ln P_i - \tfrac{1}{2} \ln |\sum_i| - \frac{1}{2}(\mathbf{x} - \mu_i)' \sum_i{}^{-1} (\mathbf{x} - \mu_i) \quad i = 1, 2, \ldots, k \tag{1.9}$$

Note that the decision function in Eq. (1.9) are hyperquadrics, since no terms higher than the second degree in the components of $\mathbf{x}$ appear in it. It can thus be stated that the Bayes maximum likelihood classifier for normal distribution of patterns provides a second order decision surface between each pair of pattern classes. An important point to be mentioned here is that if the pattern classes are truly characterized by normal densities, on an average, no other surface can yield better results. In fact, the Bayes classifier designed over known probability distribution functions, provides, on an average, the best performance for data sets which are drawn according to the distribution. In such cases, no other classifier can provide better performance on an average, because Bayes classifier gives minimum probability of misclassification over all decision rules. It is because of this characteristic of the Bayes classifier, that its performance has been compared with the genetic algorithm based classifiers developed in the later chapters.

Connectionist Approach

As already mentioned, connectionist approaches to pattern recognition have been widely studied because of their various characteristics like robustness, adaptivity, parallelism, graceful degradation under failure and real time processing. Perceptron [48, 85, 154, 183, 184], is the earliest development in this direction. It may be broadly categorized into two classes : *single layer perceptron* and *multilayer perceptron.* The single layer perceptron, given two classes of patterns, attempts to find a linear decision boundary between them. If the classes are linearly separable, the perceptron algorithm is guaranteed to find a separating hyperplane in a finite number of steps. However, as shown by Minsky and Papert [128], if the pattern space is not linearly separable, the perceptron algorithm will not converge. In fact, it is shown to fail for the simple XOR problem, which is linearly inseparable.

This limitation was overcome by Rumelhart, Hinton and Williams [182], who developed the generalized delta rule for training of multilayer perceptrons (MLP). The multilayer perceptron, known for discriminating non-linearly separable classes as well, has since then been studied and applied in various fields [117]. The MLP, with the individual processing elements (or neurons) executing the hard limiting function, partition the feature space using piecewise linear boundaries. Since, this is also a characteristic feature of the genetic algorithm based classifiers developed in this thesis, the performance of MLP has been demonstrated for comparison.

Multilayer Perceptron (MLP) [48, 85, 154]:

A Multilayer Perceptron (MLP) consists of several layers of simple neurons with full connectivity existing between neurons of adjacent layers. Fig. 1.2 shows an example of a three layer MLP which consists of an input layer (*layer 0*), one hidden layer (*layers 1*) and an output layer (*layer 2*).

The neurons in the input layer serve the purpose of fanning out the input values to the neurons of *layer 1.* Let

$$w_{ji}^{(l)}, \qquad l = 1,\ 2 \tag{1.10}$$

represent the connection weight on the link from the ith neuron in layer $l-1$ to the jth neuron in layer l. Let $\theta_j^{(l)}$ represent the threshold of the jth neuron in layer l.

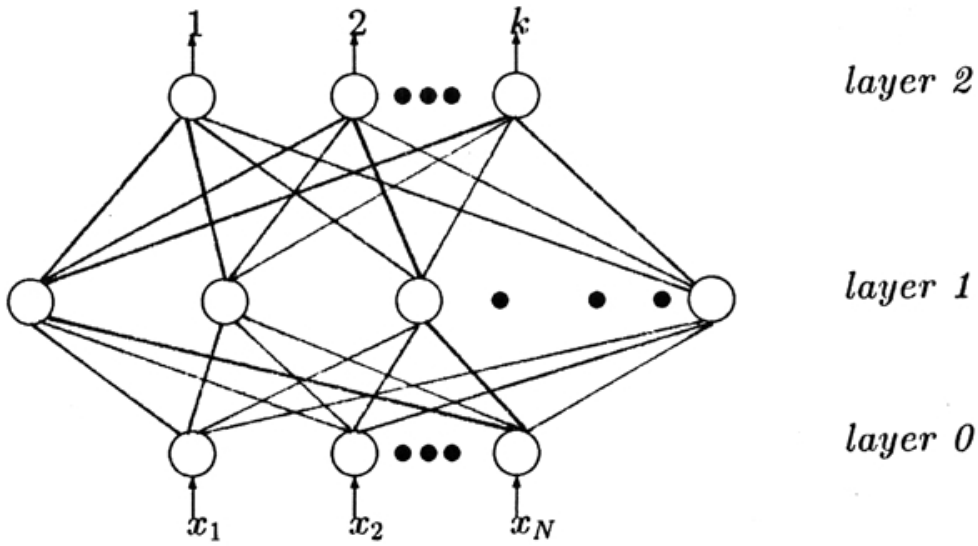


Figure 1.2: Multilayer Perceptron

The total input, $x_j^{(l)}$, received by the jth neuron in layer l is given by

$$x_j^{(l)} = \sum_i y_i^{(l-1)} \, w_{ji}^{(l)} + \theta_j^{(l)}, \tag{1.11}$$

where $y_i^{(l-1)}$ is the output of the ith neuron in layer $l-1$. For the input layer

$$y_i^{(0)} = x_i \tag{1.12}$$

where x_i is the ith component of the input vector. For the other layers

$$y_i^{(l)} = f(x_i^{(l)}) \qquad l = 1,\ 2. \tag{1.13}$$

Several functional forms like threshold logic, hard limiter, sigmoid can be used for $f(.)$.

There are several algorithms for training the network in order to learn the connection weights and the thresholds from a given training data set. Backpropagation (BP) is one such learning algorithm, where the least mean square error of the network output is computed, and this is propagated in a top down manner (i.e., from the output side) in order to update the weights. The error is computed as the difference between

the actual and the desired output when a known input pattern is presented to the network. A gradient descent method along the error surface is used in BP. More information on MLP and other neural methods are available in [48, 85, 117, 154].

Fuzzy Set Theoretic Approach

Uncertainty can arise either implicitly or explicitly in each and every phase of a pattern recognition system. It results from the incomplete or imprecise or ambiguous input information, the ill-defined and/or overlapping boundaries among the classes or regions, and the indefiniteness in defining/extracting features and relations among them. Any decision taken at a particular level will have impact on all other higher level activities. It is therefore required for a pattern recognition system to have sufficient provision for representing these uncertainties involved at every stage, so that the ultimate output of the system can be obtained with least uncertainty.

Fuzzy sets were introduced in 1965 by Lotfi A. Zadeh [225], as a way to represent vagueness in everyday life. Since this theory is a generalization of the classical set theory, it has greater flexibility to capture various aspects of incompleteness, impreciseness or imperfection in information about a situation.

The relevance of fuzzy set theory in the realm of pattern recognition [25, 27, 100, 101, 149, 150, 156, 157] is adequately justified in

- representing input patterns as array of membership values denoting the degree of possession of certain properties
- representing linguistically phrased input features for processing
- providing an estimate (representation) of missing information in terms of membership values
- representing multiclass membership of ambiguous patterns and in generating rules and inferences in linguistic form
- extracting ill-defined image regions, primitives, properties and describing relations among them as fuzzy subsets.

Other Approaches

There have been several attempts over the last decade to evolve new approaches to pattern recognition and deriving their hybrids by combining the merits of several techniques. An integration of neural network and fuzzy theory, commonly known as the neuro fuzzy computing is one such hybrid paradigm [26, 129, 147, 149, 154, 158, 165, 166, 167, 168, 169].

Application of rule based systems in pattern recognition has also gained popularity in the recent past. By modeling the rules and facts in terms of fuzzy sets, it is possible to make interfaces using the concept of approximate reasoning [137]. An approach to classifying unknown patterns by combining the k-NN rule with Dempster-Shafer theory of evidence [195] has been formulated in [53].

Recently, there have been a few attempts in using genetic algorithms in pattern recognition tasks. Since the present thesis deals with this area, a detailed discussion on the basic principles of GAs and its various applications to pattern recognition is presented separately in Section 1.3.

1.2.4 Applications of Pattern Recognition

Pattern recognition research is mostly driven by the need to process data and information obtained from the interaction between human society, science and technology. As already mentioned, in order to make machines more intelligent and human like, it must possess automatic pattern recognition capability. Some of the application areas of pattern recognition are :

1. *medicine :* medical diagnosis, image analysis, disease classification;
2. *natural resource study and estimation :* agriculture, forestry, geology, environment;
3. *man-machine communication :* automatic speech recognition and understanding, natural language processing, image processing, script recognition;
4. *vehicular :* automobile, airplane, train, boat controllers;

5. *defense :* automatic target recognition, guidance, control;

6. *police and detective :* crime and criminal detection from analyses of speech, handwriting, fingerprint, photograph;

7. *remote sensing :* detection of man-made objects and estimation of natural resources;

8. *industry :* CAD, CAM, product testing and assembly, inspection, quality control;

9. *domestic systems :* appliances;

1.3 Overview of Genetic Algorithms

Different optimization techniques that are found in the literature can be broadly classified into three categories :

- Calculus based techniques
- Enumerative techniques
- Random techniques

Calculus based techniques use indirect methods, where a set of non-linear equations is solved, or direct methods, which use tools like hill climbing. The main drawbacks of these techniques is that they are local in scope. Moreover, they require the existence of derivatives at each point of the search space, a severe constraint for many real life problems.

Enumerative techniques are conceptually very simple, involving evaluation of every point of the search space. These methods obviously breakdown for very large search spaces. Random search is a technique in which the search space is investigated at random. Obviously, in the long run, random search will do no better than enumerative search [73].

Genetic algorithms (GAs), which are efficient, adaptive and robust search and optimization processes, use the techniques in the last category viz., random choice, as

a tool for guiding the search in very large, complex and multimodal search spaces. GAs are modeled on the principles of natural genetic systems, where the genetic information of each individual or potential solution is encoded in structures called *chromosomes*. They use some domain or problem dependent knowledge for directing the search in more promising areas; this is known as the *fitness function*. Each individual or chromosome has an associated fitness function, which indicates its degree of goodness with respect to the solution it represents. Various biologically inspired operators like *selection, crossover* and *mutation* are applied on the chromosomes to yield potentially better solutions.

Note that the classical gradient search techniques perform efficiently when the problems under consideration satisfy tight constraints. But when the search space is discontinuous, noisy, high dimensional and multimodal, then GAs have been found to consistently outperform both the gradient descent method and various forms of random search [82]. Some of the applications of GAs, so far being made, include pattern classification and feature selection [68, 153, 197], image processing and scene recognition [4, 88, 145, 194], rule generation and classifier systems [30, 92, 93, 94, 95, 97], neural network design [31, 84, 123, 130, 144, 185, 191, 220], scheduling problems [40, 44], VLSI design [216], path planning [39] and the traveling salesman problem [83, 98, 138], graph coloring [46], numerical optimization [127].

The remaining part of this section deals with the principles and features of genetic algorithms, its analysis with respect to schema processing, the theoretical and empirical issues concerning GAs and finally some of its applications, particularly in the area of pattern recognition in decision making. It must be mentioned here that the research articles on integrating GAs with pattern recognition have started to come out recently; consequently the literature in this area is not rich.

1.3.1 Genetic Algorithms: Basic Principles and Features

Genetic algorithms (GAs) [46, 73, 126] are adaptive computational procedures modeled on the mechanics of natural genetic systems. They express their ability by efficiently exploiting the historical information to speculate on new offspring with expected improved performance [73].

As mentioned before, GAs encode the parameters of the search space in structures

called *chromosomes* (or *strings*). They execute iteratively on a set of chromosomes, called *population,* with three basic operators: *selection/reproduction, crossover* and *mutation.* They are different from most of the normal optimization and search procedures in four ways:

- GAs work with the coding of the parameter set, not with the parameter themselves.
- GAs work simultaneously with multiple points, and not a single point.
- GAs search via sampling (a blind search) using only the payoff information.
- GAs search using stochastic operators, not deterministic rules.

Since a GA works simultaneously on a set of coded solutions it has very little chance of getting stuck at a local optimum when used as an optimization technique. Again, the search space need not be continuous, and no auxiliary information, like derivative of the optimizing function, is required. Moreover, the resolution of the possible search space is increased by operating on coded (possible) solutions and not on the solutions themselves.

A schematic diagram of the basic structure of a genetic algorithm is shown in Fig. 1.3. The evolution starts from a set of chromosomes (representing a potential solution set for the function to be optimized) and proceeds from generation to generation through genetic operations. Replacement of an old population with a new one is known as generation (or iteration) when *generational replacement technique* (where all the members of the old population are replaced with the new ones) is used. Another population replacement technique, called *steady state reproduction* may be used, where one or more individuals are replaced at a time, instead of the whole population [46]. GAs require only a suitable objective function, which is a mapping from the chromosomal space to the solution space, in order to evaluate the suitability or *fitness* of the derived solutions.

A GA typically consists of the following components:

- A population of binary strings or coded possible solutions (biologically referred to as *chromosomes*).

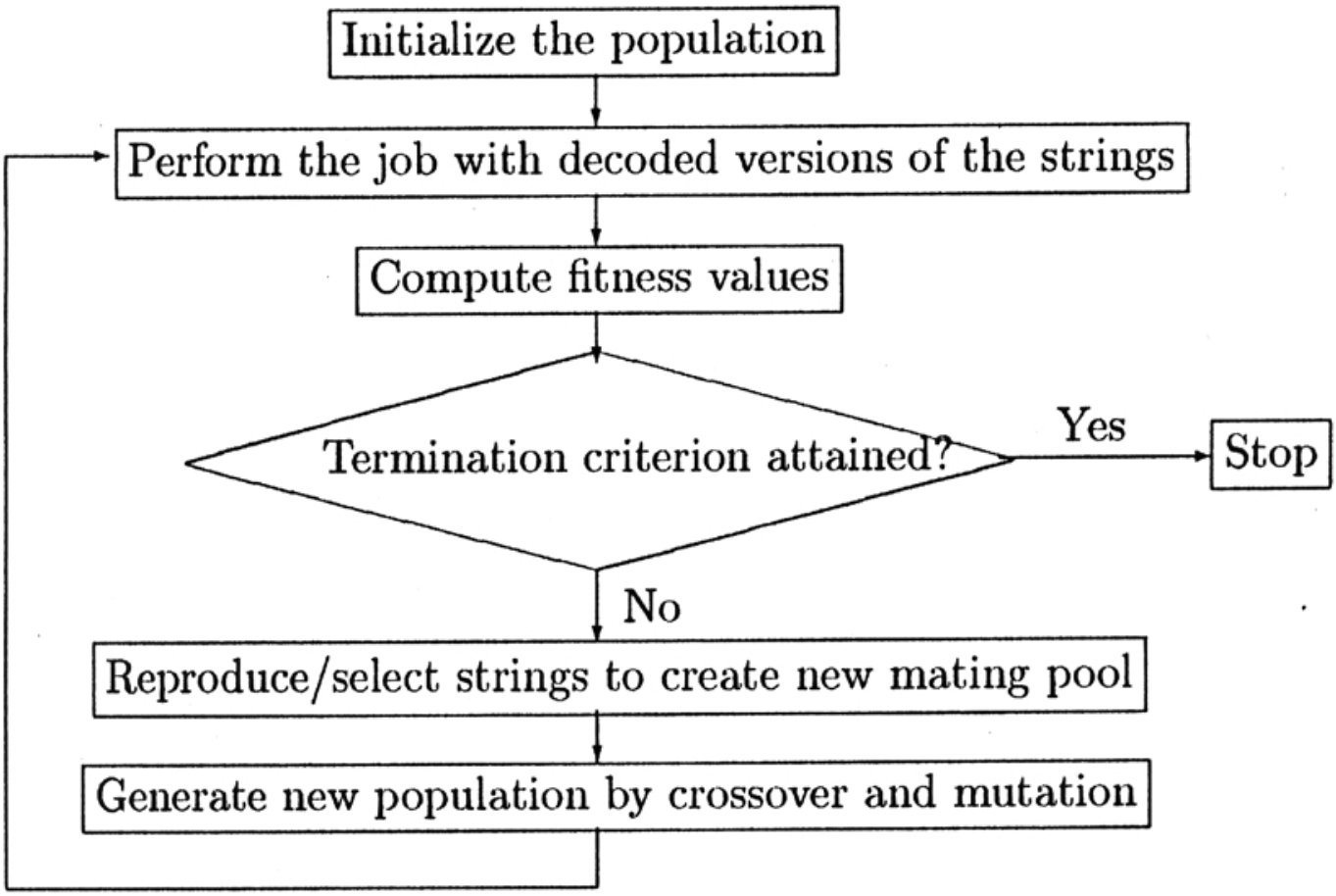


Figure 1.3: Basic steps of a genetic algorithm

- A mechanism to encode a possible solution (mostly as a binary string).
- Objective function and associated fitness evaluation techniques.
- Selection/reproduction procedure.
- Genetic operators (*crossover* and *mutation*).
- Probabilities to perform genetic operations.

These components are now briefly described.

Population: To solve an optimization problem, GAs start with the chromosomal representation of a parameter set. The parameter set is to be coded as a finite length string over an alphabet of finite length. Usually, the chromosomes are strings of 0's and 1's. For example, let $\{a_1, a_2, \ldots, a_p\}$ be a realization of the set of p parameters, and the binary representation of a_1, a_2, ..., a_p be 10110, 00100, ..., 11001, respectively. Then the string

$$10110\ 00100 \ldots 11001$$

is a chromosomal representation of the parameter set. It is evident that the number of different chromosomes (or strings) is 2^l, where l is the string length. Each chromosome actually refers to a coded possible solution. A set of such chromosomes in a generation is called a *population*, the size of which may be constant or may vary from one generation to another. A common practice is to choose the initial population randomly.

Encoding/decoding mechanism: This is one of the primary tasks in GAs. It is the mechanism of converting the parameter values of a possible solution into strings, resulting in the chromosomal representation. If the solution of a problem depends on p parameters and if we want to encode each parameter with a string of length q, then the length, l, of each chromosome will be

$$l = p * q.$$

Decoding is the task of retrieving the parameter values from the chromosomes. It proceeds in a manner that is just the reverse of the encoding process.

One commonly used principle for coding is known as the *principle of minimum alphabet* [73]. It states that for efficient coding, the smallest alphabet set, that permits a natural expression of the problem, should be chosen. In general, it has been found that the binary alphabet offers the maximum number of schemata per bit of information of any coding [73]. Hence, binary encoding is one of the commonly used strategies, although other techniques like floating point coding [51, 126] are also gaining popularity.

Objective function and associated fitness evaluation techniques: The fitness/objective function is chosen depending on the problem to be solved, in such a way that the strings (possible solutions) representing good points in the search space have high fitness values. This is the only information (also known as the payoff information) that GAs use while searching for possible solutions.

Selection/reproduction procedure: The selection/reproduction process copies individual strings (called parent chromosomes) into a tentative new population (known as mating pool) for genetic operations. Number of copies that an individual receives for the next generation is usually taken to be directly proportional to its fitness value; thereby mimicking the natural selection procedure to some extent. This scheme is commonly called the *proportional selection scheme. Roulette wheel parent selection* [73] and *linear selection* [46] are two of the most frequently used selection procedures.

As proved in [181], one problem with *proportional selection* is that this procedure cannot guarantee asymptotic convergence to the global optima. There is no assurance that any improvement made upto a given generation will be retained in future generations. To overcome this, a commonly used strategy known as the *elitist selection* [82] is adopted, thereby providing an *elitist GA* (EGA), where the best chromosome of the current generation in retained in the next generation.

Genetic operators are applied on parent chromosomes and new chromosomes (also called offspring) are generated. The frequently used genetic operators are described below.

Crossover: The main purpose of crossover is to exchange information between randomly selected parent chromosomes by recombining parts of their corresponding strings. It recombines genetic material of two parent chromosomes to produce off-

spring for the next generation. *Single point crossover* is one of the most commonly used schemes. Here, first of all, the members of the reproduced strings in the mating pool are paired at random. Then an integer position k, (known as the crossover point) is selected uniformly at random between 1 and $l-1$, where l is the string length greater than 1. Two new strings are created by swapping all characters from position $(k+1)$ to l. For example, let

$$a = 11000\ 10101\ 01000 \ldots 01111\ 10001$$
$$b = 10001\ 01110\ 11101 \ldots 00110\ 10100$$

be two strings (parents) selected from the mating pool for crossover. Let the randomly generated crossover point be 11 (eleven). Then the newly produced offspring (swapping all characters after position 11) will be

$$a' = 11000\ 10101\ 01101 \ldots 00110\ 10100$$
$$b' = 10001\ 01110\ 11000 \ldots 01111\ 10001.$$

Some other common crossover techniques are the multiple point crossover, shuffle-exchange crossover, uniform crossover [46].

The successful operation of GAs depends a lot on the coding technique, mentioned earlier, used to represent the problem variables [103, 172]. The *building block hypothesis* indicates that GAs work by identifying good building blocks, and by eventually combining them to get larger building blocks [73, 83, 89]. Unless good building blocks are coded tightly, the crossover operation cannot combine them together [76, 77]. Thus coding-crossover interaction is important for the successful operation of GAs. The problem of tight or loose coding of problem variables is largely known as the *linkage problem* [75].

Mutation: The main aim of mutation is to introduce genetic diversity into the population. Sometimes, it helps to regain the information lost in earlier generations. In case of binary representation it negates the bit value and is known as bit mutation. Like natural genetic systems, mutation in GAs is usually performed occasionally. A random bit position of a randomly selected string is replaced by another character from the alphabet. For example, let the third bit of string a, given above, be selected for mutation. Then the transformed string after mutation will be

11100 10101 01000 ... 01111 10001.

High mutation rate can lead the genetic search to a random one. It may change the value of an important bit, and thereby affect the fast convergence to a good solution. Moreover, it may slow down the process of convergence at the final stage of GAs.

Probabilities to perform genetic operations: Both the crossover and mutation operations are performed stochastically. The probability of crossover is chosen in a way so that recombination of potential strings (highly fit chromosomes) increases without any disruption. Generally, the crossover probability lies in between 0.6 to 0.9 [46, 73].

Since mutation occurs occasionally, it is clear that the probability of performing mutation operation will be very low. Typically the value lies between 0.001 to 0.01 [46, 73]. The importance of choosing the mutation probability value properly is discussed in [135, 189]. ♣

As shown in Fig. 1.3, the cycle of selection, crossover and mutation is repeated a number of times till one of the following occurs :

1. the average fitness value of a population becomes more or less constant over a specified number of generations,
2. a desired objective function value is attained by at least one string in the population,
3. the number of generations (or iterations) is greater than some threshold.

1.3.2 Schema Theorem

One of the most fundamental theoretical results of GAs is known as the *Schema Theorem.* A *schema* h is a similarity template describing a subset of strings with similarities at certain string positions. It is composed of 0,1 or the # (don't care) symbols. For example, # 1 # 1 # # 0 # # # is a schema of length 10. This schema will be subsequently referred to as h'. A schema indicates the set of all strings that match it in the positions where the schema has either a 0 or a 1.

The *defining position* of a schema is a position in it which has either a 1 or a 0. *Defining length* of a schema h, denoted by $\delta(h)$, is defined as the distance between the last defining position and the first defining position of the schema and is obtained by subtracting the first defining position from the last defining position. For the schema h' given above, the first defining position (counting from the left) is 2 and the last defining position is 7. Hence $\delta(h')$ = 7-2 = 5. The *order* of a schema h, denoted by $O(h)$, is the number of defining positions in the schema. For the schema h', $O(h')$ = 3.

A schema h_1 is said to be contained in another schema h_2 if for each defining position in h_2, the position is defined in h_1, the defining bit being the same. For example, let h_1 = # 1 1 1 0 1 0 # #, then h_1 is contained in h'. Note that if h_1 is contained in h_2, then $m(h_2, t) \geq m(h_1, t)$ where $m(h, t)$ represents the number of instances of h in the population at time t.

The Schema theorem [73] estimates the lower bound of the number of instances of different schemata at any point of time. According to this theorem, a short-length, low-order, above-average schema will receive exponentially increasing instances in subsequent generations at the expense of below average ones. This is now discussed in details. The notations that will be used for this purpose are explained below :

h	: a short length, low order, above average schema
$m(h,t)$	: m instances of a schema h in a population at generation t for the conventional genetic algorithm
$\delta(h)$	: the defining length of schema h
$O(h)$	: order of schema h
l	: length of a chromosome
$\overline{f_h}$	: average fitness value of schema h
$\overline{f}$	: average fitness value of the population

Let us assume that m instances of a particular schema h exists in the population at time t (denoted as $m(h, t)$). As already mentioned, the number of copies of each string that go into the mating pool is proportional to its fitness in the population. Accordingly, the expected number of instances of the schema h, that go into the

mating pool after selection, $m(h, t+1)$, is given by

$$m(h, t+1) = m(h, t) * \frac{\overline{f_h}}{\overline{f}}. \tag{1.14}$$

Hence above average schemata will grow exponentially, and below average ones will receive decreasing number of samples.

If a crossover site is selected uniformly at random among $(l-1)$ possible sites, a schema h will be destroyed with a probability

$$p_{d1} = \frac{\delta(h)}{(l-1)}.$$

Hence the survival probability (p_s) (when the crossover site falls outside the defining length) is

$$\begin{aligned} p_s &= 1 - p_{d1} \\ &= 1 - \frac{\delta(h)}{(l-1)}. \end{aligned}$$

If μ_c is the crossover probability,

$$p_s \geq 1 - \mu_c * \frac{\delta(h)}{(l-1)}.$$

Moreover, for survival of a schema h, all the fixed positions of h, $(O(h))$, should remain unaltered. If μ_m is the mutation probability, the probability that one fixed position of the schema will remain unaltered is $(1-\mu_m)$. Hence, for $O(h)$ number of fixed positions of a schema h to survive, the survival probability is

$$(1-\mu_m)^{O(h)}.$$

If $\mu_m \ll 1$, the above value becomes $(1 - O(h) * \mu_m)$. Therefore,

$$m(h, t+1) \geq m(h, t) * \frac{\overline{f_h}}{\overline{f}} * (1 - \mu_c * \frac{\delta(h)}{(l-1)}) * (1 - \mu_m * O(h)). \tag{1.15}$$

Neglecting the smaller term, we have

$$m(h, t+1) \geq m(h, t) * \frac{\overline{f_h}}{\overline{f}} * (1 - \mu_c * \frac{\delta(h)}{(l-1)} - \mu_m * O(h)). \tag{1.16}$$

Eq. (1.16) indicates that short (small $\delta(h)$), low order (small $O(h)$) and above average ($\overline{f_h} > \overline{f}$) schemata will receive exponentially increasing instances in subsequent generations [73, 89].

1.3.3 Some Issues in Genetic Algorithms

In Section 1.3.2 we have described the pioneering work of Holland on schema analysis. In recent years, a large amount of research interest has focussed on the development of different aspects of GAs. These include improving its theoretical foundation, examining various techniques for improving its performance, parallelizing the implementation and finding various application areas. In this section, a brief discussion on these issues is presented.

Theoretical Aspects

Premature convergence of GAs is one of the frequently studied problems [46, 50, 162]. This occurs when the population becomes homogeneous, and crossover can no longer produce more fit offspring. One of the main reasons for premature convergence in GAs is due to the selection pressure of proportional selection scheme and emphasis on crossover. In [46] (pp. 26-27) it has been suggested that under such conditions, either the GA should be restarted with a new seed, or hill-climbing heuristics should be employed to search for improvements. A new method termed *dynamic parameter encoding* has been suggested by Schraudolph and Belew [193], which dynamically resizes the available range for each parameter. On detection of premature convergence, the maximum and minimum values of the range are resized to a smaller window and the process is iterated.

Several other researchers have analyzed GAs by modeling it as states of a Markov chain [28, 47, 79, 90, 207, 214]. Goldberg and Segrest [79] assume proportional selection without mutation. Since single bit chromosomes are considered in [79], for a population size N, there are $N + 1$ possible states i, where i is the population with exactly i ones and $(N - i)$ zeros. An $(N + 1) \times (N + 1)$ transition matrix $P[i, j]$ is defined which maps the current state i to the next state j. The major focus has been the study of the pure genetic drift of proportional selection using the transition probability expression, where they investigated the expected times to absorption for the drift case. Horn [90] extended this work by modeling a niched GA using finite, discrete time Markov chains, limiting the niching operator to *fitness sharing*, introduced in [78].

Vose and Liepins [214] studied the transient and asymptotic behavior of the genetic algorithm by modeling it as a Markov chain. In [47], Davis and Principe attempted to provide a Markov chain model and an accompanying convergence theory for the simple genetic algorithm by extrapolating the existing theoretical foundation of the simulated annealing algorithm [45, 107, 212]. Rudolph has also proved the convergence of elitist GAs to the optimal string [181].

The issue of convergence of GAs to the globally optimal solution has also been pursued in [28], where GAs are again modeled as Markov chains having a finite number of states. A state is represented by a population together with a potential string. Irrespective of the choice of initial population, GAs have been proved to converge to the optimal string for infinite number of iterations, provided the conventional mutation operation is incorporated. Murthy et al. [135] have provided a stopping criterion, called ϵ-optimal stopping time, for the elitist model of the GAs. Subsequently, they have derived the ϵ-optimal stopping time for GAs with elitism under a 'practically valid assumption'.

Suzuki [207] presented a Markov chain analysis of GAs using a modified elitist strategy, by considering selection, crossover and mutation for generational change. By evaluating the eigen values of the transition matrix of the Markov chain, the convergence rate of GAs has been computed in terms of the mutation probability μ_m. It is shown that the probability of the population including the individual with the highest fitness value is lower-bounded by $1-O(|\lambda_*|^n)$, $|\lambda_*| < 1$, where n is the number of generation changes and λ_* is a specified eigen value of the transition matrix.

In [215], Vose and Wright surveyed a number of recent developments concerning the simple genetic algorithms, its formalism, and the application of the Walsh transformation to the theory of genetic algorithms. It is mainly concerned with the mathematical formalization of the simple genetic algorithm, and how the Walsh transform appertains to its theoretical development.

Some of the theoretical studies concerning deceptive problems in GAs, i.e., the class of problems which mislead the GA away from the globally optimal solution, are reported in [24, 72]. Bethke [24] used Walsh function analysis of schema averages for examining some efficient conditions for convergence of simple GAs. He quantified the deception required which would lead a GA away from the global optimum. He

demonstrated that in deceptive problems, good, low-order, short schemata must mix and match to provide bad strings. Goldberg [72] has shown that even when the problems are intentionally made hard or misleading, simple GAs usually succeeded in finding the global optimal solution. An extended schema analysis has been provided in this regard. Similar investigations with respect to deceptive problems are also presented in [116, 221].

Empirical Aspects

One important issue with respect to the effectiveness of genetic algorithms is the population size. The result of Holland [89] in this regard states that the number of schemata processed effectively is proportional to the cube of the population size. However, a study by Goldberg [74] suggests that this result should not lead one to assume very large population sizes. He has, in fact, indicated that relatively small populations are appropriate for serial implementations of GAs, while large populations are appropriate for perfectly parallel implementations. Regarding the initialization of population, there have been a few attempts. These include *random initialization* [73], where the population is initialized randomly, *extended random initialization* [33], where each member of the initial population is chosen as the best of a number of randomly chosen individuals.

In practical implementations of GAs, it is often necessary to scale the fitness values in order to keep appropriate levels of competition throughout the generations. Different mechanisms for scaling the fitness values are linear scaling, sigma truncation, power law scaling [73]. An attempt to incorporate the ancestors influence into the fitness of individual chromosomes has recently been made in [49]. This is based on the observations in nature where an individual is not an independent entity, but is highly influenced by the environment. Ghosh et al. [71] have incorporated the concept of aging of individuals for measuring their suitability for participation in genetic operations, by combining both the functional value and the age of an individual for computing its effective fitness. Results have shown that this scheme provides enhanced performance and maintains more diversity in the population.

Several alternate selection schemes have been suggested in the literature for reducing the stochastic errors associated with roulette wheel selection [9, 73]. Some of these are

deterministic sampling, remainder stochastic sampling without replacement, stochastic sampling without replacement, remainder stochastic sampling with replacement, stochastic sampling with replacement, stochastic tournament selection, ranking, etc. A study where the selection scheme itself is made adaptive can be found in [9]. Recently, a selection method called *disruptive selection*, has been proposed in [113]. This method adopts a nonmonotonic fitness function and favors both superior and inferior individuals. It has been shown here that this scheme can be used to solve nonstationary search problems, where conventional GAs fare poorly.

There have been several investigations on developing various crossover operators and studying their interaction with some other operators. In order to improve the performance, Syswerda [208] introduced a new crossover operator called *uniform crossover*, where two offspring are generated from two parents by selecting each component from either parent with a given probability. The comparative utility of single point, two point and uniform crossover operators is also investigated, where uniform crossover is found to provide a generally superior performance. This has been further confirmed by an analysis in [202], where a theoretical framework for understanding the virtues of uniform crossover is discussed. A new dynamic, knowledge-based, nonuniform crossover technique has been proposed in [118], which generalizes the uniform crossover, and constantly updates the knowledge extracted from the environment's feedback on previously generated chromosomes. Some additional information in this line are available in [52, 200, 201].

To sustain diversity (which may be lost due to crossover and very low mutation rate) in the population, Whitley and Starkweather [219] proposed a technique called *adaptive mutation*, where instead of fixed mutation rate the probability to perform mutation operation increases with increase of genetic homogeneity in the population. Bhandari et al. [29] have proposed a new mutation operator known as *directed mutation* which follows from the concept of induced mutation in biological systems [133]. This operation uses the information acquired in the previous generations rather than probabilistic decision rules. In certain environments, directed mutation will deterministically introduce a new point in the population. The new point is directed (guided) by the solutions obtained earlier, and therefore the technique is called *directed mutation*. Recently Ghannadian et al. suggested the use of random restart in place of the mutation operation in [70]. The expected time for the method to

reach an optimal solution is also derived using the Markov chain method.

Schaffer and Eshelman [192] compared crossover and mutation on binary function optimization problems. They observed that crossover generally dominates the population, which may not be necessarily beneficial to the problem. The results indicated that the utility of specific crossover and mutation operations is problem dependent.

In [82], a meta level GA is used to set the control parameters in order to search for an optimal GA for a given set of numerical optimization problems. A study regarding the optimal mutation rates has been presented in [7]. In another investigation to improve the performance of GAs, concepts of adaptive probabilities of crossover and mutation based on various fitness values of the population are recommended [205]. Here, high solutions are protected and the subaverage solutions are completely disrupted. Besides these, some other techniques for appropriate control parameter selection and investigations regarding the interaction between different operators can be found in [6, 126, 202].

A non traditional genetic algorithm, called CHC, which combines a conservative selection strategy, that always preserves the best individuals found so far, with a highly disruptive recombination operator, that produces offspring which are maximally different from both the parents, has been developed by Eshelman [57]. It incorporates an incest prevention technique where individuals are randomly paired for mating, but are only mated if their Hamming distance is above a certain threshold. Another version of genetic algorithm, known as delta coding, where the diversity of the population is constantly monitored, and the algorithm restarted when it show signs of losing diversity, is described in [222].

Recently, Mathias et al. [125] compared several types of genetic algorithms against a mutation driven stochastic hill climbing algorithm, on a standard set of benchmark functions which had Gaussian noise added to them. The genetic algorithms used in these comparisons include *elitist simple genetic algorithm*, *CHC adaptive search algorithm* [57] and *delta coding genetic algorithm* [222], where *CHC* is found to perform better than the others. One reason for this, as pointed out in [125], is that as noise component of the objective functions becomes a significant factor, the algorithms that sample the immediate neighborhood of solutions in the current populations may have an advantage in finding the global optimum.

Messy genetic algorithms (mGAs) [76, 77], another development in the area of GAs, use variable length strings that may be under or overspecified with respect to the problem being solved. mGAs divide the genetic processing into two distinct phases, a primordial phase and a juxtapositional phase. In the primordial phase, the population is first initialized to contain all possible building blocks of a specific length, where the characteristic length is chosen to encompass possibly misleading building blocks. Thereafter, the portion of good building blocks is enriched through a number of generations of reproduction only. At the same time, the population size is usually reduced by halving the number of individuals in the population at specified intervals. Juxtapositional phase proceeds with a fixed population size and the invocation of reproduction, cut and splice, and other genetic operators. It has been found that mGAs solve problems of bounded deception to global optimality in a time that grows no faster than a *polynomial function of the number of decision variables on a serial machine.*

Inspired by mGAs, Kargupta developed the gene expression messy genetic algorithm (GEMGA) which explores the computational role of gene expression (or, DNA → RNA → Protein transformation) in evolutionary linkage learning [102]. This version with quadratic sample complexity is further improved by Bandyopadhyay et al. in [10] to provide linear sample complexity.

Besides these, there are several operators, theories and techniques which have been developed in order to improve the performance of GAs. Some of these are dominance, diploidy, inversion, reordering, niching [73].

Recently, the limitation of uniprocessor computing systems and the increase in availability of multiprocessors have led to investigations of parallel genetic algorithms (PGAs). In PGAs, a single large population is divided into smaller sub populations, where each subpopulation evolves independently on different processors. While obtaining higher speed-up using multiple processors is one motivation for this division of populations, the other and more interesting motivation is that in sequential GAs the diversity may soon be lost because of selection pressure. After some generations, the majority of the chromosomes in a single population become very similar, and crossover thereafter may not be productive. However, if a number of independent populations evolves simultaneously, they will most likely search different areas of the search space. Occasional interaction between subpopulations will allow exchange of

information among widely different information carrying groups. Depending on the amount of parallelism, and the level of interaction, PGAs may be broadly classified into coarse-grain and fine-grain. Their details can be found in [114, 124, 131, 132].

1.3.4 Applications of Genetic Algorithms

As mentioned before, GAs are gradually finding applications in various business, scientific and engineering circles. In this section, a review of some of its applications, emphasizing those on pattern recognition, has been presented. These applications include design of classifier systems and knowledge based systems, automatic determination of neural network architecture, and development of image processing and pattern classification methodologies. (Note that the investigations regarding the utility of GAs for pattern recognition problems are relatively new, and the research articles in this area have started to come out [68, 153].)

A classifier system with linguistic if-then rules is designed in [92], where genetic algorithms are used to optimize the number of rules and to learn the grade of certainty associated with each rule. Other applications of GAs for developing classifier systems can be found in [30, 93, 94, 95, 115, 177]. Janikow [97] utilized GAs for optimizing the fuzzy knowledge components of a fuzzy decision tree both prior to and during its construction. The suggested method also finds application in optimizing fuzzy rule based systems. An approach of integrated genetic learning in construction of fuzzy relational structures is suggested in [159]. Another application where a GA is used for auto-fuzzy tuning in fuzzy neural network is described in [96]. In [196], methods of self tuning fuzzy modeling based on GAs are proposed. Several issues of integration of fuzzy sets with evolutionary computation are addressed recently in [160].

In [66], self adaptive GAs are used for training a recurrent neural network, which is then used for learning the best among the several stability configurations for biped locomotion robot. Integration of GAs with neural networks has recently been investigated for the problem of vehicle routing [163]. A method for the automatic generation of fuzzy controller using GAs is described in [41]. A new encoding scheme for the rules is also suggested, which results in a more compact rule base. Ishibuchi et al. have used GAs for selecting a small number of significant fuzzy if-then rules in [95].

In [110], an evolutionary concept is proposed which aims at generating rules that are interpretable and represent a relevant aspect of the system behaviour or of the control strategy, with acceptable calculation times. Recently, Xue et al. [224] have also studied the application of genetic algorithms for optimization of fuzzy systems in prediction and control.

An approach to develop general heuristics, for solving problems in knowledge-lean environments using GAs, is recently developed in [216]. The generalization is attempted over the problem space which is not seen during learning. The method described in [216] uses a new statistical measure called probability of win, which assesses the performance of heuristics in a distribution independent manner. To validate the proposed method, experimental results on generalizing heuristics learned for sequential circuit testing, VLSI cell placement and routing, and branch and bound search have been demonstrated.

Dev and Ram Murthy [54] presented a GA solution to the problem of optimal assignment of production rules in a knowledge base to a variable number of partitions. Partitioning of a knowledge base is an important problem since an efficient solution can increase its performance both in the compilation and the execution phase. The authors demonstrated that the solutions obtained using GAs compare favorably with those obtained with a clustering algorithm. There have been many attempts [153] for learning rules of classification. Some of them use decision trees [34, 171, 174]. Recently, a procedure has been suggested by integrating genetic algorithms with similarity based learning for acquiring rules for classification [198].

Methods for designing architectures of neural networks using GAs have also been reported. These are primarily divided into two parts: in one part GAs replace the learning to find appropriate connection weights of some predefined architecture. In another part, GAs themselves are used to find the architecture (connectivity) and it is evaluated using some learning algorithm.

Recently, Romanuik [178] has described an approach for the automatic construction of neural networks for pattern classification. It utilizes GAs to locally train network features using the perceptron rule. Transdimensional learning is introduced which can automatically adjust the learning bias inherent to all learning systems. Whitley et al. [220] suggested a way to apply GAs to neural network problems and they

showed how genetic search can be improved to achieve more accurate and consistent performance. The method is able to optimize the weighted connections for small neural network problems.

In [144], Pal and Bhandari incorporated GAs to find out the optimal set of weights (biases) in a layered network. Weighted mean square error over the training examples has been used as the fitness measure. They introduced a new concept of selection, called *non-linear selection,* which enhances genetic homogeneity of the population and speeds up searching. *Implementation results on both linearly separable and* non-linearly separable pattern sets are also reported.

Harp and Samad suggested an approach [84] to evolve the architecture of neural networks and their learning parameters using GAs. The network is trained separately by backpropagation algorithm. Each chromosome contains two parts: 1) *fixed length area parameter specification* which corresponds to the layer number, the number of units in it, how the units are organized, learning parameters and threshold values; 2) *projection specification field* which shows the way of connection between layers. Since the total number of layers may vary, the chromosomal length is kept variable. They also introduced a modified crossover operator which exchanges homologous segments of two chromosomes (representing two networks). Architecture, connection strengths and thresholds of individual neurons of feedforward neural networks have been simultaneously found out in another investigation [185], without using any error propagation algorithm.

An attempt has also been made for evolving architectures of Hopfield type optimum neural networks for extracting object regions from gray images using GAs [146], where each binary chromosome represents a network architecture. The presence (or absence) of connectivity between neurons is represented by 1 (or 0). The proposed GA based technique has been able to evolve network architectures whose connectivity is about two-third of the requirement of the corresponding fixed fully connected ones in order to produce comparable segmented output. The optimized networks have been found to be more noise independent. Some recent applications of GAs for determination of the architectures of neural networks are presented in [104, 186, 187].

Application of GAs to image processing problems is described in [91, 145]. A method for determining the optimal enhancement operator for both bimodal and multimodal

images is described by Pal et al. in [145]. The algorithm does not need iterative visual interaction and prior knowledge of image statistics for this purpose. The fuzziness measures are used as fitness function. A modified scheme of chromosome encoding is suggested in [91], which utilizes trees and two dimensional bit maps as its structures. This is useful in the framework of image processing, where ordinary genetic algorithms using one dimensional strings cannot be applied in the natural way because of the loss of two dimensional correlation in the process. In [139], the authors have applied GAs for partial shape matching, a challenging task when the image contains overlapping, noisy, occluded and partial shapes. For this purpose, they have used the attributed strings [210] for representing outline features of shapes. Another application of GAs to automatic scene labeling of remote sensing images, where the domain constraints are represented as semantic nets, is presented in [36].

An early attempt towards using GAs for pattern recognition and subroutine selection problems is reported by Cavicchio [37], who stored the detectors from known images, along with the associated class in the training phase. In the testing phase, the detectors of the unknown image are compared to the stored ones, and an appropriate classification is performed. The detectors are encoded as genes of a chromosome, and conventional genetic operators are applied with several variants of the mutation operation.

Kelly and Davis [105], made an attempt to hybridize the k-NN classification algorithm with GAs. The k-NN algorithm uses the training data set to compute the classification of the test data set based on a metric of nearness or similarity. Ideally, it assigns equal importance to all the attributes. But in practice, this should not be the case. Consequently, a weighting factor is assigned to each attribute which determines its contribution to the overall similarity measure. In addition, the neighbors are assigned weights, which control their effect in determining the class of an unknown data point. Obviously, the nearest neighbor is given more importance and so on. GA is used to determine the weights, thereby providing the GA-WKNN (weighted k-NN) algorithm. In order to make the performance of the GA-WKNN invariant to rotation of the data set, they proposed the GA-RWKNN algorithm [106], where GAs are again employed to search for a suitable rotation and weight vectors. The algorithm GA-RWKNN (rotated weighted k-NN), is found to significantly outperform both k-NN and ID3 [171] algorithms for two data sets.

Application of GAs for editing the set of prototypes for the k-NN rule, thereby providing a reference set, is addressed by Kuncheva [111]. In a similar fashion, GAs have been applied for selecting the initial seed points for a Radial Basis Function classifier [112]. Another attempt in this direction has been recently made by Zhao and Higuchi [227], where an evolutionary algorithm, called individual evolutionary algorithm (IEA) (which uses four basic operations viz., competition, gain, loss and retraining), is used for reducing the set of prototypes for the k-NN rule. IEA has also been used by them for obtaining reduced NN-MLP (a nearest neighbor based MLP) in terms of the number of neurons [226].

Srikanth et al. [204] described a genetic algorithmic approach to pattern classification, both crisp and fuzzy, where clusters in pattern space are approximated by ellipses or sets of ellipses in two dimensions and ellipsoids in general. A variable number of ellipsoids is searched for, which collectively classify a set of objects. Since the number of ellipsoids is not fixed a priori, a variable length chromosome is considered. The performance of the classifier is illustrated using two two-dimensional data sets. Performance on a four-dimensional data set is compared with those of neural networks for several architectures. The comparison is marginally in favor of the genetic technique presented. (Note that, at about the same time, our group also started working on similar lines, where the boundaries between classes are approximated using a number of hyperplanes. Interestingly, both the investigations, which proceeded in parallel, are reported in the same issue of a journal [68].)

Murthy and Chowdhury [136] have used GAs for finding optimal clusters, without the need for searching all possible clusters. The experimental results show that the GA based scheme may improve the final output of the K-means algorithm [209], where an improvement is possible. Another application of GA for clustering, feature selection and classification is reported in [211]. Recently, Sarkar et al. [188] have proposed a clustering algorithm using evolutionary programming-based approach, when the clusters are crisp and spherical. The algorithm automatically determines the number and the center of the clusters, and is not critically dependent on the choice of the initial cluster.

Piper has described [161] the use of GAs in chromosome classification. The fitness function takes into account constraints imposed by the context of a metaphase cell, as well as similarity of homologues. Comparison with previous classification methods

demonstrates equal classification accuracy. Incorporation of the homologue similarity constraint does not substantially improve the error rate.

Selection of a subset of principal components for classification using GAs is made in [164]. Since the search space depends on the product of the number of classes and the number of original features, this selection process by conventional means may be very computationally expensive. Results on two data sets with small and large cardinalates are presented.

A recent application of GAs for searching for error correcting graph isomorphism, an important issue in pattern recognition problems, is described in [218].

Besides these, several other applications of GAs for solving a variety of problems arising in industry are discussed in [213, 223]. Several hybrid genetic algorithms have been developed with an application on geophysical static corrections of noisy seismic data [125]. The area of genetic programming, for evolving computer programs automatically is also gaining popularity [109].

1.4 Scope of the Thesis

The present thesis deals with the development of a pattern classifier in $\mathbb{R}^N$ by utilizing the characteristics of GAs for searching for appropriate class boundaries, analyzing some of its theoretical aspects and demonstrating its effectiveness on both artificial and real life data sets. The basic philosophy of the methodology is that the decision boundaries between the pattern classes can be modeled using a number of surfaces of first or higher orders. Both fixed and variable string length GAs are used to encode the number of surfaces for constituting the boundaries. A theoretical analysis showing the relation with Bayes classifier under limiting conditions is provided. The investigation also includes incorporation of chromosome differentiation in GAs, a phenomenon commonly found in nature, for performing the crossover operation. An analogy of the classification methodology with that of MLP is found; thereby deriving the MLP architecture automatically. The effectiveness of the methodologies is demonstrated on some artificially generated pattern sets, as well as some real life problems such as speech recognition, cancer detection and satellite image classification, with the number of dimensions varying over a wide range of two to nine. The

results of investigation are summarized below on the basis of chapter headings.

1.4.1 *GA-Classifier* : Modeling the Class Boundaries using Fixed Number of Hyperplanes [11, 13, 143]

First of all, a method has been developed in Chapter 2 where the class boundaries of any given data set in $\mathbb{R}^2$ are modeled using a fixed number, say H, of lines. Two parameters viz., an angle value and a perpendicular distance value uniquely specify a line. The parameters of the H lines are encoded in binary strings of chromosomes of GAs, which are initially generated randomly. The fitness function is defined in terms of the number of points correctly classified by the arrangement of lines encoded in a chromosome. Conventional operators like selection, crossover and mutation are applied on the population of chromosomes over a number of generations till a termination criterion is achieved. The best string seen till the last generation is decoded to provide the lines constituting the final decision boundary.

This classifier is then extended to $\mathbb{R}^N$, where the decision boundaries are modeled using a fixed number (H) of hyperplanes. Since an *a priori* knowledge of a proper value of H is difficult to ascertain, it is usually overestimated; thereby giving rise to redundant hyperplanes in the final decision boundary. A scheme for automatic detection and elimination of such redundant hyperplanes is also provided here.

The aforesaid GA based classifier in $\mathbb{R}^N$ incorporating a scheme for redundancy elimination is termed as the *GA-classifier*. In a part of the experiment, the classifier is implemented with hyperspherical surfaces, instead of hyperplanes, for approximating the class boundaries. An extensive comparison of the performance of the *GA-classifier* with other conventional classifiers e.g., those based on k-NN rule, Bayes maximum likelihood ratio and MLP, is adequately demonstrated for two types of two dimensional artificial data, three dimensional speech data, four dimensional Iris data, and nine dimensional cancer data.

Note that the characteristic feature of this *GA-classifier* is that it explicitly utilizes the class boundaries for making decisions. For the other conventional classifiers, the class boundaries are generated as a consequence of the decision making process.

1.4.2 Theoretical Analysis of the *GA-classifier* [12, 14, 134]

In Chapter 3, an investigation is carried out to formulate some theoretical results regarding the behavior of the *GA-classifier*, developed in the previous chapter, for a sufficiently large number of training data points n, in an N dimensional space $\mathbb{R}^N$. This includes establishing the relationship between the Bayes classifier and *GA-classifier* under limiting conditions. (In this context one may note that in the statistical framework, the Bayes classifier is known to be the optimal one, when the class distributions and the *a priori* probabilities are known. Thus a desirable characteristic of any classifier is that its performance should approach that of the Bayes classifier under limiting conditions.)

It is proved in this chapter that for $n \rightarrow \infty$ (i.e., for sufficiently large training data set) and for a sufficiently large number of iterations, the performance of the *GA-classifier* during training approaches that of the Bayes classifier. Extensive experimental results on overlapping data sets following triangular and normal distributions with both linear and non-linear class boundaries are provided that conform to this claim. The claim also holds good when circular surfaces are considered as constituting elements/segments of boundaries. It is observed that as n increases, the performance of the *GA-classifier* during both training and testing approach those of the Bayes classifier.

It is shown that the optimum number of hyperplanes generated (after redundancy removal) by the *GA-classifier* is equal to that required to model the Bayes decision boundary when there exists only one partition of the feature space that provides the Bayes error probability. The variation of recognition score with *a priori* class probability is also shown to be similar for both the classifiers.

1.4.3 Using Variable String Lengths in *GA-classifier* [15, 19]

It is mentioned in Section 1.4.1 that an *a priori* knowledge of a proper value of H is difficult to estimate. Thus, it is usually overestimated, leading to the problem of overfitting of the data along with an associated reduction in the generalization capability of the *GA-classifier*. Additionally, it often results in the presence of redundant surfaces in the final decision boundary.

In order to overcome these limitations, the concept of variable string lengths [76] in GAs (termed VGAs) has been used in Chapter 4, for the problem of constructing class boundaries through the placement of a variable number of hyperplanes, for the purpose of classifying patterns in N dimensional feature space. The parameters of a number of hyperplanes, whose value may now vary, are encoded in a chromosome or string. Newly defined genetic operators are applied on these strings over a number of generations till a termination criterion is attained. The fitness function is defined in such a way that its maximization ensures, primarily, the minimization of the number of misclassified samples, as also the reduction of the number of hyperplanes for doing so. The *GA-classifier* utilizing variable string lengths is called the *VGA-classifier*.

As in Chapter 3, it has been theoretically established here that as the number of iterations and the size of the training data go to infinity, the performance of the classifier will approach that of the Bayes classifier. At the same time, the number of hyperplanes provided by the classifier will be the minimum.

Investigations into the effectiveness of the aforesaid *VGA-classifier*, for two sets of artificial data, speech data, Iris data, and cancer data, demonstrate that the concept of VGA, besides being able to evolve automatically the number of hyperplanes from a given range, also helps in improving its recognition capability. Moreover, empirical results on some artificially generated data sets are provided that corroborate the theoretical findings. It is demonstrated that the *VGA-classifier* is useful in approximating automatically any Bayes decision boundary (linear, non-linear) for sufficiently large size of training data and sufficiently large number of iterations.

1.4.4 Incorporation of Chromosome Differentiation in GAs (GACD) [16, 18]

The effect of incorporating some common forms of differentiation as found in natural systems, (e.g., sexual differentiation), in GAs is investigated in Chapter 5. A genetic algorithmic methodology, termed GACD, is described, which incorporates chromosome differentiation for evolutionary process.

In GACD, chromosomes are distinguished into two categories of population, termed the M and F populations, based on the value contained in the two class bits. Initially,

the two populations are of the same size, and are generated based on maximum hamming distance between them. Crossover (mating) is allowed only between individuals belonging to these categories. As a results of crossover, two offspring are created, whose classes are determined stochastically, depending on the parent chromosomes. The offspring are accordingly put into the different populations. Thus the number of chromosomes in the individual populations may vary, although the total number is kept constant.

Schema analysis of GACD, presented in Chapter 5, shows that the basic tenet of genetic algorithms holds for GACD as well; above average, short, low order schema will receive increasing number of trials (or instances) in subsequent generations. It is also shown that in certain situations, the lower bound of the number of instances of a schema sampled by GACD is greater than or equal to that of the conventional genetic algorithm.

Implementation of the *GA-classifier* by using GACD instead of GA provides the *GACD-classifier*. Its subsequent application to the aforesaid two artificial, Iris, speech and cancer data sets is found to generate better recognition scores. The number of iterations required to attain the termination criterion is also found to be less than that required by conventional GA. Moreover, in order to establish the superiority of GACD, experimental results on a large number of function optimization problems are presented in Appendix D, which demonstrate the significantly better performance of GACD over the conventional ones. It is to be noted that the concept of GACD enriches the GA literature in general, and that of the classifier in particular.

1.4.5 Relation Between *VGA-classifier* and MLP : Determination of Network Architecture [17]

Chapter 6 describes an analogy between the principles of *VGA-classifier*, developed in Chapter 4, and multilayer perceptron (MLP) (with hard limiters) based classifier. Both can model the class boundaries using a number of hyperplanes. Additionally, it is established in Chapter 4 that number of hyperplanes provided by the *VGA-classifier* for constituting the decision boundary will be the minimum. Based on these, a method called the *network construction algorithm* (NCA) for determining the MLP architecture automatically is described. It is shown that the architecture

would need atmost two hidden layers, the neurons of which are responsible for generating hyperplanes and regions.

The architecture derived using NCA has the following specifications. The input layer has N neurons, where N is the dimensionality of the feature space. These simply transmit the input signals to their outputs. The first hidden layer has H_{VGA} neurons, where H_{VGA} is the number of hyperplanes provided by the *VGA-classifier*. These are responsible for generating the equations of the H_{VGA} hyperplanes. The neurons in the second hidden and output layers perform the AND & OR functions respectively. The NCA also includes a post processing step which automatically removes any redundant neuron in the hidden/output layer.

Comparison of the MLP thus derived using NCA, with its more conventional counterparts, is extensively demonstrated for different data sets. An advantage of this investigation is that the said analogy between the *VGA-classifier* and MLP will augment the application domain of *VGA-classifier* to those areas where MLP has widespread use.

1.4.6 Application of Genetic Classifiers on Satellite Imagery

The previous chapters describe a genetic algorithm based pattern classification methodology where both fixed and variable length chromosomes are considered for constituting the decision boundary in $\mathbb{R}^N$, along with its theoretical aspects and analogy with the MLP based classification. The effect of incorporating chromosome differentiation in genetic algorithms for performing restricted crossover operations is investigated. The effectiveness of these algorithms is demonstrated extensively on several artificial and real life data sets.

Chapter 7 describes another real life application of these genetic classifiers in a different domain, viz., classification of satellite images for identifying various landcover types. This analysis has two parts. In the first part, *LANDSAT* data is considered, which is characterized by numerical feature vectors (two principal components constituting the feature space) for the classification of *Manda Granite, Romapahari Granite, Vegetation, Black Phillite* and *Alluvium.* Extensive comparison of the *GA-classifier, VGA-classifier* and *GACD-classifier* with those based on k-NN rule, Bayes maximum likelihood ratio and MLP for this data set conforms to the

findings made in the previous chapters viz., the *GACD-classifier* performs best in terms of the recognition scores, all the genetic classifiers outperform the others, and the *VGA-classifier* is able to reduce the number of hyperplanes significantly while retaining good performance.

In the second part of the investigation, we considered the problem of pixel classification from *SPOT* image of a part of the city Calcutta for segmenting different classes, namely, *pure water, turbid water, concrete, vegetation, habitation, open space* and *roads* (including bridges). Here we have compared the performance of the *GA-classifier* with those of k-NN classifier, Bayes maximum likelihood classifier and a recently developed approach based on a multivalued (fuzzy) recognition system [120, 121], for different values of H and k.

1.4.7 Conclusions and Scope for Further Research

The concluding remarks with further scope for research are presented in Chapter 8.

Chapter 2

GA-Classifier : Modeling the Class Boundaries Using Fixed Number of Hyperplanes

2.1 Introduction

Pattern classification can be viewed as 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 non-linear. In this chapter we describe a classifier where the characteristics of GAs are exploited in searching for a number of linear segments which can approximate the non-linear boundaries while providing minimum misclassification of training sample points. Such a classifier, designed using GA, is subsequently referred to as the *GA-classifier*.

The feature space is generally unbounded and continuous in nature. However, GAs can only work in finite and discrete domains. Therefore, 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 GAs 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.

The classifier is first developed for the two dimensional case, where the class boundaries are approximated using a fixed number of lines. Subsequently, it is generalized to N dimensions, for $N \geq 2$, where the decision boundaries are modeled using hyperplanes. Note that this is not a straight forward extension of the two dimensional case.

Since the optimum number of hyperplanes, H, required for proper classification of a given data set is not known *a priori*, a conservative value (or overestimated value) of H is normally used 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 towards 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 non-overlapping, non-convex

regions is extensively demonstrated on two sets of artificial data, Iris data, a speech data and cancer data, with the number of dimensions ranging from two to nine. The generalization ability of the decision boundary provided by the *GA-classifier* and the issue of removing redundancy are explained pictorially. The results are compared with those of Bayes maximum likelihood classifier (which is known to provide optimal performance from a statistical viewpoint, when the *a priori* probabilities and class conditional densities are known exactly), k-NN classifier and MLP (where both the k-NN classifier and the MLP with hard limiters are known to provide piecewise linear boundaries). These existing classifiers are described in Chapter 1.

Sections 2.2 and 2.3 describe the classification methodology in two dimensions and the implementation results, including the comparative performance of the *GA-classifier* with those of the above mentioned existing classifiers, respectively. The generalization of the classifier to N, $N \geq 2$ dimensions is presented in Section 2.4. In Section 2.5, a method for the automatic detection and removal of redundant hyperplanes is presented. Its implementation and comparative results are described in Section 2.6. An extension of the entire methodology to the case where higher order surfaces are considered is made in Section 2.7, where hyperspherical segments are considered for modeling the class boundaries. Finally the discussion and the conclusions are presented in section 2.8.

2.2 Description of *GA-classifier* in IR^2

We consider a fixed number of lines (say H) to denote a decision boundary in a two dimensional feature space $X_1 - X_2$. The value of H varies from problem to problem, depending on the number as well as the nature of the classes. These H lines need to be encoded as a single string. Note that each line provides two halfspaces - a positive halfspace and a negative halfspace, thereby yielding two regions. For H lines, the maximum number of such regions is 2^H. If the number of classes in the data set be k, then H must be chosen such that $2^H \geq k$, or $H \geq \lceil \log_2 k \rceil$.

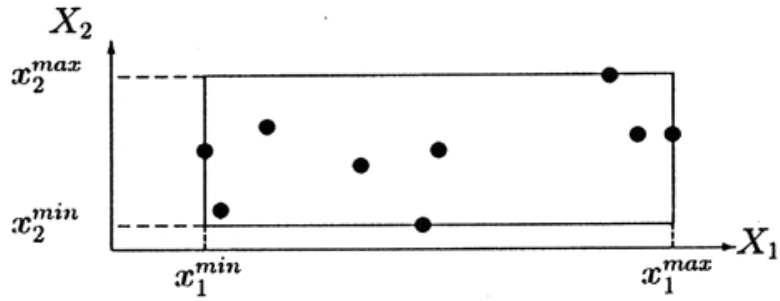


Figure 2.1: Training patterns and the enclosing rectangle

2.2.1 Chromosome Representation

As already mentioned in Section 1.3.1, the primary task in GAs is to encode the parameters of the search space in the chromosomes. Moreover, GAs are applicable to problems where the search space is finite and discrete. In case this is not so, as for this pattern classification problem, then the space has to be appropriately delimited and discretized.

2.2.2 Search Space for Lines

Let us assume that there are n training patterns available. Then, in the first step, the maximum and minimum values of each of the two features, X_1 and X_2, are computed. Let these be (x_1^{max}, x_1^{min}) and (x_2^{max}, x_2^{min}) for the features X_1 and X_2, respectively. Then the rectangle enclosing the sample points is given by the vertices $(x_1^{min}, x_2^{min}), (x_1^{min}, x_2^{max}), (x_1^{max}, x_2^{min}), (x_1^{max}, x_2^{max})$. Fig. 2.1 provides a figurative description of this. The rectangle represents the search space for the possible lines which may be considered as candidates for the formation of the decision boundary.

2.2.3 Line Representation

As is well known, one way of representing a line is by the equation

$$l_1 x_1 + l_2 x_2 = d \quad \text{such that} \quad l_1^2 + l_2^2 = 1, \tag{2.1}$$

where l_1 and l_2 are known as the direction cosines, and are given by

$$\begin{aligned} l_1 &= \cos\alpha_1 \\ l_2 &= \cos\alpha_2, \end{aligned}$$

and d is the perpendicular distance of the line from the origin. Here α_1 and α_2 are the angles that the unit normal makes with the X_1 and X_2 axes respectively. Since $\alpha_2 = \frac{\pi}{2} - \alpha_1$, we may rewrite Eq. (2.1) as

$$x_1 \cos\alpha_1 + x_2 \sin\alpha_1 = d. \tag{2.2}$$

Note that the constraint $l_1^2 + l_2^2 = 1$ holds automatically. The way of specifying the two variables α_1 and d is mentioned below.

- Angle (direction) Specification : The entire feature space will be spanned if the angle α_1 is allowed to vary in the range from 0 to 2π. If b_1 bits are used to represent an angle, then the possible values of α_1 are

  $$0, \delta * 2\pi, 2\delta * 2\pi, 3\delta * 2\pi, \ldots, (2^{b_1} - 1)\delta * 2\pi,$$

  where $\delta = \frac{1}{2^{b_1}}$. Consequently, if the b_1 bits contain a binary string having the decimal value v_1, then the angle is given by $v_1 * \delta * 2\pi$. Note that in this manner, 2^{b_1} distinct angle values are considered.

- Perpendicular Distance Specification : Once the angle α_1 is fixed, the orientation of the line becomes fixed. For a given orientation, the perpendicular distances of the two lines passing through the base points ((x_1^{min}, x_2^{min}) and (x_1^{max}, x_2^{min})) of the enclosing rectangle, from the origin, are computed from Eq. (2.2). (The perpendicular distance, d, assumes a negative value if it lies in the negative halfspace of the X_2 axis.) Among these, the one with the minimum value, d_{min}, is selected as the base line. This is demonstrated in Fig. 2.2 where the line through point 2 is the base line. The search space for lines with this orientation is restricted at one end by the base line. In other words, all lines with $d < d_{min}$, are automatically discarded from the search space. At the other end is a parallel line at a perpendicular distance of $(d_{min} + diag)$ from the origin, where $diag$ is the length of the diagonal of the rectangle given by

  $$diag = \sqrt{(x_1^{max} - x_1^{min})^2 + (x_2^{max} - x_2^{min})^2}.$$

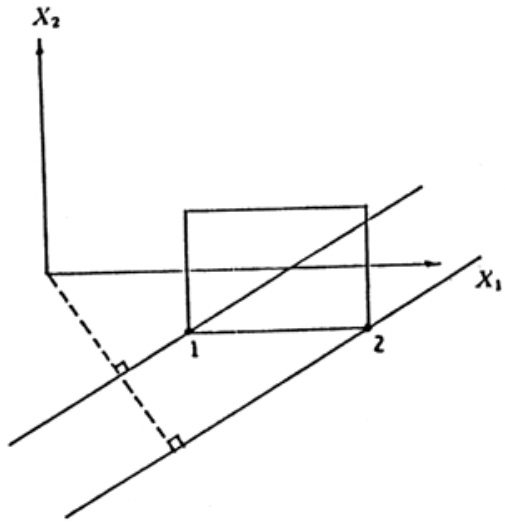


Figure 2.2: Fixing the base line

If b_2 bits are used to represent d, then a value of v_2 in these bits represents a line with the given α_1 and for which d is given by $d_{min} + \frac{diag}{2^{b_2}} * v_2$.

A line is thus specified by the two integer variables v_1 representing the angle, and v_2 representing the perpendicular distance. Let H, fixed *a priori*, be the number of lines used to model the class boundaries of any given data set. Then the length, l, of an individual or chromosome is given by

$$l = (b_1 + b_2) * H.$$

2.2.4 Genetic Operations

The fundamental operations of GA are given in Fig. 1.3. A detailed discussion on these operations is given below for the pattern classification problem.

Population Initialization

The GA generally works with a fixed population size of *Pop* where each chromosome is considered to be a string of 0s and 1s. Initially, each binary string, of length l, is generated randomly.

Example 2.1 :

Let

i : Dimension of the feature space $N = 2$

ii : Number of hyperplanes $H = 1$.

iii : Number of training points n =4.

iv : Number of bits for representing an angle $b_1 = 3$

v : Number of bits for representing the perpendicular distance $b_2 = 3$.

Then

$$\delta = \frac{1}{2^{b_1}} = \frac{1}{8}.$$

A string having the following bit pattern

$$\overbrace{0\,1\,0}^{\alpha_1} \quad \overbrace{1\,0\,0}^{d}$$

indicates a line whose normal makes angles $2 * \frac{1}{8} * 2\pi = \frac{\pi}{2}$ with the X_1 axis. Its perpendicular distance d from the origin is $d_{min} + 4 * \frac{diag}{2^3}$ $(= d_{min} + \frac{diag}{2})$, for some d_{min} and $diag$ (obtained from the $n = 4$ training data points). The equation of the line is, therefore, given by

$$x_2 = d_{min} + \frac{diag}{2}.$$

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. A string with the lowest misclassification is therefore considered to be the fittest among the population of strings. Note that every string str_i, $i = 1, 2, \ldots, Pop$ represents H lines denoted by ln_j^i, $j = 1, 2, \ldots, H$.

For each ln_j^i, the parameters l_1^{ij}, l_2^{ij} (the direction cosines) and d^{ij} (the perpendicular distance) are retrieved. For each training pattern point (x_1^m, x_2^m), $m = 1, 2, \ldots, n$, the sign with respect to the line ln_j^i, $(j = 1, 2, \ldots, H)$ i.e., the sign of the expression

$$\cos \alpha_1^{ij} x_1^m + \sin \alpha_1^{ij} x_2^m - d^{ij} \tag{2.3}$$

is found. The sign is digitized as 1 (0) if the point lies on the positive (negative) side of the line ln_j^i. The process is repeated for each of the lines, at the end of

which we have a string $sign_i^m$, 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 m^{th} training pattern. Note that each unique *sign string* corresponds to a unique region provided by the classifier.

In order to compute the misclassification associated with a particular arrangement of the lines (or a particular string), use of an auxiliary matrix (*class matrix*) is made. This is a $2^H * k$ matrix, where each row corresponds to a unique decimal value of the *sign string*, and each column, 1 to k corresponds to a class. This matrix is initialized with zeros. For each training point, the *sign string* is formulated, and its decimal value, $dec(sign)$, is computed. If the class of the point is i, then the entry in the location $[dec(sign), i]$ of matrix *class matrix* is incremented by one. At the same time, the entry in the 0th column of this row is set to 1, indicating that some training data points lie in the region represented by the *sign string*. This process is repeated for all the n training points.

In order to associate each region of the search space with a particular class, each row of *class matrix* (indicating a specific region) having a 1 in its 0th column is considered. Its maximum value is computed. Then the corresponding column number (class) is the label that is associated with the said region. All other points in this region are misclassified. The steps for computing the number of misclassified points is given in Fig. 2.3.

It is to be noted that although $sign_i^k$ can take on at most 2^H possible values (since H lines will yield a maximum of 2^H possible classifications), all of them may not occur in practice. Also, it may so happen that the maximum entries 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 2.2 :
Let there be 8 training patterns belonging to two classes, 1 and 2, in a two dimensional feature space $X_1 - X_2$. Let us assume H to be 3 i.e., 3 lines will be used to classify the points. Let the training set and a set of three lines be as shown in Fig. 2.4. Each point i_j, $i = 1, 2, \ldots, 8$, and $j = 1, 2$ indicates that it is the i^{th} training point and

```
Begin
Initialize class matrix with 0s
for i = 1 to n /** each training data point **/
Begin
    m ← class of point i.
    Compute the sign string
    Compute dec = decimal value of sign string
    Increment class matrix[dec, m]
    class matrix[dec, 0] = 1 /** Some points lie in this region **/
End
miss = 0 /** To hold the total misclassification **/
for i = 0 to 2^H
Begin
    If class matrix[i, 0] = 1
    Begin
        Compute max = maximum of class matrix[i]
        Let m be the corresponding class
        Output → Region i associated with class m
        Compute miss = miss + Σ_{j=1}^{k} class matrix[i, j] − max
    End
End
Output → Total misclassification = miss.
End
```

Figure 2.3: Steps for computing the number of misclassified training points

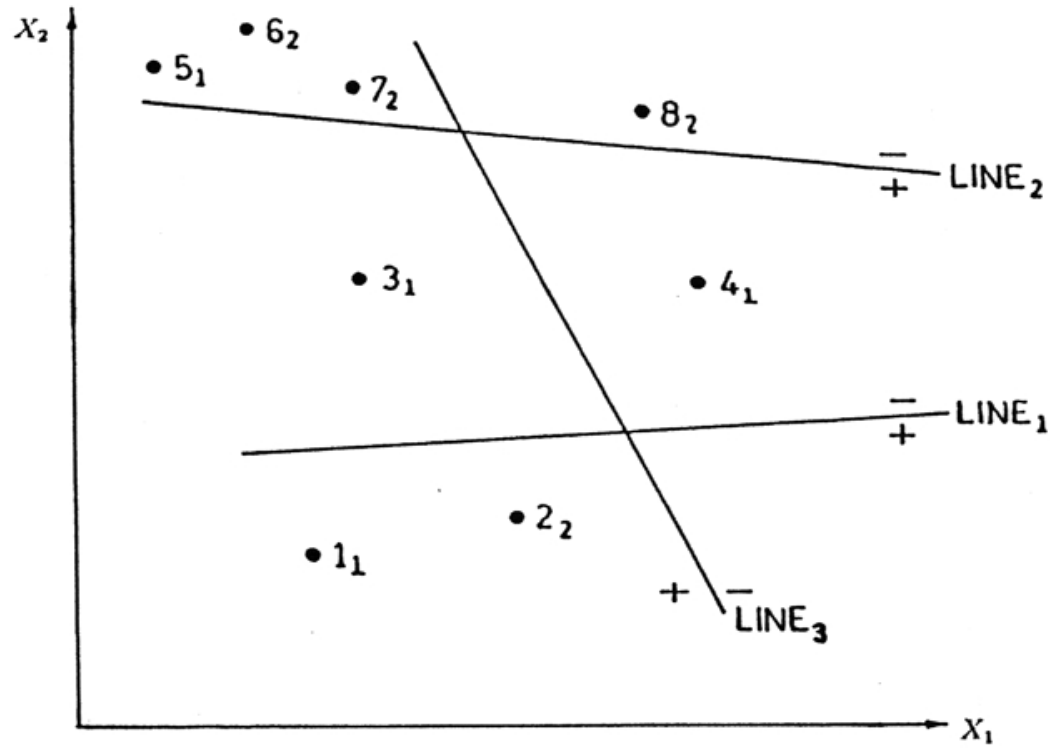


Figure 2.4: Region identification for $H = 3$ and $n = 8$

that it belongs to class j. Let the positive and the negative sides of the lines be as shown in Fig.2.4. 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 *class matrix* formed for all the eight points is shown in Fig. 2.5. It is to be noted that one region (denoted by sign string 110 or row 6) is accidentally empty, while two regions (100 and 101, or rows 4 and 5 respectively) do not exist. The number of misclassifications here is found to be $1 + 1 = 2$, one each for sign strings 001 (row 1) and 111 (row 7). Note that in this example both the strings 000 (row 0) and 001 (row 1) are providing the regions for class 2 (assuming that the tie for region 111 is resolved in favour of class 1). ♣

If the number of misclassified points for a string is denoted by *miss*, then the fitness of the string is computed as $(n - miss)$, where n is the number of training *data* points. 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* mentioned in Section 1.3.1, where

	0	1	2
0	1	0	1
1	1	1	2
2	1	1	0
3	1	1	0
4	0	0	0
5	0	0	0
6	0	0	0
7	1	1	1

Figure 2.5: *class array* for the example in Fig. 2.4

the best string seen upto the current generation is propagated to the next generation.

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.

Crossover :

A pair of strings is picked up at random and the single point crossover operator is applied according to a fixed crossover probability μ_c. For this operation, a random number cr_pt in the range of 1 to $l-1$ is generated. This is called the crossover point. The portion of the strings lying to the right of the crossover point are interchanged to yield two new strings.

Example 2.3 :
Let the two parents, for the parameters given in Example 2.1, be as follows :

$$\begin{array}{cc|cccc} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{array}$$

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

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

Mutation :

Mutation is done on a bit by bit basis (for binary strings) [61, 73] according to some mutation probability μ_m. So, theoretically, more than one bit may be flipped in the same string if μ_m so permits. The mutation probability is varied with the number of iterations. Initially it has a high value, thus ensuring a significant amount of diversity in the population. As training progresses and the GA reaches the vicinity of an optimal solution, μ_m is decreased. Finally, to ensure that the GA does not get stuck at a local optimum, μ_m is increased again.

Example 2.4:
From Example 2.3, after crossover we had child 1 = 0 0 0 0 0 1. Then mutation in bits 3 and 5 (from left) results in the string 0 0 1 0 1 1.

Termination

The process of fitness computation, selection, crossover and mutation continues for a fixed number of iterations *max_itrns* or till the termination condition (a string with misclassification number reduced to zero) is achieved.

Example 2.5 :
Let the string obtained on termination be 0 1 0 0 0 0. This represents a line lying parallel to the X_1 axis. The equation of the line is

$$x_2 = d_{min}.$$

2.3 Implementation

2.3.1 Data Sets

The effectiveness of the methodology described earlier has been demonstrated on two artificial data sets *ADS 1* and *ADS 2* (Figs. 2.6 and 2.7) and real life speech data (Fig. 2.8), none of which are linearly separable.

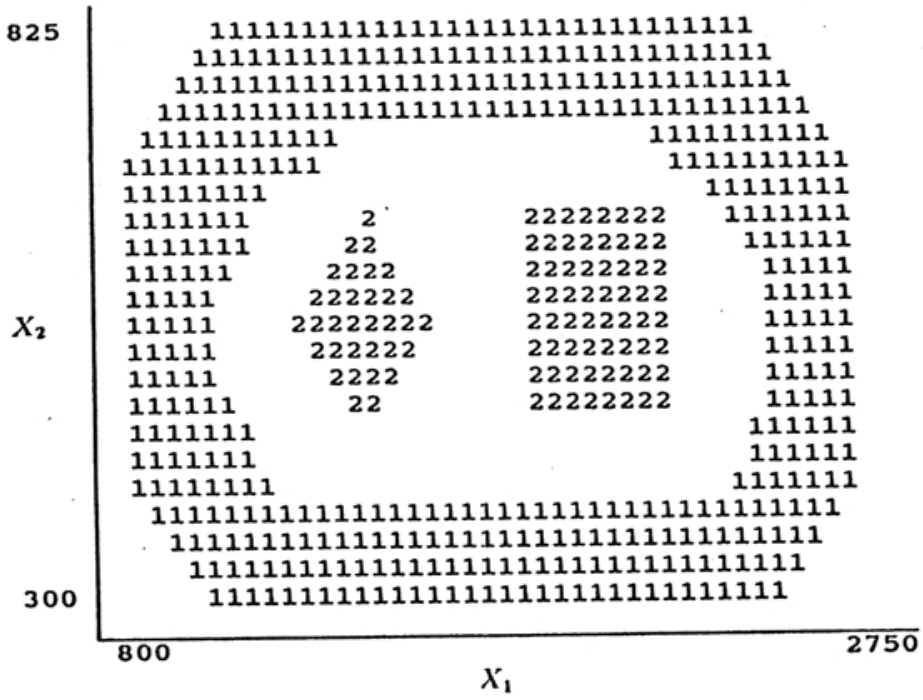


Figure 2.6: ADS 1

ADS 1 and *ADS 2*

These artificial data sets are shown in Figs. 2.6 and 2.7 respectively. Each data set has two classes, namely class 1 and class 2. *ADS 1* consists of 557 data points while *ADS 2* consists of 417 data points. The boundaries for both the data sets are seen to be highly non-linear, although the classes are separable.

Vowel Data

This data consists of 871 Indian Telugu vowel sounds [148]. 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 details of the method of extraction of formant frequencies through spectrum analysis are described in [148]. In order to apply the *GA-classifier* for $N = 2$, only the first two features F_1 and F_2 are considered. Fig. 2.8 shows the distribution of the six classes in the $F_1 - F_2$ plane. (It is known [148] that these two features are more important in characterizing the classes than F_3.) Note that the boundaries of the classes are seen to be ill-defined and overlapping.

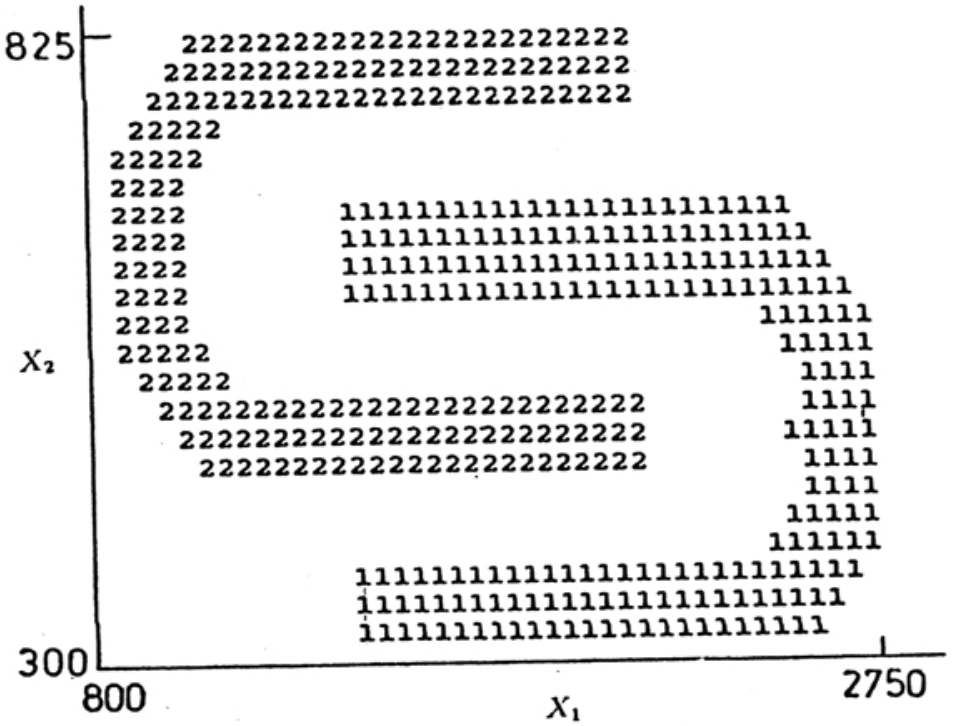


Figure 2.7: ADS 2

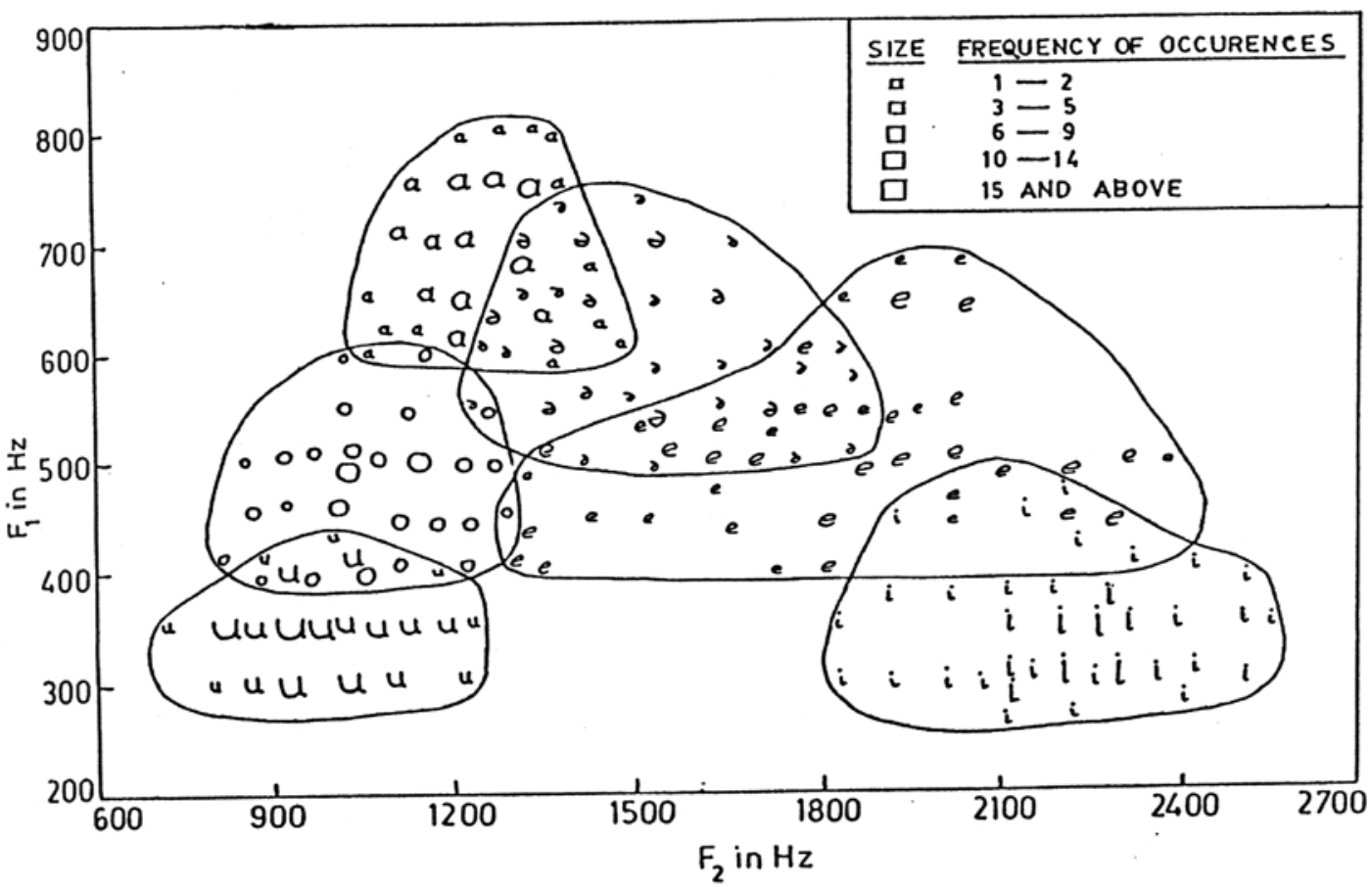


Figure 2.8: *Vowel* data in the $F_1 - F_2$ plane

2.3.2 Selection of Control Parameters

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 and crossover probability are fixed at 20 and 0.8 respectively.

The other parameter to be chosen is the mutation probability value, the importance of which has been stressed in [135, 189]. In [28], it is shown that the mutation probability must be chosen in the range of $(0, \frac{a-1}{a}]$ where a is the cardinality of the set of alphabets for the strings (cardinality= 2 for binary strings). In [135], it is proved that in many situations, the mutation probability value must be at least equal to $\frac{c}{l}$, 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 chapter we have chosen the mutation probability value 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. Then μ_m is again increased in steps to 0.333. 100 iterations are executed with each value of μ_m, thereby giving a maximum of 1500 iterations. Given that the number of bits in a string is l, the number of different possible strings are 2^l. 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 towards 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 inspite 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 [43, 73]. 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 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 that, in order to get good performance, the probability values should be high if the population size is low.

2.3.3 Results

Using *GA-Classifier*

In this section the effectiveness of the *GA-classifier* is extensively demonstrated on the data sets described earlier. Tables 2.1-2.3 present the test recognition scores corresponding to artificially generated data sets (*ADS 1* and *ADS 2*) and *Vowel* data set for different percentages of the training samples (e.g., *perc* = 5, 10, 50) and $H = 6$. The results shown are the average values computed over five runs of the algorithm from different initial populations.

It is seen from Table 2.1 that for *perc* = 50 and 10, the recognition ability of the classifier is considerably high for class 1 of the data set *ADS 1*. Class 2, on the other hand, is recognized relatively poorly. This disparity is due to the fact that 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 (see Fig. 2.6). The reversed situation for *perc* = 5 appears to be due to some error in sampling. Table 2.2 shows the classwise and overall recognition scores for *ADS 2*, which are again seen to be considerably high. Note that both the artificial data sets have non-overlapping, non-linear class boundaries.

Table 2.3 presents the results on the overlapping classes of *Vowel* data (Fig. 2.8). It is seen from the table that class δ yields a poor recognition score for *perc* = 50 and 10. This conforms to earlier findings [148, 151], when Bayes maximum likelihood

Table 2.1: Recognition scores (%) for *ADS 1* for $H = 6$

Class	*perc*=50	$perc = 10$	$perc = 5$
1	98.26	94.44	91.07
2	85.71	87.50	93.54
Overall	96.05	93.22	91.51

Table 2.2: Recognition scores (%) for *ADS 2* for $H = 6$

Class	*perc*=50	$perc = 10$	$perc = 5$
1	99.10	84.58	82.62
2	92.78	92.57	93.00
Overall	96.17	88.29	87.46

classifier, fuzzy set theoretic classifier, and MLP were used for vowel classification problem.

As expected, for all the three cases, the recognition scores are seen to improve with the value of *perc*, since, in general, increasing the size of the training data implies improving the representation of the entire data set. However, an important point to be noted here is that the performance of the *GA-classifier* remains respectable even for $perc = 5$, thereby indicating its effectiveness.

In order to demonstrate the variation of recognition score of the *GA-classifier* with H, we have considered *ADS 1* and *Vowel* data sets. Tables 2.4-2.5 show the classwise and the overall recognition scores during training for *ADS 1* and *Vowel* data sets, taking the values of H to be 8,7,6,5,4,3 and 2 successively. Tables 2.6-2.7 show the corresponding scores during testing. 10% ($perc = 10$) of the data set is used for training while the remaining 90% is used for testing. The results shown are the values averaged over five runs of the algorithm, each run starting with a different initial population.

From Tables 2.4 and 2.5 it is found that the recognition score during training improves consistently as the value of H is increased. (For *ADS 1*, any value of H beyond 5 gives

Table 2.3: Recognition scores (%) for *Vowel* Data in the $F_1 - F_2$ plane for $H = 6$

Class	*perc*=50	$perc = 10$	$perc = 5$
δ	55.56	61.53	79.27
a	71.11	76.54	76.17
i	86.04	72.25	71.34
u	77.63	86.02	72.22
e	83.65	77.54	78.68
o	91.11	72.22	93.24
Overall	81.00	75.44	72.95

Table 2.4: Variation of recognition scores (%) during training with H for *ADS 1*, with $perc = 10$

Class	Lines						
	8	7	6	5	4	3	2
1	100.00	100.00	100.00	100.00	100.00	100.00	97.82
2	100.00	100.00	100.00	66.67	60.56	66.67	44.44
Overall	100.00	100.00	100.00	94.55	93.72	94.55	89.09

100% recognition.) However, in the case of test data, the recognition score improves with H only upto a specific value. Then the score degrades (see Tables 2.6 and 2.7), since further increase in the number of lines makes the resulting decision boundary greatly dependent on the training data set. In other words, when a large number of lines 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 $H = 7$ and 8 (Table 2.6) and $H = 8$ (Table 2.7) demonstrate. (Some minor deviations can be noticed in Table 2.7 for $H = 3$ and 5. These are primarily due to small sample size and for stochastic errors in the process, but not due to overfitting of the data.)

Since *ADS 1* has a very small class totally surrounded by a larger class, the classwise

Table 2.5: Variation of recognition scores (%) during training with H for *Vowel* data in the $F_1 - F_2$ plane with $perc = 10$

Class	Lines						
	8	7	6	5	4	3	2
δ	85.71	57.14	85.71	14.28	42.85	0.00	0.00
a	87.55	100.00	87.50	87.50	37.50	0.00	0.00
i	100.00	100.00	100.00	100.00	100.00	100.00	53.33
u	93.33	100.00	86.67	86.67	100.00	100.00	100.00
e	94.99	100.00	94.99	100.00	100.00	100.00	100.00
o	83.33	72.22	77.78	83.33	77.79	77.78	94.44
Overall	91.76	90.58	89.41	85.88	84.70	77.64	72.94

Table 2.6: Variation of recognition scores (%) during testing with H for *ADS 1*, with $perc = 10$

Class	Lines						
	8	7	6	5	4	3	2
1	93.71	93.71	94.44	99.03	93.96	94.68	93.23
2	77.27	77.27	87.50	55.68	57.95	64.77	22.72
Overall	90.83	90.83	93.22	91.43	87.65	89.42	80.87

Table 2.7: Variation of recognition scores (%) during testing with H for *Vowel* data in the $F_1 - F_2$ plane with $perc = 10$

Class	Lines						
	8	7	6	5	4	3	2
δ	58.46	20.00	61.53	21.53	12.30	0.00	0.00
a	77.78	93.82	76.54	83.95	59.25	0.00	0.00
i	86.45	82.58	72.25	80.64	85.16	82.58	84.51
u	72.79	94.11	86.02	75.00	92.64	94.11	73.52
e	72.72	74.86	77.54	81.28	82.35	85.02	89.30
o	62.96	67.90	72.22	65.43	78.39	74.07	98.14
Overall	72.77	75.69	75.44	72.13	75.69	68.06	70.86

recognition score in this case is of greater importance than the overall score. Thus, it is seen from Tables 2.4 and 2.6, $H = 2$ is not a good choice in this case since the recognition of class 2 is very poor, although the overall score is reasonably good. The reason for the good overall recognition score is that it is more affected by the score of class 1 than that of class 2 (since, as already mentioned, there are many more points belonging to the former than to the latter).

For the *Vowel* data, since we have the number of classes, k, equal to 6, the minimum number of lines required (as mentioned in Section 2.2) for its proper classification is 3 ($3 \geq \lceil \log_2 6 \rceil$). This is evident from Tables 2.5 and 2.7, where it is found that the first two classes are not recognized at all for $H = 2$. Inspite of this, the algorithm manages to attain a fairly respectable overall recognition score, since those two lines are used to differentiate between the remaining four classes very well. Note that because of the complexity of the data set, even $H = 3$ is not sufficient for proper classification, where again the first two classes are not recognized at all. Also, as expected, class δ is classified very poorly as this is the class with maximum overlap. As mentioned before, this was also found in [148, 151] where fuzzy set theoretic classifier and Bayes maximum likelihood classifier were used for vowel classification problem.

The decision boundary obtained for *ADS 1*, with $H = 6$ is shown in Fig 2.9. The training pattern points are underscored in the figure. This boundary was obtained

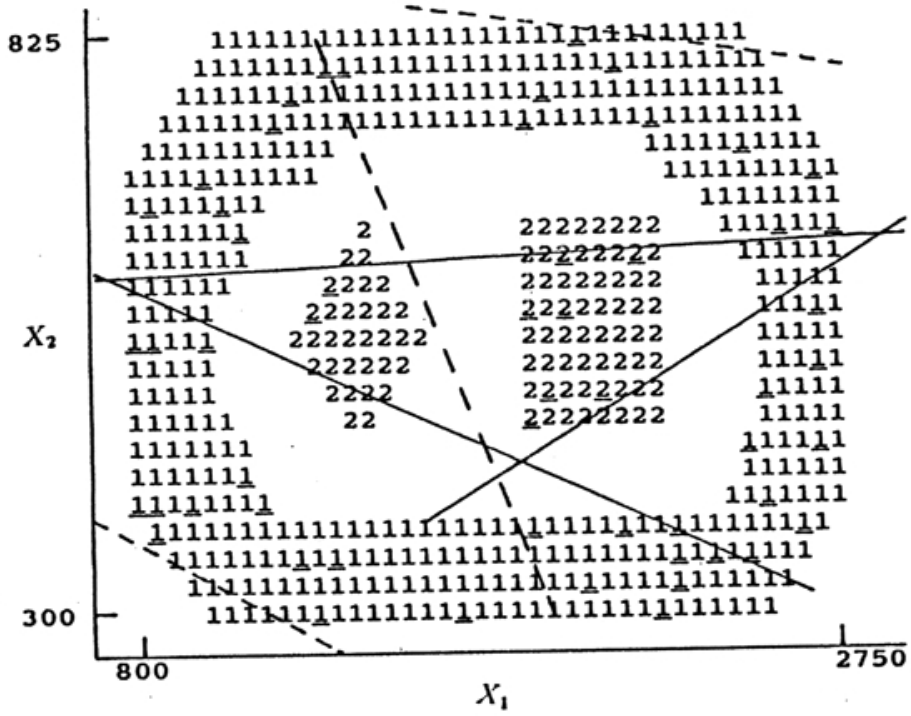


Figure 2.9: Classification of 90% test data set using the final boundary generated with 10% training data set for $H = 6$. Training patterns are underscored. Dotted lines indicate redundant lines

on termination of training after 538 iterations when a string with no misclassified points had been found. The dotted lines are obviously redundant.

Comparison with Existing Methods

The performance of the *GA-classifier* is compared with those of MLP (for three different architectures), Bayes maximum likelihood classifier (where *a priori* probabilities and the class conditional densities are estimated from the training data set) and k-NN classifier. For MLP, conventional back propagation (BP) algorithm, with the neurons executing the sigmoid function, is applied for learning. The learning rate η is fixed at 0.8, the momentum factor at 0.2 and the maximum number of generations at 3000. Online weight updation is performed, where the connection weights are updated every time an input pattern is presented to the network. The three different architectures considered consist of two hidden layers, with 5, 10 and 20 nodes in each layer respectively. The number of nodes in the input layer is two, and that in the output layer is equal to the number of classes in the data set.

k-NN algorithm is executed taking k equal to $\sqrt{n}$, where n is the number of training samples. It is known that as 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 [42, 67]. One such value of k for which the limiting conditions are satisfied is $\sqrt{n}$. For the Bayes maximum likelihood classifier, we assume a multivariate normal distribution of the samples with unequal dispersion matrices and unequal *a priori* probabilities ($= \frac{n_i}{n}$ for n_i patterns from class i) in each case. Comparison with these three methods is performed for all the data sets. Although for some data sets, application of Bayes maximum likelihood classifier, with the assumption of normal distribution may not be meaningful, the results are included to keep parity with others.

The comparative results are presented in Tables 2.8-2.10 for $H = 6$ and $perc = 10$. For both the artificial data sets, *ADS 1* and *ADS 2*, the GA based algorithm is found to provide good performance. In fact, for *ADS 1*, it provides the best result, followed by that of the k-NN classifier. Note that k-NN classifier is reputed to partition well this type of non-overlapping, non-linear regions. MLP with 5 nodes in both the layers is seen to yield the best result for *ADS 2*. The result for MLP is found to degrade as the number of nodes in the hidden layers is increased for this data set indicating the situation of over fitting of the training data for large architectures. *GA-classifier* performs better than k-NN, Bayes maximum likelihood classifier and one case of MLP. It must be noted that MLP totally fails to recognize class 2 for *ADS 1* for all the three cases; consequently, its performance on *ADS 1* is also poor.

Table 2.10 shows the result for the *Vowel* data in the $F_1 - F_2$ plane where Bayes maximum likelihood classifier is seen to provide the best score (79.13%), followed by the result of the *GA-classifier* (75.44%). The Bayes maximum likelihood classifier is known to perform well for this data set [148], assuming multivariate normal densities for the classes. As expected, all the classifiers yield poor recognition scores for class δ. MLP is seen to perform poorly for this data set.

Table 2.8: Comparative recognition scores (%) for *ADS 1* for $perc = 10, H = 6$

Class	Bayes	k-NN	MLP			GA
			2:5:5:2	2:10:10:2	2:20:20:2	
1	100.00	96.85	100.00	100.00	100.00	94.44
2	18.18	59.09	0.00	0.00	0.00	87.50
Overall	85.65	90.23	82.47	82.47	82.47	93.22

Table 2.9: Comparative recognition scores (%) for *ADS 2* for $perc = 10, H = 6$

Class	Bayes	k-NN	MLP			GA
			2:5:5:2	2:10:10:2	2:20:20:2	
1	84.08	84.58	95.01	100.00	100.00	84.58
2	77.71	86.29	88.62	76.57	68.10	92.57
Overall	81.11	85.63	91.92	89.09	85.10	88.29

Table 2.10: Comparative recognition scores (%) for *Vowel* data in the $F_1 - F_2$ plane for $perc = 10, H = 6$

Class	Bayes	k-NN	MLP			GA
			2:5:5:6	2:10:10:6	2:20:20:6	
δ	46.15	35.38	27.16	24.69	23.07	61.53
a	85.18	81.48	47.69	46.15	66.67	76.54
i	81.93	85.80	90.96	82.58	84.50	72.25
u	89.70	76.47	91.91	74.26	91.17	86.02
e	83.42	75.40	59.89	54.01	69.51	77.54
o	72.83	77.16	69.75	85.18	14.80	72.22
Overall	79.13	75.31	69.21	65.90	60.81	75.44

2.4 Extension of the *GA-classifier* to IR^N

The previous discussion described the *GA-classifier* in two dimensions only. However, in order to be effective, the classifier must be applicable to higher dimensions as well. In this section, the extension of the classifier to N dimensions, $N \geq 2$ is described. The decision boundary, which was approximated by lines in two dimensions, will require planes in three dimensions, and hyperplanes in higher dimensions. For the sake of generality, we refer to the decision boundaries as being approximated by hyperplanes.

2.4.1 Description

We consider a fixed number of hyperplanes (say H) to denote a decision boundary in an N dimensional feature space $X_1, X_2, \ldots, X_N$. These H hyperplanes need to be encoded as a single string. As before, $k \leq 2^H$ or $H \geq \lceil \log_2 k \rceil$, where k represents the number of classes in the data set.

2.4.2 Search Space for Hyperplanes

The search space for the hyperplanes is restricted to the hyper rectangle constructed around the training data points, given by the vertices
$(x_1^{min}, x_2^{min}, \ldots, x_N^{min}), (x_1^{max}, x_2^{min}, \ldots, x_N^{min}), (x_1^{min}, x_2^{max}, x_3^{min}, \ldots, x_N^{min})$
$, \ldots, (x_1^{max}, x_2^{max}, x_3^{min}, \ldots, x_N^{min}), \ldots, (x_1^{max}, x_2^{max}, \ldots, x_N^{max})$.

where x_i^{min} and x_i^{max}, $i = 1, 2, \ldots, N$, represent the minimum and maximum values of ith feature. Fig. 2.10 shows such an example for three dimensional 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.

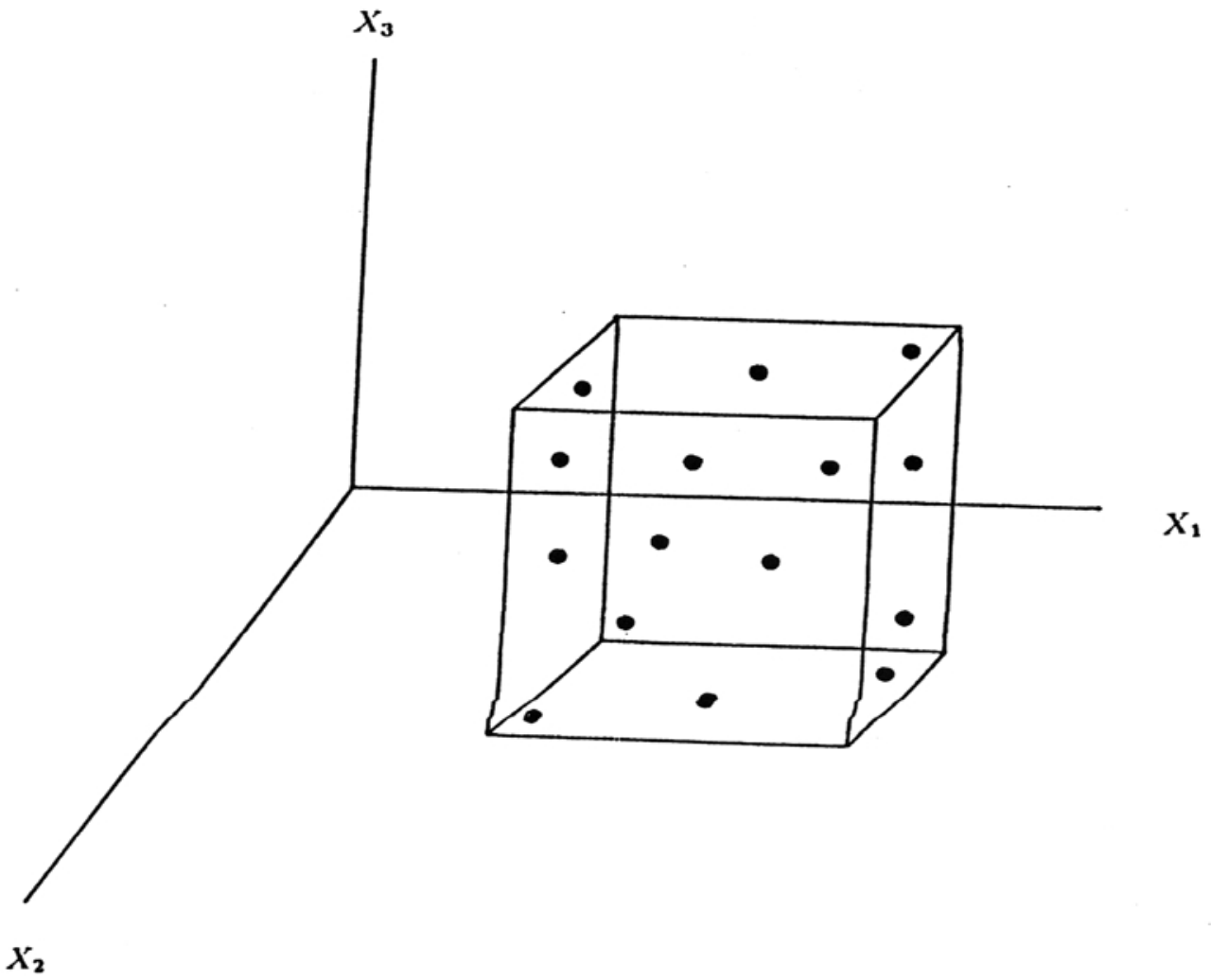


Figure 2.10: Training samples and the enclosing hyper rectangle

2.4.3 Hyperplane Specification

Extending the concepts of the two dimensional case, one may write the equation of a hyperplane as [203, 209]

$$l_1 x_1 + l_2 x_2 + \ldots + l_N x_N = d, \tag{2.4}$$

where $l_1 = \cos\alpha_1, l_2 = \cos\alpha_2, \ldots, l_N = \cos\alpha_N$. Here $\alpha_1, \alpha_2, \ldots, \alpha_N$ are the angles that the unit normal to the hyperplane makes with the N axes $X_1, X_2, \ldots, X_N$ respectively; d is the perpendicular distance of the hyperplane from the origin. It also follows that

$$l_1^2 + l_2^2 + \ldots + l_N^2 = 1. \tag{2.5}$$

Hence, N variables are needed to specify a hyperplane viz. $N-1$ angle variables (say $\alpha_1, \alpha_2, \ldots, \alpha_{N-1}$) and d. Angle α_N is implicitly specified through Eq. (2.5).

Although this representation seems natural, it has certain problems. Because of the constraint stated in Eq. (2.5), not all combinations of α_i, $1 \leq i \leq N$ are possible. Those violating the constraint equation do not represent valid hyperplanes. On the other hand, it is difficult to restrict GA so that only valid strings (representing valid hyperplanes) are formed. Although there are several techniques for performing constrained optimization in genetic algorithms, like the addition of a high penalty value [126, 176], the first approach should be to try to reformulate the problem, so that it becomes unconstrained in nature. Fortunately, this is possible for the hyperplane representation case.

The equation of a hyperplane in N dimensional space $(X_1 - X_2 - \cdots - X_N)$ may be rewritten recursively as

$$x_N \cos\alpha_{N-1} + \beta_{N-1} \sin\alpha_{N-1} = d, \tag{2.6}$$

where $\beta_{N-1} = x_{N-1}\cos\alpha_{N-2} + \beta_{N-2}\sin\alpha_{N-2}$

$\beta_{N-2} = x_{N-2}\cos\alpha_{N-3} + \beta_{N-3}\sin\alpha_{N-3}$

$\vdots$

$\beta_1 = x_1 \cos\alpha_0 + \beta_0 \sin\alpha_0.$

The various parameters are as follows :

X_i : the i th feature of the training points.

$(x_1, x_2, \ldots, x_N)$: a point on the hyperplane

α_{N-1} : the angle that the unit normal to the hyperplane makes with the X_N axis.
α_{N-2} : the angle that the projection of the normal in the $(X_1 - X_2 - \cdots - X_{N-1})$ space makes with the X_{N-1} axis.
$\vdots$
α_1 : the angle that the projection of the normal in the $(X_1 - X_2)$ plane makes with the X_2 axis.
α_0 : the angle that the projection of the normal in the (X_1) plane makes with the X_1 axis $= 0$.
Hence, $\beta_0 \sin \alpha_0 = 0$.
d : the perpendicular distance of the hyperplane from the origin.

Thus the N tuple $< \alpha_1, \alpha_2, \ldots, \alpha_{N-1}, d >$ specifies a hyperplane in N dimensional space. As before, each angle α_j, $j = 1, 2, \ldots, N-1$ is allowed to vary in the range of 0 to 2π. If b_1 bits are used to represent an angle, then the possible values of α_j are

$$0, \delta * 2\pi, 2\delta * 2\pi, 3\delta * 2\pi, \ldots, (2^{b_1} - 1)\delta * 2\pi,$$

where $\delta = \frac{1}{2^{b_1}}$. A value v_1^j in the b_1 bits corresponding to α_j represents an angle given by $v_1^j * \delta * 2\pi$.

For the purpose of specifying d, the above mentioned hyper rectangle enclosing the training points is considered. The vertices of the enclosing hyper rectangle are now represented as :

$$(x_1^{ch_1}, x_2^{ch_2}, \ldots, x_N^{ch_N}),$$

where each ch_i, $i = 1, 2, \ldots, N$ can be either max or min. (Note that there will be 2^N vertices.) The length of the diagonal, $diag$, of this hyper rectangle is given by

$$diag = \sqrt{(x_1^{max} - x_1^{min})^2 + (x_2^{max} - x_2^{min})^2 + \ldots + (x_N^{max} - x_N^{min})^2}.$$

A hyperplane is designated as the *base hyperplane* with respect to a given orientation (i.e., for some $\alpha_1, \alpha_2, \ldots, \alpha_{N-1}$) if

i : it has the same orientation, $\alpha_1, \alpha_2, \ldots, \alpha_{N-1}$,

ii : it passes through one of the vertices of the enclosing rectangle,

iii : its perpendicular distance from the origin is minimum (among the hyperplanes passing through the other vertices). As before, let this distance be d_{min}.

If b_2 bits are used to represent d, then a value of v_2 in these bits represents a hyperplane with the given orientation and for which d is given by $d_{min} + \frac{diag}{2^{b_2}} * v_2$.

2.4.4 Genetic Operations

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. Thus each chromosome is of a fixed length given by

$$l = H((N-1) * b_1 + b_2).$$

Pop chromosomes are initially generated randomly to form a population.

A detailed flowchart for the method described here is given in Fig. 2.11. The operations of generating initial population, fitness computation, selection, crossover and mutation are as described earlier. The only difference is that now the feature space can be of any dimension $N \geq 2$. An example of a string in three dimensional space is included here for ease of understanding.

Example 2.6 :
Let n =4, $b_1 = 3$, $b_2 = 3$, $N = 3$ and $H = 1$. Then a string

$$\overbrace{0\,0\,0}^{\alpha_1}\quad\overbrace{0\,1\,0}^{\alpha_2}\quad\overbrace{1\,0\,0}^{d}$$

indicates a plane whose

i : normal makes an angle $(2 * \frac{2\pi}{8} = \frac{\pi}{2})$ with the X_3 axis,

ii : projection of the normal in the $X_1 - X_2$ plane makes an angle 0 with the X_2 axis.

In other words, the plane lies parallel to the $X_1 - X_3$ plane. The equation of the plane is

$$x_2 = d_{min} + 4 * \frac{diag}{2^3} = d_{min} + \frac{diag}{2}.$$

2.5 Postprocessing (Deletion of Redundant Hyperplanes)

As mentioned earlier, for a k class problem we have the following inequality

$$H \geq \lceil \log_2 k \rceil.$$

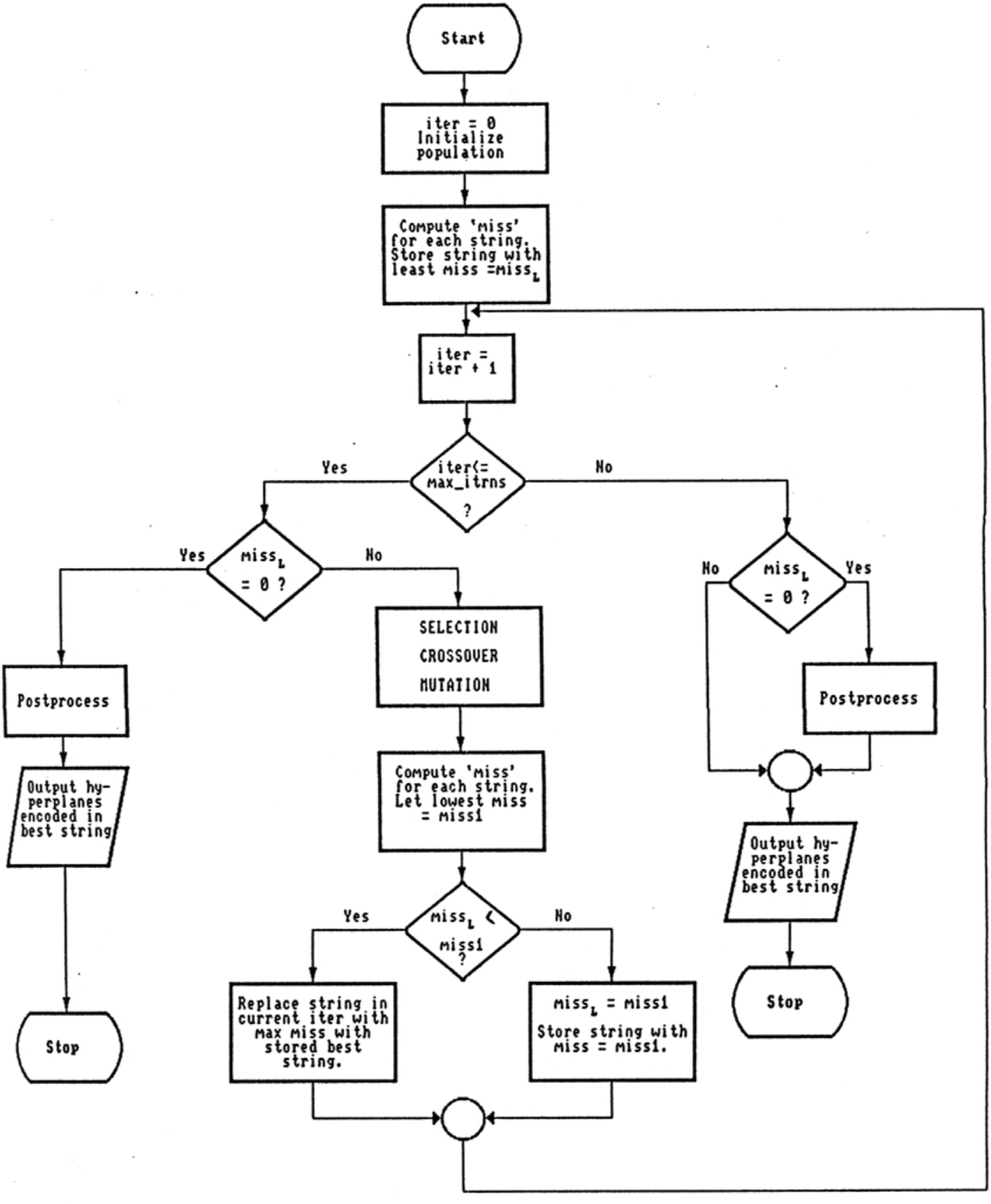


Figure 2.11: Flowchart for the GA based pattern classification method

In order to tackle the intricacy of certain data sets, consideration of $H = \lceil \log_2 k \rceil$ may not be sufficient for proper classification. For this reason, we try to make an overestimation of H (i.e., H sufficiently greater than $\lceil \log_2 k \rceil$) 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. 2.12. 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). A and S are the inputs to the block "Postprocess" (Fig. 2.11) for elimination of redundant hyperplanes.

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 $S2$ ($S2 = \{h_1, h_2, \ldots, h_p\}$, $p \leq H$) is also formed such that one side (subsequently referred to as *unique* side, since all the points on this side belong to exactly one class) 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 $S2$ 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. 2.12 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 for similar processing. In case *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 the original A and the hyperplanes in S' as the set S.

The processing block (Fig. 2.12) 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 by the process described earlier. The detailed flowchart of *check_red* is presented in Fig. 2.13. It eliminates those hyperplanes in NR which are not essential for classifying the sample 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 2.2.4. To return *True*, the *class matrix* formed for NR must have atmost a single entry for each row corresponding to *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 e.g., consider Fig. 2.14 in two dimensional 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

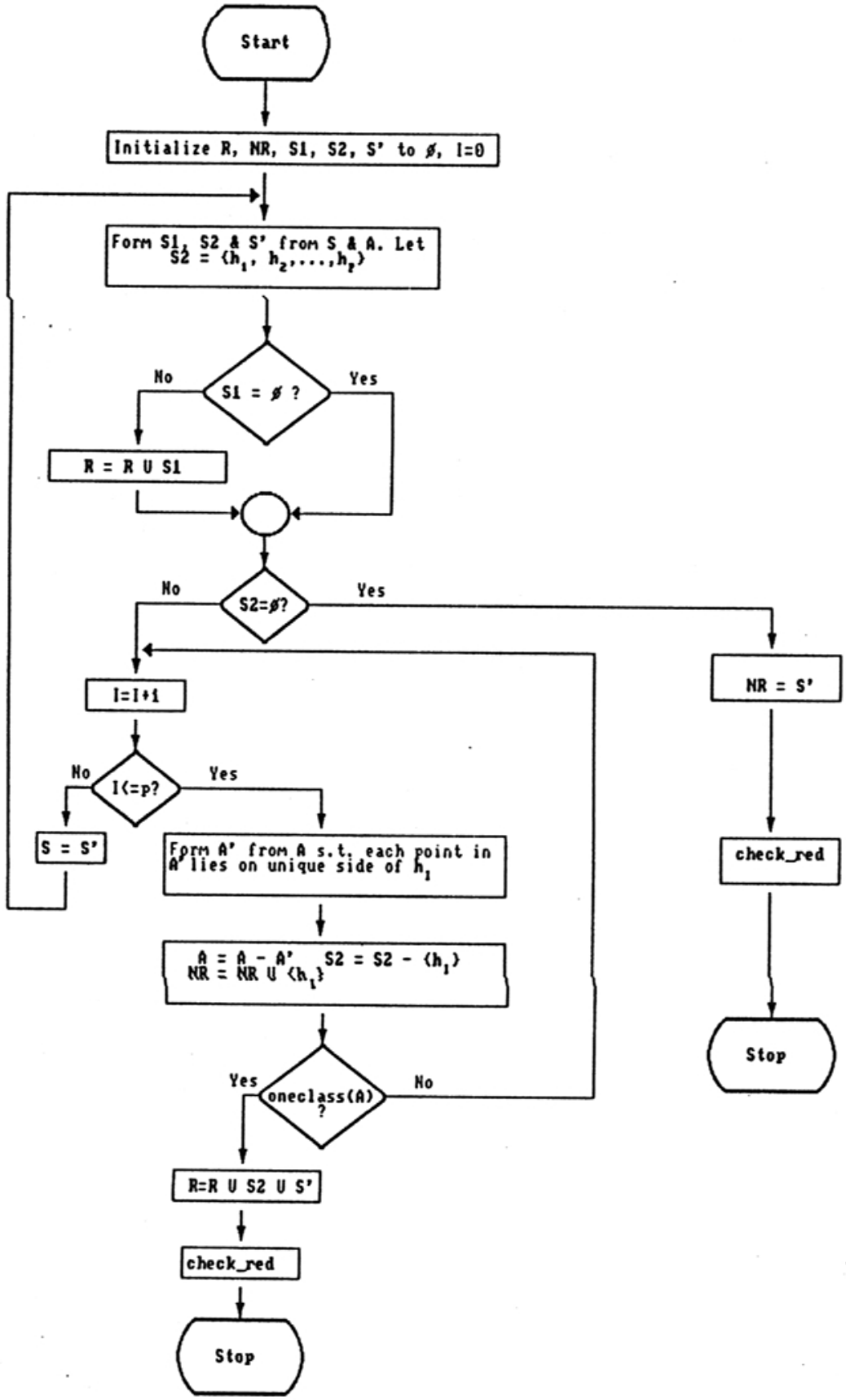


Figure 2.12: Flowchart for the process of elimination of redundant hyperplanes

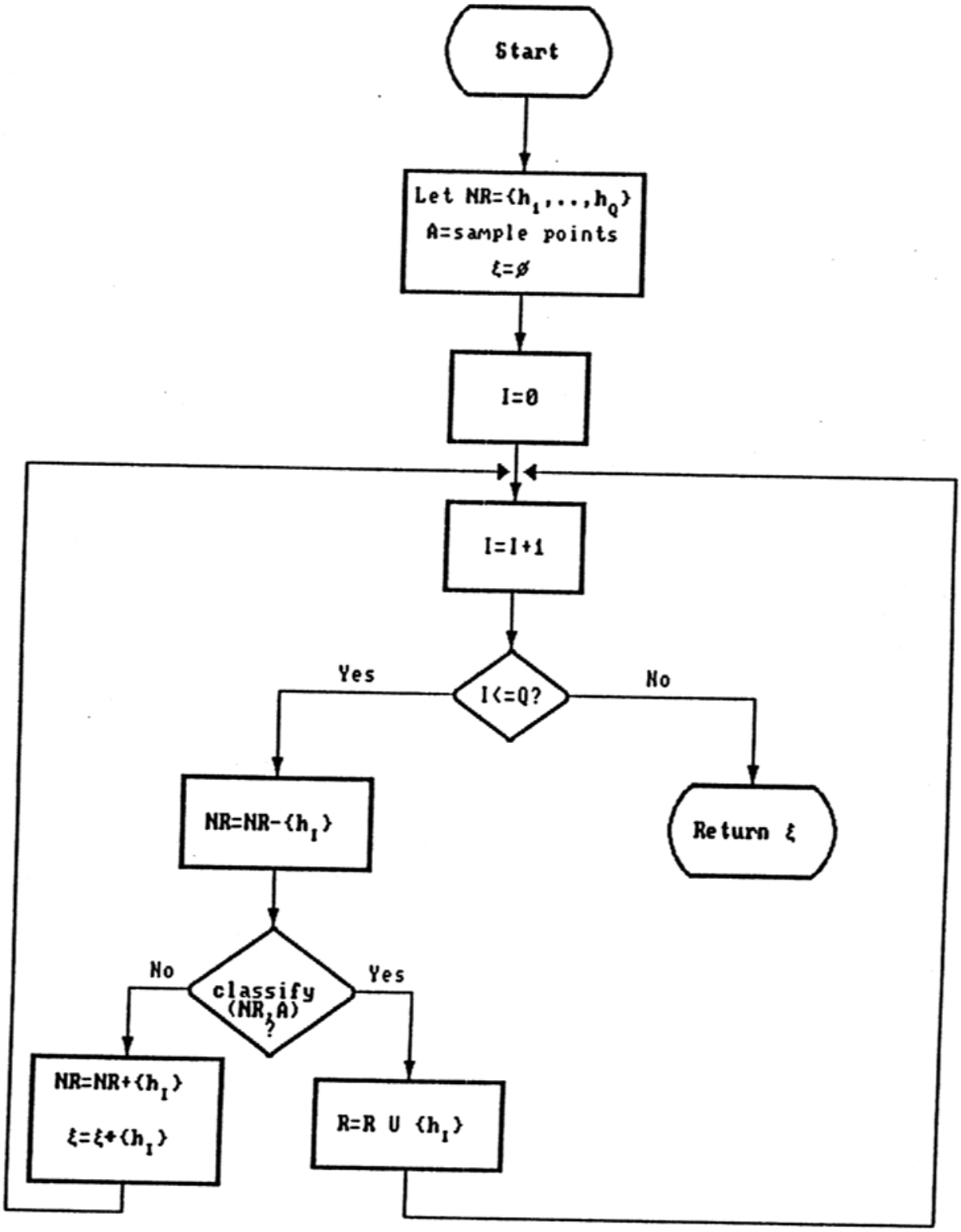


Figure 2.13: Flowchart for the process *check_red*

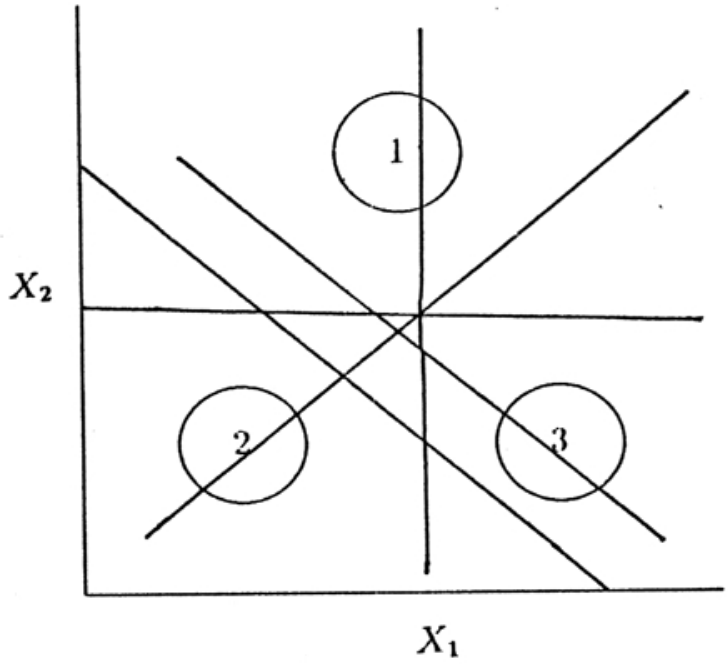


Figure 2.14: Three classes and a set S of five lines for partitioning them

three and two lines are shown in Figs. 2.15 and 2.16 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. 2.16 may be deleted. *check_red* would then declare the three lines shown in Fig. 2.15 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. Inspite 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 inter relationships into account.

2.6 Implementation

In this section, the results of the application of *GA-classifier* to data sets of higher dimensionality, along with an indication of the number of redundant hyperplanes, are presented. The parameters are same as the case for $N = 2$.

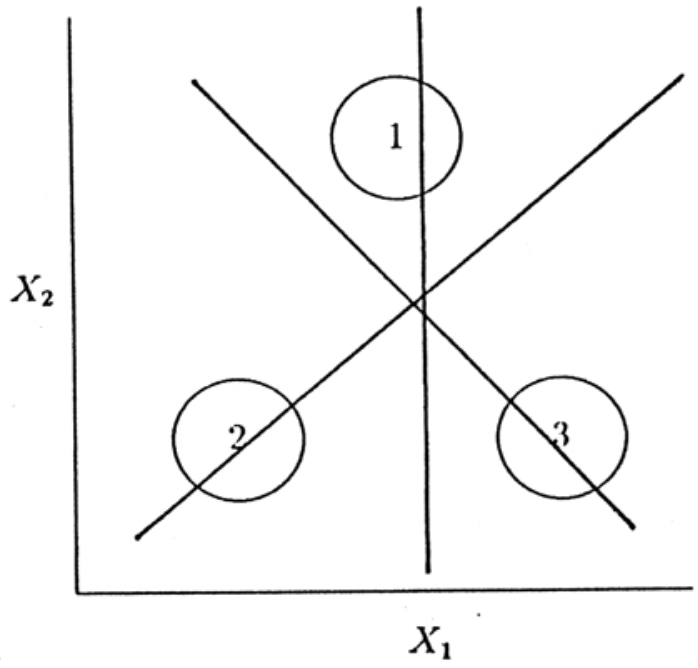


Figure 2.15: Subset of S comprising three lines which can successfully partition the three classes in Fig. 2.14

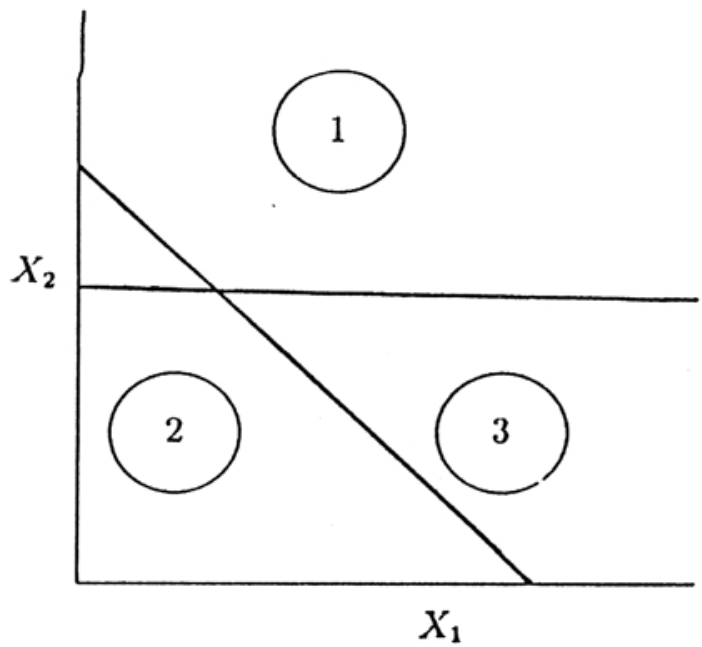


Figure 2.16: Subset of S comprising two lines which can successfully partition the three classes in Fig. 2.14

2.6.1 Data Sets in $\mathbb{R}^N$

Vowel Data : The description of this data is provided in Section 2.3.1. For implementing the *GA-classifier* for $N > 2$, we considered all the three features F_1, F_2 and F_3 for classifying the six vowel classes $\{\delta, a, i, u, e, o\}$.

Iris Data : This data represents different categories of irises having four feature values. The four feature values represent the sepal length, sepal width, petal length and the petal width in centimeters [62]. It has three classes 1, 2 and 3 with 50 samples per class.

Cancer Data : This breast cancer database, obtained from the University of Wisconsin Hospital, Madison [122], is used for the purpose of demonstrating the effectiveness of the classifier in classifying high dimensional patterns. It has 683 samples belonging to two classes *Benign* (class 1) and *Malignant* (class 2), and nine features corresponding to *clump thickness, cell size uniformity, cell shape uniformity, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli* and *mitoses.*

2.6.2 Experimental Results

Tables 2.11, 2.12 and 2.13 present the test recognition scores for the *Vowel*, *Iris* and *Cancer* data sets respectively, for different values of H. For the *Vowel data*, the score is found to improve with the value of H upto $H = 7$ which provides the maximum score. On application of the post processing step, two out of these seven hyperplanes were found to be redundant. However, $H = 5$ provided a comparatively lower score. This may be due to the fact that with $H = 7$, the classifier has more flexibility in placing the five hyperplanes appropriately, than for $H = 5$. It is also to be noted that the *GA-classifier* will be able to place the hyperplanes appropriately even with $H = 5$ if it is executed for sufficiently more iterations. (A similar observation was made when the post processing redundancy elimination step was applied to *ADS 1* for $H = 6$. It was found that 3 lines were actually required to provide the class boundaries. However, it is evident from Tables 2.4 and 2.6 that consideration of $H = 3$ provides neither good training nor test performance.) For the *Iris* data, redundancy removal process yielded, in each case, exactly two non redundant hyperplanes. At

Table 2.11: Variation of recognition scores (%) during testing with H for *Vowel* data with $perc = 10$

Class	Surfaces						
	8	7	6	5	4	3	2
δ	67.69	21.53	46.15	49.23	52.30	55.38	0.00
a	75.30	86.42	32.09	30.86	0.00	0.00	0.00
i	79.35	77.42	84.51	78.06	76.12	81.29	80.00
u	88.97	90.44	75.73	77.94	84.55	56.61	0.00
e	69.31	80.75	78.07	83.95	86.63	72.19	73.26
o	66.04	67.28	41.35	74.07	70.98	48.14	97.50
Overall	74.56	74.68	71.99	71.37	69.21	57.50	53.30

the same time, $H = 2$ and 3 are found to provide the best score. Similarly for the *Cancer* data, the recognition scores decrease as H increases, with $H = 2$ and 3 providing the best values.

A point to be noted here is that inclusion of F_3 increases the confusion of the *Vowel* data set, thereby reducing the recognition scores for all values of H (except $H = 8$). See Tables 2.7 and 2.11 in this regard. This was also found to be the case in [148].

The comparative recognition scores of the *Vowel*, *Iris* and *Cancer* data sets are presented in Tables 2.14, 2.15 and 2.16 respectively. As for the two dimensional case,

Table 2.12: Variation of recognition scores (%) during testing with H for *Iris* with $perc = 10$

Class	Surfaces						
	8	7	6	5	4	3	2
1	100.00	97.78	100.00	100.00	100.00	100.00	100.00
2	75.55	82.22	84.44	82.22	82.22	93.33	93.33
3	97.78	93.33	95.55	97.78	97.78	100.00	100.00
Overall	91.11	91.11	93.33	93.33	93.33	97.78	97.78

Table 2.13: Variation of recognition scores (%) during testing with H for *Cancer* data with $perc = 10$

Class	Surfaces						
	8	7	6	5	4	3	2
1	91.25	96.75	96.75	93.80	93.50	94.49	95.75
2	79.62	75.46	86.11	91.05	92.59	94.90	92.59
Overall	87.17	89.28	93.01	92.90	93.18	94.64	94.64

the Bayes maximum likelihood classifier is found to provide the best performance, for *Vowel* data, followed by the score of the *GA-classifier*. Note that although the *GA-classifier* score is presented for H=6, this is much less compared to the one for $H = 7$ (74.68% from Table 2.11). MLP is again found to perform poorly for this data set.

For *Iris* data, as seen from Table 2.15, *GA-classifier* is found to provide a significantly better score than all other classifier, even for H=6. (Note from Table 2.12 that the score is even better for $H = 2$ and 3). Both the Bayes maximum likelihood classifier and MLP perform poorly for this data set.

From Table 2.16 it is found that both k-NN classifier and MLP perform well for the *Cancer* data. In fact it is found that all the classifiers provide relatively high recognition scores for this data.

2.7 Consideration of Higher Order Surfaces

It is to be noted that instead of hyperplanes, the class boundary of any given data set can be modeled using any other higher order surface. This section describes the method of utilizing a fixed number of hyperspherical segments to constitute the decision surface in N dimensional space, $N \geq 2$.

Table 2.14: Comparative recognition scores (%) for *Vowel* data for $perc = 10, H = 6$

Class	Bayes	k-NN	MLP			GA
			3:5:5:6	3:10:10:6	3:20:20:6	
δ	49.23	35.38	12.30	47.69	47.23	46.15
a	83.95	74.07	43.20	16.04	27.62	32.09
i	81.29	83.87	69.67	79.35	80.58	84.51
u	75.00	80.88	80.14	77.94	83.29	75.73
e	79.14	74.33	68.44	90.37	87.30	78.07
o	83.33	56.17	77.16	54.93	51.70	41.35
Overall	77.73	70.35	65.26	67.55	68.48	71.99

Table 2.15: Comparative recognition scores (%) for *Iris* for $perc = 10, H = 6$

Class	Bayes	k-NN	MLP			GA
			4:5:5:3	4:10:10:3	4:20:20:3	
1	100.00	100.00	100.00	98.81	66.67	100.00
2	57.55	97.77	66.67	64.77	71.11	84.44
3	92.11	73.33	77.78	80.51	97.78	95.55
Overall	83.22	90.37	81.48	81.36	78.51	93.33

Table 2.16: Comparative recognition scores (%) for *Cancer* data for $perc = 10, H = 6$

Class	Bayes	k-NN	MLP			GA
			9:5:5:2	9:10:10:2	9:20:20:2	
1	88.75	98.50	97.25	97.25	97.25	96.75
2	96.29	90.74	90.74	90.74	90.74	86.11
Overall	91.39	95.77	94.97	94.97	94.97	93.01

2.7.1 Description

The equation of a hypersphere in N dimensions, with center $(h_1, h_2, \ldots, h_N)$, and radius r is given by

$$(x_1 - h_1)^2 + (x_2 - h_2)^2 + \ldots + (x_N - h_N)^2 = r^2. \tag{2.7}$$

Thus the $N+1$ parameters $h_1, h_2, \ldots, h_N$ and r correspond to a unique hypersphere. As for the case of hyperplanes, we construct a hyper rectangle, around the training data points. The lengths of the N sides of this hyper rectangle, $l_1, l_2, \ldots, l_N$ are given by

$$\begin{aligned} l_1 &= x_1^{max} - x_1^{min} \\ l_2 &= x_2^{max} - x_2^{min} \\ &\vdots \\ l_N &= x_N^{max} - x_N^{min}. \end{aligned}$$

This rectangle will subsequently be referred to as the *inner rectangle*. A possible encoding scheme for a number (say C) of hyperspherical segments is to represent the parameters $h_1, h_2, \ldots, h_N$ and r as a bit string, such that $h_1, h_2, \ldots, h_N$ are constrained to lie within the *inner rectangle*, and r in the range [0,*diag*], where *diag* is the length of the diagonal of the *inner rectangle*. Thus if we assume the following :

$b_{h_1}, \ldots, b_{h_N}$ = number of bits for representing $h_1, \ldots, h_N$ respectively
b_r = number of bits for representing r
$v_{h_1}, \ldots, v_{h_N}$ = value in the $h_1, \ldots, h_N$ fields of a chromosome respectively
v_r = value in the r field of a chromosome

then these parameters decode to the hypersphere with the following specifications :

$$\begin{aligned} h_1 &= x_1^{min} + \frac{v_{h_1}}{2^{b_{h_1}}} * l_1 \\ h_2 &= x_2^{min} + \frac{v_{h_2}}{2^{b_{h_2}}} * l_2 \\ &\vdots \\ h_N &= x_N^{min} + \frac{v_{h_N}}{2^{b_{h_N}}} * l_N \\ r &= \frac{v_r}{2^{b_r}} * diag \end{aligned}$$

There is a subtle problem with this representation. Since the center of the hypersphere is constrained to lie within the *inner rectangle* and the radius in the range [0,*diag*], the decision boundaries generated by a finite number of such hyperspheres

becomes restricted. Thus any complicated boundary cannot be generated easily with this technique.

In order to bypass this problem, the following alternative has been adopted. Surrounding the *inner rectangle*, let us consider a larger rectangle, to be referred to as the *outer rectangle*, whose sides are of length $l'_1, l'_2, \ldots, l'_N$, where

$$\begin{aligned} l'_1 &= (2p+1) * l_1 \\ l'_2 &= (2p+1) * l_2 \\ &\vdots \\ l'_N &= (2p+1) * l_N, \end{aligned}$$

and p is an integer, $p \geq 0$. Then the center of the hypersphere (which will be considered for constituting the decision boundary) is allowed to lie anywhere within the *outer rectangle*. p can be chosen sufficiently large in order to approximate any form of the decision boundary.

The center $(h_1, h_2, \ldots, h_N)$ of the hypersphere is a randomly chosen point within the outer rectangle. For computing the radius, the nearest distance of the center $(h_1, h_2, \ldots, h_N)$ from the vertices of *inner rectangle* is determined. Let this be d_1. Similarly, let the farthest distance be d_2. Now if $(h_1, h_2, \ldots, h_N)$ lies within the *inner rectangle*, then the radius can take on values in the range [0,d_2]. Otherwise the range is considered as $[d_1, d_2]$. Given any chromosome, the detailed steps of decoding the parameter values are given in Fig. 2.17.

The number of bits required for encoding one hypersphere is, therefore, given by

$$b_{h_1} + b_{h_2} + \ldots + b_{h_N} + b_r.$$

For a fixed number of hyperspheres C, the string length becomes

$$l = (b_{h_1} + b_{h_2} + \ldots + b_{h_N} + b_r) * C.$$

In general $b_{h_1} = b_{h_2} = \ldots = b_{h_N} = b_1$. Hence the string length will be

$$l = (N * b_1 + b_r) * C.$$

The regions corresponding to the different classes are determined from the training data set using Eq. (2.7). Fitness of a string is determined by the number of points

Step 1 : Computation of the center of the hypersphere.
Let $v_{h_1}, v_{h_2}, \ldots, v_{h_N}$ and v_r be the values in the $h_1, h_2, \ldots, h_N$ and r bits respectively. Let p be input by the user. Compute

$$h_1 = (x_1^{min} - p * l_1) + \frac{v_{h_1}}{2^{b_{h_1}}} * (2 * p \ + \ 1) \ l_1$$
$$h_2 = (x_2^{min} - p * l_2) + \frac{v_{h_2}}{2^{b_{h_2}}} * (2 * p \ + \ 1) \ l_2$$
$$\vdots$$
$$h_N = (x_N^{min} - p * l_N) + \frac{v_{h_N}}{2^{b_{h_N}}} * (2 * p \ + \ 1) \ l_N$$

Step 2 : Computation of maximum and minimum distances.
Compute the distances of all the vertices of the *inner rectangle* from $(h_1, h_2, \ldots, h_N)$. Let d_1 and d_2 be the minimum and maximum of these values.
Step 3 : Whether center lies within the *inner rectangle* or not.
If $(h_1, h_2, \ldots, h_N)$ lies within the *inner rectangle*, go to step 5. (i.e., if $x_1^{min} \leq h_1 \leq x_1^{max}$, $x_2^{min} \leq h_2 \leq x_2^{max}$, ... and $x_N^{min} \leq h_N \leq x_N^{max}$, go to step 5). Otherwise go to step 4.
Step 4 : Center outside *inner rectangle* - compute radius.
Compute r in the range $[d_1, d_2]$. Or,

$$r = d_1 + \frac{v_r}{2^{b_r}} * (d_2 - d_1).$$

Go to step 6.
Step 5 : Center inside *inner rectangle* - compute radius.
Compute r in the range $[0, d_2]$. Or,

$$r = \frac{v_r}{2^{b_r}} * d_2.$$

Step 6 : Proceed

Figure 2.17: Steps for decoding the parameters of hyperspheres

correctly classified by the string. The conventional selection, crossover and mutation operators are applied till a termination criterion (string with zero misclassification) is achieved, or a maximum number of iterations have been executed. As before, an elitist strategy is used.

2.7.2 Implementation

The variation of recognition scores with C during testing for *ADS 1* and *Vowel* data, when circular and spherical segments respectively are used for constituting the class boundaries provided by the *GA-classifier* is presented in Tables 2.17 and 2.18. Keeping the other genetic parameters the same, p is chosen to be 8.

Comparing with the results in Tables 2.6 and 2.11, it is found on an average that for both these cases, use of higher order surfaces helps to improve the recognition over those with hyperplanes. This is expected, since the class boundaries of both these data sets are non-linear in nature. Overall, for *ADS 1*, three circular surfaces are required to model the decision boundary. (This conforms to our findings in Section 2.6.2 after application of redundancy removal step, when hyperplanes are used for modeling the class boundaries, where three lines were found to be necessary.) The remaining surfaces for the cases with $C > 3$ are redundant, except for $C = 6$, where four surfaces are required. Note that, because of the associated overfitting, the score for $C = 6$ is also poorer.

For *Vowel* data, unlike the case with hyperplanes, no redundant spherical surfaces are found, even with $C = 8$. The recognition scores are also consistently better than those with hyperplanes (see Tables 2.18 and 2.11). Thus it appears that even $C = 8$ is not sufficient for proper approximation of the class boundaries of *Vowel* data. This also seems to be appropriate, in view of the overlapping classes for *Vowel* data (Fig. 2.8).

2.8 Discussion and Conclusions

A method of generating class boundaries in $\mathbb{R}^N$, $N \geq 2$, using GAs has been described in this chapter, along with its demonstration on both artificial and real life

Table 2.17: Variation of recognition scores (%) during testing with C for *ADS 1*, with $perc = 10$, $p = 8$

Class	Surfaces						
	8	7	6	5	4	3	2
1	96.37	96.37	87.43	95.18	94.20	97.58	98.79
2	96.59	96.59	97.72	87.21	87.50	86.36	55.68
Overall	96.41	96.41	89.24	93.20	93.02	95.62	91.23

Table 2.18: Variation of recognition scores (%) during testing with C for *Vowel* data with $perc = 10$

Class	Surfaces						
	8	7	6	5	4	3	2
δ	67.69	32.31	50.77	50.77	24.61	0.00	0.00
a	85.24	86.42	81.48	65.43	91.35	95.06	0.00
i	85.22	87.09	86.45	80.64	84.51	87.74	93.54
u	94.11	89.71	91.91	92.65	88.23	91.91	89.71
e	75.40	70.58	71.65	76.47	72.72	82.35	73.79
o	69.75	72.84	69.75	69.75	67.28	64.19	79.01
Overall	76.23	76.08	75.97	75.44	74.55	75.82	67.81

data having overlapping, and non-overlapping, non-convex 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 maximum likelihood classifier (which is well known for discriminating overlapping classes), k-NN classifier and MLP (where both the k-NN classifier and the MLP with hard limiters are known to provide piecewise linear boundaries) is also presented. The results of the proposed algorithm are seen to be comparable to, often better than, them in discriminating both overlapping and non-overlapping, non-convex regions. The effectiveness of the classifier is also demonstrated on a high dimensional data *Cancer* data set. (Although we have presented the results of the k-NN classifier for k=$\sqrt{n}$ for the reason mentioned in Section 2.3.3, experiments were also conducted with k = 1 and 3. For example in the case of *Cancer* data, the corresponding recognition scores were found to be 95.45% and 95.60% respectively.)

Instead of hyperplanes, some other higher order surface can also be used for approximating the class boundaries. Such an attempt has also been presented here where hyperspherical segments are used for modeling the classes. Its effectiveness has been demonstrated on *ADS 1* and *Vowel* data. The results point to the fact that higher order segments provide superior performance as compared to linear ones when the class boundaries are complicated. Since in most real life situations, the class boundaries are unknown, and generally non-linear, use of any higher order surfaces is better and natural. This provides better approximation, albeit at the cost of increased computational complexity, since an additional parameter has to be encoded in this case. For example, for N dimensional space, $N + 1$ parameters must be encoded when using hyperspherical segments for modeling the class boundaries.

The generalization capability of the decision boundary is demonstrated diagrammatically. Its variation with different values of H is also presented empirically. It is observed that an increase in the value of H or C does not necessarily result in an increase in the generalization capability of the classifier. The reason is that a large number of surfaces 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 chapter 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 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 [28] that as the number of iterations goes towards 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 GAs [77] 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 reported in Chapter 4 of this thesis. 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 [126].

The GA based classifier is designed for points in $\mathbb{R}^N$. 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. Here 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 [29] 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. A modification of the fitness function by incorporating the information on relative position of the boundary from the training data may constitute another part of further investigation.

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 depending on the value of b_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.

In this context it should be mentioned that it has been proved in [28] that an elitist model of GA will surely provide the optimal string as the number of iterations goes to infinity, provided conventional mutation operation is incorporated. This proof is irrespective of the values of the control parameters. However, since in practice, it is not possible to have infinitely many iterations, proper selection of control parameters is necessary for good performance and fast convergence.

Regarding the sensitivity analysis of the system with respect to the discretization parameters, one may note that such an analysis would also require the knowledge of relationship between the genetic parameters and the discretization parameters. However, considering the difficulty in proper selection of these parameters, and the fact that the GA process is a random one and the theoretical results on its convergence are asymptotic [28], such an investigation would be gigantic in nature.

Another aspect of the work that needs to be investigated is the replacement of the conventional GA used here by other more sophisticated operators and models. A

preliminary attempt in this regard has already been made where CHC algorithm [57], which incorporates an incest prevention technique, is used as the search and optimization tool. For example, in the case of *ADS 1* and *Vowel* data sets, the results obtained during testing with CHC are 94.82% and 69.81% respectively when 10% of the data set is used for training.

Since simulated annealing [45, 107, 212] is another very well known search and optimization strategy which also deals with an encoding of the parameters of the search space in strings, it is appropriate that an investigation be carried out to test its effectiveness for designing a classifier based on the aforesaid approach. Such a study has been made in [20], where it is found that the two perform comparably when the criterion to be minimized is the number of misclassified training data points. However, if the error rate (obtained by dividing the number of misclassified points by the size of the training data) is to be minimized, then the performance of the simulated annealing based classifier degrades considerably, while that of the *GA-classifier* remains unchanged.

Chapter 3

Theoretical Analysis of the GA-classifier

3.1 Introduction

In this chapter we provide a theoretical investigation of the performance of the *GA-classifier* described in Chapter 2. This mainly includes establishing the relation between Bayes classifier (where the class *a priori* probabilities and conditional densities are known) and the *GA-classifier* under limiting conditions. It is known from the literature that Bayes classifier [209] is the best possible classifier if the class conditional densities and the *a priori* probabilities are known. No classifier can provide better performance than Bayes classifier under such conditions. (Let a denote the error probability associated with Bayes classifier or the Bayes decision rule. If any other decision rule is used and the error probability associated with that rule is b, then $a \leq b$. Bayes classifier is the best classifier because of this property.) In practice, it is difficult to use Bayes classifier because the class conditional densities and the *a priori* probabilities may not be known. Hence new classifiers are devised and their performances are compared to that of the Bayes classifier. The desirable property of any classifier is that it should approximate or approach the Bayes classifier under limiting conditions. Such an investigation for MLP was performed in [180] to show that the MLP, when trained as a classifier using back-propagation, approximates the Bayes optimal discriminant function. As already mentioned in Section 2.3.3, k-NN classifier also approaches the Bayes classifier under certain conditions as the size of the training data set goes to infinity.

There are many ways in which the performance of a classifier is compared to that of the Bayes classifier. One such way is to investigate the behavior of the error rate (defined as the ratio of the number of misclassified points to the size of the data set) as the size of the training data goes to infinity, and check whether the limiting error rate is equal to a. Such an investigation is performed in this chapter.

It is proved in this chapter that for $n \rightarrow \infty$ (i.e., for sufficiently large training data set) and for a sufficiently large number of iterations, the error rate of the *GA-classifier* during training will approach the Bayes error probability. However, as mentioned above, the Bayes classifier, with known class distributions and *a priori* probabilities, always provides the best generalization capability. It has also been shown theoretically that for n tending to infinity, the number of hyperplanes found by the *GA-classifier* to constitute the decision boundary will be equal to the op-

timum number of hyperplanes (i.e., which provide the Bayes decision boundary) if exactly one partition provides the Bayes error probability. Otherwise the number of hyperplanes found by the *GA-classifier* will be greater than or equal to the optimum value.

The theoretical findings have also been experimentally verified on a number of training data sets following triangular and normal distributions having both linear and non-linear boundaries. Performance on test data sets has also been studied. Experiments have been conducted using different values of H. Instead of using hyperplanes, circular surfaces in two dimensions, have also been considered as constituting the decision boundary. The generalization capability of the classifier has been studied as a function of the class *a priori* probabilities (for two class problems). The empirical findings show that as the training data size (n) increases, the performance of the *GA-classifier* approaches that of Bayes classifier for all the data sets.

In Section 3.2 we present a theoretical investigation to find a relationship between the *GA-classifier* and Bayes classifier. A critical discussion of the proof is also given in the same section. This is followed by a description of the process of redundancy elimination along with the associated study of the relationship between the optimal number of hyperplanes required to model the Bayes boundary and those provided by the *GA-classifier* in Section 3.3. Section 3.4 contains the experimental results and their analysis. Finally, the conclusions are presented in Section 3.5.

3.2 Relationship with Bayes Error Probability

In this section we investigate theoretically the relationship between the *GA-classifier* and Bayes classifier. The mathematical notations and preliminary definitions are described first. This is followed by the claim that for $n \rightarrow \infty$, the error rate of the *GA-classifier* during training will approach the Bayes error probability a. Finally some critical comments about the proof are mentioned.

Let there be k classes $C_1, C_2, \ldots, C_k$ with *a priori* probabilities $P_1, P_2, \ldots, P_k$ and continuous class conditional densities $p_1(\mathbf{x}), p_2(\mathbf{x}), \ldots, p_k(\mathbf{x})$. Let the mixture density

be

$$p(\mathbf{x}) = \sum_{i=1}^{k} P_i p_i(\mathbf{x}). \tag{3.1}$$

(According to the Bayes rule, a point is classified to class i iff

$$P_i p_i(\mathbf{x}) \geq P_j p_j(\mathbf{x}), \quad \forall j = 1, \ldots, k \ \ and \ \ j \neq i.)$$

Let $X_1, X_2, \ldots, X_n, \ldots$ be independent and identically distributed (i.i.d) N dimensional random vectors with density $p(\mathbf{x})$. This indicates that there is a probability space $(\Omega, \mathcal{F}, Q)$, where $\mathcal{F}$ is a σ field of subsets of Ω, Q is a probability measure on $\mathcal{F}$, and

$$X_i : (\Omega, \mathcal{F}, Q) \longrightarrow (I\!R^N, B(I\!R^N), P) \quad \forall i = 1, 2, \ldots, \tag{3.2}$$

such that

$$\begin{aligned} P(A) &= Q(X_i^{-1}(A)) && (3.3) \\ &= \int_A p(\mathbf{x}) d\mathbf{x} && (3.4) \end{aligned}$$

$\forall A \in B(I\!R^N)$ and $\forall i = 1, 2, \ldots$.

Here $B(I\!R^N)$ is the Borel σ field of $I\!R^N$.

Let

$$\begin{aligned} \mathcal{E} = \{E : E = (S_1, S_2, \ldots, S_k), S_i \subseteq I\!R^N, S_i \neq \emptyset \\ \forall i = 1, \ldots, k, \bigcup_{i=1}^{k} S_i = I\!R^N, S_i \bigcap S_j = \emptyset, \forall i \neq j\}. \end{aligned} \tag{3.5}$$

$\mathcal{E}$ provides the set of all partitions of $I\!R^N$ into k sets as well as their permutations, i.e.,

$$E_1 = (S_1, S_2, S_3 \ldots, S_k) \in \mathcal{E},$$

$$E_2 = (S_2, S_1, S_3, \ldots, S_k) \in \mathcal{E},$$

then $E_1 \neq E_2$. Note that the decision rule $E = (S_{i1}, S_{i2}, \ldots, S_{ik})$ denotes that each S_{ij}, $1 \leq j \leq k$, is the decision region corresponding to class C_j.

Let $E_0 = (S_{01}, S_{02}, \ldots, S_{0k}) \in \mathcal{E}$ be such that each S_{0i} is the decision region corresponding to the class C_i in $\mathbb{R}^N$ and these are obtained by using Bayes decision rule. Then

$$a = \sum_{i=1}^{k} P_i \int_{S_{0i}^c} p_i(\mathbf{x}) d\mathbf{x} \leq \sum_{i=1}^{k} P_i \int_{S_{1i}^c} p_i(\mathbf{x}) d\mathbf{x} \tag{3.6}$$

$\forall E_1 = (S_{11}, S_{12}, \ldots, S_{1k}) \in \mathcal{E}$. Here a is the error probability obtained using the Bayes decision rule.

It is known from the literature that such an E_0 exists and it belongs to $\mathcal{E}$ because Bayes decision rule provides an optimal partition of $\mathbb{R}^N$ and for every such $E_1 = (S_{11}, S_{12}, \ldots, S_{1k}) \in \mathcal{E}$, $\sum_{i=1}^{k} P_i \int_{S_{1i}^c} p_i(\mathbf{x}) d\mathbf{x}$ provides the error probability for $E_1 \in \mathcal{E}$. One such E_0 is given below :

$$\begin{aligned} S_{01} &= \{\mathbf{x} : P_1 p_1(\mathbf{x}) \geq P_j p_j(\mathbf{x}) \ \ \forall\, j \neq 1\}, \\ S_{0i} &= \{\mathbf{x} : P_i p_i(\mathbf{x}) \geq P_j p_j(\mathbf{x}) \ \ \forall\, j \neq i\} \bigcap [\bigcup_{l=1}^{i-1} S_{0l}]^c, \quad i = 2, 3, \ldots, k. \end{aligned}$$

Note that E_0 need not be unique.

Assumptions : Let H_{opt} be a positive integer and let there exist H_{opt} hyperplanes in $\mathbb{R}^N$ which provide an optimal decision rule (i.e., an optimal E_0). Let H_{opt} be known *a priori.*

Let

$$\begin{aligned} \mathcal{C}_n(\omega) = \{&E : E = (S_1, S_2, \ldots, S_k), E \in \mathcal{E}, \\ &\text{E is generated by } H_{opt} \text{ hyperplanes for the training sample set} \\ &(X_1(\omega), X_2(\omega), \ldots, X_n(\omega))\}. \end{aligned} \tag{3.7}$$

Note that $\mathcal{C}_n(\omega)$ is uncountable. Note also that each E in $\mathcal{C}_n(\omega)$ provides a value of the number of misclassifications for the training sample set $(X_1(\omega), X_2(\omega), \ldots, X_n(\omega))$. Let us now define a relation for the elements in $\mathcal{C}_n(\omega)$ as follows :

An element $E_1 = (S_{11}, S_{12}, \ldots, S_{1k})$ of $\mathcal{C}_n(\omega)$ is said to be related to element $E_2 = (S_{21}, S_{22}, \ldots, S_{2k})$ of $\mathcal{C}_n(\omega)$ if

$$X_i(\omega) \in S_{1j} \Rightarrow X_i(\omega) \in S_{2j} \ \ \forall\, i = 1, 2, \ldots, n, \ \forall\, j = 1, 2, \ldots, k. \tag{3.8}$$

Let for an element $E_1 = (S_{11}, S_{12}, \ldots, S_{1k})$, $\mathcal{S}_{E_1} = \{E : E \in \mathcal{C}_n(\omega),\ E \text{ is related to } E_1\}$. Note that for any two points E_1 and E_2 in $\mathcal{C}_n(\omega)$, either $\mathcal{S}_{E_1} = \mathcal{S}_{E_2}$ or $\mathcal{S}_{E_1} \bigcap \mathcal{S}_{E_2} = \emptyset$. Note that the number of such different $\mathcal{S}_{E_i}$s are finite since the training sample set is finite.

Let the *GA-classifier* be applied on the training sample set $(X_1(\omega), X_2(\omega), \ldots, X_n(\omega))$ in such a way that the total number of possible rules under consideration, denoted by $\mathcal{T}_n$, gives rise to atleast one element of the different $\mathcal{S}_{E_i}s$. Note that the *GA-classifier* described in this thesis satisfies this criterion with suitable choice of the discretization parameters. Hence, if the number of iterations is sufficiently large, the *GA-classifier* will provide the minimum number of misclassified points [28], and correspondingly an element $E \in \mathcal{C}_n(\omega)$ (in one of the $\mathcal{S}_{E_i}s$).

Let $\mathcal{A} = \{A : A \text{ is a set consisting of } H_{opt} \text{ hyperplanes in } I\!R^N\}$. Let $\mathcal{A}_0 \subseteq \mathcal{A}$ be such that it provides all the optimal decision rules in $I\!R^N$ i.e., each element of $\mathcal{A}_0$ provides the regions which are also obtained using the Bayes decision rule. Note that each $A \in \mathcal{A}$ generates several elements of $\mathcal{E}$ which result from different permutations of the list of k regions. Let $\mathcal{E}_A \subseteq \mathcal{E}$ denote all possible $E = (S_1, S_2, \ldots, S_k) \in \mathcal{E}$ that can be generated from A.

Let

$$G = \bigcup_{A \in \mathcal{A}} \mathcal{E}_A. \tag{3.9}$$

Note that achieving optimality in $\mathcal{E}$ is equivalent to achieving optimality in G since H_{opt} hyperplanes also provide Bayes decision regions. Thus, henceforth, we shall concentrate on G.

Note that $\mathcal{T}_n(\omega)$ denotes all possible decision rules that are considered in the *GA-classifier* when the training sample set is $\{X_1(\omega), X_2(\omega), \ldots, X_n(\omega)\}$. Also, $\mathcal{T}_n(\omega) \subseteq \mathcal{T}_{n+1}(\omega)$ for every n. $\lim_{n\to\infty} \mathcal{T}_n(\omega) \subseteq G$. It can also be seen that as $n \to \infty$, there will be some decision rules in $\mathcal{T}_n(\omega)$ which will converge to an optimal decision rule (i.e., an optimal E_0), everywhere ω.

Let $Z_{iE}(\omega) = 1$ if $X_i(\omega)$ is misclassified when E is used as a decision rule where $E \in G$, $\forall \omega \in \Omega$.

$= 0$ otherwise.

Let $f_{nE}(\omega) = \frac{1}{n}\sum_{i=1}^{n} Z_{iE}(\omega)$, when $E \in G$ is used as a decision rule.

That is, $f_{nE}(\omega)$ is the error rate associated with the decision rule E for the training sample set $\{X_1(\omega), X_2(\omega), \ldots, X_n(\omega)\}$.

In particular, $f_{nE}(\omega) = \frac{1}{n}\sum_{i=1}^{n} Z_{iE}(\omega)$ when $E \in \mathcal{T}_n(\omega)$.

Let

$$f_n(\omega) \quad = Inf\{f_{nE}(\omega) : E \in \mathcal{T}_n(\omega)\}. \tag{3.10}$$

It is to be noted that the pattern classification algorithm mentioned in Chapter 2 uses $n * f_{nE}(\omega)$, the total number of misclassified points, as the objective function which it attempts to minimize. This is equivalent to searching for a suitable $E \in \mathcal{T}_n(\omega)$ such that the term $f_{nE}(\omega)$ is minimized, i.e., for which $f_{nE}(\omega) = f_n(\omega)$. As already mentioned, it is known that for infinitely many iterations the elitist model of GAs will certainly be able to obtain such an E.

Theorem 3.1 : For sufficiently large n, $f_n(\omega) \not> a$, (i.e., for sufficiently large n, $f_n(\omega)$ cannot be greater than a) almost everywhere.

Proof : Let $Y_i(\omega) = 1$ if $X_i(\omega)$ is misclassified according to Bayes rule $\forall \omega \in \Omega$.
$= 0$ otherwise.

Note that $Y_1, Y_2, \ldots, Y_n, \ldots$ are i.i.d random variables. Now

$$\begin{aligned} Prob(Y_i = 1) &= \sum_{j=1}^{k} Prob(Y_i = 1/X_i \; is \; in \; C_j) P(X_i \; is \; in \; C_j) \\ &= \sum_{j=1}^{k} P_j Prob(\omega : X_i(\omega) \in S_{0j}^c \; given \; that \; \omega \in C_j) \\ &= \sum_{j=1}^{k} P_j \int_{S_{0j}^c} p_j(\mathbf{x}) d\mathbf{x} = a. \end{aligned}$$

Hence the expectation of Y_i, $E(Y_i)$ is given by

$$E(Y_i) = a, \; \forall i.$$

Then by using Strong Law of Large Numbers [56], $\frac{1}{n}\sum_{i=1}^{n} Y_i \longrightarrow a$ almost everywhere.

i.e., $P(\omega : \frac{1}{n}\sum_{i=1}^{n} Y_i(\omega) \not\longrightarrow a) = 0$.

Let $B = \{\omega : \frac{1}{n}\sum_{i=1}^{n} Y_i(\omega) \longrightarrow a\} \subseteq \Omega$. Then $Q(B) = 1$.

Note that $f_n(\omega) \leq \frac{1}{n}\sum_{i=1}^{n} Y_i(\omega)$, $\forall n$ and $\forall \omega$, since the set of regions $(S_{01}, S_{02}, \ldots, S_{0k})$ obtained by the Bayes decision rule is also provided by some $A \in \mathcal{A}$ and consequently it will be included in G and $\mathcal{T}_n(\omega)$. Note that $0 \leq f_n(\omega) \leq 1$, $\forall n$ and $\forall \omega$. Let $\omega \in B$. For every $\omega \in B$, $U(\omega) = \{f_n(\omega); n = 1, 2, \ldots\}$ is a bounded, infinite set. Then by Bolzano-Weierstrass theorem [5], there exists an accumulation point of $U(\omega)$. Let y = Sup$\{y_0 : y_0$ is an accumulation point of $U(\omega)\}$. From elementary mathematical analysis we can conclude that $y \leq a$, since $\frac{1}{n}\sum_{i=1}^{n} Y_i(\omega) \longrightarrow a$ almost everywhere and $f_n(\omega) \leq \frac{1}{n}\sum_{i=1}^{n} Y_i(\omega)$. Thus it is proved that for sufficiently large n, $f_n(\omega)$ cannot be greater than the error probability a for $\omega \in B$.

Corollary : $\lim_{n\to\infty} f_n(\omega) = a$ for $\omega \in B$.

Proof : Consider a sequence $f_{i_k}(\omega)$ in $U(\omega)$ such that $\lim_{n\to\infty} f_{i_k}(\omega) = b$ and $b < a$. We shall show that $b < a$ is not possible. Suppose $b < a$ is possible. Let $\varepsilon = \frac{a-b}{4}$. Then for $k > k_0$, $f_{i_k}(\omega) \in (b - \varepsilon, b + \varepsilon)$. That is

$$f_{i_k}(\omega) < a - \varepsilon, \quad \forall\ k > k_0. \tag{3.11}$$

For sufficiently large n, and for a given ω, whatever decision rule $E \in G$ that one considers,

$$\frac{\text{number of misclassified samples with respect to } E}{n} \geq a - \varepsilon, \tag{3.12}$$

from the properties of the Bayes classifier. This contradicts (3.11). Thus $b < a$ is impossible. Hence for any convergent sequence in $U(\omega)$ with limit as b, $b \geq a$. From Theorem 3.1, we have $b \leq a$, since b is an accumulation point. This shows that $b = a$. Since every bounded infinite set has atleast one accumulation point, and every accumulation point in $U(\omega)$ here is a, $\lim_{n\to\infty} f_n(\omega) = a$.

Remarks

- It is to be noted that when the class conditional densities and the *a priori* probabilities are known, the error probability associated with any decision rule whatsoever (including the one provided by the *GA-classifier*) can never be

less than a. Our claim is that the error rate of the *GA-classifier* (obtained by dividing the number of misclassified data points by the size of the training data) during training is less than or equal to a. As the size of the training set goes to infinity, from the corollary to Theorem 3.1, the error rate provided by the *GA-classifier* approaches a. Thus the performance of the *GA-classifier* approaches that of the Bayes classifier as n goes to infinity. This indicates that the boundary generated by the *GA-classifier* approaches the Bayes boundary if it exists and is unique under limiting conditions. In case the Bayes boundary is not unique, then the *GA-classifier* will generate any such boundary where its error rate is a.

- The proof is not typical of the GA based classifier. It will hold for any other classifier where the criterion is to find a decision boundary that minimizes the number of misclassified points. For example, if simulated annealing is used instead of genetic algorithm, then too the results will hold under limiting conditions.

- Instead of hyperplanes, any other higher order surface could have been used in *GA-classifier* for obtaining the decision boundaries, provided the number of higher order surfaces is finite and known, and the Bayes boundary is indeed provided by such surfaces. It would lead to only minor modifications to the proof presented earlier, with the basic inference still holding good.

- Note that although the class conditional density functions are assumed to be continuous, the above results will still hold good even if the density functions are not continuous.

- The proof assumes that there exists H_{opt} hyperplanes that can model the Bayes decision boundary. Although theoretically any curved surface can be approximated by a number of hyperplanes, this may become an unrealistic assumption in practice. However, results are included in this chapter to demonstrate that as the size of the data set increases, the hyperplanes provided by the *GA-classifier* approximate the highly curved Bayes boundary better.

- From Theorem 3.1 and its corollary, it is evident that for small values of n, the

result of the *GA-classifier* is such that during training

$$\frac{\text{number of misclassified points}}{n} < a,$$

and also

$$a - \frac{\text{number of misclassified points}}{n}$$

could be significant. As n goes to infinity,

$$\frac{\text{number of misclassified points}}{n} \rightarrow a.$$

Thus, while conducting the experiments with known *a priori* probabilities and densities, we would find that during training

$$\frac{\text{number of misclassified points}}{n} < a,$$

and

$$a - \frac{\text{number of misclassified points}}{n}$$

could be large or small depending on n being small or large respectively. It is to be noted that

$$\frac{\text{number of misclassified points}}{n} < a$$

is not a complete indication of the performance of the classifier. One should check the performance using test data sets also. Several experiments have been conducted in this chapter to judge the performance of the classifier *vis a vis* Bayes classifier when the experiments included the testing phase also. ♣

The results proved analytically in this section are also verified experimentally in Section 3.4 under different situations. Note that in the proof described above it is assumed that H_{opt} is known *a priori*. However in practice, as mentioned earlier, H_{opt} may not be known. In that case we try to use a conservative value, which may lead to the presence of some redundant hyperplanes in the resultant decision boundary. The next section describes a method of eliminating the redundant hyperplanes thereby generating the optimum number of hyperplanes H_{GA} of the *GA-classifier*. A relationship of H_{GA} and the optimum hyperplanes H_{opt} yielding the Bayes boundary is also established in the next section.

3.3 Relationship between H_{opt} and H_{GA}

Here, we first present a technique for obtaining H_{GA} hyperplanes from the initial over estimation of H. Subsequently we establish that H_{GA} is equal to H_{opt} when there exists exactly one partition of the feature space that provides the Bayes error probability a. In case more than one partition can provide the Bayes error probability, then all we can say is that H_{GA} will be greater than or equal to H_{opt}.

In this context one must note that a method of redundancy removal was described in Section 2.5, which can not be used here for finding H_{GA}, because it was dependent on the order in which the hyperplanes were considered for removal. Hence, it did not guarantee that the minimal number of hyperplanes would always be provided. On the other hand, the method that will be described here is exhaustive, and guarantees that the optimal value H_{GA} will be obtained.

3.3.1 Obtaining H_{GA} from H

A hyperplane is considered to be redundant if its removal has no effect on the recognition score of the classifier for the given training data set. In order to arrive at the optimal number of hyperplanes, one of the ways is to consider first of all, all possible combinations of the H hyperplanes. For each such combination, the first hyperplane is removed and it is checked whether the remaining hyperplanes can successfully classify all the patterns. If so, then this hyperplane is deleted, and the test is repeated for the next hyperplane in the combination.

Obviously, testing all possible combinations results in an algorithm with exponential complexity in H. To avoid this, a branch and bound technique is adopted where the search within a combination is discontinued (before considering all the H hyperplanes) if the number of hyperplanes found to be non redundant so far, is greater than or equal to the number of hyperplanes declared to be non redundant by some earlier combination. The complexity may be reduced further by terminating the algorithm if the combination being tested provides a set of $\lceil \log_2 k \rceil$ non redundant hyperplanes, since this is the minimum number of hyperplanes that is required in order to provide k distinct regions (see Section 2.2). This method guarantees removal of all the redundant hyperplanes from the decision boundary, thereby yielding

exactly H_{GA} hyperplanes.

3.3.2 How H_{GA} is related to H_{opt}

Let H_{GA} be the number of hyperplanes (after elimination of redundancy) found by the *GA-classifier* to provide an error rate $= f_n(\omega)$, which is less than or equal to a (Bayes error probability) when $n \rightarrow \infty$. If H_{opt} is the optimal number of hyperplanes, then obviously H_{GA} cannot be less than H_{opt}. It is now our task to ascertain whether $H_{GA} > H_{opt}$ or $H_{GA} = H_{opt}$. For this we must first consider the following situations:

a) The number of partitions which provides the Bayes error probability is exactly one. Since this partition, formed from H_{opt} hyperplanes, provides the Bayes error probability, which is known to be optimal, hence for $n \rightarrow \infty$, the regions provided by the H_{GA} hyperplanes must be exactly same as the regions provided by the H_{opt} hyperplanes. Thus H_{GA} must be same as H_{opt} for large values of n.

b) On the contrary, the number of partitions that provide the Bayes error probability may be greater than one. For example, this will be the case if at least one of the k classes is totally disconnected from the other classes. Another example is provided in Appendix A. In these cases, the regions provided by the H_{GA} hyperplanes may not be identical to the ones provided by the H_{opt} hyperplanes. Consequently, H_{GA} can be greater than H_{opt} for such situations although the classifier still provides an error rate equal to $f_n(\omega)$.

3.3.3 Some Points Related to n and H

In practice we always deal with finite data sets (or finite n). Obviously, in that case, additional hyperplanes, beyond H_{opt}, may be placed appropriately in order to further reduce the number of misclassified points; at the cost of possibly reduced generalizability. These hyperplanes will not be eliminated by the redundancy removal process. However, as n increases, as expected the effect of introducing additional hyperplanes will decrease and also the performance of the classifier in terms of the test data will gradually improve. In the limiting case, for $n \rightarrow \infty$, only the optimum number of hyperplanes with a specific arrangement will provide the requisite decision boundary. Any additional hyperplane will obviously be detected as redundant. At

the same time, the generalization of the classifier will be optimum.

3.4 Experimental Results

Extensive empirical results are provided in this section, which are seen to conform to the theoretical findings in Section 3.2. It is also found experimentally that the decision boundary obtained by *GA-classifier* approaches that of Bayes classifier as n increases. Data sets following triangular and normal distributions are considered, having both linear and non-linear class boundaries. All the data sets have considerable amount of overlap. In a part of the experiment, instead of lines, circular segments (in two dimension) are considered as the constituting elements of the decision boundaries. Its objective is to demonstrate the theoretical claims made in Section 3.2 for higher order surfaces. Effect of class *a priori* probability on the recognition score has also been experimentally investigated.

Different situations considered for conducting the experiments are as follows :

i : The decision boundary is provided by H_{opt} hyperplanes, and H_{opt} is known *a priori.*

ii : The decision boundary is provided by H_{opt} higher order surfaces, and H_{opt} is known *a priori.*

iii : It is known that the decision boundary can be approximated by H_{opt} hyperplanes but the value of H_{opt} is not known.

iv : It is known that the decision boundary can be approximated by H_{opt} higher order surfaces but the value of H_{opt} is not known.

v : It is known that no finite number of hyperplanes can approximate the decision boundary.

vi : It is known that no finite number of higher order surfaces can approximate the decision boundary.

vii : Nothing is known about the given data set. In that case we may try to approximate the boundary by a fixed number of hyperplanes or any other higher order surfaces.

This section is divided into three parts. The description of the data sets is given in the first part. The decision boundaries and recognition scores (during training and testing) obtained by *GA-classifier* (using linear as well as circular surfaces) are then compared with those of Bayes classifier for different sizes of the training data in the second part. Finally, the third part demonstrates the variation of the generalization capability of the classifier as a function of the class *a priori* probability, for two class problems.

3.4.1 Data sets

Four types of data sets are used for empirically verifying the theoretical results in Section 3.2.

Data Set 1 : This is a two dimensional $(X - Y)$ data set, generated using a triangular distribution of the form shown in Fig. 3.1 for the two classes, 1 and 2. The range for class 1 is [0,2]× [0,2] and that for class 2 is [1,3]× [0,2] with the corresponding peaks at (1,1) and (2,1) respectively. Fig. 3.1 shows the distribution along the X axis since only this axis has discriminatory capability. The distribution along the X axis may be formally quantified as

$$\begin{aligned} f_1(x) &= 0 \quad for\ x \leq 0 \\ f_1(x) &= x \quad for\ 0 < x \leq 1 \\ f_1(x) &= 2 - x \quad for\ 1 < x \leq 2 \\ f_1(x) &= 0 \quad for\ x > 2 \end{aligned}$$

for class 1. Similarly for class 2

$$\begin{aligned} f_2(x) &= 0 \quad for\ x \leq 1 \\ f_2(x) &= x - 1 \quad for\ 1 < x \leq 2 \\ f_2(x) &= 3 - x \quad for\ 2 < x \leq 3 \\ f_2(x) &= 0 \quad for\ x > 3. \end{aligned}$$

The distribution along the Y axis for both the classes is

$$f(y) = 0 \quad for\ y \leq 0$$

$$f(y) = \quad y \quad for\ 0 < y \leq 1$$
$$f(y) = \quad 2 - y \quad for\ 1 < y \leq 2$$
$$f(y) = \quad 0 \quad for\ y > 2.$$

If P_1 is the *a priori* probability of class 1 then using elementary mathematics, we can show that Bayes classifier will classify a point to class 1 if its X coordinate is less than $1 + P_1$. This indicates that the Bayes decision boundary is given by

$$x = 1 + P_1. \tag{3.13}$$

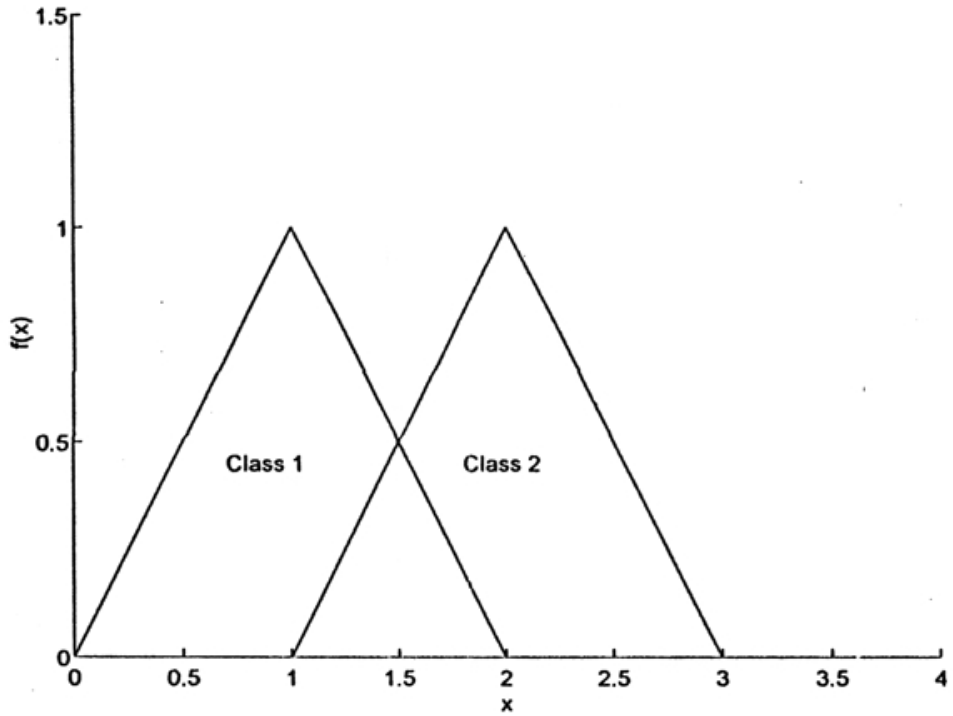


Figure 3.1: Triangular distribution along X axis for *Data Set 1* having two classes.

Data Set 2 : Fig. 3.2 shows the normally distributed data set consisting of two classes. The mean (μ_1,μ_2) and covariance values (Σ_1, Σ_2) for the two classes are :

$$\mu_1 = (0.0, 0.0),\ \mu_2 = (1.0, 0.0) \text{ and}$$

$$\Sigma_1 = \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}$$

$$\Sigma_2 = \begin{pmatrix} 4.0 & 0.5 \\ 0.5 & 4.0 \end{pmatrix}$$

respectively.

The classes are assumed to have equal *a priori* probability (= 0.5). Mathematical analysis shows that the Bayes decision boundary for such a distribution of points will be of the following form :

$a_1{x_1}^2 + a_2{x_2}^2 + 2a_3x_1x_2 + 2b_1x_1 + 2b_2x_2 + c = 0.$

The Bayes boundary for such a class distribution is also shown in Fig. 3.2.

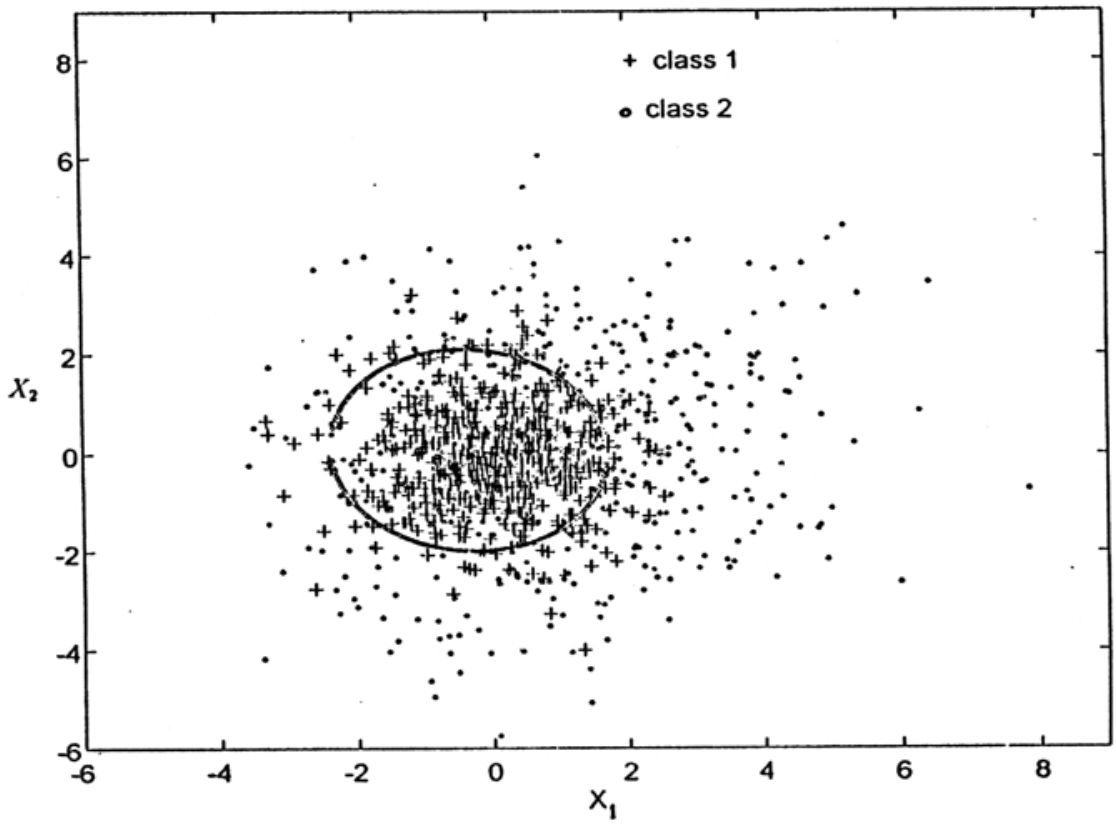


Figure 3.2: *Data Set 2* for $n = 1000$ along with the Bayes decision boundary for two classes.

Data Set 3 : To consider a multi class problem, a nine class triangular distribution of data points is considered. All the classes are assumed to have equal *a priori* probabilities (= 1/9). The $X - Y$ ranges for the nine classes are as follows :

Class 1 : [-3.3, -0.7] × [0.7, 3.3]
Class 2 : [-1.3, 1.3] × [0.7, 3.3]
Class 3 : [0.7, 3.3] × [0.7, 3.3]
Class 4 : [-3.3, -0.7] × [-1.3, 1.3]
Class 5 : [-1.3, 1.3] × [-1.3, 1.3]
Class 6 : [0.7, 3.3] × [-1.3, 1.3]

Class 7 : [-3.3, -0.7] × [-3.3, -0.7]
Class 8 : [-1.3, 1.3] × [-3.3, -0.7]
Class 9 : [0.7, 3.3] × [-3.3, -0.7]

Thus the domain for the triangular distribution for each class and for each axis is 2.6. Consequently, the height will be $\frac{1}{1.3}$ (since $\frac{1}{2} * 2.6 * height = 1$). We have chosen the classes in such a way that (i) each class has some overlap with everyone of its adjacent classes and (ii) the class boundaries are represented in a simple way (e.g., $x = 1$). In any direction the non-overlapping portion is 1.4 units, the overlapping portion is 1.2 units and the length of the Bayes decision region is 2 units. The resulting Bayes boundary along with the data set is shown in Fig. 3.3.

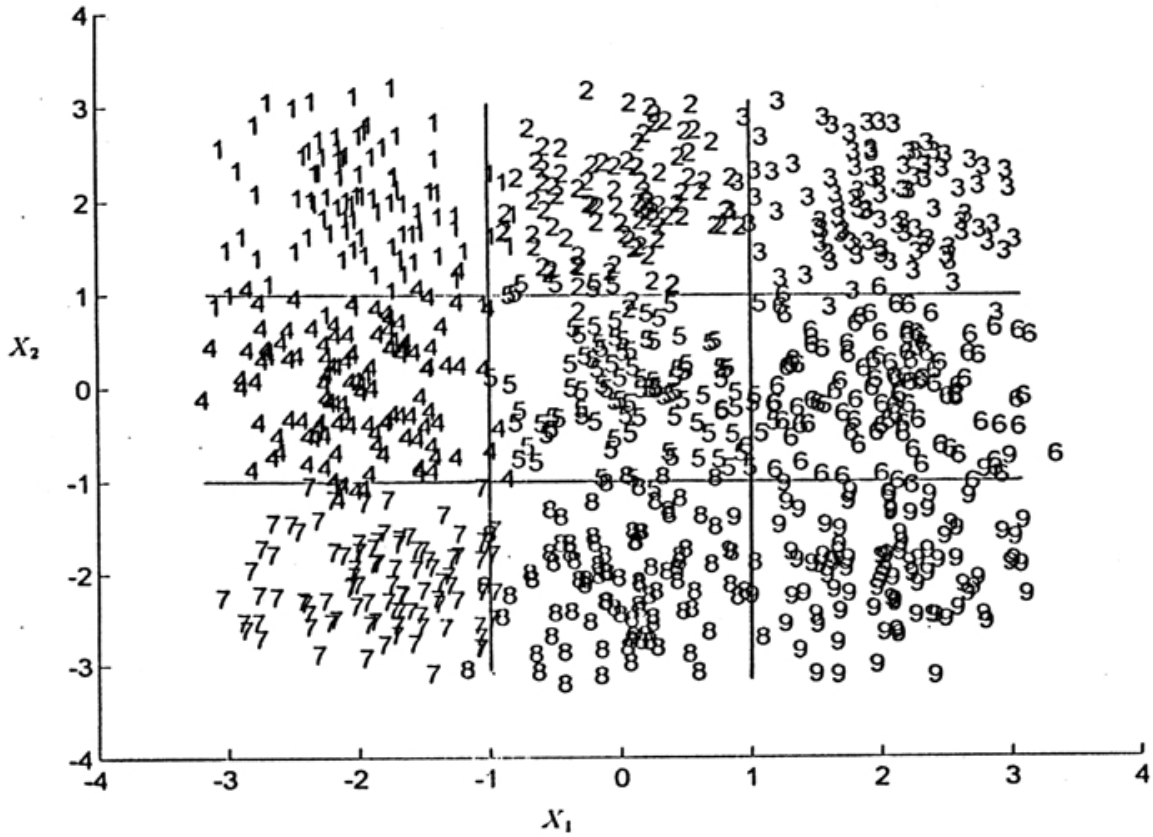


Figure 3.3: *Data Set 3* for $n = 900$ along with the Bayes decision boundary for nine classes.

Data Set 4 : This two class two dimensional data set is used specifically for the purpose of investigating the generalization capability of the *GA-classifier* as a function of the *a priori* probability of class 1. The data set is normally distributed with the following parameters :

$$\mu_1 = (0.0, 0.0),\ \mu_2 = (1.0, 0.0) \text{ and}$$

$$\Sigma_1 = \Sigma_2 = \Sigma = \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}.$$

Since $\Sigma_1 = \Sigma_2$, the Bayes boundary will be linear [67] of the following form :

$$(\mu_2 - \mu_1)^T \Sigma^{-1} X + \frac{1}{2}(\mu_1^T \Sigma^{-1} \mu_1 - \mu_2^T \Sigma^{-1} \mu_2). \ \clubsuit$$

Note :

- For each data set and for a particular value of n, two different training data sets are generated using different seed values. Training is performed with each data set for five different initial populations. This means that a total of 10 runs are performed for a given n. The results of the *GA-classifier* presented here are average values over these 10 runs.

- The test data set comprises 1000 points. Testing of the *GA-classifier* is performed for each of the 10 training runs. Results shown are the average values over these 10 runs.

- Roulette wheel selection strategy with elitism is used in this investigation. Due to memory constraints (data sets range from 100 - 5000 points in size), small population size of 20 is chosen. Consequently, as before, a high crossover probability (μ_c) of 0.8 is fixed for single point crossover scheme. The mutation probability value (μ_m) varies over the range of [0.015, 0.333], initially having a high value, then gradually decreasing and finally increasing again in the later stages of the algorithm. 100 iterations of the GA are performed for each μ_m. A total of 1500 iterations is performed.

  As mentioned in Section 2.8, an elitist model of GA will always provide the optimal string as the number of generations goes to infinity, provided the probability of going from any population to the one containing the optimal string is greater than zero. This proof is irrespective of the population sizes and different probability values. However, appropriate selection of GA parameters is necessary in view of the fact that it has to be terminated after a finite number of iterations.

3.4.2 Learning the Class Boundaries and Performance on Test Data

Utilizing linear surfaces

An extensive comparison of the performance of the *GA-classifier* with that of Bayes classifier is performed for *Data Set 1*, *Data Set 2* and *Data Set 3*. The following cases are considered for Bayes classifier :

Case 1 : The distributions as well as the class *a priori* probabilities (P_1 and P_2) are known.

Case 2 : The distribution is known, with $P_1 = P_2 = 0.5$.

Case 3 : The distribution is known, with $P_1 = n_1/n$ and $P_2 = n_2/n$. Here n_1 and n_2 are the number of points belonging to classes 1 and 2 respectively, $n = n_1 + n_2$.

Case 4 : Normal distribution with unequal covariance matrices, while P_1 and P_2 are known .

Case 5 : Normal distribution with unequal covariance matrices and $P_1 = P_2 = 0.5$.

Case 6 : Normal distribution with unequal covariance matrices, and $P_1 = n_1/n$ and $P_2 = n_2/n$, where n_1 and n_2 are defined above.

a) *Data Set 1 :* The different values of n considered are 100, 500, 1000, 1500, 2000, 3000 and 4000 and $P_1 = 0.4$. The Bayes boundary for this data set is a straight line ($x = 1.4$). The training recognition scores of Bayes classifier and *GA-classifier* are shown for $H = 1$ and 3 (Table 3.1). As expected, for each value of n, the recognition score during training of the *GA-classifier* with $H = 3$ was found to be at least as good as that with $H = 1$. Table 3.2 shows the recognition scores for the test data. Here, it is found that the Bayes classifier yields a better performance than the *GA-classifier*. Also, *GA-classifier* with $H = 3$ provides a consistently lower score than with $H = 1$. This is expected since a larger H leads to overfitting of the training data, thereby yielding better training performance (Table 3.1), while the generalization capability of the classifier degrades (Table 3.2).

It is seen from Table 3.1 that for all values of n, the recognition scores for *GA-classifier* during training are better than those of Bayes (all cases). However this difference in performance gradually decreases for larger values of n. This is demonstrated in Fig. 3.4, which plots α as a function of n, where α = overall training recognition score of the *GA-classifier* − overall recognition score of the Bayes clas-

sifier (case 1). This indicates that as n increases, the decision boundaries provided by the *GA-classifier* gradually approach the Bayes boundary. This observation is demonstrated in Figs 3.5, 3.6, 3.7 and 3.8 for $n = 100$, 1000, 2000 and 4000 respectively.

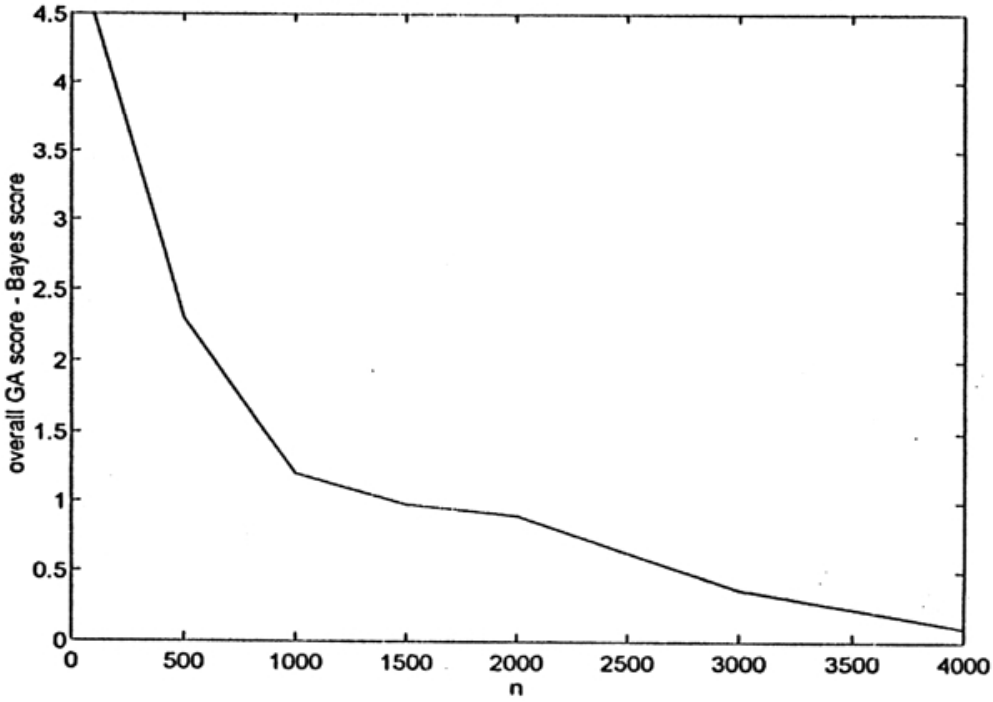


Figure 3.4: Variation of α (= overall GA score - Bayes score) with n for *Data Set 1*.

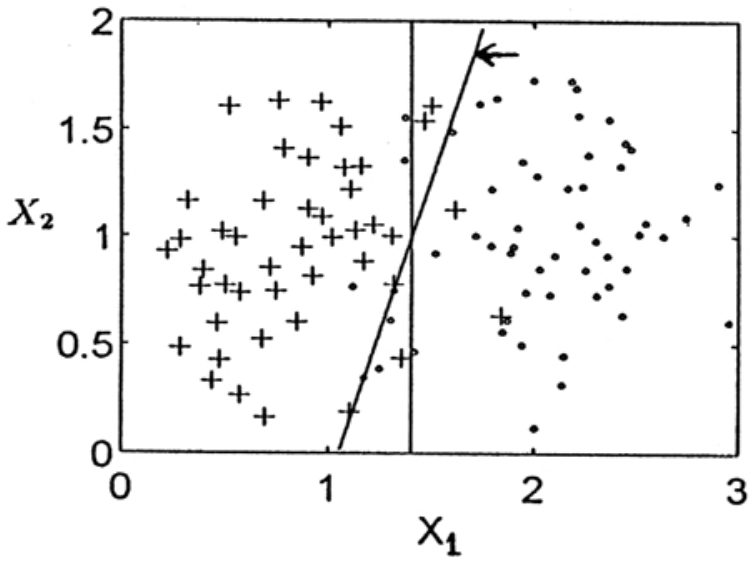


Figure 3.5: *Data Set 1* for $n = 100$ and the boundary provided by *GA-classifier* for $H = 1$ (marked with an arrow) along with Bayes decision boundary. Class 1 is represented by '+' and class 2 by 'o'.

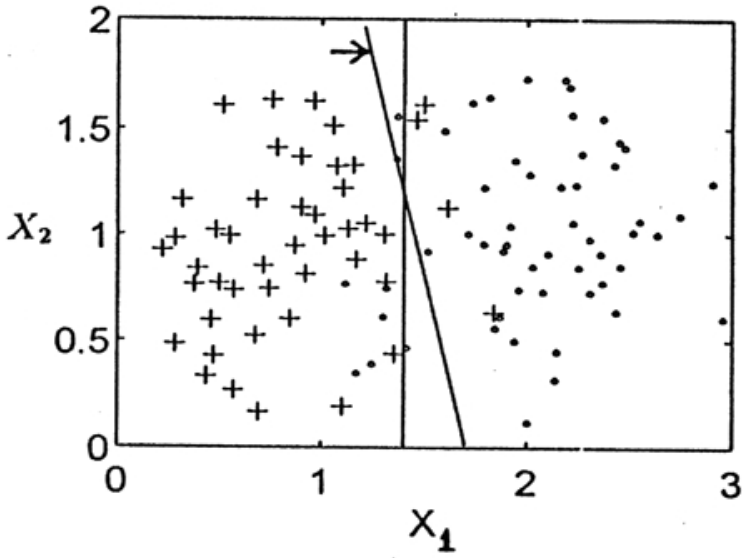


Figure 3.6: *Data Set 1* for $n = 1000$ and the boundary provided by *GA-classifier* for $H = 1$ (marked with an arrow) along with Bayes decision boundary. Class 1 is represented by '+' and class 2 by 'o'. (Only 100 data points are plotted for clarity.)

A point to be mentioned here is that although the result of Bayes classifier (case 1) is the ideal one considering the actual distribution and *a priori* probabilities, interestingly some of the other cases (e.g., case 5 for n=100, case 3 for n=500) are seen to produce better scores. A reason for this discrepancy may be because of statistical variations. It is also found that results for case 1 and case 4 are similar (having same values for n=100, 1000 and 1500, and close values for other values of n). The reason for this is that normal distribution with proper variance has been found to be able to approximate the triangular distribution closely.

b) *Data Set 2 :* Four values of n considered here are $n = 500$, 1000, 2000 and 5000. Cases 1, 3, 4 and 6 are investigated since only these are relevant in this context. Table 3.3 presents the results corresponding to them during training. Fig. 3.2 shows the data set for $n = 1000$ and the corresponding Bayes boundary. (It is found that the Bayes boundary totally surrounds class 1.) Due to practical limitations, we have considered H=4 and 6 only for executing the *GA-classifier*, although from the complexity of the decision surface it appears that a still larger number of lines may be required for proper approximation.

For smaller values of n (500 and 1000), it is found, as in the case of *Data Set 1*, that the *GA-classifier* yields a better score than the Bayes classifier for the training data set (Table 3.3). Fig. 3.9 shows the decision boundary obtained for n=1000 and

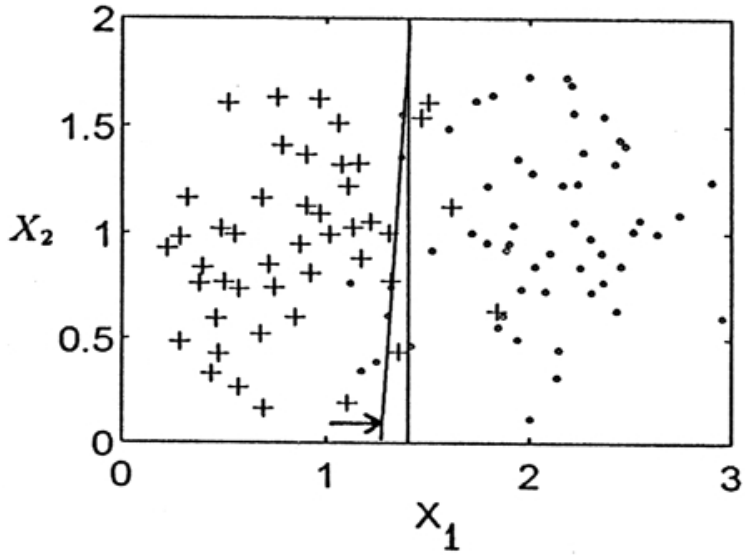


Figure 3.7: *Data Set 1* for $n = 2000$ and the boundary provided by *GA-classifier* for $H = 1$ (marked with an arrow) along with Bayes decision boundary. Class 1 is represented by '+' and class 2 by 'o'. (Only 100 data points are plotted for clarity.)

H =4, along with the Bayes boundary. Although all the four lines of *GA-classifier* are found to be necessary, and they surround class 1, the boundary formed by them could not approximate the Bayes boundary well. In spite of this fact, its recognition score during training is relatively larger than that of the Bayes classifier. Increasing H to a value 6 improves both the approximation of the boundary and the recognition score during training (Table 3.3).

For $n = 2000$, one out of the four lines is found to be redundant by the *GA-classifier* (Fig. 3.10) and they fail to surround class 1 for the same number (1500) of iterations. The training score is accordingly found to be lower than that of the Bayes classifier. For H=6 (Fig. 3.11), the recognition score during training exceeds the one obtained by Bayes, and the approximation to the Bayes boundary improves (although only 5 lines are utilized effectively).

For a further increase in n to 5000, the training recognition scores (Table 3.3) for both $H = 4$ and 6 are seen to be worse than Bayes scores for the same number of iterations. However, the approximation to Bayes boundary is seen to be much improved here as compared to $n = 1000$ and 2000. This is evident from Figs. 3.12 and 3.13, where $H = 6$, as expected, provides better performance than $H = 4$. From Figs. 3.10 and 3.13 it is interesting to note that one line is found to be redundant and at the same time the score is worse than that of Bayes. This may be attributed

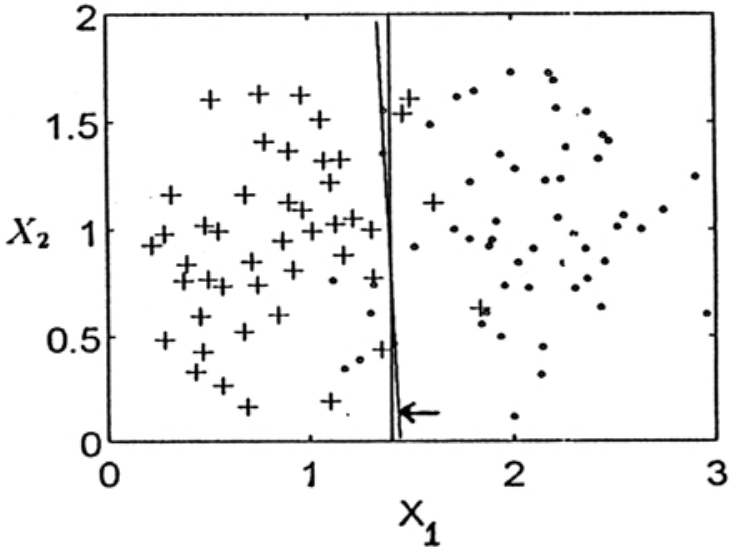


Figure 3.8: *Data Set 1* for $n = 4000$ and the boundary provided by *GA-classifier* for $H = 1$ (marked with an arrow) along with Bayes decision boundary. Class 1 is represented by '+' and class 2 by 'o'. (Only 100 data points are plotted for clarity.)

to the premature termination of GA.

Table 3.4 shows the recognition scores for *Data Set 2* during testing. Again, it is found that the *GA-classifier* provides poorer scores than Bayes classifier. Also, *GA-classifier* with $H = 6$ provides better performance than with $H = 4$. This is expected since $H = 6$ can better approximate the complex decision surface.

c) *Data Set 3 :* In order to extend the results for problems where $k > 2$, we utilized a nine class, two dimensional data set. The data set for $n = 1800$ along with the corresponding Bayes boundary is shown in Fig. 3.3. The experiments on training as well as test data sets have been conducted for $H = 4$ and $n = 450$, 900, 1350 and 1800. Only the training results are presented here. As before, the test results, show a superior performance of the Bayes classifier. These are not included here. Comparison with the Bayes classifier is made for case 1 only. The results for the overall training recognition scores are shown in Table 3.5. Fig. 3.14 shows the boundary obtained using the *GA-classifier* along with the Bayes decision boundary for n=1800 where it is seen that the four lines more or less approximate the Bayes boundary. It is found from Table 3.5 that although for each value of n the training scores of the *GA-classifier* and the Bayes classifier are comparable, the latter one appears to provide a slightly better performance. The *GA-classifier* is not able to approximate the Bayes boundary very closely. One of the reasons may again be the

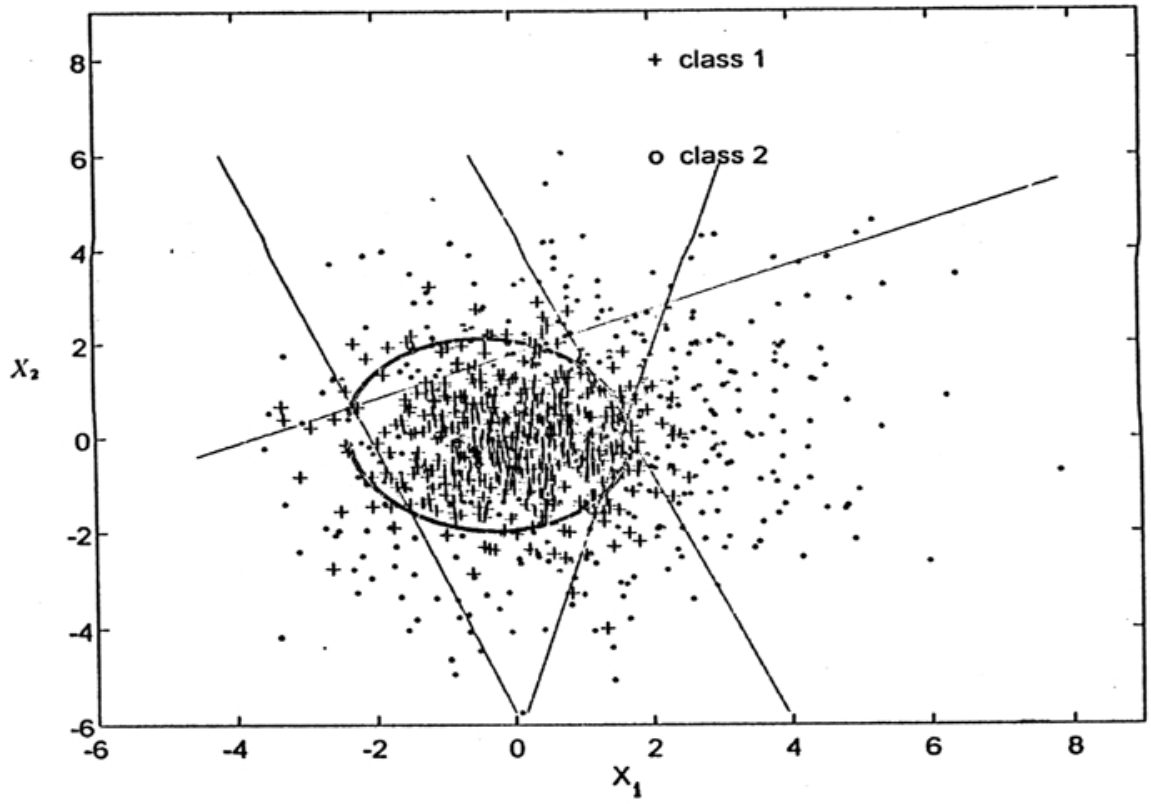


Figure 3.9: *Data Set 2* for $n = 1000$ and the boundary provided by *GA-classifier* for $H = 4$ along with Bayes decision boundary (circular one).

premature termination of the GA. Another factor may be the coding itself which does not allow encoding of the actual Bayes lines (this may be due to an insufficiency of the precision defined for the perpendicular distance of the lines).

Utilizing Higher Order Surfaces

It has been pointed out earlier (Section 3.2) that instead of approximating the decision boundary by hyperplanes, we can assume any other higher order surface with similar effect. This subsection describes the results of utilizing a fixed number of circular segments to constitute the decision surface in two dimensional space. The method of using circular/hyperspherical segments for modeling the class boundaries is described in Chapter 2.

The results obtained during both training and testing, when circular segments are used to model the decision boundaries for *Data Set 1* and *Data Set 2*, are shown in Table 3.6. The value of p is assumed to be 8 in order to be able to approximate any arbitrary boundary appropriately.

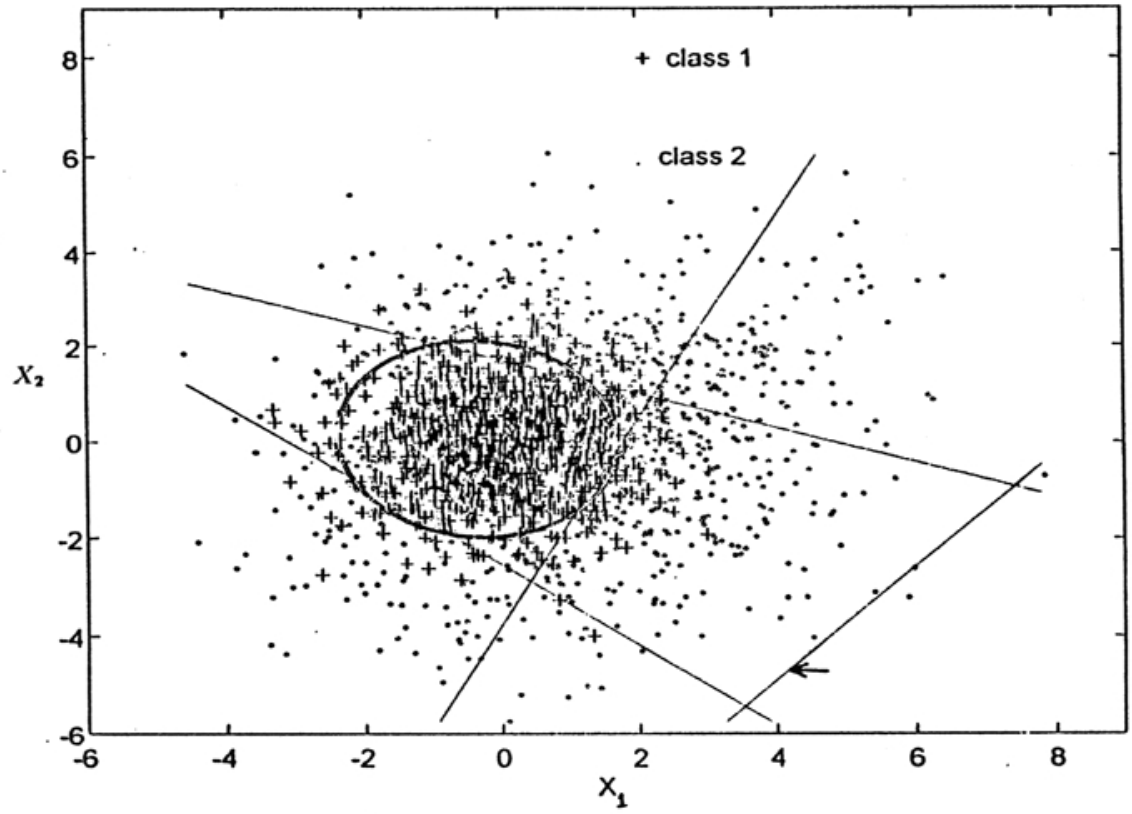


Figure 3.10: *Data Set 2* for $n = 2000$ and the boundary provided by *GA-classifier* for $H = 4$ along with Bayes decision boundary (circular one). The redundant line is marked with an arrow.

Interestingly, the approximation of the decision boundary for *Data Set 1* and *Data Set 2* could be made better using linear and circular segments respectively. This is followed for both data sets. Figs. 3.15 and 3.16 show the GA and Bayes boundaries obtained for *Data Set 1*, $n = 1000$ and $C = 1$, and *Data Set 2* , $n = 5000$ and $C = 6$ (where one segment is found to be redundant as in Fig. 3.13) respectively.

The test scores for *Data Set 1* with circular segments are found to be poorer compared to those with linear segments. The case is reversed for *Data Set 2* (except with $n = 2000$ and $H = 6$, where the two are comparable). This is expected since the decision boundary is linear for *Data Set 1*, while it is circular for *Data Set 2*. Note that, all these results are however inferior to the Bayes scores for the test data (Tables 3.2 and 3.4).

Note that for $p = 8$, the circular property of the segments does not appear to be fully exploited (Fig. 3.16) because of large radius. This was done in order to be able to approximate better any type of decision boundary using circular segments only.

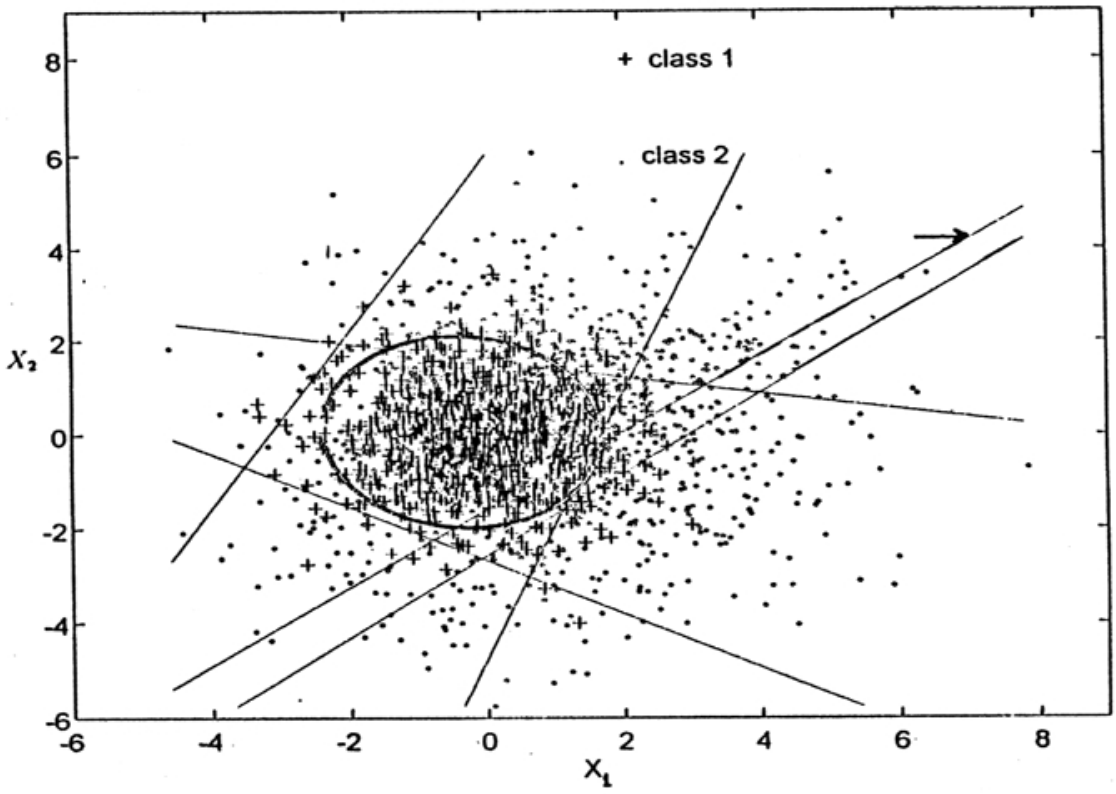


Figure 3.11: *Data Set 2* for $n = 2000$ and the boundary provided by *GA-classifier* for $H = 6$ along with Bayes decision boundary (circular one). The redundant line is marked with an arrow.

Fig. 3.17 shows another result for $n = 2000$, $p = 0$ and $C = 2$ which corresponds to smaller radius, and hence shows better circular characteristics. Here, the recognition scores over the training and the test data are 75.10% and 74.65% respectively.

3.4.3 Variation of Recognition Scores with P_1

In this section we investigate the variation of the test recognition scores of the *GA-classifier* with P_1 (*a priori* probability of class 1), and demonstrate that the variation is similar to that of the Bayes classifier for *Data Set 1*, *Data Set 2* and *Data Set 4*. Figs. 3.18 ($H = 1$ and 3), 3.19 ($H = 6$), and 3.20 ($H = 1$) show the results for the three data sets respectively. Here one training data set of size 200 and two test data sets of size 1000 are taken. Training of the *GA-classifier* is performed for five initial conditions. Subsequently, testing is performed for the two test data sets. The results shown are the average values over the ten runs. Note that the variation of the test recognition scores of the *GA-classifier* with P_1 is similar to that of Bayes

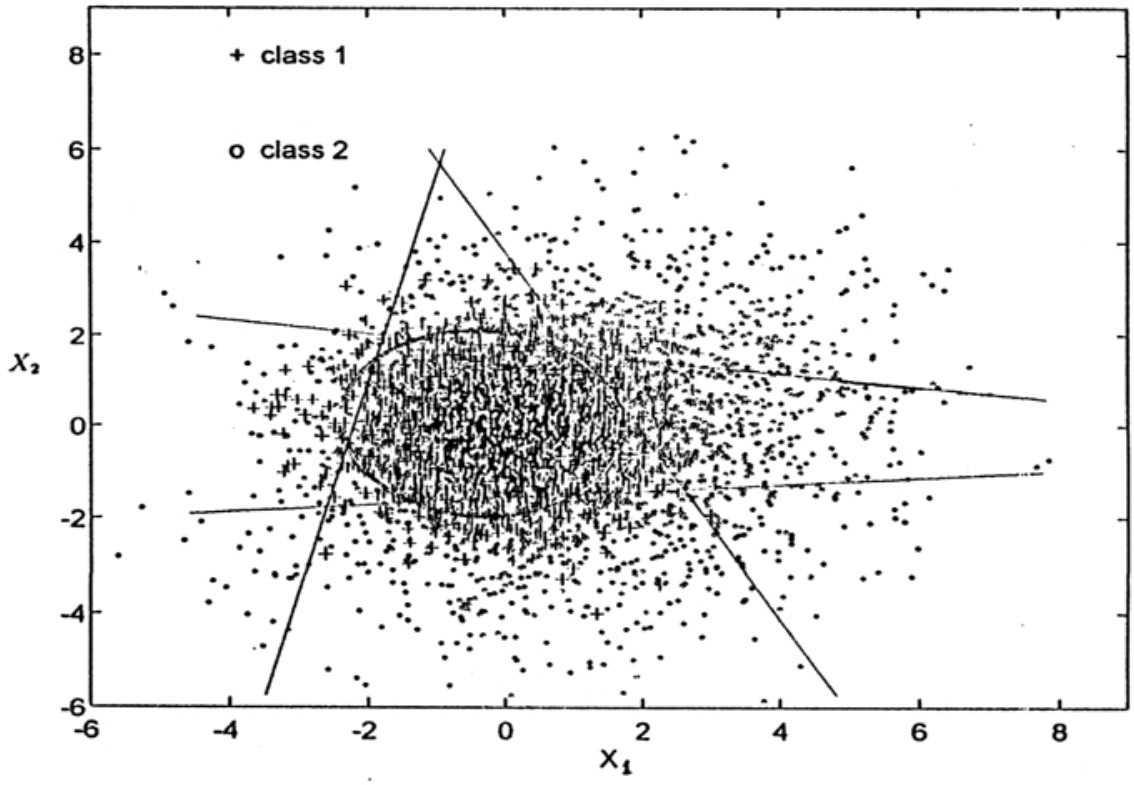


Figure 3.12: *Data Set 2* for $n = 5000$ and the boundary provided by *GA-classifier* for $H = 4$ along with Bayes decision boundary (circular one).

classifier for all the three data sets. (For the convenience of the readers, the above mentioned variation for Bayes classifier is discussed theoretically in Appendix B with reference to triangular and normal distribution of data points.)

It is shown in Appendix B that for triangular distribution the error probability varies symmetrically with P_1 having the maximum value for $P_1 = 0.5$. Similar observations were made in the investigation for *Data Set 1*. Consequently, results are presented for P_1 lying in the range [0,0.5] only. The test scores for the *GA-classifier* with H=3 are found to be consistently poorer than those with $H = 1$ for this data set (Fig. 3.18). It is to be mentioned here that recognition of the training data was better for $H = 3$ than for $H = 1$ (Table 3.1). This again supports the observations that increasing H beyond the optimum value leads to better training performance but poorer generalization of the *GA-classifier*.

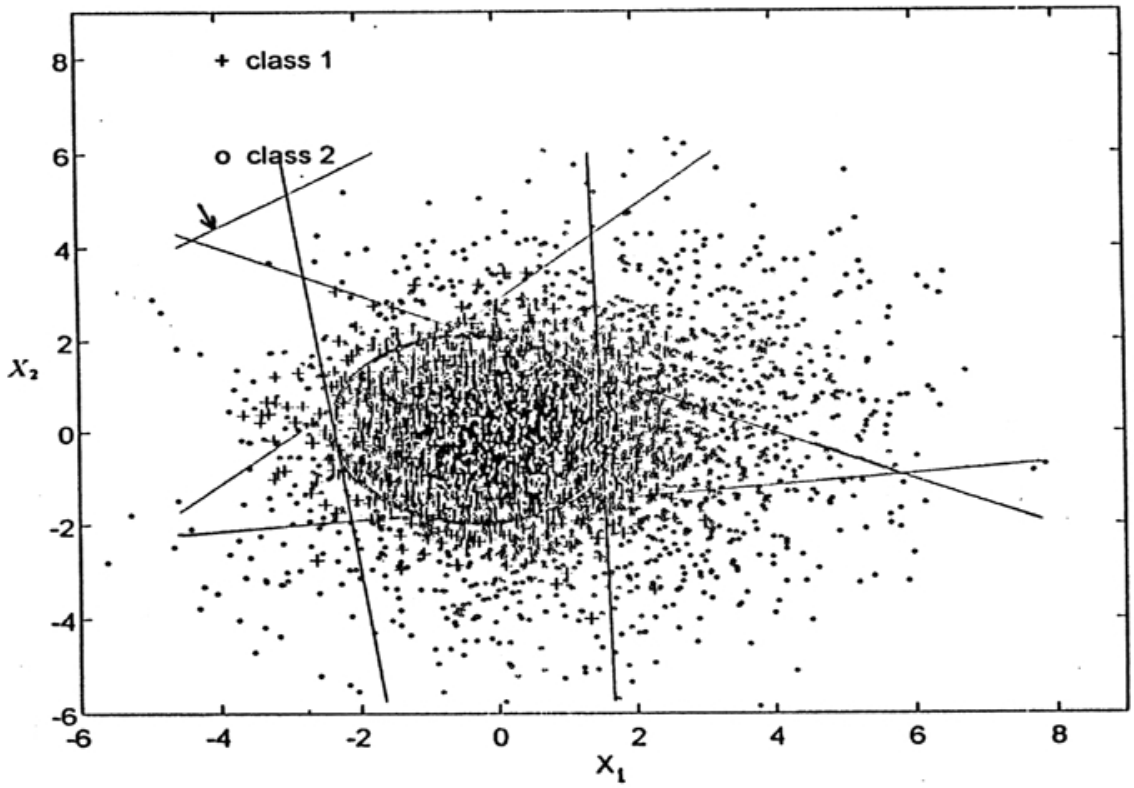


Figure 3.13: *Data Set 2* for $n = 5000$ and the boundary provided by *GA-classifier* for $H = 6$ along with Bayes decision boundary (circular one). The redundant line is marked with an arrow.

3.5 Discussion and Conclusions

A theoretical investigation is made here to find the relationship between a GA based classifier developed in Chapter 2 and the Bayes classifier. It is shown that for a sufficiently large number of training data points, and for a sufficiently large number of iterations the performance of the *GA-classifier* approaches that of the Bayes classifier when the number of hyperplanes (or higher order surfaces) giving rise to the Bayes decision regions are known.

It is also shown that the number of hyperplanes provided by the *GA-classifier* for constituting the decision boundary will be optimum if the minimal partition providing the Bayes error probability is unique. Otherwise it will be greater than or equal to the optimum value (see Section 3.3). In either case, the classifier will still be optimal in terms of the number of misclassified training data points.

The experimental results on the different distribution of overlapping data with both

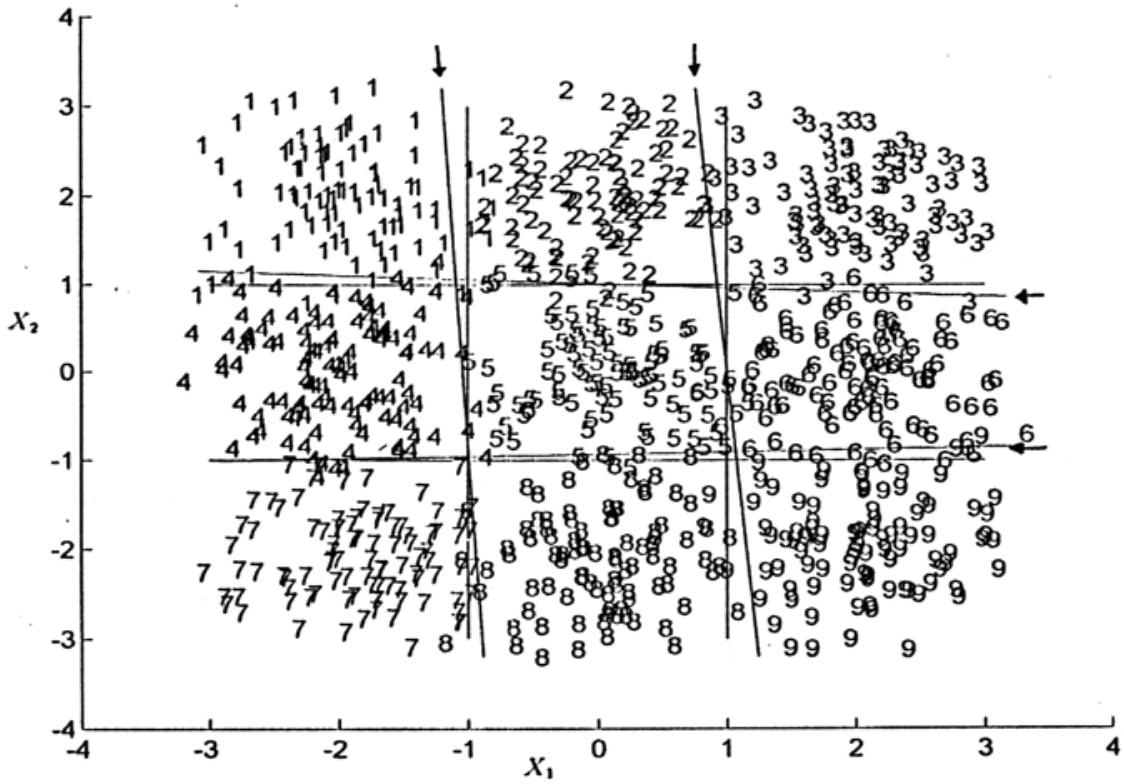


Figure 3.14: *Data Set 3* and the boundary provided by *GA-classifier* for $n = 1800$ and $H = 4$ (marked with arrows) along with Bayes decision boundary. (Only 900 points are shown.)

linear and non-linear boundaries show that the decision regions provided by *GA-classifier* gradually approach the ones provided by Bayes classifier as the number of training data points increases. Specifically, it has been found that for the *GA-classifier*

(i) $\frac{\text{number of misclassified points}}{n} \leq a$ during training,

(ii) $\frac{\text{number of misclassified points}}{n} \geq a$ during testing, and

(iii) $\frac{\text{number of misclassified points}}{n} \rightarrow a$ during training and testing as $n \rightarrow \infty$,

where a is the Bayes error probability. For *Data Set 2*, where the decision boundaries are non-linear, the *GA-classifier* using circular segments performs better than the one using linear surfaces (Table 3.6, for *Data Set 2*). On the other hand, for *Data Set 1* where, the decision boundary is actually linear, the *GA-classifier* with lines performs better. In real life, data sets are often overlapping with non-linear boundaries. Thus using higher order surfaces for modeling the decision boundary appears to be better in real life problems. This also conforms to the findings in

Chapter 2, where the approximation of the class boundaries could be made better by using hyperspherical surfaces for *ADS 1* and *Vowel* data.

As far as the generalization capability is concerned, the Bayes classifier generalizes better than the *GA-classifier* for all the data sets, and also for all values of *a priori* class probability P_1. It must be noted that when the class *a priori* probabilities and the probability distributions are known, the Bayes classifier provides the optimal performance. No other classifier, in that case, can provide a better performance. However, in most practical cases, these quantities are usually unknown. It is in such situations that the *GA-classifier*, which is non parametric in nature, can be utilized. It has already been shown in the previous chapter that the performance of the *GA-classifier* is comparable to (sometimes better than) that of some other standard classifiers.

Table 3.1: Comparative classwise and overall recognition scores (%) during training for *Data Set 1*

n	class	*GA-classifier*		Bayes classifier					
		H=1	H = 3	Case 1	Case2	Case 3	Case 4	Case 5	Case 6
100	1	88.77	89.26	81.08	85.94	81.08	82.60	88.77	83.71
	2	93.24	95.00	91.21	87.88	90.30	89.60	87.18	89.60
	overall	91.50	92.70	87.00	87.5	87.00	87.00	88.00	87.50
500	1	76.48	78.24	80.29	87.75	81.41	83.43	87.51	82.87
	2	95.69	95.59	89.39	84.73	90.15	87.87	85.38	87.85
	overall	88.00	88.65	85.70	85.90	86.20	86.10	86.20	86.00
1000	1	86.91	86.17	82.68	87.64	83.03	84.32	87.40	84.80
	2	91.01	92.00	92.93	87.71	92.75	91.84	87.81	91.58
	overall	89.35	89.67	88.80	87.70	88.85	88.80	87.65	88.50
1500	1	87.17	87.70	82.26	88.18	82.70	84.38	88.33	84.59
	2	90.73	90.45	92.07	88.17	92.10	91.11	87.94	90.54
	overall	89.08	89.35	88.10	88.16	88.33	88.10	88.10	88.16
2000	1	80.77	81.00	83.13	88.71	83.25	84.24	87.97	84.24
	2	94.81	94.93	91.71	86.77	91.54	91.37	87.52	91.29
	overall	89.15	89.36	88.25	87.55	88.20	88.50	87.70	88.45
3000	1	75.96	78.72	80.82	85.59	80.57	82.41	85.34	82.24
	2	95.68	94.02	91.86	87.65	91.86	90.75	87.76	90.81
	overall	87.83	87.90	87.46	86.83	87.36	87.43	86.80	87.40
4000	1	82.54	82.50	87.73	87.39	81.89	84.35	87.39	83.83
	2	91.81	92.08	91.52	86.88	91.93	90.18	86.67	90.59
	overall	88.22	88.25	88.12	87.07	88.05	87.92	86.95	87.97

Table 3.2: Comparative overall recognition scores (%) during testing for *Data Set 1*

n	*GA-classifier*		Bayes classifier
	H=1	H = 3	(case 1)
100	85.30	85.00	87.90
500	85.90	85.75	87.90
1000	86.20	85.87	87.90
1500	86.21	86.00	87.90
2000	86.55	86.41	87.90
3000	87.10	87.10	87.90
4000	87.32	87.20	87.90

Table 3.3: Comparative classwise and overall recognition scores (%) during training for *Data Set 2*

n	class	*GA-classifier*		Bayes classifier			
		H=4	H=6	Case 1	Case 3	Case 4	Case 6
500	1	89.23	89.23	85.38	86.54	85.77	87.31
	2	62.92	65.83	65.83	64.17	64.17	63.33
	overall	76.60	78.00	76.00	75.80	75.40	75.80
1000	1	83.92	84.90	83.33	84.12	83.33	84.71
	2	67.96	67.44	66.53	65.92	65.71	65.10
	overall	76.10	76.35	75.10	75.20	74.70	75.10
2000	1	87.98	87.58	85.27	83.73	84.77	85.22
	2	61.78	63.37	64.77	65.43	65.17	65.18
	overall	74.85	75.45	75.00	74.87	74.95	74.98
5000	1	85.18	83.04	85.42	85.50	85.74	85.82
	2	65.04	68.39	66.33	66.01	66.09	65.81
	overall	75.18	75.76	75.94	75.82	75.98	75.88

Table 3.4: Comparative overall recognition scores (%) during testing for *Data Set 2*

n	*GA-classifier*		Bayes classifier
	$H = 4$	$H = 6$	(case 1)
500	72.75	73.20	74.80
1000	72.92	73.35	74.80
2000	73.05	73.62	74.80
5000	73.34	74.15	74.80

Table 3.5: Comparative overall recognition scores (%) during training for *Data Set 3*

n	overall recognition score	
	GA-classifier (H=4)	Bayes classifier(case 1)
450	93.56	93.78
900	93.22	93.11
1350	90.08	92.52
1800	92.25	92.50

Table 3.6: Overall recognition scores (%) during training and testing for higher order surfaces

Data Set	n	No. of Circles C	*GA-classifier* (circular surface)		*GA-classifier* (linear surface $H = C$) from Tables 3.1 and 3.3	
			Training	Testing	Training	Testing
Data Set 1	1000	1	89.40	86.05	89.35	86.20
	2000	1	89.10	86.20	89.15	86.55
	2000	4	74.97	73.10	74.85	73.05
Data Set 2	2000	6	75.55	73.60	75.45	73.62
	5000	6	76.00	74.45	75.76	74.15

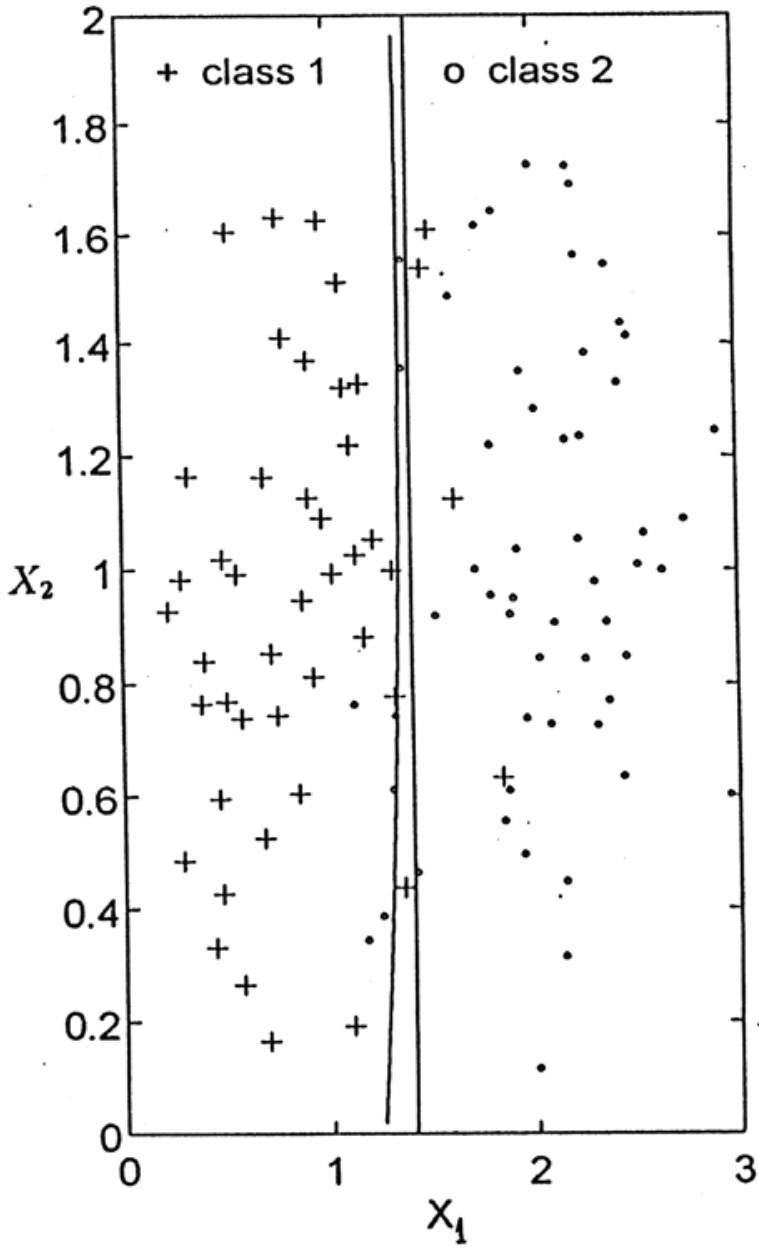


Figure 3.15: *Data Set 1* for $n = 1000$ and the circular boundary (left one) provided by *GA-classifier* when circular segments are considered to constitute the decision boundary for $C = 1$, $p = 8$ along with Bayes decision boundary. (Only 100 data points are plotted for clarity.)

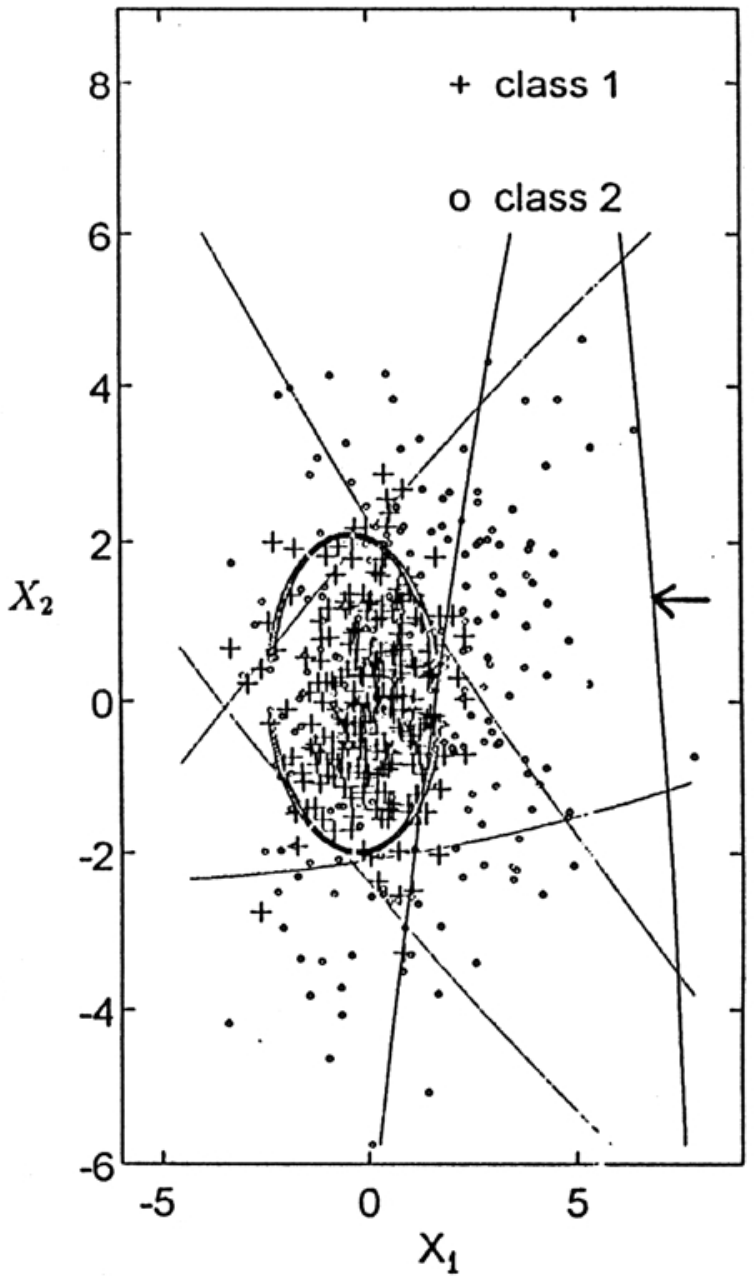


Figure 3.16: *Data Set 2* for $n = 5000$ and the boundary provided by *GA-classifier* when circular segments are considered to constitute the decision boundary for $C = 6$, $p = 8$ along with Bayes decision boundary. The redundant segment is marked with an arrow. (Only 500 data points are plotted for clarity.)

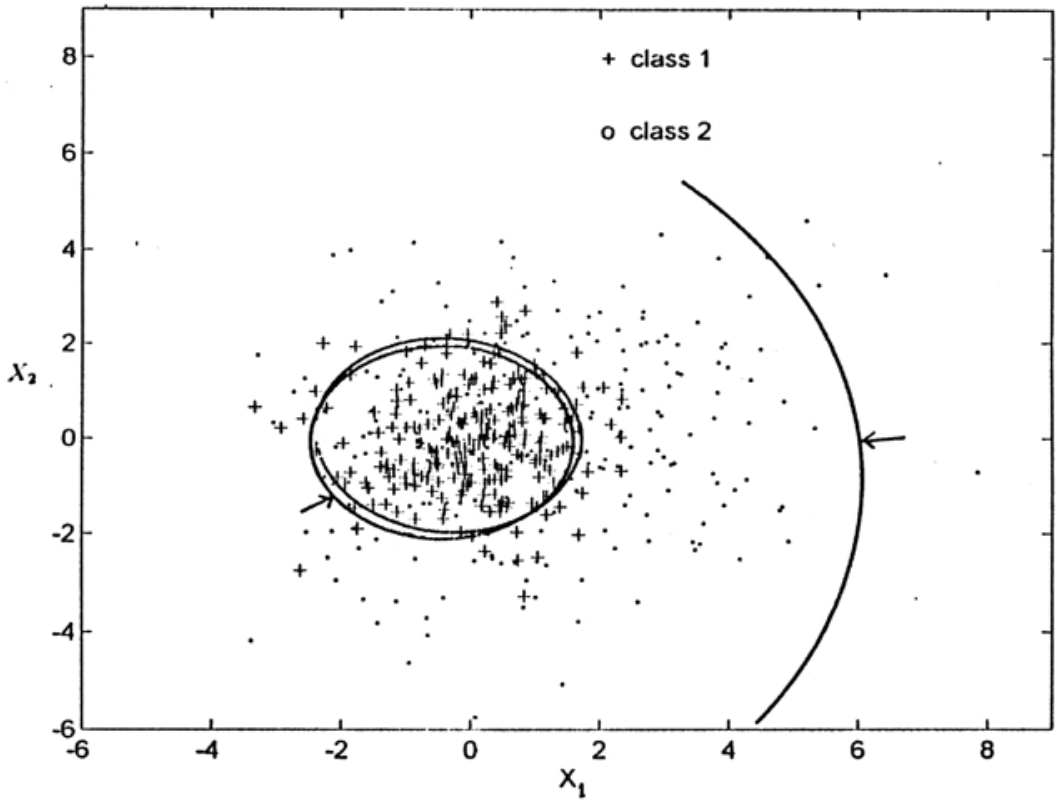


Figure 3.17: *Data Set 2* for $n = 2000$ and the boundary provided by *GA-classifier* (marked with arrows) when circular segments are considered for $C = 2$ and $p = 0$. The Bayes decision boundary is also shown. (Only 500 data points are plotted for clarity.) The arc on the right is obviously redundant.

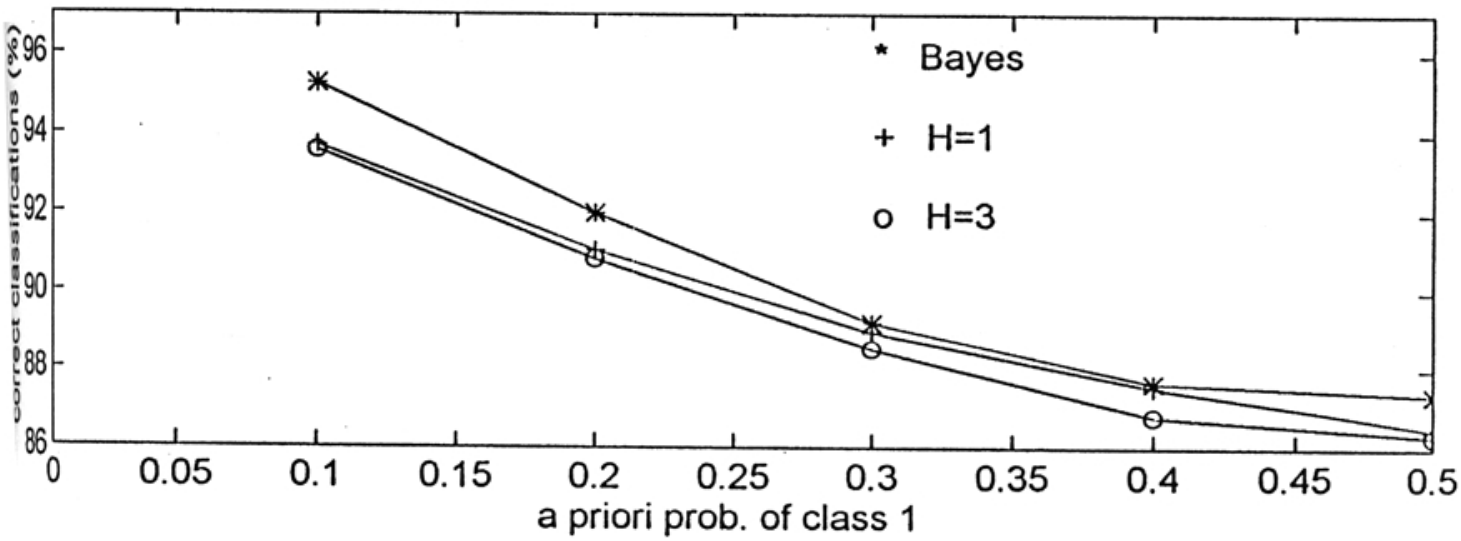


Figure 3.18: Variation of recognition score of test data with P_1 for *Data Set 1* corresponding to Bayes classifier and *GA-classifier* ($H = 1$ and 3). $n = 200$.

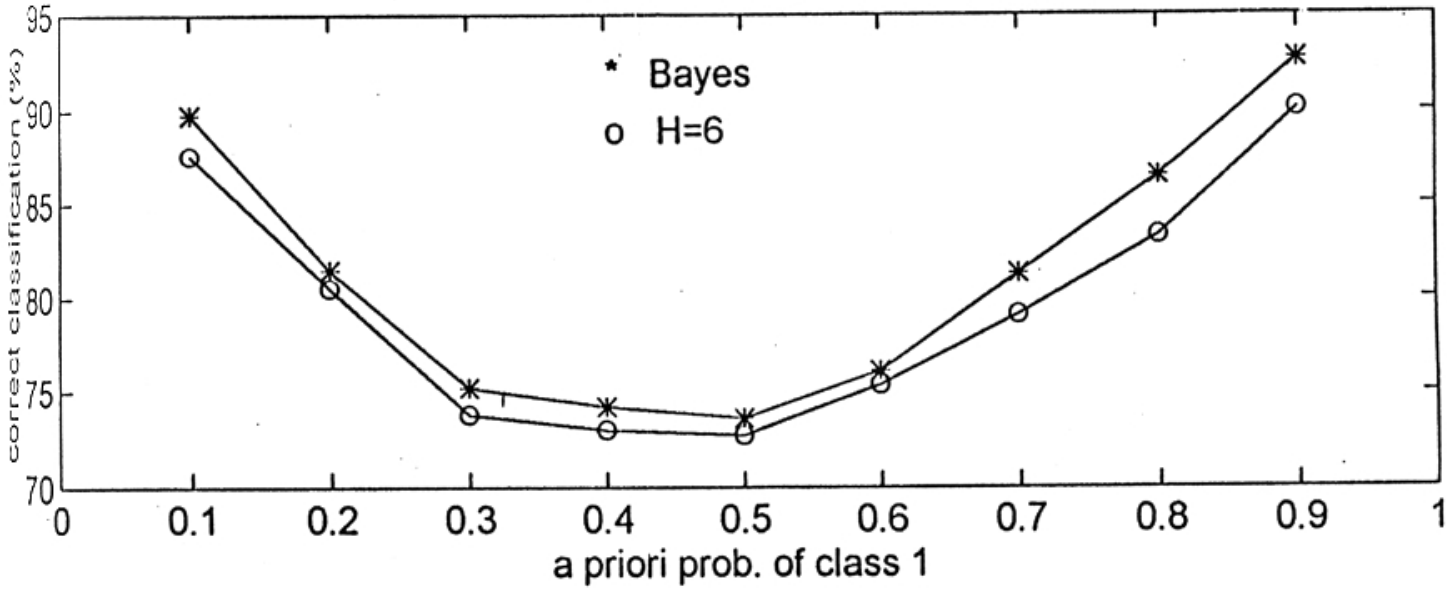


Figure 3.19: Variation of recognition score of test data with P_1 for *Data Set 2* corresponding to Bayes classifier and *GA-classifier* ($H = 6$). $n = 200$.

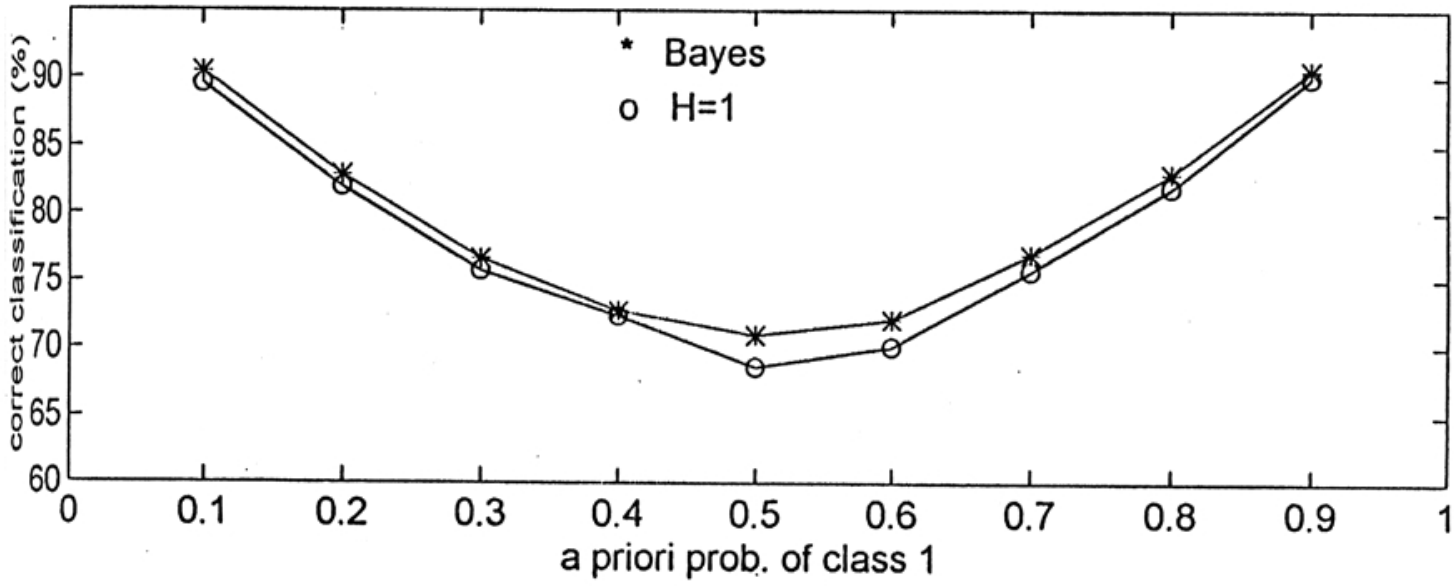


Figure 3.20: Variation of recognition score of test data with P_1 for *Data Set 4* corresponding to Bayes classifier and *GA-classifier* ($H = 1$). $n = 200$.

Chapter 4

Using Variable String Lengths in GA-classifier

4.1 Introduction

We have seen in the previous chapters that the estimation of a proper value of H is crucial for a good performance of the *GA-classifier*. Since this is difficult to achieve, one may frequently use a conservative value of H while designing the classifier. This first of all leads to the problem of an overdependence of the algorithm on the training data, especially for small sample size. In other words, since a large number of hyperplanes can readily and closely fit the classes, this may provide good performance during training but poor generalization capability. Secondly, 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*.)

In this chapter, a method has been described to automatically evolve the value of H as a parameter of the problem. For this purpose, the concept of variable length strings in GA has been adopted. Unlike the conventional GA, here the length of a string is not fixed. 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*.

Issues of minimum misclassification error and minimum number of hyperplanes generated by the *VGA-classifier* are theoretically analyzed under limiting conditions. Moreover, it is proved that for infinitely large training data set, and infinitely many iterations, the error rate provided by the *VGA-classifier* during training will approach the Bayes error probability. As mentioned in Section 3.1, the Bayes classifier is known to provide optimal performance when the class conditional densities and the *a priori* probabilities are known. Also, a desirable property of any classifier is that it should approach the Bayes classifier under limiting conditions.

Experimental results on two sets of artificial data, *ADS 1* and *ADS 2*, *Iris* data, *Vowel* data and *Cancer* data (described in Chapter 2) show that our *VGA-classifier* is able to reduce the number of hyperplanes significantly while providing improved generalization capability as compared to the *GA-classifier*. In order to demon-

strate the theoretical claims mentioned above, some additional results on artificially generated data sets with known class distributions and *a priori* probabilities are included. It is shown that the *VGA-classifier* is able to provide the minimum number of hyperplanes as also the minimum number of misclassified points. Moreover, the classifier is able to approximate automatically any Bayes decision boundary (linear, non-linear) for sufficiently large data sets and sufficiently large number of iterations.

One may note the difference between the *VGA-classifier* and the one described by Srikanth et al. in [204], also using the similar concept of variable length strings. In the latter method, the decision boundary was modeled by a variable number of ellipsoids which have a higher degree of complexity than hyperplanes. The fitness function of the string was determined from the number of misclassified samples only. Thus there is no incentive for reducing the number of ellipsoids although a factor favouring more compact ellipsoids was introduced. Additionally, no theoretical basis was provided to demonstrate the validity of the methodology. However, in a part of the experiment, we have implemented their operators in our *VGA-classifier* for the purpose of comparison.

Section 4.2 describes briefly some previous attempts to use variable string lengths in GA, and discusses the criteria for the application of VGA to the classification problem. A detailed description of *VGA-classifier*, including the modified genetic operators, is then presented in Section 4.3, along with suitable examples. Section 4.4 contains the theoretical analyses of *VGA-classifier* in terms of the number of hyperplanes and error probability. The implementation results on different data sets are provided in Section 4.5. Finally, Section 4.6 presents the discussion and conclusions of this chapter.

4.2 Genetic Algorithm with Variable String Length and the Classification Criteria

The concept of variable string lengths in genetic algorithms has been used earlier in [199] to encode sets of fixed length rules. Messy genetic algorithm [76, 77] also uses the concept of variable string lengths for constructing the chromosomes which may be under or over specified. Use of GA with variable string length has been made in

[84] for encoding variable number of fixed length blocks in order to construct layers of a neural network, and in [123] for the genetic evolution of the topology and weight distribution of neural networks.

As mentioned in Section 4.1, the *GA-classifier* with fixed H, and consequently fixed string length is rigid, and therefore has several limitations like overfitting of the training data and presence of redundant hyperplanes in the decision boundary when a conservative value of H is used. To overcome these limitations, the use of variable length strings representing variable number of hyperplanes for modeling optimally the decision boundary therefore seems natural and appropriate. This would eliminate the need for fixing the value of H, evolving it adaptively instead; thereby providing an optimal value of H.

It is to be noted that 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 harmful with respect to the generalization capability of the classifier. Thus the fitness function should be defined in such a way, maximization of which ensures primarily the minimization of the number of misclassified samples and also the requisite number of hyperplanes.

While incorporating the concept of variable string lengths, one may note that it is necessary to either modify the existing genetic operators or introduce new ones. In order to utilize the existing operators as much as possible, a new representation scheme involving the consideration of the ternary alphabet set {0, 1, # }, where # represents the don't care position, is used. For applying the conventional crossover operator, the two strings, which may now be of unequal lengths, can be made of equal length by appropriately padding one of them with #s. However, some extra processing steps have to be defined in order to tackle the presence of #s in the strings. Similarly, the mutation operator needs to be suitably modified such that it has sufficient flexibility to change the string length while retaining the flavour of the conventional operator. (As will be evident in the next section, the genetic operators are defined in such a way that the inclusion of # in the strings does not affect their binary characteristics for encoding and decoding purposes.) The classifier thus formed using variable string length GA (or VGA) is referred to as the *VGA-classifier*.

Therefore the objective of the *VGA-classifier* is to place an appropriate number of hyperplanes in the feature space such that it, first of all, minimizes the number of misclassified samples and then attempts to reduce the number of hyperplanes. Using 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. Note that the *GA-classifier*, with fixed length strings, does not have this flexibility. The description of such a classifier is given in the next section.

4.3 Description of *VGA-Classifier*

The basic operations of GAs are followed sequentially as shown in Fig. 1.3. However, the operators themselves are suitably defined for this problem.

4.3.1 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. As mentioned in Section 2.4.3, in $\mathbb{R}^N$, N parameters are required for representing one hyperplane. These are $N-1$ angle variables, $\alpha_1^i, \alpha_2^i, \ldots, \alpha_{N-1}^i$, indicating the orientation of hyperplane i ($i = 1, 2, \ldots, H$ when H hyperplanes are encoded in the chromosome), 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

$$l_H = (N-1) * b_1 + b_2 \tag{4.1}$$

$$l_{max} = H_{max} * l_H \tag{4.2}$$

respectively.

Let string i represent H_i hyperplanes. Then its length l_i is

$$l_i = H_i * l_H.$$

Unlike the conventional *GA-classifier*, the number of hyperplanes encoded in a string may vary, and is therefore stored. 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.

Example 4.1 :
Let $N = 2$, $H_{max} = 3$, $b_1 = 2$ and $b_2 = 2$, indicating that the features are in 2 dimensional space, and a maximum of 3 hyperplanes are allowed to model the decision boundary. Also, 2 bits are allowed to encode each of the angle and perpendicular distance variables. Then

$$l_{max} = 3 * (1 * 2 + 2) = 12,$$

i.e., maximum string length in the population will be 12. In the initial population, let strings i and j comprise 2 and 3 hyperplanes respectively. Or $H_i = 2$, $H_j = 3$. Therefore

$$l_i = 8 \quad \text{and} \quad l_j = 12.$$

Let the randomly generated strings be

$$\begin{array}{lcccccccccccc} string_i & = & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & & & & \\ string_j & = & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}$$

In the *GA-classifier* (Chapter 2), if $l = 12$, then all the strings must essentially contain 12 bits.

4.3.2 Fitness Computation

As mentioned in Section 4.2, the fitness function (which is maximized) is defined in such a way that

i : 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 first of all minimizes the number of misclassified points, and then

ii : 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 r_i distinct regions which contain at least one training data point. (Note that although $r_i \leq 2^{H_i}$, in reality it is upper 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 r_i regions constitutes the total misclassification $miss_i$ associated with the string. Accordingly, the fitness of string i may be defined as

$$fit_i = (n - miss_i) - \alpha H_i \quad 1 \leq H_i \leq H_{max} \tag{4.3}$$

$$= 0, \qquad otherwise, \tag{4.4}$$

where n = size of the training data set and $\alpha = \frac{1}{H_{max}}$. 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 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,

$$\begin{aligned} fit_i &= (n - miss_i) - \alpha H_i, \\ fit_j &= (n - miss_j) - \alpha H_j. \end{aligned}$$

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 Eq. (4.4)) 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 $H_i > H_j$.

Example 4.2 :
From Fig. 4.1, for a two dimensional (N=2), two class problem where $n = 15$, let the two lines ($H_i = 2$) encoded in a chromosome be as shown. There are 3 corresponding regions, *Region 1*, *Region 2* and *Region 3*. Points represented by 1 and 2 indicate that they belong to classes 1 and 2 in the training data set respectively. Hence, as mentioned above, *Region 1* and *Region 3* are associated with class 2 (since the majority of points that lie in this region are from class 2). Similarly, *Region 2* is associated with class 1. The total misclassification associated with the string is 3 (i.e., $miss = 3$). Since here $H_i = 2$, for $H_{max} = 3$ (and hence $\alpha = \frac{1}{3}$) and $n = 15$, the fitness of the string is

15 - 3 - 1/3 * 2 = 11.333.

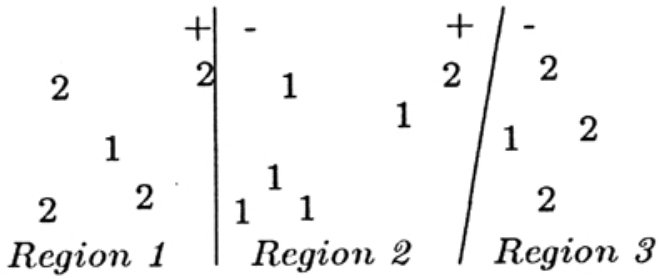


Figure 4.1: Example of fitness computation

Note that in the case of fixed length strings, the term αH_i would not have any significance, since its value would be the same for all the strings (= 1).

4.3.3 Genetic Operators

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

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 [73] 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 Eq. 4.1). Then, for each incomplete hyperplane, all the #s are set to defined bits (either 0 or 1 randomly) with probability $\frac{u}{t}$. 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.

Example 4.3 :
Let us consider single point crossover between $string_i$ and $string_j$ described in Example 4.1. They are written again for convenience.

$$string_i = 1\ 0\ \ 1\ 1\ \ \ 0\ 0\ \ 1\ 0$$
$$string_j = 0\ 1\ \ 0\ 0\ \ \ 1\ 0\ \ 1\ 1\ \ \ 0\ 0\ \ 0\ 1$$

First $string_i$ is padded with #s to make both the strings equal in length. Therefore now,

$$\begin{array}{lcccccccccccc|c} string_i & = & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & \# & \# & \# & \# \\ string_j & = & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}$$

Similar to the standard practice, the crossover point is determined randomly in the range 1 to 12, since l_j=12. Let the crossover point be 11 (the left most bit being 1) as shown above. Hence after crossover the two strings become :

$$\begin{array}{lcccccccccccc|c} offspring_i & = & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & \# & \# & \# & 1 \\ offspring_j & = & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & \# \end{array}$$

Note that the 3rd hyperplane of both the offspring is incomplete. Considering $offspring_i$ only,

$$u \text{ (no. of defined bits)} = 1, \quad t \text{ (total no. of bits for a hyperplane)} = 4.$$

Therefore the #s in the first three positions in $offspring_i$ are set to defined bits (0 or 1 randomly) with probability $\frac{1}{4}$. Otherwise, the 1 in the last position is set to #. Similarly, probability $\frac{3}{4}$ determines whether the # in position 12 of $offspring_j$ is set to defined bit or not. Otherwise, the defined bits in positions 9 - 11 are set to #. Let the resulting two strings be

$$\begin{array}{lcccccccccccccc} offspring_i & = & 1 & 0 & 1 & 1 & & 0 & 0 & 1 & 0 & & \# & \# & \# & \# \\ offspring_j & = & 0 & 1 & 0 & 0 & & 1 & 0 & 1 & 1 & & 0 & 0 & 0 & 1 \end{array}$$

There is no need for pushing the #s in $offspring_i$ to the end of the string, since they are already there. Thus

$$offspring_i = \quad 1\,0\;\;1\,1 \quad 0\,0\;\;1\,0.$$

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 [73] 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. 4.2.

Note that mutation may result in some incomplete hyperplanes, and these are handled in a manner, as done for crossover operation. For example, the operation on the defined bits, i.e., when $k \leq l_i$ in Fig. 4.2, 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.

Example 4.4 :
Let us consider $offspring_i$ from Example 4.3, which is of length 8. So it is padded

```
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 /* for 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., for #s */
            if rnd2 < μ_m2 do /* set to defined */
                Position k of string i is set to 0 or 1 randomly
            endif
        endif
    endfor
End
```

Figure 4.2: Mutation operation for string i

with 4 #s to make the length 12. Or,

$$offspring_i = 1\ 0\ \ 1\ 1\ \ \ 0\ 0\ \ 1\ 0\ \ \ \#\ \#\ \ \#\ \#$$

Applying the different mutation operators described above, let the resultant string be

$$1\ 0\ \ 1\ 1\ \ \ 0\ \#\ \ 1\ \#\ \ \ 0\ 1\ \ \#\ 0.$$

The #s in position 6 and 8 in the 2nd hyperplane are set to defined randomly with probability $\frac{1}{2}$, since there are 2 defined bits of a total of 4 positions. Otherwise, the defined bits (positions 5 and 7 in the 2nd hyperplane) are set to #s. Similarly, the # in position 11 of the 3rd hyperplane is set to a random defined bit with probability $\frac{3}{4}$, since there are 3 defined bits of a total of 4 positions. Otherwise, the defined bits (in positions 9, 10 and 12 of the 3rd hyperplane) are set to #s the . Let the resultant string be

1 0 1 1 # # # # 0 1 1 0.

The #s are pushed to the end of the string to result in

1 0 1 1 0 1 1 0 # # # #

or, 1 0 1 1 0 1 1 0.

Note that the operations defined here for designing the *VGA-classifier* are different from those used in [76, 84, 123, 199, 204]. ♣

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 upto the current generations 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.

4.4 Theoretical Study of *VGA-classifier*

This section has two parts. In the first part we provide a theoretical investigation on the values of *miss* and H of the *VGA-classifier*. The relationship between the *VGA-classifier* and the Bayes classifier is established in the second part.

4.4.1 Issues of Minimum *miss* and H

In this section we prove that the above mentioned *VGA-classifier* will provide the minimal misclassification error during training, for infinitely large number of iterations. At the same time it will require minimum number of hyperplanes in doing so.

For proving this we use the result of [28], where it has been established that for an infinitely large number of iterations, an elitist model of GA will surely provide the optimal string. In order to prove this convergence they assumed that the probability of going from any string to the optimal one is always greater than zero, and the probability of going from a population containing the optimal string to one not

containing the optimal one is zero. Since the mutation operation and elitism of the proposed VGA ensure that both these conditions are met, the result of [28], regarding the convergence to the optimal string is valid for VGA as well.

Let us now consider the fitness function for string i given by Eq. (4.3). Maximization of the fitness function means minimization of

$$miss_i + \alpha H_i = err_i, \quad \text{say}$$

where $\alpha = \frac{1}{H_{max}}$. Let us call this the error function (err_i).

Let for any size of the training data set (n), the minimum value of the error function as obtained by the *VGA-classifier* be

$$err_{min} = miss_{opt} + \alpha H_{opt}$$

after it has been executed for infinitely large number of iterations. Then according to Bhandari et al. [28], this corresponds to the optimal string. Therefore we may write

$$miss_{opt} + \alpha H_{opt} \leq miss + \alpha H, \quad \forall\ miss,\ H. \tag{4.5}$$

Theorem 4.1 : For any value of H, $1 \leq H \leq H_{max}$, the minimal number of misclassified points is $miss_{opt}$.

Proof : The proof is trivial and follows from the definition of the fitness function in Eq. (4.3) and the fact that $miss_{opt} + \alpha H_{opt} \leq miss + \alpha H$, $\forall\ miss,\ H$ (see Eq. (4.5)).

Theorem 4.2 : H_{opt} is the minimal number of hyperplanes required for providing $miss_{opt}$ number of misclassified points.

Proof : Let the converse be true, i.e., there exists some H', $H' < H_{opt}$, that provides $miss_{opt}$ number of misclassified points. In that case, the corresponding fitness value would be $miss_{opt} + \alpha H'$. Note that now $miss_{opt} + \alpha H_{opt} > miss_{opt} + \alpha H'$. This violates Eq. (4.5). Hence $H' \not< H_{opt}$, and therefore H_{opt} is the minimal number of hyperplanes required for providing $miss_{opt}$ misclassified points. ♣

From Theorems 4.1 and 4.2, it is proved that for any value of n, the *VGA-classifier* provides the minimum number of misclassified points for infinitely large number of iterations, and it requires minimum number of hyperplanes in doing so.

4.4.2 Error Rate

In this part, we attempt to establish that for $n \longrightarrow \infty$, the error rate provided by the *VGA-classifier* during training will approach the the Bayes error probability for infinitely many iterations.

The fitness function of the *VGA-classifier* as defined in Eq. (4.3) is

$$fit = (n - miss) - \alpha H.$$

Dividing the right hand side by n, we get

$$1 - \frac{miss}{n} - \alpha\frac{H}{n}.$$

For $n \longrightarrow \infty$, if the limit exists, then it goes to

$$1 - \lim_{n\longrightarrow\infty}[\frac{miss}{n}].$$

The term $\frac{miss}{n}$ provides the error rate of the *VGA-classifier*.

It is to be noted that the *VGA-classifier* attempts to minimize $f_{nE}(\omega)$ (Eq. 3.10), the average number of misclassified samples, for large n. Assuming that $H_{max} \geq H_{opt}$, we may analyze as in Section 3.2, and state the following theorem and corollary.

Theorem 4.3 : For sufficiently large n, $f_n(\omega) \not> a$, (i.e., for sufficiently large n, $f_n(\omega)$ cannot be greater than a) almost everywhere.

Proof : The proof is the same as that of Theorem 3.1.

Corollary : $\lim_{n\to\infty} f_n(\omega) = a$ for $\omega \in B$.

Proof : The proof is the same as that of corollary to Theorem 3.1.

As for the case with *GA-classifier*, Theorem 4.3 and its corollary indicate that the performance of the *VGA-classifier* approaches that of the Bayes classifier under limiting conditions. Or in other words, the decision boundary generated by the *VGA-classifier* will approach the Bayes decision boundary for infinitely large training data set and number of iterations, if the Bayes boundary is unique. Otherwise the *VGA-classifier* will provide one such boundary for which the error rate is equal to the Bayes error probability a. Some experiments validating the theoretical results proved here are reported in the next section for data sets with both linear and non-linear Bayes boundary.

4.5 Implementation

In this section the effectiveness of VGA in automatically determining the value of H of the classifier is demonstrated on a number of artificial and real life data sets. The recognition scores of the *VGA-classifier* are also compared with those of the fixed length *GA-classifier*. A comparison of our concept of using variable string lengths in GA with another similar approach [204] is provided. The theoretical findings in Section 4.4 are verified empirically on a number of data sets where the Bayes boundary is linear and hence H_{opt} (the optimal value of the number of hyperplanes) is known exactly, as well as on *Data Set 2* (described in Chapter 3), where the boundary is highly curved and therefore H_{opt} is not known exactly. The data sets are described first followed by the results and their analysis.

4.5.1 Data Sets

The data sets *ADS 1*, *ADS 2*, *Vowel*, *Iris* and *Cancer*, described in Chapter 2, have been used here for demonstrating the effectiveness of the *VGA-classifier*. Additionally, a few more data sets with known H_{opt} are used, for the purpose of verifying the theoretical results obtained in Section 4.4. These are described below.

Uniform Data : This data set consists of two linearly separable classes having equal *a priori* probabilities. The points of each class are distributed uniformly over a circle of unit radius as shown in Fig. 4.3. The classes are non-overlapping. The separation between the classes, *sep*, is varied by changing the center of the circles appropriately. 500 points are generated for each value of *sep* to yield 5 data sets.

Constant Data : This overlapping data is generated so that there is no randomness in it (Fig. 4.4). The meanings of the various symbols used in Fig. 4.4 are as follows :
1 : One sample from class 1.
2 : One sample from class 2.
𝟚 : One sample each from classes 1 and 2.
𝔸 : Two samples from class 1 and one sample from class 2.
𝔹 : Two samples from class 2 and one sample from class 1.
Note that a single line provides the optimal decision boundary for the training data set when 40 samples are misclassified. It should be mentioned here that this data set

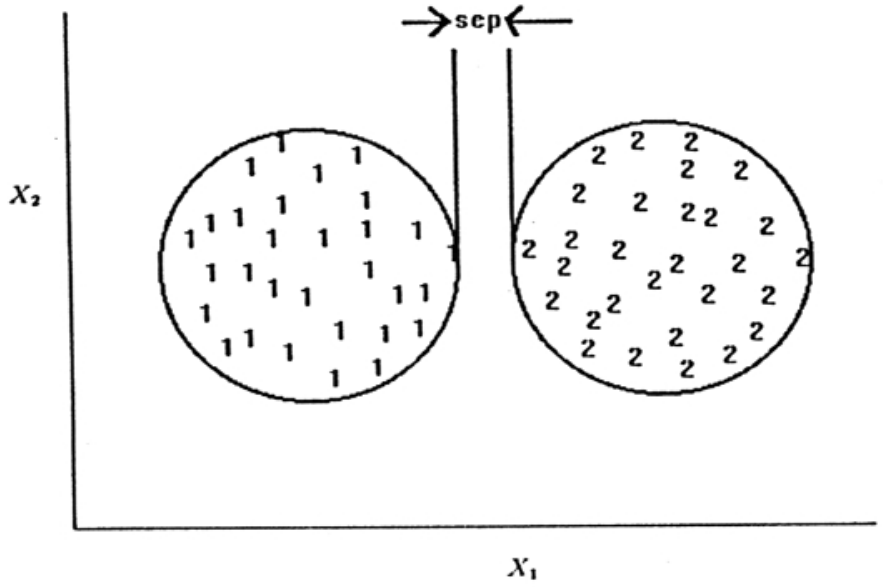


Figure 4.3: *Uniform Data*

could have been generated, with a very small probability, by a suitable distribution.

Triangular Data : This data set, consisting of two classes in triangular distribution along X axis and uniform distribution along Y axis, is similar to the triangular data in Chapter 3. The ranges for classes 1 and 2 are [0,2]×[0,2] and [0.1,2.1]×[0,2] respectively. It is evident from the selected ranges along the X axis that this data set has considerably more overlap than *Data Set 1* of Chapter 3. The Bayes boundary is a line at $x = 1.05$ (see Fig. 4.5).

We have also used *Data Set 2*, described in Chapter 3 (Fig. 3.2), for demonstrating that the *VGA-classifier* is indeed able to approximate better even the highly curved Bayes boundary as the size of the training data is increased. The values of n used for this purpose are 1000, 5000 and 15000.

4.5.2 Results

As in Chapter 2, a fixed population size of 20 is chosen. *Roulette wheel strategy* is used to implement proportional selection. *Single point crossover* is applied with a fixed crossover probability of 0.8. A variable value of mutation probability μ_m is selected from the range [0.015, 0.333]. 200 iterations are performed with each mutation probability value. (Note that the number of hyperplanes constituting the search space for *VGA-classifier* now ranges from 1 to H_{max}. So, unlike the case

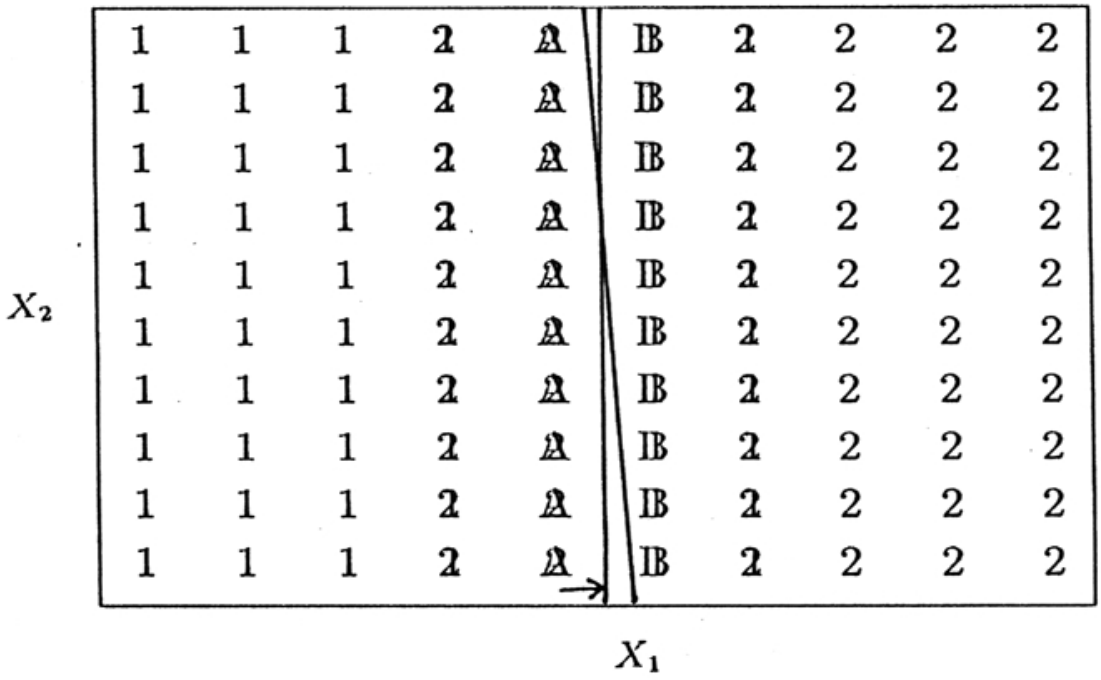


Figure 4.4: *Constant Data* along with the optimal Bayes boundary (shown with an arrow) and *VGA* boundary. Here, 1 indicates one sample from class 1, 2 indicates one sample from class 2, 2 indicates one sample each from classes 1 and 2, A indicates two samples from class 1 and one sample from class 2, B indicates two samples from class 2 and one sample from class 1

with fixed H, the number of iterations for each value of μ_m is increased from 100 to 200.) The values of μ_{m_1} and μ_{m_2} mentioned in Section 4.3.3 are set to 0.1. The process is executed for a maximum of 3000 iterations. *Elitism* is incorporated as in Chapter 2. The recognition scores provided here are the average values obtained over five different runs of the algorithm. H_{max} is set to 10, so $\alpha = 0.1$. Training sets are the same as used in Chapter 2.

Performance of the VGA-classifier

Tables 4.1 and 4.2 show the number of hyperplanes H_{VGA}, for a particular run, as determined automatically by the *VGA-classifier* for modeling the class boundaries of *ADS 1*, *ADS 2*, *Vowel*, *Iris* and *Cancer* data sets when the classifier is trained with 10% and 50% samples respectively. Two different values of H_{max} are used for this purpose viz., $H_{max} = 6$ and $H_{max} = 10$. The overall recognition scores obtained during testing of the *VGA-classifier* along with their comparison with those obtained for the fixed length version (i.e., *GA-classifier*) with $H = 6$ and 10 are also shown.

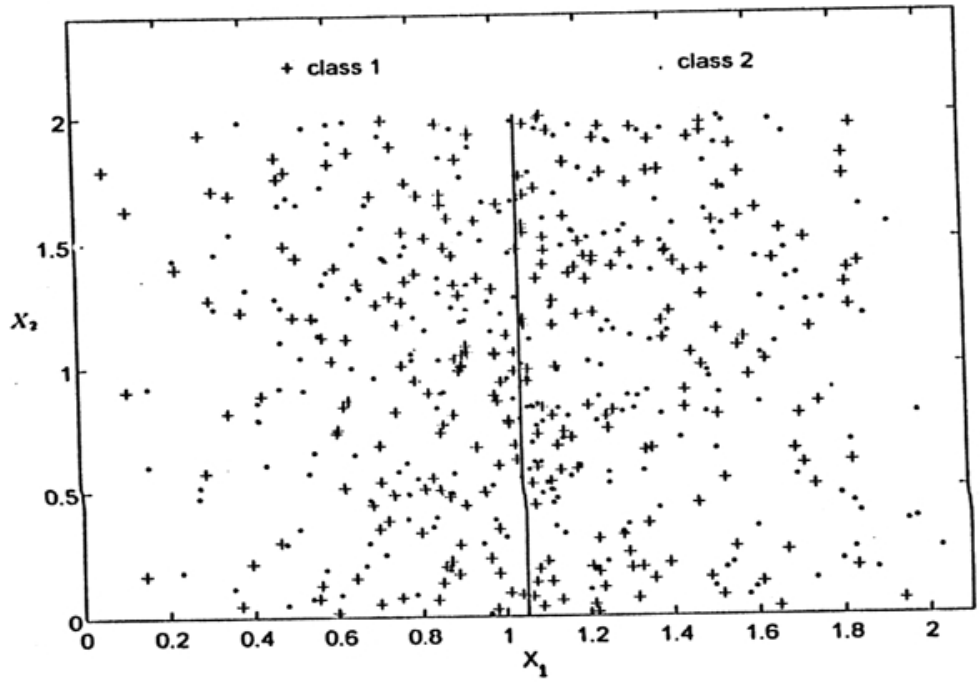


Figure 4.5: *Triangular Data* along with the Bayes boundary

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 4.1). In this case, $H_{max} = 6$ does not appear to be a high enough value for modeling the decision boundaries of *Vowel* classes with *VGA-classifier*. This is 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 as compared to 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

Table 4.1: 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
ADS 2	6	88.16	84.04	5	88.35	88.29
Vowel	6	73.66	69.21	6	71.19	71.99
Iris	2	95.56	76.29	2	95.81	93.33
Cancer	2	93.34	77.27	1	95.45	93.01

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 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. 4.6.

Comparison with the Method in [204]

In this section an investigation is made to compare the performance of our concept of using variable string length in GA with that of another similar approach of Srikanth et al. [204]. For this purpose the operators used in [204] are implemented here for the same problem of pattern classification using hyperplanes, and the resulting performance is compared to those of our *VGA-classifier* for the five data sets. Before providing the results, let us describe in brief the method of incorporating variable string lengths in GAs as proposed in [204].

The initial population is created randomly such that each string encodes the parameters of only one hyperplane. The fitness of a string is characterized by just the number of training points it classifies correctly, irrespective of the number of

Table 4.2: 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
ADS 2	5	95.22	94.56	3	96.26	96.17
Vowel	6	78.26	77.77	6	77.11	76.68
Iris	2	97.60	93.33	2	97.67	97.33
Cancer	2	93.69	92.73	2	97.66	93.67

hyperplanes encoded in it. Among the genetic operators, traditional selection and mutation are used. A new form of crossover, called *modulo crossover* is used which keeps the sum of the lengths of the two chromosomes constant both before and after crossover.

Two other operators are used in conjunction with the *modulo crossover* for the purpose of faster recombination and juxtaposition. These are the *insertion* and *deletion* operators. During *insertion*, a portion of the genetic material from one chromosome is inserted at a random *insert-location* in the other chromosome. Conversely, during *deletion*, a portion of a chromosome is deleted to result in a shorter chromosome.

Tables 4.3 and 4.4 show the comparative overall recognition scores during both training and testing of the *VGA-classifier* for the above mentioned four data sets when our approach of incorporating variable string length is compared with that adopted in [204] for 10% and 50% training data respectively. Other parameters are kept the same as before. Results shown are the average values taken over five different runs. For keeping parity, the VGA of Srikanth et al. is implemented such that no more than 10 hyperplanes are used for modeling the decision boundary of the data sets. The table also shows the number of hyperplanes, H_{VGA}, generated by the two methods for one particular run. Since the VGA of Srikanth et al. does not take care of the minimization of the number of hyperplanes while maximizing the fitness function, the H_{VGA} is usually higher than that of our method.

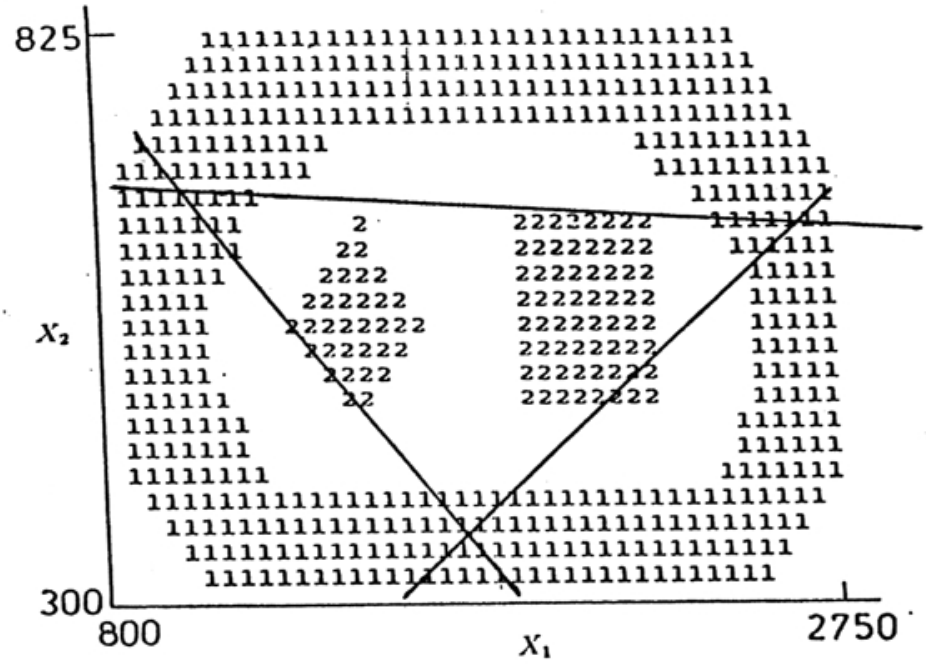


Figure 4.6: *ADS 1* along with the *VGA* boundary for $H_{max} = 10$

As is evident from the tables, the performance of the classifier during training is better for the VGA in [204] than the proposed one for all the data sets. The former, in general, uses more hyperplanes (of which many were found to be redundant on investigation), which results in an increase in the execution time. From the training performance, it appears that the operators used by Srikanth et al., are better able to recombine the subsolution blocks into larger blocks. However this is seen, in general, to result in comparatively poorer scores during testing. To consider a typical example, in one of the cases for the *Vowel* data set when 10% data was used for training, 10 hyperplanes were used to provide a training recognition score of 97.47%, while the recognition score during testing fell to 68.95%.

It is also found that with increase in the size of the training data, the number of hyperplanes for modeling the class boundaries increase for the algorithm of Srikanth et al. Furthermore, as expected, the performance of all the classifiers is improved with increase in the size of the training data from 10% to 50%.

Experimental Verification of the Theoretical Results

The non-overlapping uniformly distributed data set *Uniform data* (Fig. 4.3) is used to establish that the *VGA-classifier* is able to arrive at the optimal value of H_{opt}, irrespective of the value of H_{max}, provided $H_{max} \geq H_{opt}$. The simplicity of the data

Table 4.3: Comparative classification performance of *VGA-classifier* for H_{max}=10 using two types of variable string lengths (when 10% of the data set is used for training and the remaining 90% for testing)

Data set	Proposed VGA			VGA [204]		
	Training score (%)	Test score (%)	H_{VGA}	Training score (%)	Test score (%)	H_{VGA}
ADS 1	100	95.62	3	100	93.16	6
ADS 2	92.68	88.16	6	99.10	90.50	6
Vowel	80.00	73.66	6	97.36	70.22	9
Iris	100	95.56	2	100	94.98	2
Cancer	98.57	93.34	2	100	92.81	3

set ensures that this can be demonstrated within a reasonable number of iterations. The value of H_{opt} for this data is known to be 1, which is sufficient to properly classify all the given points (i.e., $miss_{opt} = 0$). Here $n = 500$. The genetic parameters described earlier are the same. The separation *sep* between the classes is progressively decreased from 0.08 to 0.001. Table 4.5 presents the number of generations required to attain the optimal values (i.e., $H_{opt} = 1$ and $miss_{opt} = 0$) for different values of *sep*. Two values of H_{max} are taken, namely $H_{max} = 5$, so $\alpha = 0.2$, and $H_{max} = 10$, so $\alpha = 0.1$. The results are, as before, the average values obtained over 5 runs of the algorithm.

It is found that the *VGA-classifier* is able to reduce the number of lines to 1 in all the experiments while attaining zero misclassification for both the values of H_{max}. As expected, the computational effort increases as the value of *sep* decreases. Again, the number of iterations required to attain the optimal value is larger for $H_{max} = 10$ than that for $H_{max} = 5$. This is also expected since $H_{max} = 10$ indicates a larger search space.

In order to demonstrate the effectiveness of the methodology for overlapping data set where H_{VGA} is known, we used the *Constant Data* ($H_{opt} = 1$) of Fig. 4.4 and the *Triangular Data* ($H_{opt} = 1$) of Fig. 4.5 for $H_{max} = 10$. For *Constant Data*, a single line is seen to provide the optimal decision boundary for the training data set when

Table 4.4: Comparative classification performance of *VGA-classifier* for H_{max}=10 using two types of variable string lengths (when 50% of the data set is used for training and the remaining 50% for testing)

Data set	Proposed VGA			VGA [204]		
	Training score (%)	Test score (%)	H_{VGA}	Training score (%)	Test score (%)	H_{VGA}
ADS 1	98.18	96.41	4	100.00	96.01	9
ADS 2	97.21	95.22	5	100.00	94.85	7
Vowel	79.73	78.26	6	85.48	78.37	9
Iris	100	97.60	2	100.00	94.67	5
Cancer	98.82	93.69	2	100	93.61	5

40 samples are misclassified (Fig. 4.4, the line shown with an arrow). Increasing the number of lines to any value is not going to improve the recognition score. The *VGA-classifier* with $H_{max} = 10$ is also found to provide a single line in all the 5 different runs of the algorithm (see Fig. 4.4), yielding the optimal decision regions.

For the *Triangular data*, the experiment is performed for different values of n ranging from 100 to 3000, and $H_{max} = 10$. Interestingly, unlike the *Constant Data*, in none of the cases the *VGA-classifier* is found to be able to arrive at the optimal value of H_{VGA}. However, the recognition score during training is found to be consistently better than that obtained using the Bayes decision boundary. A plot of the difference between the average misclassification obtained by the Bayes boundary and that obtained by the VGA boundary during training, versus n is presented in Fig. 4.7. It shows that this difference gradually decreases with increasing n. This reflects the trend that as $n \longrightarrow \infty$, the two values will tend to be the same. At the same time, since the *VGA-classifier* is guaranteed to provide the optimal string (with maximum fitness value) for infinitely many iterations, the number of hyperplanes will also be optimal.

Note that, since it is always possible, for this highly overlapping data set, to reduce the number of misclassified points by increasing the number of hyperplanes, the optimal number of lines could not be obtained in the case of *Triangular Data*. Larger

Table 4.5: Number of generations to attain optimal values of H and *miss* for *Uniform data* for $H_{max} = 5$ and 10.

sep	generations	
	$H_{max} = 5$	$H_{max} = 10$
0.08	184.5	300.4
0.03	418.5	469.8
0.02	522.2	757.4
0.004	560.6	782.4
0.001	777.2	1231.0

number of hyperplanes results in enclosing of small regions such that all or majority of the points in the region belong to a single class; thereby reducing the classification error during training. (As noted in Chapter 2, this may not be beneficial to the generalization capability of the classifier.) But with increasing n, the likelihood of such an event decreases. In the extreme case when $n = \infty$, only the optimal Bayes boundary is able to provide the minimum average misclassification.

Data Set 2 with $n = 1000$, 5000 and 15000 is used to demonstrate that the boundary provided by the *VGA-classifier* starting with H_{max} hyperplanes (say, $H_{max} = 10$) approaches the Bayes boundary even when the latter is highly curved. In view of the complexity of the data set, the number of iterations is increased beyond 3000 to 6000.

Figs. 4.8 and 4.9 present the boundaries obtained after 3000 and 6000 iterations respectively when $n = 1000$. The corresponding results for $n = 5000$ and 15000 are presented in Figs. 4.10 and 4.11, and Figs. 4.12 and 4.13 respectively. As expected, the boundary obtained after 6000 iterations for $n = 15000$ provides the best approximation to the Bayes boundary among all the cases. Note that the *VGA-classifier* used five lines for $n = 15000$ and four lines for $n = 1000$ and 5000 for modeling the decision boundary.

As seen from Figs. 4.8 and 4.9 for $n = 1000$, the redundant line present after 3000 iterations has been removed after execution of 6000 iterations. This is unlike the cases for $n = 5000$ and 15000 where one redundant line is seen to be present even

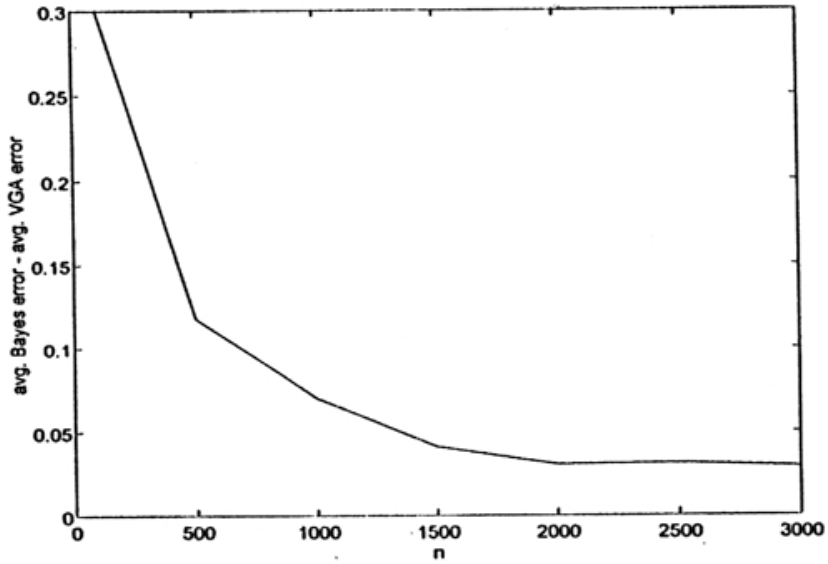


Figure 4.7: Variation of (avg. Bayes error - avg. *VGA* error during training) with n for *Triangular Data*

after 6000 iterations. The reason for this may be that the *VGA-classifier* took most of its time for reducing the number of misclassified points leaving very little time to remove the redundancy. Obviously, if the *VGA-classifier* was allowed to execute for more iterations, all redundancies would have been removed. The approximation to the Bayes boundary may also have improved further.

4.6 Discussion and Conclusions

The problem of fixing the value of H *a priori* of the *GA-classifier* has been resolved by introducing the concept of variable string lengths in the classification algorithm. New genetic operators are defined to deal with the concept of variable string lengths for formulating the classifier. The fitness function has been defined so that its maximization indicates minimization of the number of misclassified samples as well as the required number of hyperplanes. It is proved that for infinitely large number of iterations the method is able to arrive at the minimum number of misclassified samples and will need minimum number of hyperplanes for this purpose. It is also established that as the size of the training data set n and the number of iterations go to infinity, the performance of the *VGA-classifier* will approach that of the Bayes

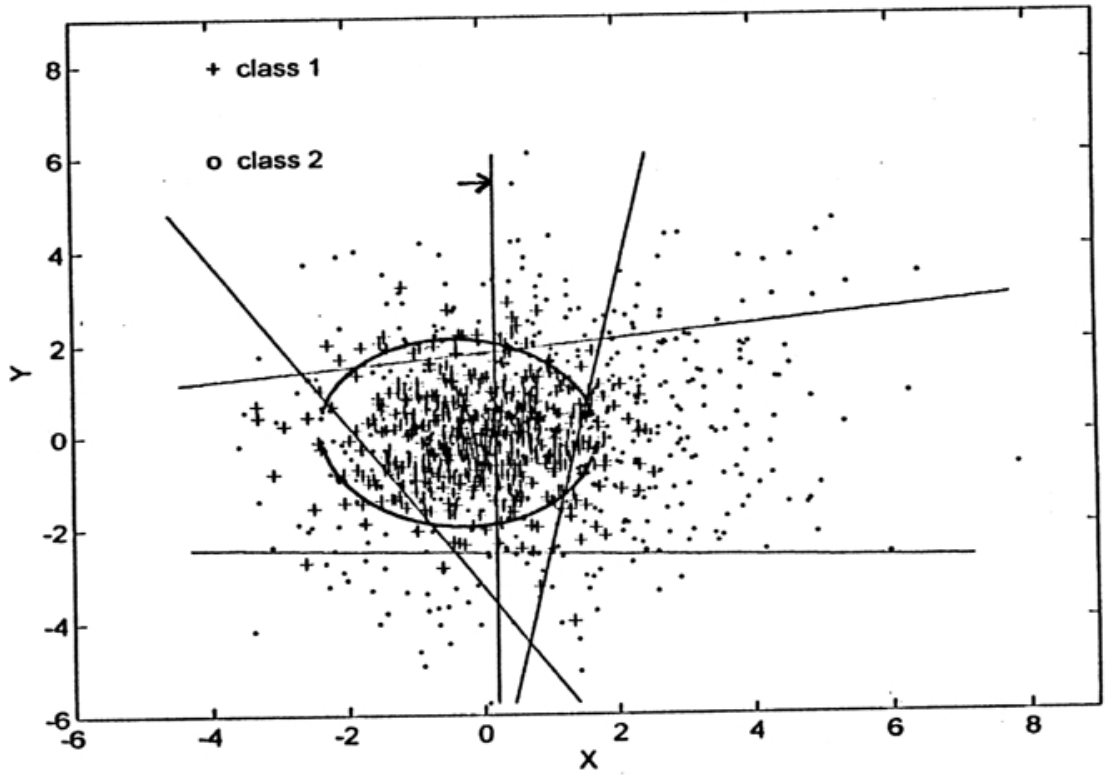


Figure 4.8: *Data Set 2* for $n = 1000$ and the boundary provided by *VGA-classifier* for $H_{max} = 10$ after 3000 iterations along with Bayes decision boundary (circular one). The redundant line is marked with an arrow.

classifier.

Experimental evidence for different percentages of training and test data indicates that given a value of H_{max}, the algorithm can not only be able to automatically evolve an appropriate value of H for a given data set, but also result in improved performance of the classifier. The theoretical findings are also substantiated experimentally on several artificial data sets with known class distributions and *a priori* probabilities. Although it is proved that the classifier will always provide the minimum number of hyperplanes for infinitely large number of iterations, in practice, one may sometimes find redundancy in the output boundary due to premature termination. These would surely be removed if the algorithm is executed for sufficiently large number of iterations. One such situation is found for *Data Set 2* with 1000 points, when the redundant line present after 3000 iterations is seen to be removed automatically after 6000 iterations.

It is to be noted that since any non-linear boundary can be approximated by a number

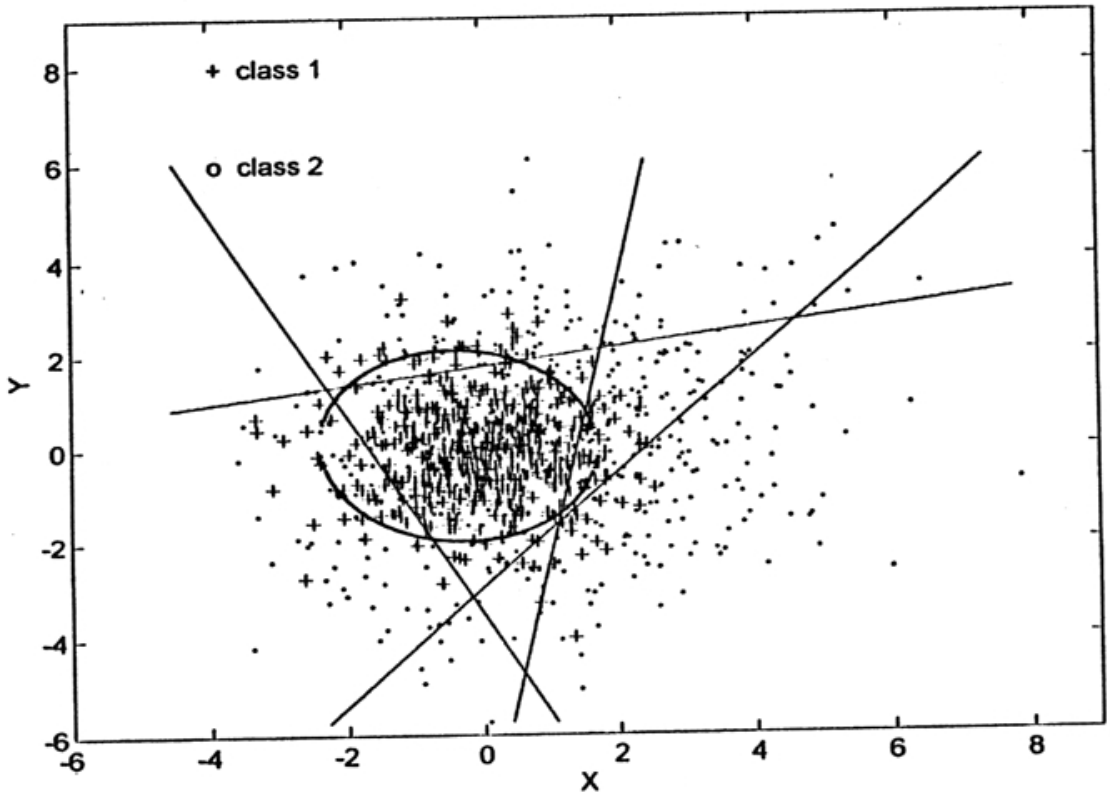


Figure 4.9: *Data Set 2* for $n = 1000$ and the boundary provided by *VGA-classifier* for $H_{max} = 10$ after 6000 iterations along with Bayes decision boundary (circular one).

of linear surfaces, the *VGA-classifier* with a large value of H_{max} can be applied to data sets having any kind of non-linear class boundaries. This is demonstrated for *Data Set 2*, where the boundary provided by the *VGA-classifier* is found to approach the highly curved Bayes boundary as the number of training data points is increased. Note that any choice of $H_{max} \geq H_{opt}$ would provide similar performance of the *VGA-classifier* if n and the number of iterations are taken to be sufficiently large. Thus given a large enough value of H_{max}, the *VGA-classifier* can automatically evolve an appropriate number of hyperplanes to model any kind of decision boundary for large n and number of iterations. Moreover, being non parametric in nature, the *VGA-classifier* can be applied to any kind of data set. (In this context we mention about the proof that exists in the literature [67] showing that the k-NN classifier approaches the Bayes classifier if the values of k and $\frac{k}{n}$ can be made to approach ∞ and 0 respectively as $n \rightarrow \infty$. Several choices of k are therefore possible that satisfy this criterion, with each choice providing a different approximation of the boundary and hence different performance of the k-NN classifier. Similarly, it is known that

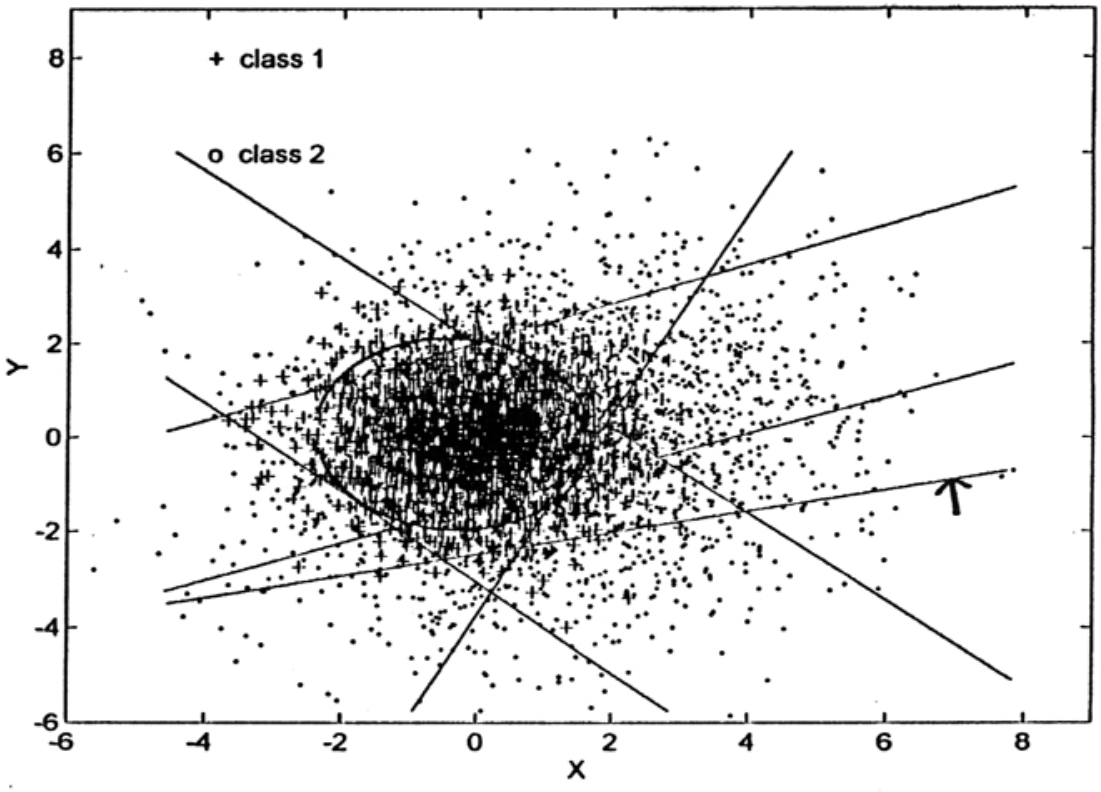


Figure 4.10: *Data Set 2* for $n = 5000$ and the boundary provided by *VGA-classifier* for $H_{max} = 10$ after 3000 iterations along with Bayes decision boundary (circular one). The redundant line is marked with an arrow.

when the MLP, given an appropriate architecture, is trained using back propagation, it approximates the Bayes optimal discriminant function under limiting conditions [180]. However, a proper choice of the architecture is not an easy task; thereby making the performance of MLP vary widely with different architectures.)

Regarding the choice of the number of iterations to be executed before termination and the size of the training data set (n) of the *VGA-classifier*, one may note the following. Since the class boundaries are explicitly generated in the *VGA-classifier*, one indication to terminate the algorithm may be when the boundaries undergo very little or no change over a number of iterations. In this context one may refer to the investigation in [135], where the concept of ϵ-optimal stopping time for GAs is formulated. Estimation of a proper value of n which would ensure that the performance of the *VGA-classifier* is within some factor of the optimal performance is an open issue and requires further research.

The method of using variable string length in the algorithm of Srikanth et al. is

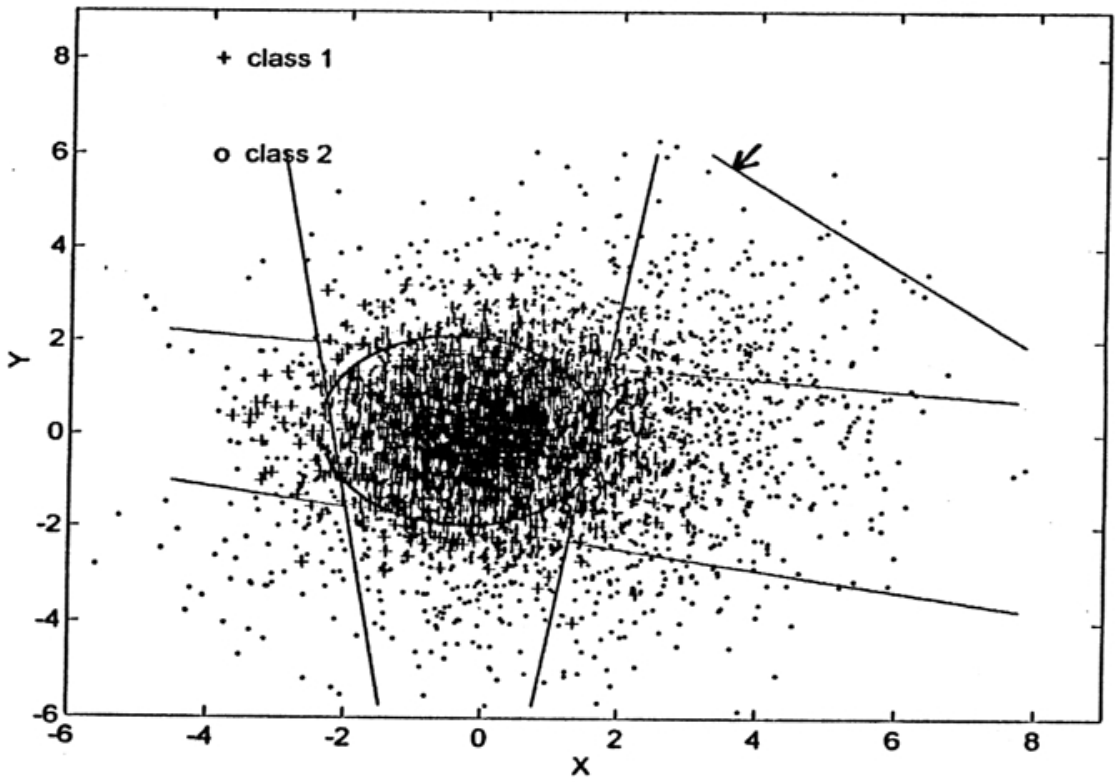


Figure 4.11: *Data Set 2* for $n = 5000$ and the boundary provided by *VGA-classifier* for $H_{max} = 10$ after 6000 iterations along with Bayes decision boundary (circular one). The redundant line is marked with an arrow.

also implemented in our *VGA-classifier* for comparison. Since the former method does not include a factor for reducing the number of surfaces, it is found to use more hyperplanes for constituting the decision boundary. This results in better performance during training, mostly at the cost of reduced generalization capability. Additionally, the execution time is also more since no explicit effort is made to decrease the number of hyperplanes.

In this connection one may also note that the genetic operators and processing steps of the *VGA-classifier* described here entail very little disruption of those in the conventional GA. On the other hand this is not true for the method of Srikanth et al. which introduces two new processing steps viz., insertion and deletion, besides using a significantly different crossover operator. Further, the former method requires the specification of H_{max}, whereas such a constraint is not there for the latter one.

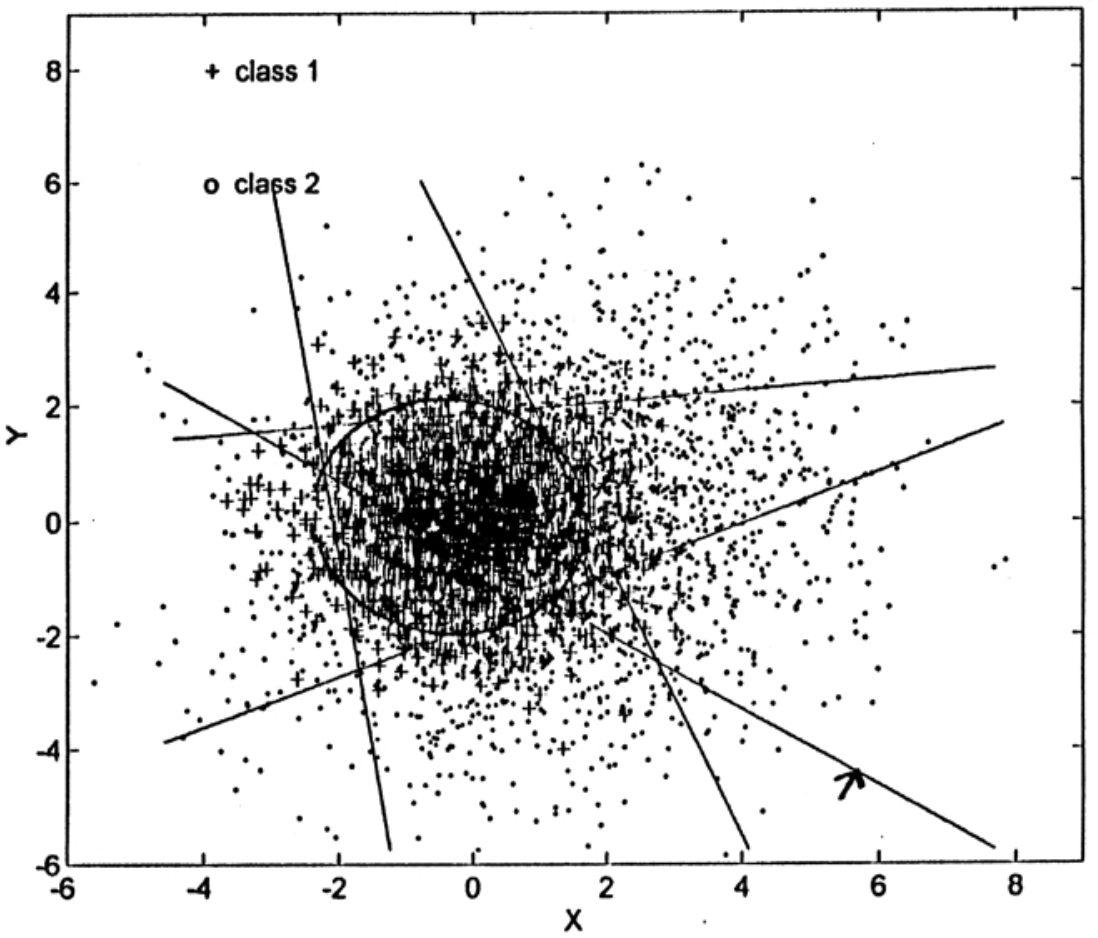


Figure 4.12: *Data Set 2* for $n = 15000$ and the boundary provided by *VGA-classifier* for $H_{max} = 10$ after 3000 iterations along with Bayes decision boundary (circular one). The redundant line is marked with an arrow.

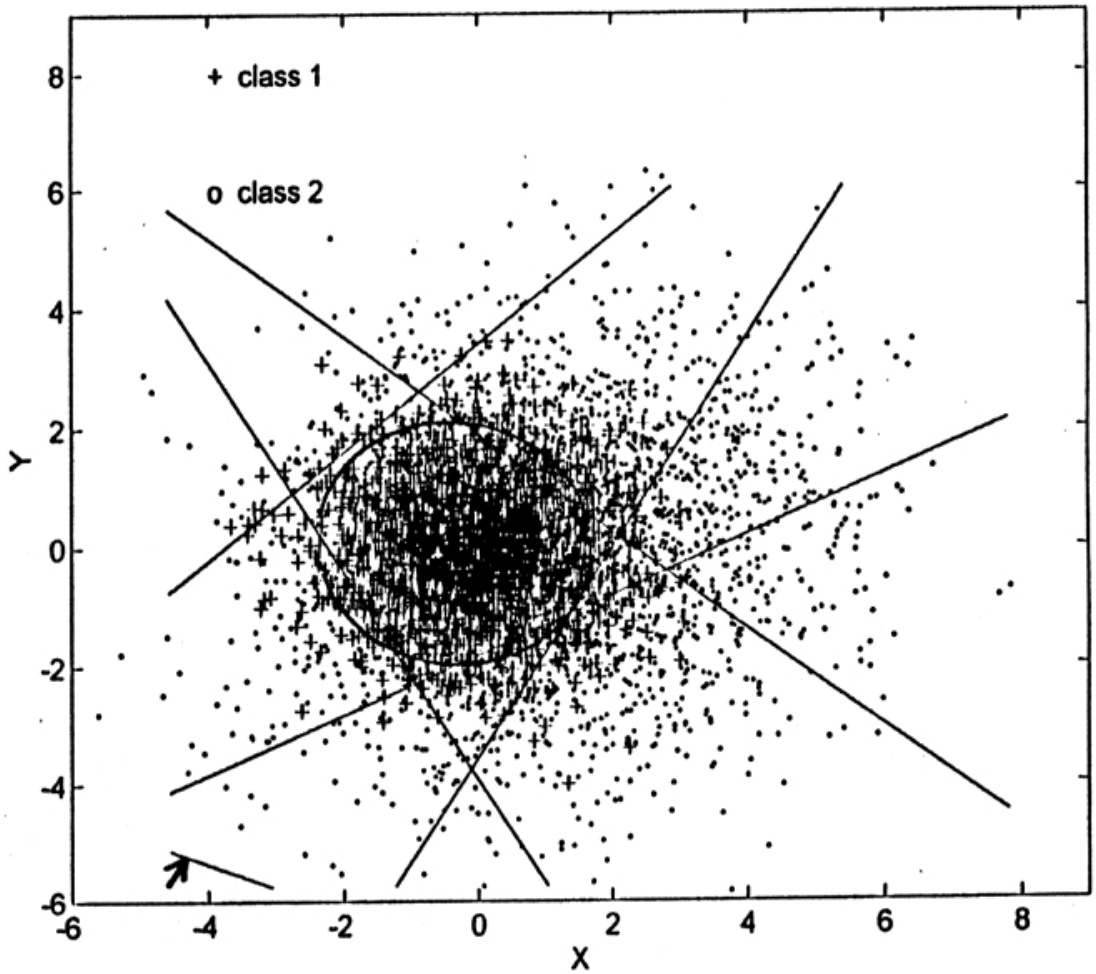


Figure 4.13: *Data Set 2* for $n = 15000$ and the boundary provided by *VGA-classifier* for $H_{max} = 10$ after 6000 iterations along with Bayes decision boundary (circular one). The redundant line is marked with an arrow.

Chapter 5

Incorporation of Chromosome Differentiation in GAs (GACD)

5.1 Introduction

The power of GAs lies in their ability to encode complex information and parameters of the search space in simple structures called *chromosomes* or *genotypes*, which are usually of a fixed length. Since the chromosomes are treated as individuals of the same type, unrestricted mating is allowed during crossover. However, nature generally differentiates the individuals of a species into more than one type or class (typical example being sexual differentiation). Cross breeding is preferred to inbreeding because of the various advantages it offers e.g., healthier offspring, introduction of greater variety etc. The widespread existence of this sort of differentiation and breeding styles in almost all living beings indicates the need for investigating a corresponding concept in genetic algorithms. Motivated by this, an attempt has been made in this chapter to differentiate the chromosomes into two distinct classes, M and F respectively and to examine subsequently, its effect on the *GA-classifier* developed in Chapter 2. The methodology of incorporating two classes of chromosomes M and F is now referred to as GACD (GA with chromosome differentiation), and the corresponding classifier is named as *GACD-classifier*. Thus the present investigation is an attempt to enrich the literature of GA in general, and that of *GA-classifier* in particular.

In addition to developing the methodology of GACD, a modified schema theorem is also presented here. It shows that the basic tenet of GAs holds for GACD also; short, low order, above average schemata will receive increasing number of trials in subsequent generations. Extensive empirical investigation has also been made for a variety of function optimization and pattern classification problems. These show an overall better performance of GACD both in terms of the best value obtained and the number of generations required to attain this value.

Section 5.2 presents the motivation and a detailed description of GACD. Its analysis with respect to schema sampling, along with an empirical investigation is provided in Section 5.3. Section 5.4 presents the *GACD-classifier*, along with its comparative performance with respect to *GA-classifier* for *ADS 1*, *ADS 2*, *Iris*, *Vowel* and *Cancer* data sets, for two different values of crossover probability μ_c. Finally, the discussion and conclusions are provided in Section 5.5.

5.2 GACD : Incorporating Differentiation in GA

5.2.1 Criterion

As mentioned before, nature generally differentiates the individuals of a species into more than one class. Sexual differentiation is a typical example, where the individuals of a species generally belong to either male or female class. The prevalence of this form of differentiation indicates an associated advantage which appears to be in terms of co-operation between two dissimilar individuals, who can at the same time specialize in their own fields. This co-operation and specialization should give rise to healthier and more fit offspring [73]. Appendix C provides an analysis in this regard.

These observations led to the investigation into the effects of differentiating the chromosomes of a population into two different classes, namely M and F respectively, thereby giving rise to two separate populations. Since, in addition, we would like to make the two populations most dissimilar, (this requirement is artificially imposed), these are initially generated in such a way that the Hamming distance between them is maximized. Crossover is allowed between individuals belonging to the two distinct populations only. Note that the concept of restricted mating through Hamming distance was also used in [57, 59]. The other genetic operators are applied in the usual way.

As crossover is allowed between these two dissimilar groups only, a greater degree of diversity is introduced in the population leading to greater exploration in the search. At the same time conventional selection is performed over the entire population which serves to exploit the information gained so far. Thus it appears that GACD attains a greater balance between exploration and exploitation which is crucial for any adaptive system; thereby making GACD superior to conventional GA (CGA).

5.2.2 Description of GACD

The basic steps of GA as shown in Fig. 1.3 are followed in GACD as well. However, the individual processes are modified. These are now discussed in detail.

Population Initialization :

The structure of a chromosome of GACD is shown in Fig. 5.1. Here the l bits, termed *data bits* encode the parameters of the problem. The initial two bits, termed the *class bits* indicate the class (M or F) of the chromosome.

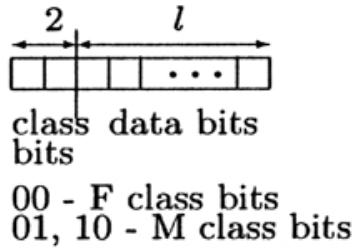


Figure 5.1: Structure of a chromosome in GACD

Two separate populations, one containing the M chromosomes (M population) and the other containing the F chromosomes (F population), are maintained over the generations. The sizes of these two populations, p_m and p_f respectively, may vary. Let $p_m + p_f = p$, where p is fixed (equivalent to the population size of CGA). Initially $p_m = p_f = \frac{p}{2}$. The data bits of each M chromosome are first generated randomly. One of the two *class bits*, chosen randomly, is initialized to 0 and the other to 1. The data bits of the F chromosomes are initially generated in such a way that the Hamming distance between the two populations (in terms of the data bits) is maximum. The Hamming distance between two chromosomes c_1 and c_2 (where c_1 and c_2 are bit strings), denoted by $h(c_1, c_2)$, is defined as the number of bit positions in which the two chromosomes differ. Hamming distance between two populations, P_1 and P_2, denoted by $h(P_1, P_2)$, is defined as follows :

$$h(P_1, P_2) = \sum_i \sum_j h(c_i, c_j), \quad \forall c_i \in P_1, \forall c_j \in P_2. \tag{5.1}$$

A method of generating p_f number of F chromosomes such that the above mentioned restriction is satisfied while allowing a certain amount of randomness is developed whose pseudo code is shown in Fig 5.2. Here $M(i,j)$ and $F(i,j)$ indicate the jth bit of the ith M and F chromosomes in the populations respectively. $check(i,j)$ is an auxiliary data structure used to keep track of the bits of the M chromosomes that have been chosen for complementation. The *class bits* of each F chromosome are initialized to 0s.

```
Begin
        i=0 to p_m
              j=0 to l − 1
                    check(i, j) = 0  /* Initialization */
        i=0 to p_f
              j=0 to l − 1
                    repeat
                          k = random(p_m)  /* returns an integer in
                                              the range of 0 to p_m − 1 */
                    until (check(k, j)=0)  /* M(k, j) not chosen
                                              before */
                    check(k, j)=1  /* M(k, j) now chosen.
                                      Not to be chosen again */
                    F(i, j)= complement(M(k, j))
end
```

Figure 5.2: Algorithm for initializing the F population from the initial M population in GACD

Fitness Computation :

Only the l data bits are used to compute the fitness of the chromosomes in a problem specific manner, similar to the one described in Section 2.2.4.

Selection :

Selection is performed over all the p ($= p_m + p_f$) chromosomes, (i.e. disregarding the class information) using their fitness values. In other words, all the chromosomes compete with one another for survival. The selected chromosomes are placed in the mating pool.

Crossover :

Crossover is applied with probability μ_c between an M and an F parent chromosome. Each parent contributes one class bit to the offspring. Since the F parent can only contribute a 0 (its class bits being 00), the class of the child is primarily determined by the M parent which can contribute a 1 (yielding an M child) or a 0 (yielding an F child) depending upon the bit position (among the two class bits) of the M parent

chosen. This process is performed for both the offspring whereby either two M or two F or one M and one F offspring will be generated.

Crossover is carried on until (a) there are no chromosomes in the mating pool, or (b) there are no M (or F) chromosomes in the mating pool. In the former case the crossover process terminates. In the latter case, the remaining F (or M) chromosomes are mated with the best M (or F) chromosome. Note that if at the start of the crossover process, it is found that the mating pool contains chromosomes of only one class, then the crossover process is discontinued.

Mutation :

Bit by bit mutation is performed over the data bits only with probability μ_m. The *class bits* are not mutated.

Note : Elitism is incorporated by preserving the best chromosome, among both the M and F chromosomes, seen till the current generation in a location outside the population.

5.3 Schema Theorem for GACD

In this section the schema theorem (Section 1.3.2) is modified appropriately to incorporate the ideas of the GACD algorithm. The terminology used for presenting the analysis is first described. Note that, some of the symbols used here are also defined in Section 1.3.2. They are mentioned here again, for the sake of completeness. In general the M and F parameters are denoted by subscripts m and f respectively, while parameters with no subscript denote that these are applicable over both the M and F populations. Finally an analysis of GACD with respect to schema sampling is presented where it is shown that in most situations the lower bound of the number of instances of a schema sampled by GACD is better than that of CGA.

5.3.1 Terminology

p : the total population size which is assumed to be constant.

$p_m(t)$: the M population size at time t.

$p_f(t)$: the F population size at time t.

$\overline{f}$: the average fitness of the entire population.

h : a schema.

$\overline{f_h}$: the average fitness of instances of schema h over the entire population.

$\overline{f_m}$: the average fitness of the M population.

$\overline{f_f}$: the average fitness of the F population.

l : the length of a string.

$m(h,t)$: no. of instances of schema h in the population at time t.

$m_m(h,t)$: no. of instances of schema h in the M population at time t.

$m_f(h,t)$: no. of instances of schema h in the F population at time t.

$\delta(h)$: the defining length of the schema h.

$O(h)$: the order of the schema h.

μ_c : the probability of crossover.

μ_m : the probability of mutation.

Superscripts s and c with any of the above mentioned symbols indicate the corresponding values after selection and crossover respectively. It is to be noted that the following equalities will hold for GACD for any value of t.

$$p = p_m(t) + p_f(t) \tag{5.2}$$

$$\overline{f} = \frac{\overline{f_m} * p_m + \overline{f_f} * p_f}{p} \tag{5.3}$$

$$m(h,t) = m_m(h,t) + m_f(h,t) \tag{5.4}$$

5.3.2 Analysis of GACD

Let us consider the effects of each operation, *selection, crossover* and *mutation* separately.

Selection : Proportional selection is performed over the entire population. Hence, similar to the treatment provided in Chapter 1, Section 1.3.2, the expected number of instances of the schema h after selection will be given by

$$m^s(h, t+1) = m(h, t) * \frac{\overline{f_h}}{\overline{f}}. \tag{5.5}$$

The expected number of instances of the schema h that will be present in the M and F populations respectively must obviously be proportional to the fraction present in the two populations before selection takes place. Or, in other words,

$$m^s_m(h, t+1) = m^s(h, t+1) * \frac{m_m(h, t)}{m(h, t)}. \tag{5.6}$$

Similarly,

$$m^s_f(h, t+1) = m^s(h, t+1) * \frac{m_f(h, t)}{m(h, t)}. \tag{5.7}$$

Crossover : To analyze the effect of crossover (assuming single point crossover) on the instances of the schema h, its probability of disruption is first calculated. Instances of the schema that are members of the M population are considered first. The analysis for the F population is analogous. For the present, let us assume that an instance of the schema from the M population, if not disrupted by crossover, is placed in the M population again.

Schema h will most likely be disrupted due to crossover if all the following conditions hold.

1. Crossover occurs (with probability μ_c).
2. Crossover site falls within the first and the last defining positions (with probability $\frac{\delta(h)}{l-1}$).
3. Crossover occurs with an instance of some schema h^* in the F population such that h^* is not contained in h (with probability $1 - \frac{m^s_f(h,t+1)}{p^s_f(t+1)}$).

(Note that if h^* is contained in h, then crossover can never disrupt h i.e., schema h will survive in both the offspring. Schema h^*, on the other hand, may not survive crossover at all.)

Taking the above mentioned three conditions into account, the probability of disruption of h in one instance of the schema may be written as

$$\mu_c * \frac{\delta(h)}{l-1} * \left(1 - \frac{m_f^s(h, t+1)}{p_f^s(t+1)}\right). \tag{5.8}$$

Hence the probability of survival of one instance of the schema in the M population is given by

$$1 - \mu_c * \frac{\delta(h)}{l-1} * \left(1 - \frac{m_f^s(h, t+1)}{p_f^s(t+1)}\right). \tag{5.9}$$

Consequently, considering $m_m^s(h, t+1)$ instances (expected after selection), after crossover we get

$$m_m^c(h, t+1) \geq m_m^s(h, t+1)\left(1 - \mu_c * \frac{\delta(h)}{l-1} * \left[1 - \frac{m_f^s(h, t+1)}{p_f^s(t+1)}\right]\right). \tag{5.10}$$

The greater than sign comes because even after disruptive crossover, the schema h may survive.

Similarly the expected number of instances of h that will survive crossover in the F population is given by the relation

$$m_f^c(h, t+1) \geq m_f^s(h, t+1)\left(1 - \mu_c * \frac{\delta(h)}{l-1} * \left[1 - \frac{m_m^s(h, t+1)}{p_m^s(t+1)}\right]\right). \tag{5.11}$$

It had previously been assumed that if an instance of h is present in the M (or F) population, and if h is not disrupted due to crossover, then it survives in the M (or F) population. In reality the situation may not be so. Let P_1 be the probability that h survives in the M population, when it is originally present in the M population. Hence $(1 - P_1)$ is the probability that h goes to the F population after crossover. Similarly let P_2 and $(1 - P_2)$ be the probabilities that h survives in the M and F populations respectively, when it is originally present in the F population.

Thus the modified equation for schema survival due to crossover is :

$$m_m^{*c}(h, t+1) = P_1 * m_m^c(h, t+1) + P_2 * m_f^c(h, t+1).$$

The second term is introduced on considering the instances of h that are present in the F population, which survive crossover but are placed in the M population. Similarly,

$$m_f^{*c}(h, t+1) = (1 - P_2) * m_f^c(h, t+1) + (1 - P_1) * m_m^c(h, t+1).$$

Therefore the expected number of instances of h present in the entire population after crossover is

$$\begin{aligned} m^c(h, t+1) &= P_1 * m_m^c(h, t+1) + P_2 * m_f^c(h, t+1) + (1 - P_2) * m_f^c(h, t+1) \\ &\quad + (1 - P_1) * m_m^c(h, t+1) \\ &= m_m^c(h, t+1) + m_f^c(h, t+1). \end{aligned}$$

Or,

$$\begin{aligned} m^c(h, t+1) \geq \quad & m_m^s(h, t+1) \left\{ 1 - \frac{\mu_c \delta(h)}{l-1} \left[1 - \frac{m_f^s(h, t+1)}{p_f^s(t+1)} \right] \right\} \\ & + m_f^s(h, t+1) \left\{ 1 - \frac{\mu_c \delta(h)}{l-1} \left[1 - \frac{m_m^s(h, t+1)}{p_m^s(t+1)} \right] \right\}. \end{aligned} \tag{5.12}$$

Using Eqs.(5.6) and (5.7), the r.h.s. of inequality (5.12) may be written as

$$\begin{aligned} & \frac{m^s(h, t+1)}{m(h, t)} \left\{ m_m(h, t) + m_f(h, t) - \frac{\mu_c \delta(h)}{l-1} \left[m_m(h, t) \left(1 - \frac{m_f^s(h, t+1)}{p_f^s(t+1)} \right) \right. \right. \\ & \qquad \left. \left. + m_f(h, t) \left(1 - \frac{m_m^s(h, t+1)}{p_m^s(t+1)} \right) \right] \right\} \end{aligned}$$

$$= m^s(h, t+1) \left(1 - \frac{\mu_c \delta(h)}{(l-1)} \left[1 - \left\{ \frac{m_m(h, t) m_f^s(h, t+1)}{p_f^s(t+1) m(h, t)} + \frac{m_f(h, t) m_m^s(h, t+1)}{p_m^s(t+1) m(h, t)} \right\} \right] \right).$$

Let us denote the term in the curly brackets by α, i.e.,

$$\alpha = \frac{m_m(h, t) m_f^s(h, t+1)}{p_f^s(t+1) m(h, t)} + \frac{m_f(h, t) m_m^s(h, t+1)}{p_m^s(t+1) m(h, t)}. \tag{5.13}$$

Therefore,

$$m_{GACD}^c(h, t+1) \geq m^s(h, t+1) \left(1 - \mu_c \frac{\delta(h)}{l-1} [1 - \alpha] \right). \tag{5.14}$$

In this context a slight modification of the schema theorem of Section 1.3.2 is called for which provides a better lower bound of the number of instances of h that survive

after selection and crossover. An instance of schema h may be disrupted due to crossover *iff* it is crossed with an instance of another schema h^* such that h^* is not contained in h and the other conditions for disruptive crossover hold. Accounting for this detail, the disruption probability should be recalculated as

$$\mu_c \frac{\delta(h)}{l-1} \left(1 - \frac{m^s(h, t+1)}{p}\right).$$

Hence after selection and crossover

$$m^c_{CGA}(h, t+1) \geq m^s(h, t+1) \left\{1 - \mu_c \frac{\delta(h)}{l-1} \left[1 - \frac{m^s(h, t+1)}{p}\right]\right\}. \tag{5.15}$$

Let us denote the term $\frac{m^s(h,t+1)}{p}$ by β. Thus

$$m^c_{CGA}(h, t+1) \geq m^s(h, t+1) \left\{1 - \mu_c \frac{\delta(h)}{l-1} [1 - \beta]\right\}.$$

Mutation : Since the conventional bit by bit mutation is applied on the strings with a probability μ_m, the probability of disruption of one bit of the schema is μ_m. Probability of its survival is $1 - \mu_m$. Hence the probability of survival of the schema is $(1 - \mu_m)^{O(h)}$. Thus the expected number of instances of the schema h that are present in the population at time $t + 1$ (after selection, crossover and mutation) is given by

$$m_{GACD}(h, t+1) \geq m^s(h, t+1) \left\{1 - \frac{\mu_c \delta(h)}{(l-1)} (1 - \alpha)\right\} \left\{(1 - \mu_m)^{O(h)}\right\}.$$

Approximating the r.h.s., the inequality may be written as

$$m_{GACD}(h, t+1) \geq m^s(h, t+1) \left\{1 - \frac{\mu_c \delta(h)}{l-1} (1 - \alpha) - \mu_m O(h)\right\}. \tag{5.16}$$

Similarly, the equation for CGA is given by

$$m_{CGA}(h, t+1) \geq m^s(h, t+1) \left\{1 - \frac{\mu_c \delta(h)}{l-1} (1 - \beta) - \mu_m O(h)\right\}. \tag{5.17}$$

♣

In order to compare $m_{GACD}(h, t+1)$ and $m_{CGA}(h, t+1)$, we have to consider the following cases.

Case i : Let $m_m(h,t) = m_f(h,t) = m_1$. In that case $m_m^s(h,t+1) = m_f^s(h,t+1) = m_2$. Note that $m(h,t) = 2m_1$, $m^s(h,t+1) = 2m_2$ and $\beta = \frac{2m_2}{p}$. Then

$$\begin{aligned}\alpha &= \frac{1}{2m_1}\left(\frac{m_1 m_2}{p_f^s(t+1)} + \frac{m_1 m_2}{p_m^s(t+1)}\right) \\ &= \frac{m_2}{2}\left(\frac{p}{p_f^s(t+1)p_m^s(t+1)}\right) \\ &= \beta\left[\frac{p^2}{4p_f^s(t+1)p_m^s(t+1)}\right].\end{aligned}$$

The minimum value of the term in square brackets is 1 when $p_m^s(t+1) = p_f^s(t+1)$. Hence $\alpha \geq \beta$. This in turn indicates that $lower\_bound(m_{GACD}(h,t+1)) \geq lower\_bound(m_{CGA}(h,t+1))$, i.e., the lower bound of the number of instances of some schema h sampled by GACD is better than that of CGA.

Case ii : Let $m_m(h,t) \neq m_f(h,t)$. Let $m_m(h,t) = \gamma m_f(h,t)$ where $\gamma \neq 1$. Then $m_m^s(h,t+1) = \gamma m_f^s(h,t+1)$. Note that $m(h,t) = m_f(h,t)(1+\gamma)$, $m^s(h,t+1) = m_f^s(h,t+1)(1+\gamma)$ and $\beta = \frac{m_f^s(h,t+1)(1+\gamma)}{p}$. Thus, we may write

$$\begin{aligned}\alpha &= \frac{1}{m_f(h,t)(1+\gamma)}\left(\frac{\gamma m_f(h,t) m_f^s(h,t+1)}{p_f^s(t+1)} + \frac{m_f(h,t)\gamma m_f^s(h,t+1)}{p_m^s(t+1)}\right) \\ &= \frac{m_f^s(h,t+1)\gamma}{(1+\gamma)}\left(\frac{p}{p_f^s(t+1)p_m^s(t+1)}\right) \\ &= \frac{m_f^s(h,t+1)(1+\gamma)}{p}\frac{\gamma}{(1+\gamma)^2}\frac{p^2}{p_f^s(t+1)p_m^s(t+1)} \\ &= \beta\frac{\gamma}{(1+\gamma)^2}\frac{p^2}{p_f^s(t+1)p_m^s(t+1)}.\end{aligned}$$

Now, in this case $\alpha \geq \beta$ if the following holds :

$$\frac{\gamma}{(1+\gamma)^2}\frac{p^2}{p_f^s(t+1)p_m^s(t+1)} \geq 1.$$

Or,

$$\frac{p}{\sqrt{p_f^s(t+1)p_m^s(t+1)}} \geq \frac{1+\gamma}{\sqrt{\gamma}}. \tag{5.18}$$

Since the above mentioned condition (inequality 5.18) cannot be always ensured, we cannot conclude that $lower\_bound(m_{GACD}(h,t+1)) \geq$

$lower\_bound(m_{CGA}(h, t+1))$. (Note also that both the functions $\frac{(1+\gamma)}{\sqrt{\gamma}}$ and $\frac{p}{\sqrt{p_f^s(t+1)p_m^s(t+1)}}$ have minimum value 2.)

In order to experimentally compare the values of m_{CGA} and m_{GACD}, an optimization problem is considered. Let $f(x) = x^2$ be the function to be optimized. A population size of 30 (initially 15 M and 15 F chromosomes are considered for GACD) and chromosome length of 10 are taken, with $\mu_c = 0.8$ and $\mu_m = 0.01$. The variation of the number of instances of four schemata with different characteristics is presented over the first five generations in Figs. 5.3-5.6. Note that, any instance of the schema 1######### will have x values greater than or equal to 512, whereas, that of 0######### will have values less than or equal to 511. In other words, the resulting strings corresponding to the former one have higher fitness values than those of the latter one.

From the nature of the problem it is obvious that the number of instances of the first three schemata should increase while that of the last schema should decrease in subsequent generations. As expected, this is the case for both CGA and GACD, except a minor temporary experimental deviation in each case (e.g., 2nd generation in Fig. 5.3, 5th generation in Fig. 5.4, 2nd generation in Fig. 5.5 and 2nd generation in Fig. 5.6). However, the growth (decay) rates for schemata with high (low) fitness values are greater for GACD as compared to CGA, although both could find the optimal value of x (=1023). ♣

In the next section we describe an application of GACD for pattern classification with two different crossover probability values μ_c. Unlike the previous chapters, we considered here two values of μ_c to demonstrate the effectiveness of the new concept of chromosome differentiation. (Besides this, another application of GACD to function optimization is shown in Appendix D. This further strengthens the superiority of GACD over conventional GAs.)

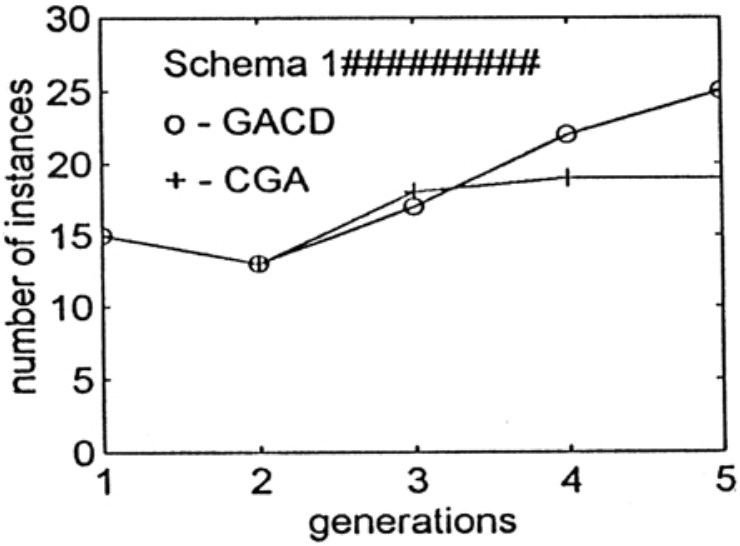


Figure 5.3: Variation of number of instances of schema 1 # # # # # # # # # # with generations for the function $f(x) = x^2$.

5.4 *GACD-classifier* : Pattern Classification Using GACD

The conventional GA (CGA) in *GA-classifier* is replaced by GACD to provide the *GACD-classifier*, whose effectiveness is now being demonstrated on *ADS 1, ADS 2, Vowel, Iris* and *Cancer* data sets described in Chapter 2. Population size of 20 is chosen for this problem. Therefore, initially the number of M and F chromosomes is 10 each. As before, the crossover probability is fixed at 0.8 while the mutation probability is varied in the range [0.015, 0.333] over every 100 generations for a maximum of 1500 generations. Roulette wheel selection, single point crossover and bit mutation are applied. Training sets are the same as used in Chapter 2 and 4.

Comparative results for *GA-classifier* (based on CGA) and *GACD-classifier* (based on GACD) are presented in Tables 5.1 and 5.2 in terms of number of generations for attaining termination criterion (of zero misclassification, or the maximum 1500 generations), and average recognition scores during training and testing.

(Note that it has already been demonstrated in Chapter 2 that the performance of the *GA-classifier* is comparable to, sometimes better than, that of some other classifiers like Bayes maximum likelihood classifier, k-NN rule and multilayered perceptron for these data sets.) The entries shown in Tables 5.1 and 5.2 are the average

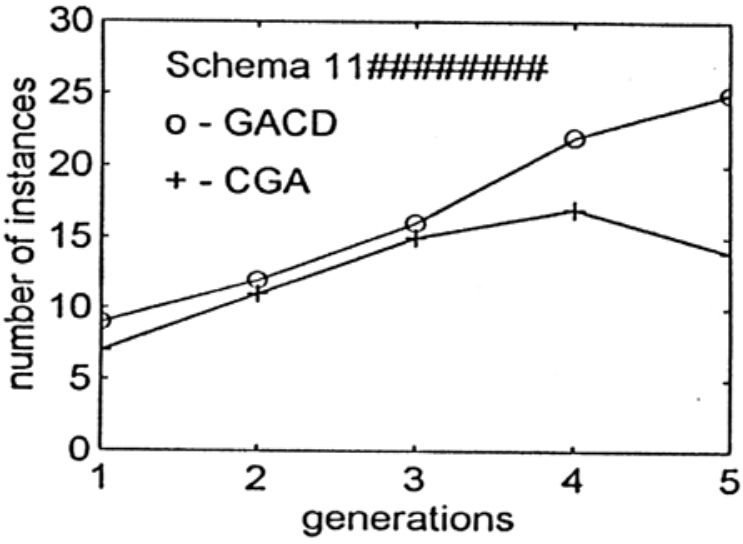


Figure 5.4: Variation of number of instances of schema 1 1 # # # # # # # # with generations for the function $f(x) = x^2$.

values obtained over five different runs of the algorithm from different starting configurations. 10% of the data set is used for training. The remaining 90% data points are used for testing. The value of H is chosen to be 6 based on the previous experiments (in Chapters 2 and 4) with these data sets.

Table 5.1: Comparative results for the pattern classification problems for $\mu_c = 0.8$, $H = 6$

Data Set	Avg. no. of generations		Training (*Avg. recog.*) (*score %*)		Testing (*Avg. recog.*) (*score %*)	
	GACD	CGA	GACD	CGA	GACD	CGA
ADS 1	415.20	512.60	100.00	100.00	97.25	93.22
ADS 2	774.50	898.20	100.00	100.00	95.48	88.29
Iris	14.50	22.20	100.00	100.00	94.67	93.33
Vowel	1500	1500	94.12	82.35	75.19	71.99
Cancer	58.60	280.40	100.00	100.00	93.99	93.01

For all the data sets it is seen from Tables 5.1 and 5.2 that in terms of the recognition scores, *GACD-classifier* performs considerably better than *GA-classifier* for the test data, indicating its better generalization capability. Regarding the training

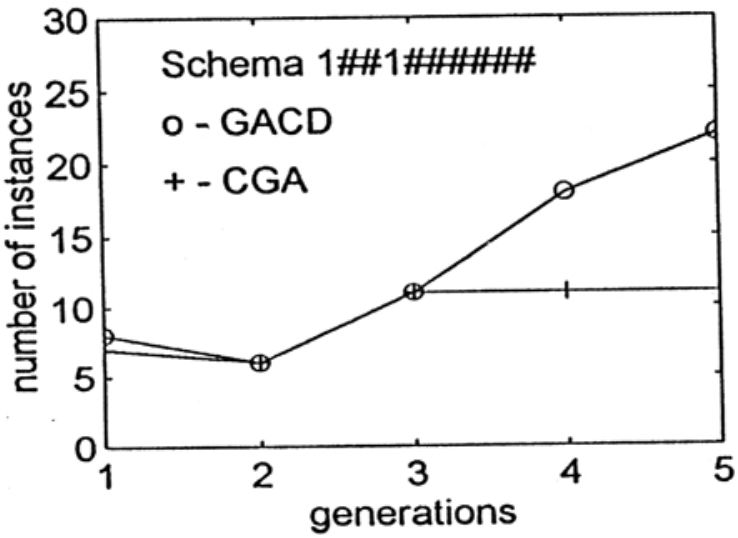


Figure 5.5: Variation of number of instances of schema 1 # # 1 # # # # # # with generations for the function $f(x) = x^2$.

scores, both *GA-classifier* and *GACD-classifier* attain 100% scores for *ADS 1*, *ADS 2*, *Iris* and *Cancer* for $\mu_c = 0.8$ and $\mu_c = 0.5$ (except for *ADS 1*, $\mu_c = 0.5$, when zero misclassification is attained seven out of ten times). Inspite of this, the number of generations required to attain 100% score during training is comparatively less for *GACD-classifier*, indicating its faster convergence.

Since the *Vowel data* has considerable amount of overlap, both *GACD-classifier* and *GA-classifier* fail to attain zero misclassification even until 1500 generations for both values of μ_c. The training recognition scores are found to be much better for *GACD-classifier*.

To demonstrate the variation of the recognition score with H, we present the results for only *Vowel data*, which has considerable amount of overlapping between the classes. Table 5.3 shows the results for $\mu_c = 0.8$. Again, the *GACD-classifier* is found to provide a performance superior to that of *GA-classifier* for all values of H. As expected, it is found that the misclassification during training gradually decreases as H increases for both *GACD-classifier* and *GA-classifier*. However, the results on the test data indicate that increasing the number of hyperplanes does not necessarily increase their generalization capability.

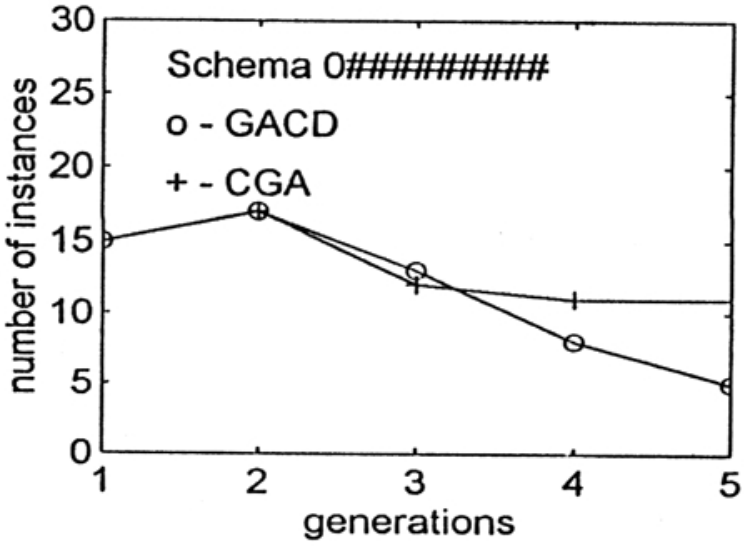


Figure 5.6: Variation of number of instances of schema 0 # # # # # # # # # # with generations for the function $f(x) = x^2$.

5.5 Discussion and Conclusions

The effect of differentiating the chromosomes into two distinct classes in GAs, and its subsequent application to the pattern classification problem have been studied in this chapter. A new genetic methodology called GACD (thereby generating the *GACD-classifier*) is formulated in which two separate populations are maintained and crossover is allowed only between individuals belonging to these two different classes.

GACD is shown to satisfy the schema theorem. It has been proved that in many cases the lower bound of the number of instances of a schema h sampled by GACD is greater than or equal to that of CGA. Because of this, GACD is better able to exploit the information gained so far. Again, initializing the M and F populations in such a way so as to maximize the hamming distance between them, and allowing mating between individuals from these two dissimilar populations, enhances the exploration capability of GACD. As a result, GACD appears to strike a better balance between exploration and exploitation, which is crucial for any adaptive optimization technique, thereby giving it an edge over the conventional GAs.

Experimental evidence for a simple function optimization problem is provided to show that the growth and decay rate for above and below average schemata respectively are greater for GACD as compared to CGA. The superiority of GACD over CGA

Table 5.2: Comparative results for the pattern classification problems for $\mu_c = 0.5$, $H = 6$

Data Set	Avg. no. of generations		Training (*Avg. recog.*) (*score %*)		Testing (*Avg. recog.*) (*score %*)	
	GACD	CGA	GACD	CGA	GACD	CGA
ADS 1	520.20	905.50	100.00	98.71	95.02	94.22
ADS 2	433.50	585.40	100.00	100.00	91.18	85.13
Iris	44.50	195.40	100.00	100.00	94.67	94.67
Vowel	1500	1500	96.47	88.23	76.71	71.37
Cancer	116.40	353.40	100.00	100.00	92.04	87.17

(both in terms of the best value obtained and the average number of iterations required in finding this value) is extensively established through a series of function optimization and pattern classification problems. Although the results demonstrated here assume elitist model, experiments were also conducted for the non-elitist version, and the conclusions as mentioned above still hold.

Two bits are utilized for differentiating the chromosomes into two classes (keeping analogy with the X, Y chromosomes of human beings). Obviously this is not the unique choice. An alternative could have been to use only one bit. However, since we want the class of the offspring to be determined by both the parents, one bit proves to be insufficient. Again, more than two bits could have been used for this purpose; but this would lead to increased computational complexity. Note that we have allowed both 10 and 01 to represent M chromosomes, while only 00 for the F chromosomes. Since the sizes of the M and F populations are kept the same initially, this does not inflict any bias in the process towards the M chromosomes.

In the present method we have incorporated differentiation into two categories. Similar differentiation into more than two classes can be formulated within the same framework in case nature demands so. Further investigations need to be performed to check whether any relationship exists between the size of an individual population (M or F) and the presence of the optimum string in it, and to determine the effect

Table 5.3: Variation of the average recognition score (%) during testing for *Vowel data* with H for $\mu_c = 0.8$

H	Training		Testing	
	GACD-classifier	*GA-classifier*	*GACD-classifier*	*GA-classifier*
4	91.76	82.35	75.83	69.21
5	91.76	83.35	72.90	71.37
6	94.12	82.35	75.19	71.99
7	95.29	88.23	74.94	74.68

of individual population sizes on the overall performance.

It has been proved in [28] that any elitist model of GAs will definitely converge to the optimal string as the number of iterations tends to infinity provided the probability of going from any population to the one containing the optimal string is greater than zero. Note that the conventional mutation operation alone ensures that this probability is greater than zero. Since GACD utilizes the conventional mutation operation and incorporates elitism, the above mentioned criteria are fulfilled. Thus GACD is also guaranteed to provide the optimal string as the number of iterations goes to infinity.

Chapter 6

Relation Between VGA-classifier and MLP : Determination of Network Architecture

6.1 Introduction

In Chapter 4, the *VGA-classifier* has been described where genetic algorithm with variable string length is used for evolving automatically an appropriate number of hyperplanes for modeling the class boundaries (linear, non-linear) of a data set. It has been theoretically shown that for infinitely large training data set, and infinitely large number of iterations, the performance of the *VGA-classifier* approaches that of the Bayes classifier; at the same time the number of hyperplanes required for modeling the class boundaries is minimum.

It is known that the Multilayer Perceptron (MLP) [117, 183, 184], with the neurons executing hard limiting non-linearities, also approximates the class boundaries using piecewise linear segments. Thus, a clear analogy exists between the two methodologies, viz. classification based on MLP and variable string length GA (or, *VGA*). If the parameters of the hyperplanes produced by the *VGA-classifier* are encoded in the connection weights and threshold values of MLP, then the performance provided by *VGA-classifier* and MLP will be the same. The architecture (including connection weights) of MLP can thus be determined from the results of the *VGA-classifier*. (In turn, this will eliminate the need of training the MLP and hence the learning parameters.)

Based on this realization, we describe, in this chapter, a methodology where the architecture along with the connection weights MLP, with each neuron executing the hard limiting function, can be determined automatically using the principle of pattern classification with *VGA* (Chapter 4). During the design procedure, it is guaranteed that the number of hidden layers (excluding the input and output layers) of the MLP thus formed, will be atmost two. The neurons of the first hidden layer are responsible for generation of the equations of hyperplanes. The neurons in the second hidden layers are responsible for generating the regions by performing the AND function, whereas those in the output layer are responsible for producing a combination of different regions by performing the OR function. The algorithm also includes a post processing step which removes the redundant neurons, if there are any, in the hidden/output layers. The performance of the resulting MLP is compared with those of its conventional version and some more with different architectures, for the same set of artificial data, Iris data, speech data and cancer data as described in

Chapter 2. Note that this investigation may also be considered as an application of the *VGA-classifier* for determination of MLP architecture and connection weights with hard limiting neurons.

In this context it should be mentioned that there are several approaches for determining the network architecture and connection weights [32, 60, 123, 173], including those [31, 84, 130, 144, 185, 191, 220] mentioned in Section 1.1. In [123], parallel genetic algorithm is used for evolving the topology and weights of feedforward artificial neural networks. Here both the connectivity and the weights are encoded in the chromosomes. Additionally, the granularity i.e., the number of bits used for encoding the weights is also encoded as a parameter of the problem. This method, thus, utilizes variable string lengths for topology and weight determination. Another method based on the construction of Voronoi diagrams over the set of training patterns is described in [32], where the number of layers, number of neurons in each layer and the connection weights are automatically determined. Fahlman devised the cascade-correlation algorithm for automatic network construction [60]. The algorithm begins with a minimal network, then automatically trains and adds new hidden units one by one, creating a multilayer network. Pruning a network of a large size is another approach towards determination of proper network architecture. A detailed survey on this can be found in [173]. Principles of some of the other methods in this regard are already discussed in Section 1.3.4.

The analogy between the principles of the VGA and MLP based classifiers is described in Section 6.2. This is followed by a detailed description of the network construction algorithm in Section 6.3, along with an example. Section 6.4 describes a post processing step which removes redundant neurons in the second hidden/output layer. The implementation results are presented in Section 6.5. Finally, Section 6.6 provides the discussion and conclusions.

6.2 Analogy between Multilayer Perceptron and *VGA-classifier*

A description of the multilayer perceptron (MLP) is already presented in Section 1.2.3. It is known in the literature [117] that MLP with hard limiting non-

linearities approximates the decision boundaries 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 (e.g., the set of initial random weights in MLP, initial population in VGA). Both of them 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 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 to a large extent. Using the concept of variable string length in a *GA-classifier* further allows it not to 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. In this context one must note that since the *VGA-classifier* has to be terminated after finitely many iterations, and the size of the data set is also finite, it may not always end up with the minimum number of hyperplanes. Consequently, the problem of overfitting exists for *VGA-classifier* also, although it is comparatively reduced.

6.3 Deriving the MLP Architecture and the Connection Weights

In this section we describe how the principle of fitting a number of hyperplanes using GA, for approximating the class boundaries, can be exploited in determining the appropriate architecture along with the connection weights of MLP. Since our aim is to model the equation of hyperplanes, we use the hard limiting function in the neurons of the MLP, defined as

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

6.3.1 Terminology

Let us assume that the *VGA-classifier* provides H_{VGA} hyperplanes, designated by

$$\{Hyp_1, Hyp_2, \ldots, Hyp_{H_{VGA}}\},$$

r regions designated by

$$\{R_1, R_2, \ldots, R_r\},$$

and the k classes designated by

$$\{C_1, C_2, \ldots, C_k\}.$$

Note that more than one region may be labeled with a particular class, indicating that $r \geq k$.

Let R^1 be the region representing class C_1, and let it be a union of r_1 regions given by

$$R^1 = R_{j_1^1} \bigcup R_{j_2^1} \bigcup \ldots R_{j_{r_1}^1}, \qquad 1 \leq j_1^1, j_2^1, \ldots, j_{r_1}^1 \leq r.$$

Generalizing the above, let R^i $(i = 1, 2, \ldots, k)$ be the region representing class C_i, and let it be a union of r_i regions given by

$$R^i = R_{j_1^i} \bigcup R_{j_2^i} \bigcup \ldots \bigcup R_{j_{r_i}^i}, \qquad 1 \leq j_1^i, j_2^i, \ldots, j_{r_i}^i \leq r.$$

Note that each R^i is disjoint, i.e.,

$$R^i \bigcap R^j = \phi \qquad i \neq j, \quad i, j = 1, 2, \ldots, k.$$

6.3.2 Network Construction Algorithm

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

guarantees that the total number of hidden layers (excluding the input and output layers) will be atmost 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 [87]. However, nothing has been stated about the selection of connection weights and the neuronal functions.)

The hyperplanes in *VGA-classifier* are represented by the equation Eq. (2.6). The output of the *VGA-classifier* is the parameters of the H_{VGA} hyperplanes. These are obtained as follows :

$$\begin{array}{cccc} \alpha_1^1, & \alpha_2^1, \ldots, & \alpha_{N-1}^1, & d^1, \\ \alpha_1^2, & \alpha_2^2, \ldots, & \alpha_{N-1}^2, & d^2, \\ & \vdots & & \\ \alpha_1^{H_{VGA}}, & \alpha_2^{H_{VGA}}, \ldots, & \alpha_{N-1}^{H_{VGA}}, & d^{H_{VGA}}. \end{array}$$

where α_is and d are the angles and perpendicular distance values respectively.

Step 1 : Allocate N neurons in the input layer, *layer 0*, where N is the number of input features. The N dimensional input vector is presented to this layer, where the neurons 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 i th hyperplane ($i = 1, 2, \ldots, H_{VGA}$) be

$$c_1^i x_1 + c_2^i x_2 + \ldots + c_N^i x_N - d = 0,$$

where from Eq. (2.6) we may write

$$\begin{aligned} c_N^i &= \cos\alpha_{N-1}^i \\ c_{N-1}^i &= \cos\alpha_{N-2}^i \sin\alpha_{N-1}^i \\ c_{N-2}^i &= \cos\alpha_{N-3}^i \sin\alpha_{N-2}^i \sin\alpha_{N-1}^i \\ &\vdots \\ c_1^i &= \cos\alpha_0^i \sin\alpha_1^i \ldots \sin\alpha_{N-1}^i \\ &= \sin\alpha_1^i \ldots \sin\alpha_{N-1}^i \end{aligned}$$

since $\alpha_0^i = 0$.

Then the corresponding weight on the link to the ith neuron in *layer 1* from the jth neuron in *layer 0* is

$$w_{ij}^1 = c_j^i \qquad j = 1, 2, \ldots, N$$

and the threshold is

$$\theta_i^1 = -d^i$$

(since the bias term 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 $(i = 1, 2, \ldots, r)$ lies on the positive side of the jth hyperplane Hyp_j $(j = 1, 2, \ldots, H_{VGA})$, then the weight on the link between the ith neuron in *layer 2* and the jth neuron in *layer 1* is

$$w_{ij}^2 = +1.$$

Otherwise

$$w_{ij}^2 = -1,$$

and

$$\theta_i^2 = -(H_{VGA} - 0.5).$$

Note that the neurons in this layer effectively serve the AND function, such that the output is high (+1) if and only if all the inputs are high (+1). Otherwise, the output is low (-1).

Step 4 : Allocate k neurons in *layer 3* (output layer), 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 $(i = 1, 2, \ldots, k)$ be a combination of r_i regions. That is,

$$R^i = R_{j_1^i} \bigcup R_{j_2^i} \bigcup \ldots \bigcup R_{j_{r_i}^i}.$$

Then the ith neuron of *layer 3*, $(i = 1, 2, \ldots, k)$, is connected to neurons $j_1^i, j_2^i \ldots j_{r_i}^i$ of *layer 2* and,

$$w_{ij}^3 = 1 \qquad j \in \{j_1^i, j_2^i \ldots j_{r_i}^i\}$$

whereas

$$w_{ij}^3 = 0 \qquad j \notin \{j_1^i, j_2^i \ldots j_{r_i}^i\}$$

and

$$\theta_i^3 = r_i - 0.5.$$

Note that the neurons in this layer effectively serve the OR function, such that the output is high (+1) if atleast one of the inputs is high (+1). Otherwise, the output is low (-1). For any given point, the output of atmost one neuron in the output layer, corresponding to the class of the said point, will be high. Also, none of the outputs of the neurons in the output layer 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.

6.3.3 An Example

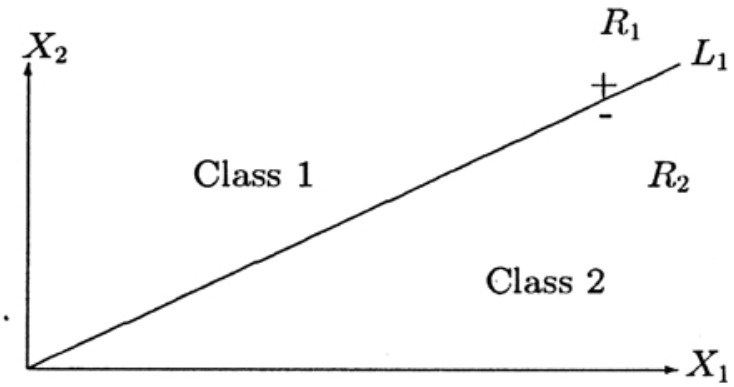


Figure 6.1: Problem for demonstrating the network construction algorithm

In order to demonstrate the functioning of the algorithm, let us consider the problem in Fig. 6.1 for $N = 2$ (two dimensional feature space). Let L_1 be the line result-

ing from the application of *VGA-classifier* for partitioning the two classes shown. The corresponding angle (considering angles from the normal to the X_2 axis in the anticlockwise direction) and perpendicular distance values are

$$L_1 \rightarrow \alpha_1^1 = 315^o, d^1 = 0.$$

In other words, the equation of L_1 is given by

$$x_2 \cos(315) + x_1 \sin(315) = 0,$$

or,

$$\frac{x_2}{\sqrt{2}} - \frac{x_1}{\sqrt{2}} = 0.$$

As can be seen from Fig. 6.1, there are 2 distinct regions viz. R_1, andR_2, of which R_1 represents the region for class 1 and R_2 represents the region for class 2. Also,

$$\begin{aligned} R_1 &\rightarrow \text{+ve side of } L_1 \\ R_2 &\rightarrow \text{-ve side of } L_1. \end{aligned}$$

Applying NCA, we obtain the following in steps :

Step 1 : 2 neurons in the input layer, since N =2.

Step 2 : 1 neuron in *layer 1*, since $H_{VGA} = 1$. The connection weights and the threshold are as follows :

$$\begin{aligned} w_{11}^1 &= \cos\alpha_0^1 * \sin\alpha_1^1 \\ &= -\frac{1}{\sqrt{2}} \\ w_{12}^1 &= \cos\alpha_1^1 \\ &= \frac{1}{\sqrt{2}} \\ \theta_1^1 &= -d^1 \\ &= 0.0 \end{aligned}$$

Step 3 : 2 neurons in *layer 2*, since there are two distinct regions, $r = 2$. The connection weights and the thresholds are as follows :

$$\begin{aligned} w_{11}^2 &= 1 \\ \theta_1^2 &= -0.5 \\ w_{21}^2 &= -1 \\ \theta_2^2 &= -0.5 \end{aligned}$$

Step 4 : 2 neurons in the output layer, *layer 3*, since there are two classes, $k = 2$. The connection weights and the thresholds are as follows :

$$\begin{aligned} w_{11}^3 &= 1 \\ w_{12}^3 &= 0 \\ \theta_1^2 &= 0.5 \\ w_{21}^3 &= 0 \\ w_{22}^3 &= 1 \\ \theta_2^2 &= 0.5 \end{aligned}$$

Note that the zero weights effectively mean that the corresponding connections do not exist. The resulting network is shown in Fig. 6.2.

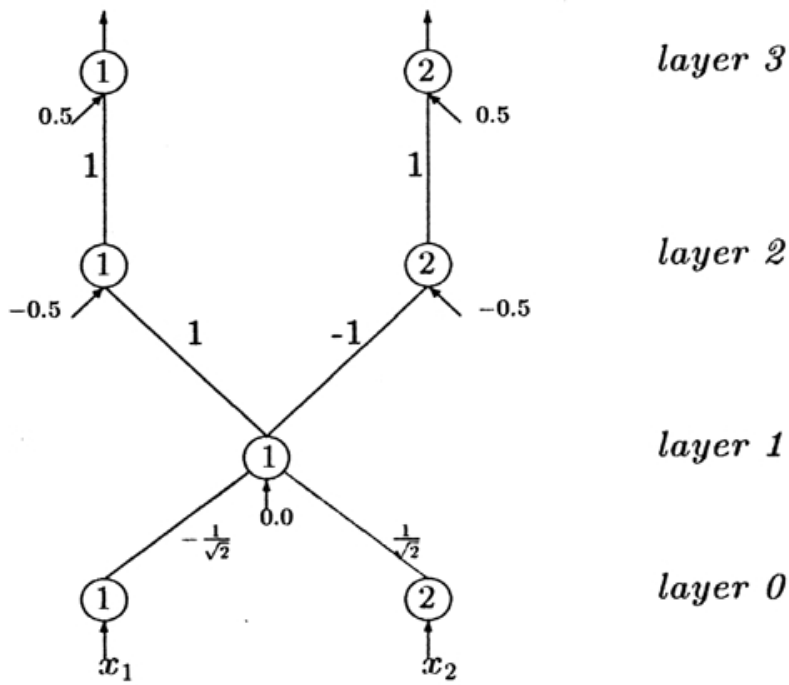


Figure 6.2: Network for the problem in Fig. 6.1

6.4 Post Processing Step

The network obtained from the application of NCA may be further optimized in terms of the links and neurons in the output layer. A neuron in *layer 3* that has an input connection from only one neuron in *layer 2* may be eliminated completely. Mathematically, let for some i, $1 \leq i \leq k$,

$$\begin{aligned} w_{ij}^3 &= 1 \quad \text{if } j = j' \\ &= 0 \quad \text{otherwise.} \end{aligned}$$

Then neuron i of *layer 3* is eliminated and is replaced by neuron j' of *layer 2*. Its output then becomes the output of the network. Note that this step produces a network where a neuron in layer i is 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 will be the case when $r = k$, i.e., a class is associated with exactly one region formed by the H_{VGA} hyperplanes.

Applying the post processing step to the network obtained in Fig. 6.2, we find that neurons 1 and 2 of *layer 3* have links only from neurons 1 and 2 of *layer 2* respectively. Consequently, one entire layer may be removed and this results in a network as shown in Fig. 6.3.

6.5 Implementation

The effectiveness of the network construction algorithm (NCA) is demonstrated on the same data sets mentioned in Chapter 2. As in Chapter 4 a fixed population size of 20 is chosen. *Roulette wheel strategy* is used to implement proportional selection. *Single point crossover* is applied with a fixed crossover probability of 0.8. A variable value of mutation probability μ_m is selected from the range [0.015, 0.333]. 200 iterations are performed with each mutation probability value. The values of μ_{m_1} and μ_{m_2} are set to 0.1. The process is executed for a maximum of 3000 iterations. *Elitism* is incorporated in the process. The recognition scores provided here are the

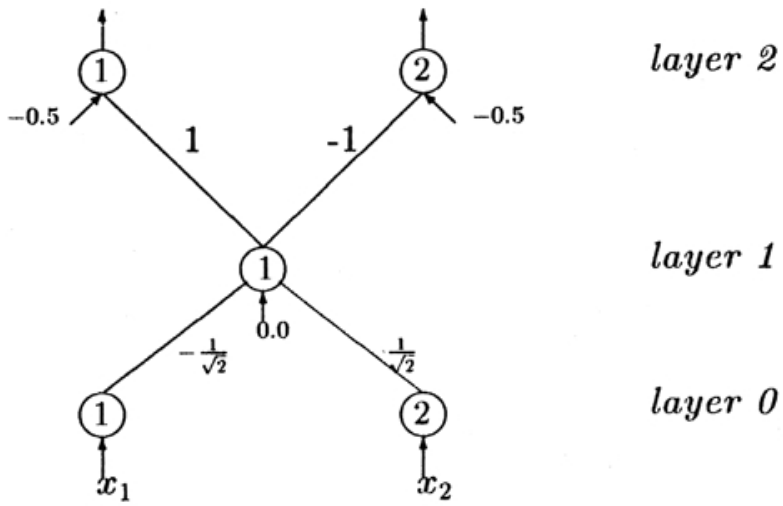


Figure 6.3: Modified network after post processing

average values obtained over five different runs of the algorithm. H_{max} is set to 10, so $\alpha = 0.1$. Training sets are the same as used in the experiments of previous chapters.

The MLP is executed using both hard limiters and the sigmoid function in the neurons. The sigmoid function is defined as

$$f(x) = \frac{1}{1 + e^{-x}}.$$

As in Chapter 2, the learning rate and momentum factor are fixed at 0.8 and 0.2 respectively. Online weight updation, i.e., updation after each training data input, is performed for a maximum of 3000 iterations.

The performance of *VGA-classifier* and consequently that of the MLP derived using NCA (i.e., where the architecture and the connection weights have been determined using NCA) 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 which were used in Chapter 2 viz., those having two hidden layers with 5, 10 and 20 nodes in each layer respectively. Tables 6.1-6.5 summarize the results obtained. The MLP architecture is denoted by Arch. in the tables.

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. These are used to select the MLP architectures as shown in columns 10-11 and 12-13.

From Table 6.1 corresponding to *ADS 1*, it is found that the MLP trained using BP does not succeed in learning the boundaries of class 2, for all the architectures (columns 4-11). In fact, as seen from Fig. 2.6, class 2 is totally surrounded by class 1. The *VGA-classifier*, on the other hand, is able to place the lines appropriately, thereby yielding a significantly better score both during training and testing (columns 2-3). Consequently, the network derived using NCA (which has the performance same as that of the *VGA-classifier*) also provides a significantly better score (columns 12-13).

Similar is the case for *Vowel* and *Iris* data (Tables 6.3 and 6.4 respectively) where the *VGA-classifier*, and consequently the MLP derived using NCA provide a superior performance than the MLPs trained with BP. For *ADS 2* (Table 6.2) and *Cancer* data (Table 6.5), the situation is different where MLPs trained with BP provide superior performance (except one case for *ADS 2*). The overall recognition score during testing for *Vowel* is found to increase with the increase in the number of nodes in the hidden layers (columns 5, 7 and 9) since the classes are highly overlapping. For *Iris* data, the reverse is true, indicating a case of overfitting the classes.

Note that the Arch. values of the MLPs mentioned in columns 10-11 and 12-13 of the tables are the ones obtained without the application of the post processing step. These values are put in order to clearly represent the mapping from *VGA-classifier* to MLP, in terms of the number of hyperplanes and regions, although the post processing task could have reduced the size of the network while keeping the performance same. For example in the case of *Iris* data, the number of hyperplanes and regions are 2 and 3 (columns 2-3) respectively. Keeping analogy with this, the Arch. value in column 12-13 are mentioned to be 4:2:3:3. In practice, after post processing, the said values became 4:2:3. Similarly, for *ADS 1*, *Vowel* and *Cancer* data, the values after post processing were found to be 2:3:4:2, 3:6:2:6 and 9:2:3:2 respectively. In the case of *ADS 2*, there was no change before and after post processing.

6.6 Discussion and Conclusions

A method for automatic determination of MLP architecture and the associated connection weights is described, based on its analogy with the *VGA-classifier* (described in Chapter 4) in terms of placement capability of hyperplanes for approximating the class boundaries. The method guarantees that the architecture will involve atmost 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 providing a combination of regions for the classes.

This investigation may also be considered as an application of the *VGA-classifier*. It not only finds a relation of *VGA-classifier* with the MLP, but also provides a way of determining an appropriate archtecture and connection weights for MLP. Moreover, the said analogy will augment the application domain of *VGA-classifier* to those areas where MLP has widespread use.

It has already been shown in Chapter 4 that the performance of the *VGA-classifier* approaches that of the Bayes classifier as the size of the training data set and the number of iterations go to infinity, while the number of hyperplanes required to model the boundaries becomes minimum. This, in turn, indicates that the MLP derived using the network construction algorithm (NCA) will be the optimum, and its performance will also approach that of the Bayes under limiting conditions.

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.

Table 6.1: Classwise and overall recognition scores (%) for *ADS 1*

Class	*VGA-classifier* $H_{VGA}=3, r=5$		MLP									
			SIGMOID								Hard Limiter	
			Arch.=2:5:5:2		Arch.=2:10:10:2		Arch.=2:20:20:2		Arch.=2:3:5:2		Arch.=2:3:5:2	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
1	100.00	95.89	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	95.89
2	100.00	94.31	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.00	94.31
overall	100.00	95.62	83.63	82.47	83.63	82.47	83.63	82.47	83.63	82.47	100.00	95.62

Table 6.2: Classwise and overall recognition scores (%) for *ADS 2*

Class	*VGA-classifier* $H_{VGA}=6, r=12$		MLP									
			SIGMOID								Hard Limiter	
			Arch.=2:5:5:2		Arch.=2:10:10:2		Arch.=2:20:20:2		Arch.=2:6:12:2		Arch.=2:6:12:2	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
1	100.00	93.52	100.00	95.01	100.00	100.00	100.00	100.00	100.00	100.00	100.00	93.52
2	84.21	78.85	100.00	88.62	84.21	76.57	73.68	68.10	89.47	86.85	84.21	78.85
overall	92.68	88.16	100.00	91.92	92.68	89.09	87.80	85.10	95.12	93.88	92.68	88.16

Table 6.3: Classwise and overall recognition scores (%) for *Vowel*

Class	VGA-classifier $H_{VGA} = 6, r = 7$		MLP									
			SIGMOID								Hard Limiter	
			Arch.=3:5:5:6		Arch.=3:10:10:6		Arch.=3:20:20:6		Arch.=3:6:7:6		Arch.=3:6:7:6	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
ð	10.30	8.21	85.71	12.30	87.51	47.69	100.00	47.23	85.71	6.15	10.30	8.21
a	100.00	91.35	75.00	43.20	100.00	16.04	100.00	27.62	87.50	13.58	100.00	91.35
i	94.11	84.51	100.00	69.67	100.00	79.35	100.00	80.58	100.00	78.70	94.11	84.51
u	73.33	66.91	86.67	80.14	100.00	77.94	100.00	83.29	100.00	91.91	73.33	66.91
e	89.99	85.56	80.00	68.44	94.98	90.37	100.00	87.30	80.00	59.35	89.99	85.56
o	83.33	75.92	88.89	77.16	94.44	54.93	94.44	51.70	77.78	43.21	83.33	75.92
Overall	80.00	73.66	87.05	65.26	95.82	67.55	98.82	68.48	88.23	56.36	80.00	73.66

Table 6.4: Classwise and overall recognition scores (%) for *Iris*

Class	*VGA-classifier* $H_{VGA} = 2, r = 3$		MLP									
			SIGMOID								Hard Limiter	
			Arch.=4:5:5:3		Arch.=4:10:10:3		Arch.=4:20:20:3		Arch.=4:2:3:3		Arch.=4:2:3:3	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
1	100.00	100.00	100.00	100.00	100.00	98.81	100.00	66.67	100.00	100.00	100.00	100.00
2	100.00	93.33	100.00	66.67	100.00	64.77	100.00	71.11	100.00	77.78	100.00	93.33
3	100.00	93.33	100.00	77.78	100.00	80.51	100.00	97.78	100.00	66.67	100.00	93.33
overall	100.00	95.56	100.00	81.48	100.00	81.36	100.00	78.51	100.00	81.48	100.00	95.56

Table 6.5: Classwise and overall recognition scores (%) for *Cancer*

Class	*VGA-classifier* $H_{VGA} = 2, r = 4$		MLP									
			SIGMOID								Hard Limiter	
			Arch.=9:5:5:2		Arch.=9:10:10:2		Arch.=9:20:20:2		Arch.=9:2:4:2		Arch.=9:2:4:2	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing	Training	Testing
1	97.72	98.50	100.00	97.25	100.00	97.25	100.00	97.25	97.65	96.75	97.72	98.50
2	100.00	83.79	100.00	90.74	100.00	90.74	100.00	90.74	100.00	93.75	100.00	83.79
overall	95.50	93.34	100.00	94.97	100.00	94.97	100.00	94.97	98.51	95.70	95.50	93.34

Chapter 7

Application of Genetic Classifiers on Satellite Imagery

7.1 Introduction

In the previous chapters, we have described the effectiveness of GAs in searching for an appropriate number of surfaces for approximating the class boundaries in $\mathbb{R}^N$. Both fixed and variable number of surfaces have been considered in this regard, thereby requiring the implementation of fixed and variable string length GAs, and providing *GA-classifier* and *VGA-classifier* respectively. The theoretical analysis of these genetic classifiers has been provided along with an analogy with the MLP based classification. The effect of incorporating chromosome differentiation for performing restricted crossover operation has also been studied, which led to the generation of *GACD-classifier*. The effectiveness of the genetic classifiers has been extensively demonstrated on several artificial and real life data sets.

In this chapter, another real life application of the classifiers, in a different domain of classification of satellite imagery data, has been shown. Two types of satellite images are considered namely, *LANDSAT* data [140, 141], and the *SPOT* image data [152].

For the *LANDSAT* data, the task of the genetic classifiers is to partition the two dimensional feature space (constituted by the first two principal components of green, red, near infra red, infra red bands) for identifying an unknown pattern as a member of *Manda Granite, Romapahari Granite, Vegetation, Black Phillite,* and *Alluvium* classes. In the case of *SPOT* image, which is characterized by three bands (green, red, near infra red), the *GA-classifiers* are used for pixel classification in order to segment the image space into seven classes viz, *Turbid Water, Concrete, Pure Water, Vegetation, Habitation, Open Space* and *Roads* (including bridges).

An extensive comparison of the performance of genetic classifiers with those of k-NN rule, Bayes maximum likelihood classifier and MLP is presented. For the *LANDSAT* data, where the performance is measured in terms of recognition score, the results again confirm the superiority of the genetic classifiers over the others. For *SPOT* image classification, the segmented output results are also compared with those of a recently developed fuzzy set theoretic multivalued classifier [119, 120, 121].

The comparative performance of all the genetic classifiers developed in this thesis with those of Bayes maximum likelihood classifier, k-NN and rule MLP based classifiers is first provided in Section 7.2 for *LANDSAT* data. Section 7.3 presents the

similar results on *SPOT* satellite image of a part of Calcutta city along with a comparison with Bayes maximum likelihood classifier, k-NN rule, and the multivalued recognition system. A brief description of the multivalued recognition system, which was not described earlier, is included here for convenience. Finally the discussion and conclusions are provided in Section 7.4.

7.2 Classification of *LANDSAT* Data

The data set is generated with the help of the remotely sensed images recorded by the satellite LANDSAT-V, which has two sensors, Multispectral scanner (MSS) and Thematic mapper (TM).

The data used here is obtained by the MSS. The intensity due to any given pixel is resolved on the electromagnetic spectrum into four bands, which are taken to be four features. The four bands are : Green band of wavelength 0.5-0.6 μm, Red band of wavelength 0.6-0.7 μm, Near Infrared band of wavelength 0.7-0.8 μm, Infrared band of wavelength 0.8-1.1 μm.

The area of the earth's surface covered by each pixel is 79m$\times$79m. Since the features are highly correlated, principal component analysis was done to reduce the four features to two principal features. Two classes, Quartzite and Bhusani granite, were removed from the data set, in order to reduce the overlapping to some extent. The details of the data set can be found in [140, 141].

The reduced data set, demonstrated in Fig. 7.1, shows the satellite imagery data of rocks, vegetation and soil. It has 795 samples with five classes, viz. *Manda Granite, Romapahari Granite, Vegetation, Black Phillite* and *Alluvium* [141]. These are referred to as classes 1, 2, 3, 4 and 5 respectively.

7.2.1 Parameters

The experimental parameters are chosen as follows :

Training data set	= 10%
Test data set	= Remaining 90%

Genetic Algorithm :

Maximum iterations (*GA* and *GACD* classifiers)	= 1500
Maximum iterations (*VGA-classifier*)	= 3000
Population size	= 20
Single point crossover probability	= 0.8
Mutation probability range	= [0.015, 0.333]
Selection strategy	Roulette wheel [73]

MLP :

Maximum iterations	= 3000
Learning rate	= 0.8
Momentum factor	= 0.2
Architecture (*Arch 1*)	= 2:5:5
Architecture (*Arch 2*)	= 2:10:10:5
Weight updation	online
Neuron function	sigmoid

k-NN :

k	$= \sqrt{n}$, n = size of training data set

Bayes maximum likelihood classifier :

Class *a priori* probabilities	$= \frac{n_i}{n}$, n_i = number of samples
Unequal dispersion matrices	in class i.

7.2.2 Results

Table 7.1 shows the comparative performance of the classifiers discussed in this thesis on the *LANDSAT* data. Class 3 is seen to be amenable to a consistently good recognition score (except for MLP) 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 (except for MLP) since this is totally overlapped with other classes, especially, with classes 4 and 5.

The *GACD-classifier* with $H = 4$ is seen to provide the best recognition score for this data set while MLP performs very poorly. Interestingly, the different versions of

Table 7.1: Comparative recognition scores (%) during testing for *LANDSAT data* for $perc = 10$

		Class					
		1	2	3	4	5	overall
GA-classifier	$H = 4$	97.62	40.74	100.00	98.45	45.26	82.98
	$H = 6$	76.98	45.37	98.45	94.32	81.05	83.26
GACD-classifier	$H = 4$	90.47	37.96	98.45	96.90	75.78	84.52
	$H = 6$	90.47	57.40	91.75	86.08	46.31	78.80
VGA-classifier	H_{max} =10	86.50	44.44	98.45	89.69	78.94	83.26
Bayes		82.54	75.92	96.91	60.82	71.58	78.10
k-NN		77.77	40.74	100.00	95.36	53.68	79.77
MLP	Arch 1	65.07	70.37	57.21	90.20	33.68	66.38
	Arch 2	42.85	72.22	76.20	79.89	36.31	61.50

the genetic classifiers (except *GACD-classifier* with $H = 6$) yield better recognition scores as compared to those of Bayes maximum likelihood ratio, k-NN rule and MLP based classifiers. It was found that the *VGA-classifier* starting with H_{max}=10 succeeded in reducing the number of surfaces to 5, while providing a good performance at the same time. However, *GACD-classifier*, with only 4 surfaces gives a better score; thereby indicating that *VGA-classifier* provided a suboptimal result. It must be noted that, as proved in Chapter 4, the *VGA-classifier* would have provided an improved performance if more iterations were executed.

The result presented in this section once again underlines our earlier finding that the performance of the genetic classifiers is superior to the conventional classifiers like Bayes maximum likelihood classifier and k-NN rule, as also the MLP based classifier. The *GACD-classifier* is found to outperform *GA-classifier* for $H = 4$ (when it provides the best performance), although for $H = 6$ its recognition score is less. *VGA-classifier* again succeeds in reducing the number of hyperplanes considerably, while keeping the recognition score high.

7.3 Pixel Classification of *SPOT* Image

In this section, an application of *GA-classifier* for classifying the pixels in a *SPOT* image of a part of the city of Calcutta is described, and its performance is compared to those of Bayes maximum likelihood classifier, k-NN classifier and a multivalued (fuzzy) recognition system (MVRS). Since the MVRS is a new addition here, its short description is first of all given before presenting the results.

7.3.1 Multivalued Recognition System (MVRS)

The multivalued recognition system (MVRS), developed by Mandal et al. [120] consists of two phases - Learning and Fuzzy Processing.

In the learning phase, depending on the geometric structure and the relative positions of the pattern classes in the feature space, the training sample set of each pattern class is decomposed into a few sample groups. Accordingly, each individual feature axis is divided into a number of sub-domains, called the feature sub-domains, to highlight the sample groups. Each of the feature sub-domains is extended to an extent to incorporate the portions possibly uncovered by the training samples. Thus the whole feature space is decomposed into some overlapping space sub-domains. Using the space sub-domains, a relational matrix is generated which denotes their compatibility to the different classes.

The fuzzy processing or classification phase, consists of three parts, namely, feature extraction, fuzzy classification and decision making. The feature extractor finds the characteristic vector for each input pattern, whose components denote the degree of belonging of the pattern to the corresponding sub-domain. The fuzzy classifier then uses the modified compositional rule of inference between the characteristic vector and the relational matrix to compute the similarity vector, the components of which denote the degree of similarity of the pattern to a class. The similarity vector is analyzed during decision making to provide multiple class choices viz., single choice, first choice, combined choice, second choice and null choice. The details of the algorithm are mentioned in [120]. The results shown here correspond to first/single choice.

7.3.2 Data Set

The French satellites SPOT (Systems Probataire d'Observation de la Terre) [175], launched in 1986 and 1990, carry two imaging devices that consist of a linear array of charge coupled device (CCD) detectors. Two imaging modes are possible, the multispectral and panchromatic modes. The image considered in this experiment has three bands in the multispectral mode (shown in Figs. 7.2 to 7.4). These bands are :

Band 1 - green band of wavelength 0.50 - 0.59 μm
Band 2 - red band of wavelength 0.61 - 0.68 μm
Band 3 - near infra red band of wavelength 0.79 - 0.89 μm.

The training data set comprises 2105 data points belonging to seven classes, and three features corresponding to the above mentioned three bands. The seven classes are *turbid water, concrete, pure water, vegetation, habitation, open space* and *roads* (including bridges).

Some Characteristic Regions in the Image

Some important landcovers are present in the image (Figs. 7.2 to 7.4). Most of these can be identified, from a knowledge about the area, more easily in Band 3 of the input image (Fig. 7.4). These are the following :

The prominent black stretch across the figure is the river *Hooghly*. Portions of a bridge (referred to as the *second bridge*), which was under construction when the picture was taken, protrude into the *Hooghly* near its bend around the center of the image. There are two distinct black, elongated patches below the river, on the left side of the image. These are water bodies, the one to the left being *Garden Reach lake* and the one to the right being *Khidirpore dockyard*. Just to the right of these water bodies, there is a very thin line, starting from the right bank of the river, and going to the bottom edge of the picture. This is a canal called the *Talis nala*. Above the *Talis nala*, on the right side of the picture, there is a triangular patch, the *race course*. On the top, right hand side of the image, there is a thin line, stretching from the top edge, and ending on the middle, left edge. This is the *Beleghata canal* with a road by its side. There are several roads on the right side of the image, near the

middle and top portions. These are not very obvious from the images. A bridge cuts the river near the top of the image (see Fig. 7.2). This will be referred to as the *first bridge.*

7.3.3 Issue of large value of H

In view of the complexity of the data set, high values of H like 15 and 20 for *GA-classifier* were considered for this image. Since the maximum number of regions provided by H hyperplanes is equal to 2^H, and the number of hyperplanes for this data set is quite large, the value of the number of regions $= 2^H$ became prohibitively large, exceeding the memory limitations of the system. This became a practical limitation of the method. However, an important point that needs to be taken into consideration is that the possible number of regions can never be larger than the number of points n in the training data set. Also, $n << 2^H$ for $H = 15, 20$. Thus we need to consider atmost n regions while tackling this problem. In fact, the number of regions for this problem was found to be considerably less than n as well.

7.3.4 Parameters

The parameters are kept the same as described in Section 7.2.1, except that 100% of the data is used for training. Different values of H considered for the *GA-classifier* are 10, 15 and 20. The values of k, for k-NN classifier, are 1, 3 and $\sqrt{n}$.

7.3.5 Results

The following symbols have been used to identify the different classes in the output images: turbid water - Tu W, pure water - Pu W, concrete - Concr., vegetation - Veg., habitation - Hab., open space - Op Sp, roads (including bridges) - R and unclassified pixels - Uncls. Figs. 7.5 to 7.7 present the output classified images of the *GA-classifier* corresponding to $H = 10$, 15 and 20. Figs. 7.8 to 7.10 present the output images of the classifier based on k-NN rule corresponding to k = 1, 3 and $\sqrt{n}$. Fig. 7.11 presents the results for the Bayes maximum likelihood classifier and Fig. 7.12 presents the classified image based on the first/single choice of MVRS.

From the results it is found that most of the classifiers are able to identify *Hooghly, first bridge, Garden Reach lake, Khidirpore dockyard* and *race course* properly. Overall, the prevalence of concrete on the right bank of *Hooghly* (corresponding to the Calcutta metropolis area), and vegetation, open space and habitation on the left bank (corresponding to Howrah region) is also correct. Also, the pixels corresponding to the *second bridge*, are identified by the classifiers (though to a less extent by the *GA-classifier* for H =15), as those of either concrete or road class.

As seen from Figs. 7.5 to 7.7, *GA-classifier* with H =15 appears to perform better than those with H = 10 and 20. The *Beleghata canal* (which comes out as a mixture of road and water) and some parts of *Talis nala* are seen to be better identified with H = 15 (see Fig. 7.6). A triangular lighter outline within the *race course*, an open space, corresponding to the tracks is identified by *GA-classifier*, with both H =10 and 15, but not with H =20. A point to be noted is that with H =20, many points (10848 out of 262144) remained unclassified. (Such unclassified points are a result of getting test points in regions from where no training point came.) The corresponding values are found to be 2323 and 625 for *GA-classifier* with H = 15 and 10. This is expected, since increasing the number of lines results in closer fit of the data set, thereby decreasing the size of regions that are labelled during training. MVRS also produces a significant number (6613) of unclassified pixels.

It is found that both k-NN rule, with k=1 (Fig. 7.8) and MVRS (Fig. 7.12) fail to retain the shape of the *race course*. In fact, the result for k-NN, with k=1 is found to be quite poor, since it is unable to classify roads, on the right side; being able to identify some scattered road classes (white dots) instead. This is mostly the case for MVRS also (Fig. 7.12). The performance of k-NN rule is found to improve with the value of k, being best for k = $\sqrt{n}$, where the extracted features are found to be much more sharp. However, even for this value of k, *Beleghata canal* comes out as road interspersed with concrete, while *Talis nala* is recognized more as road. MVRS also recognizes *Talis nala* more as road.

Bayes maximum likelihood classifier (Fig. 7.11) is also able to identify most of the structures correctly. However, it also classifies *Talis nala* as mostly road. Moreover, the thick road structures along both banks of *Hooghly* is an overestimation.

7.4 Discussion and Conclusions

In this chapter, another real life application of the *GA-classifier*, for classifying satellite image data has been demonstrated. Two data sets namely, *LANDSAT* data, and *SPOT* data, are used for classifying different landcover types.

For *LANDSAT* data, *GACD-classifier* is found to provide the best result for $H = 4$. The concept of variable string lengths is once again found to be useful for evolving a proper value of H. Overall, all the genetic classifiers performed better than those of the Bayes, k-NN and MLP, thereby strengthening the earlier findings in Chapters 2-5. For *LANDSAT* data, the performance is measured in terms of recognition score, whereas it is the quality of the segmented image output which was considered for analyzing the performance of the classifier for *SPOT* image.

Most of the classifiers are found to identify some characteristic regions in the *SPOT* image, namely, *Hooghly, first bridge, Garden Reach lake, Khidirpore dockyard* and *race course* properly. The performance of the *GA-classifier* is found to be better with H =15 than with $H = 10$ and 20. The number of points that remain unclassified in the image increases with the value of H. It may be noted that the output image obtained by MVRS also contained a significant number of unclassified points. The quality of the segmented image provided by k-NN classifier improves with the value of k. The Bayes maximum likelihood classifier is also able to identify most of the regions in the image correctly, although it overestimates some road structures.

Implementation of the *GA-classifier* for classifying *SPOT* image necessitated a large value of H. This led to the problem of having to consider 2^H, a considerably large value, for the number of regions. This problem has been solved by noting that in practice, the number of regions can never exceed the number of training points n.

Figure 7.1: LANDSAT data

Figure 7.2: SPOT image of Calcutta in the green band (Band 1)

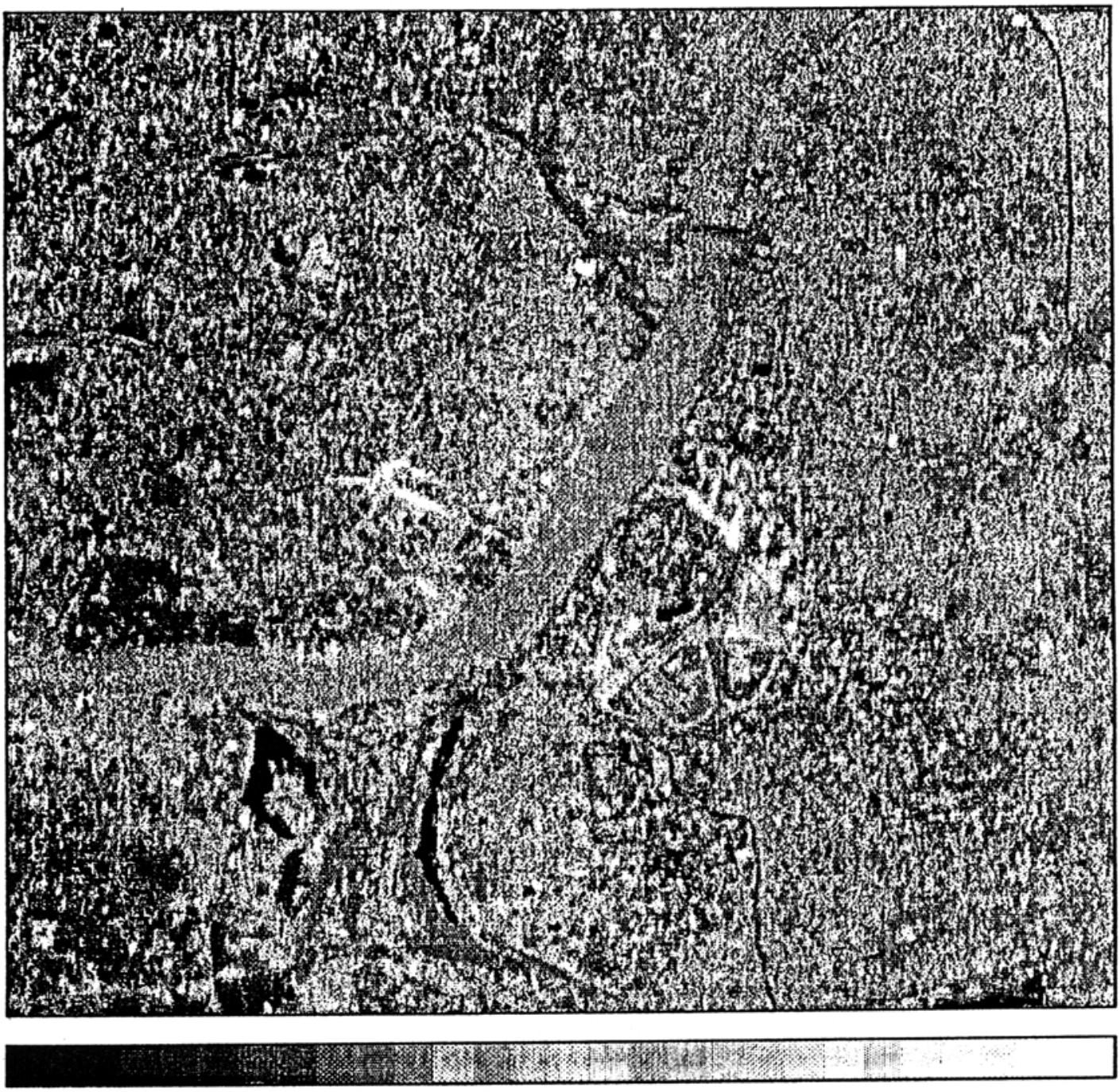


Figure 7.3: SPOT image of Calcutta in the red band (Band 2)

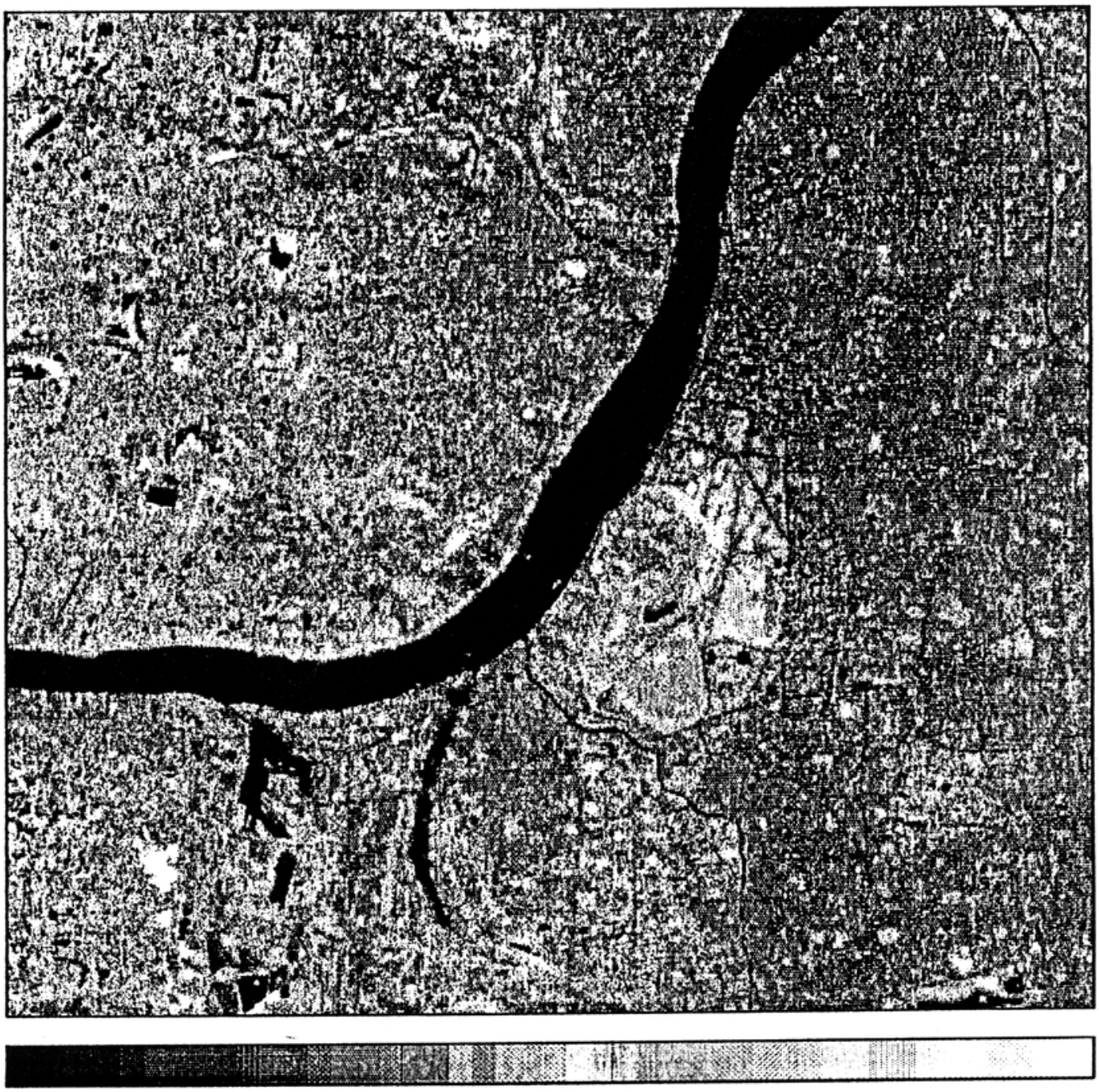


Figure 7.4: SPOT image of Calcutta in the near infra red band (Band 3)

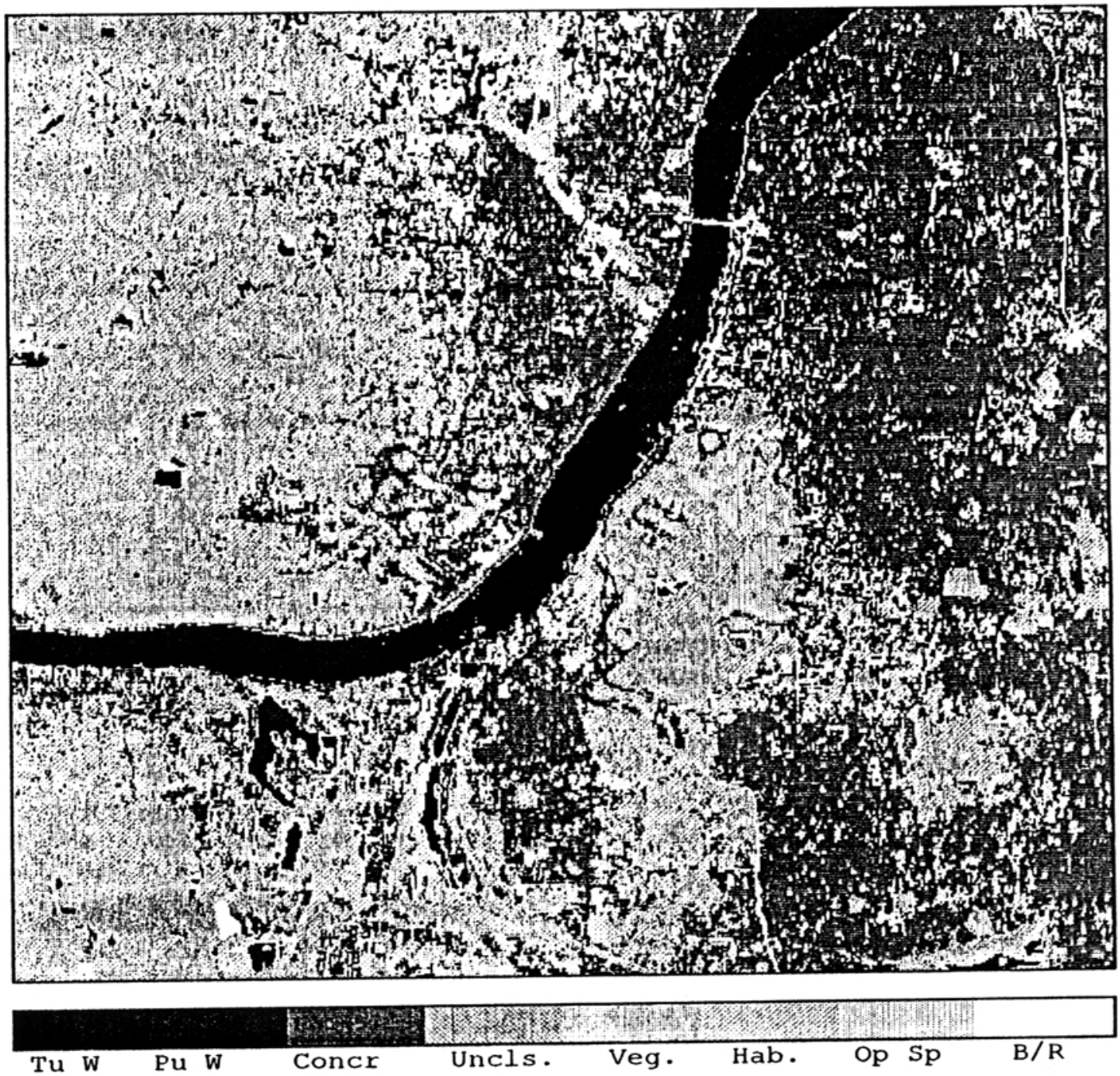


Figure 7.5: Classified image using *GA-classifier*, H =10

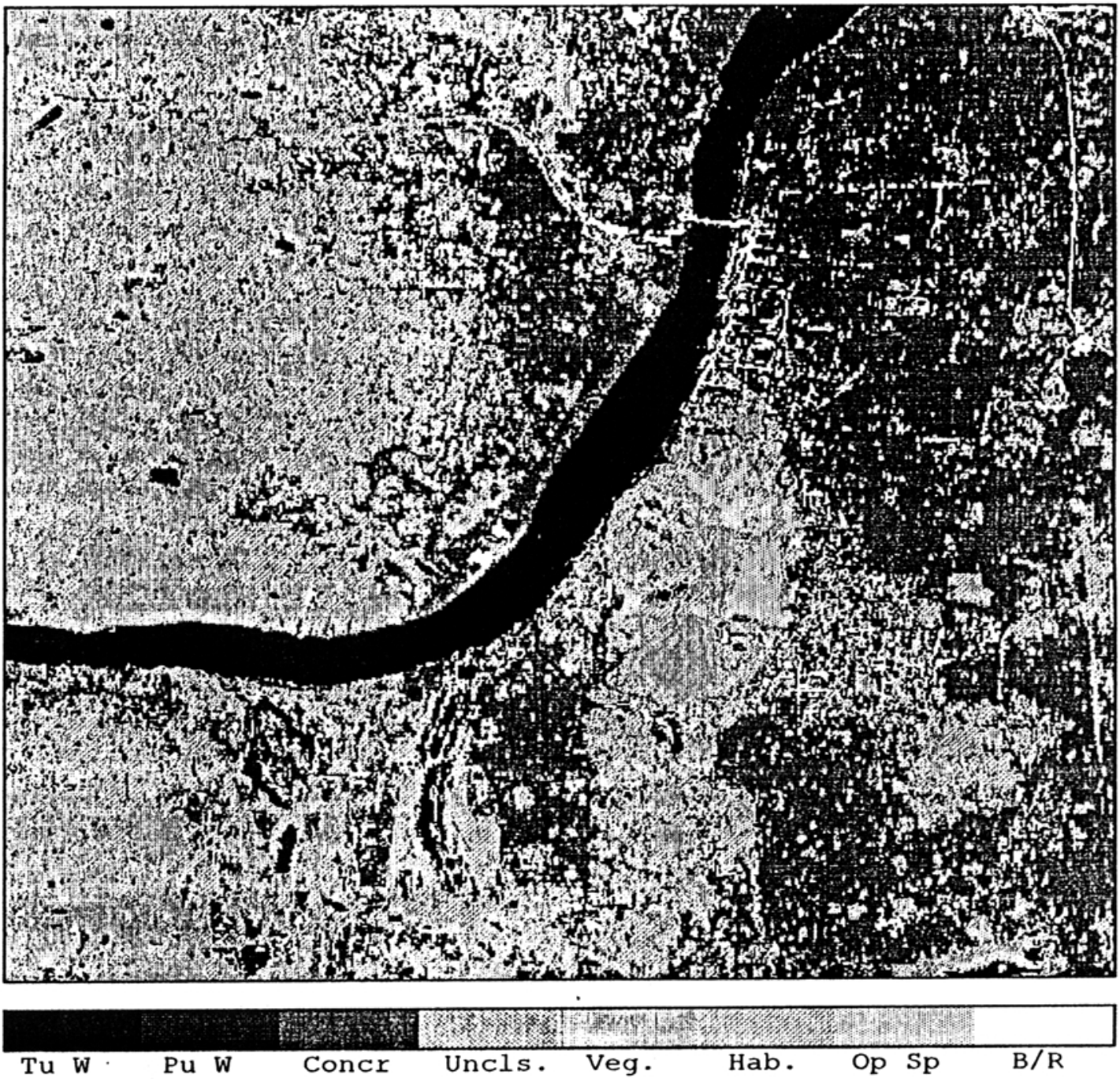


Figure 7.6: Classified image using *GA-classifier*, H =15

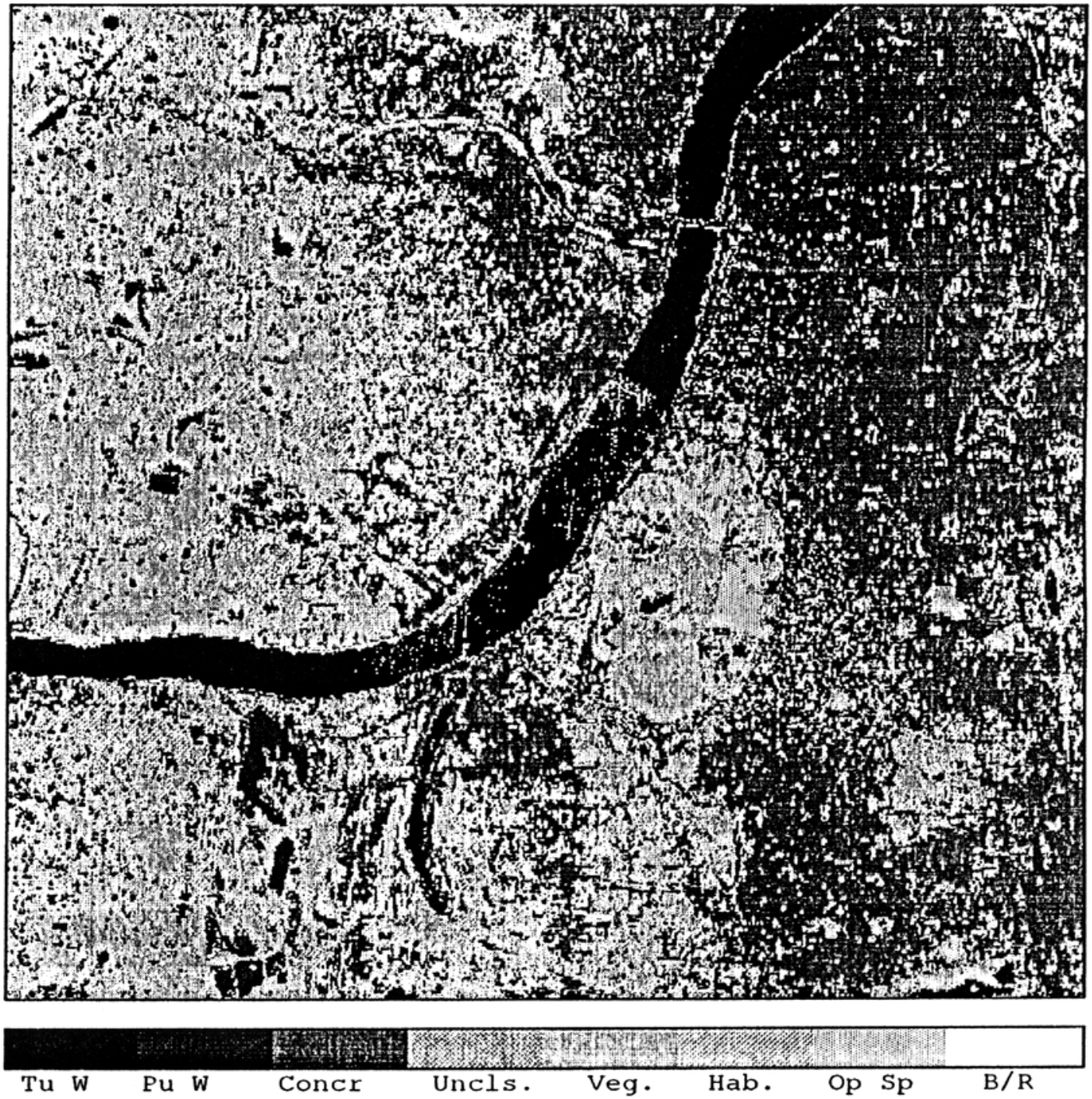


Figure 7.7: Classified image using *GA-classifier*, H =20

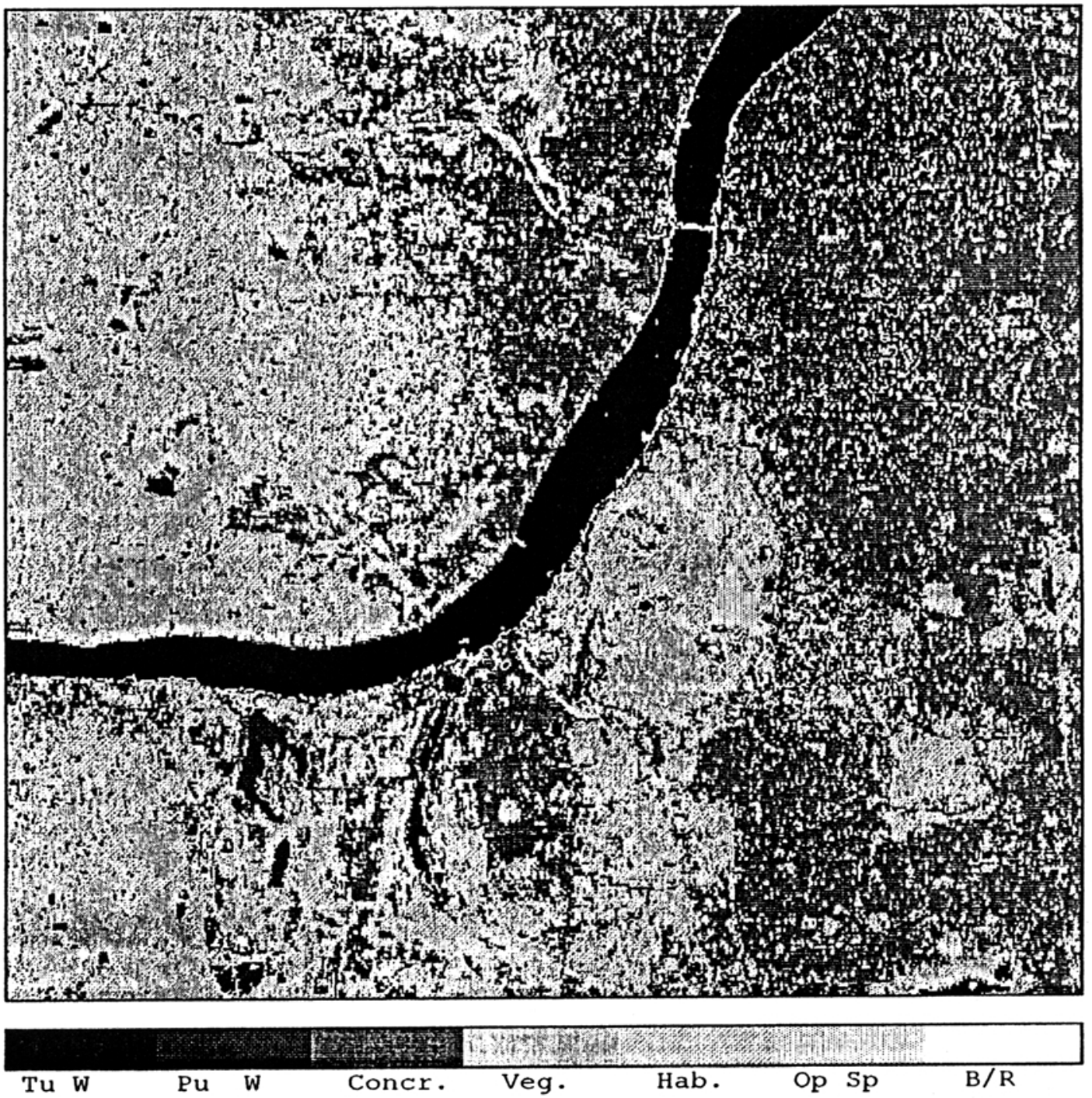


Figure 7.8: Classified image using k-NN, k = 1

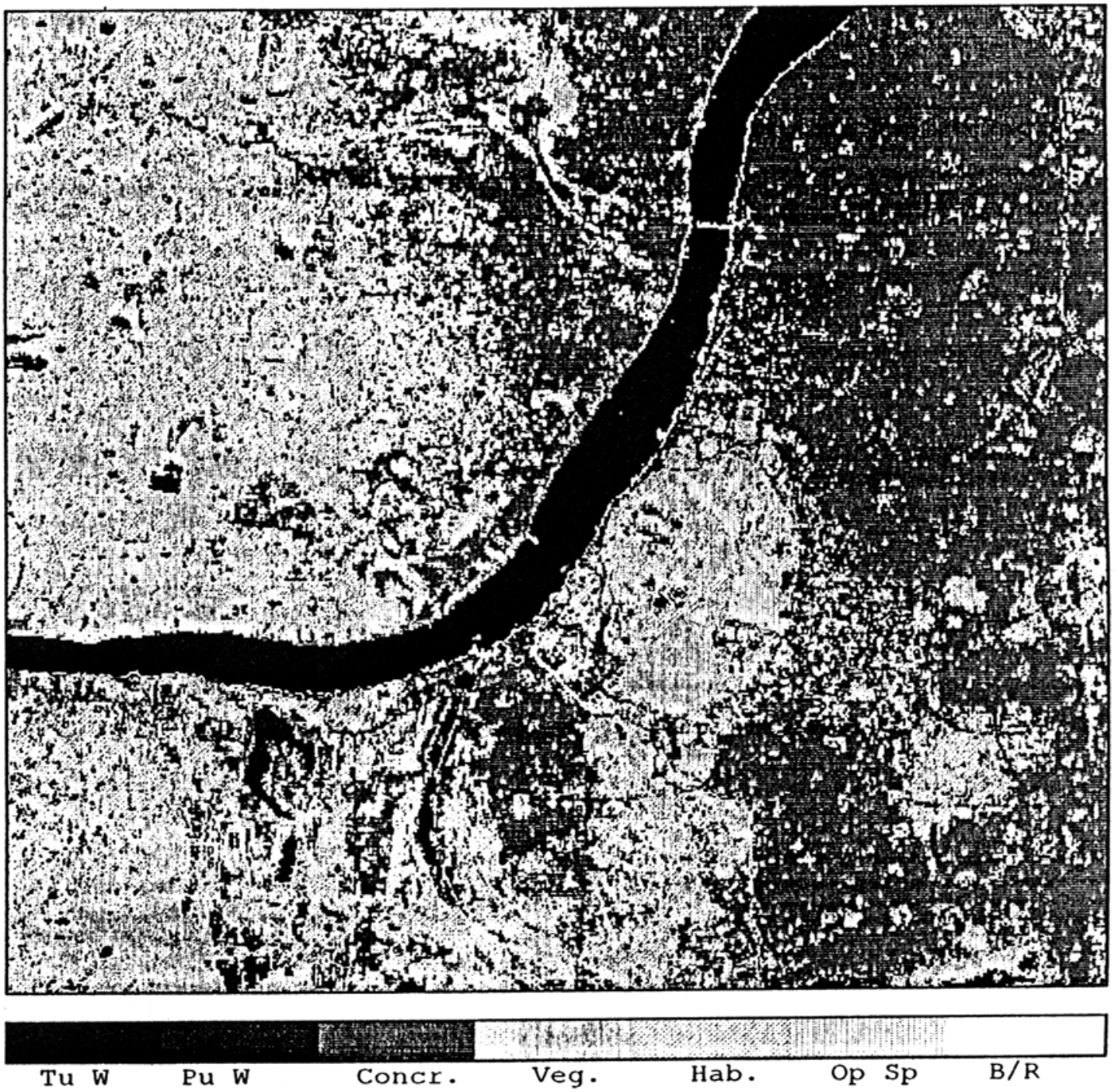


Figure 7.9: Classified image using k-NN, k = 3

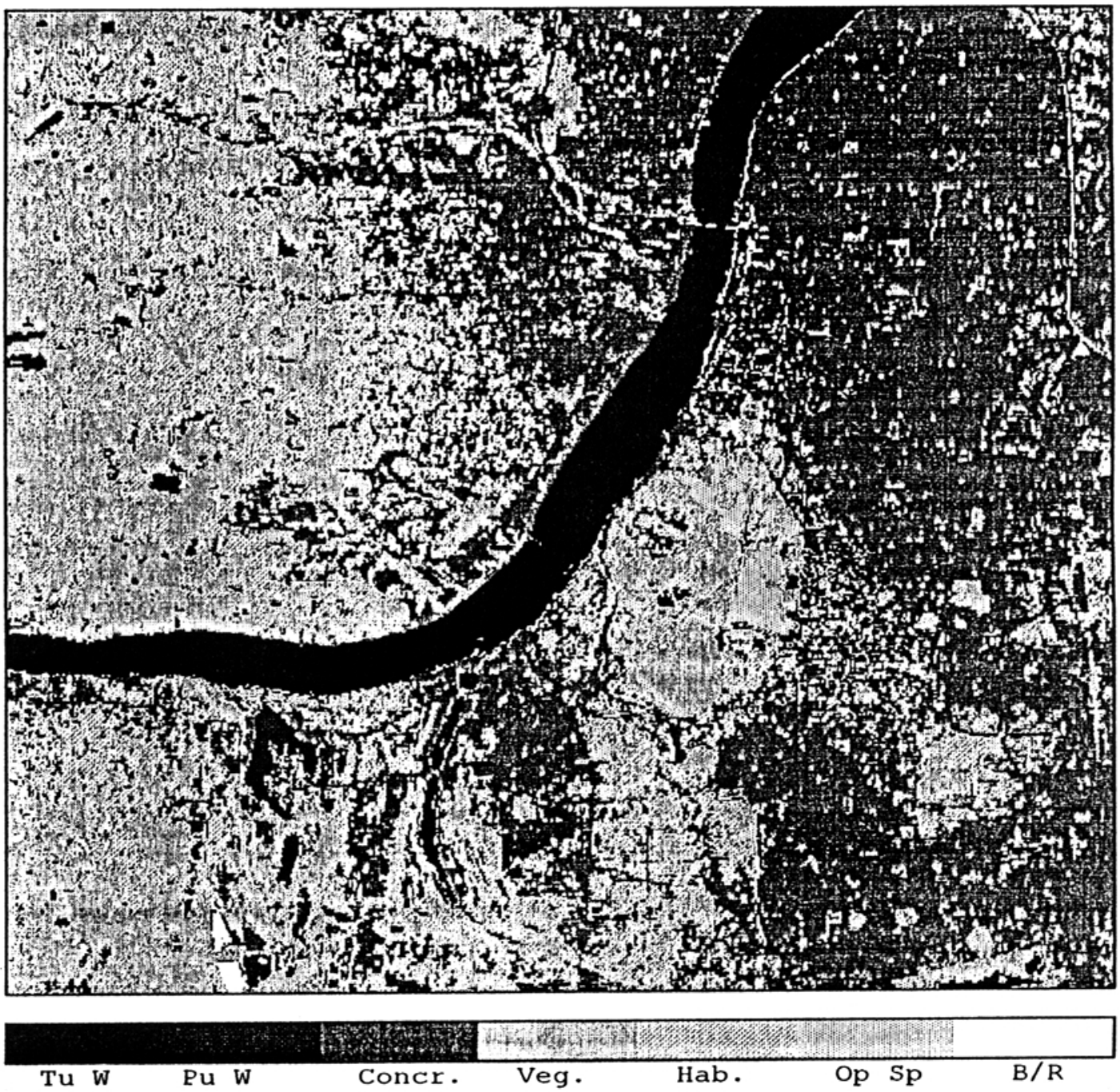


Figure 7.10: Classified image using k-NN, $k = \sqrt{n}$

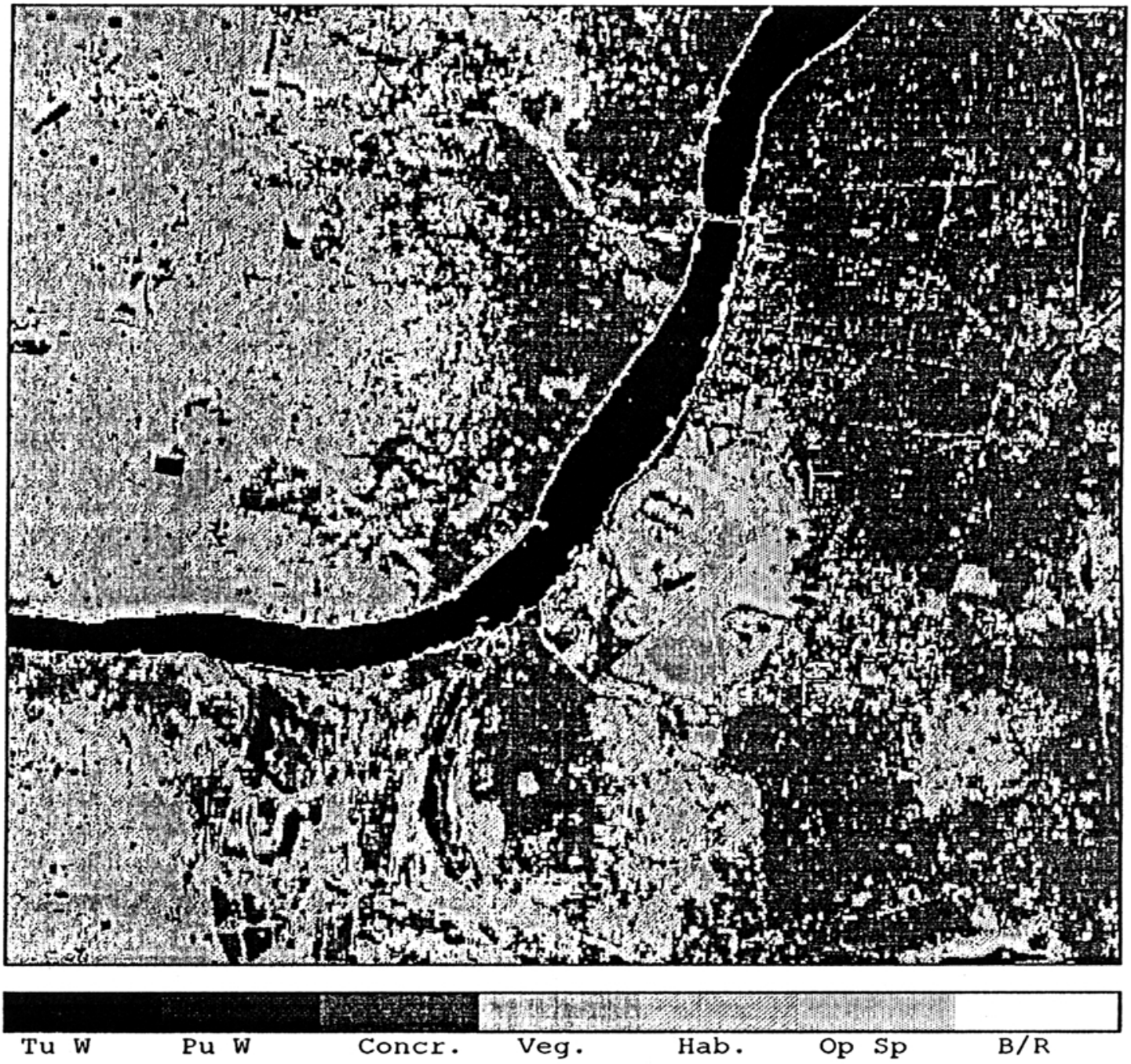


Figure 7.11: Classified image using Bayes rule

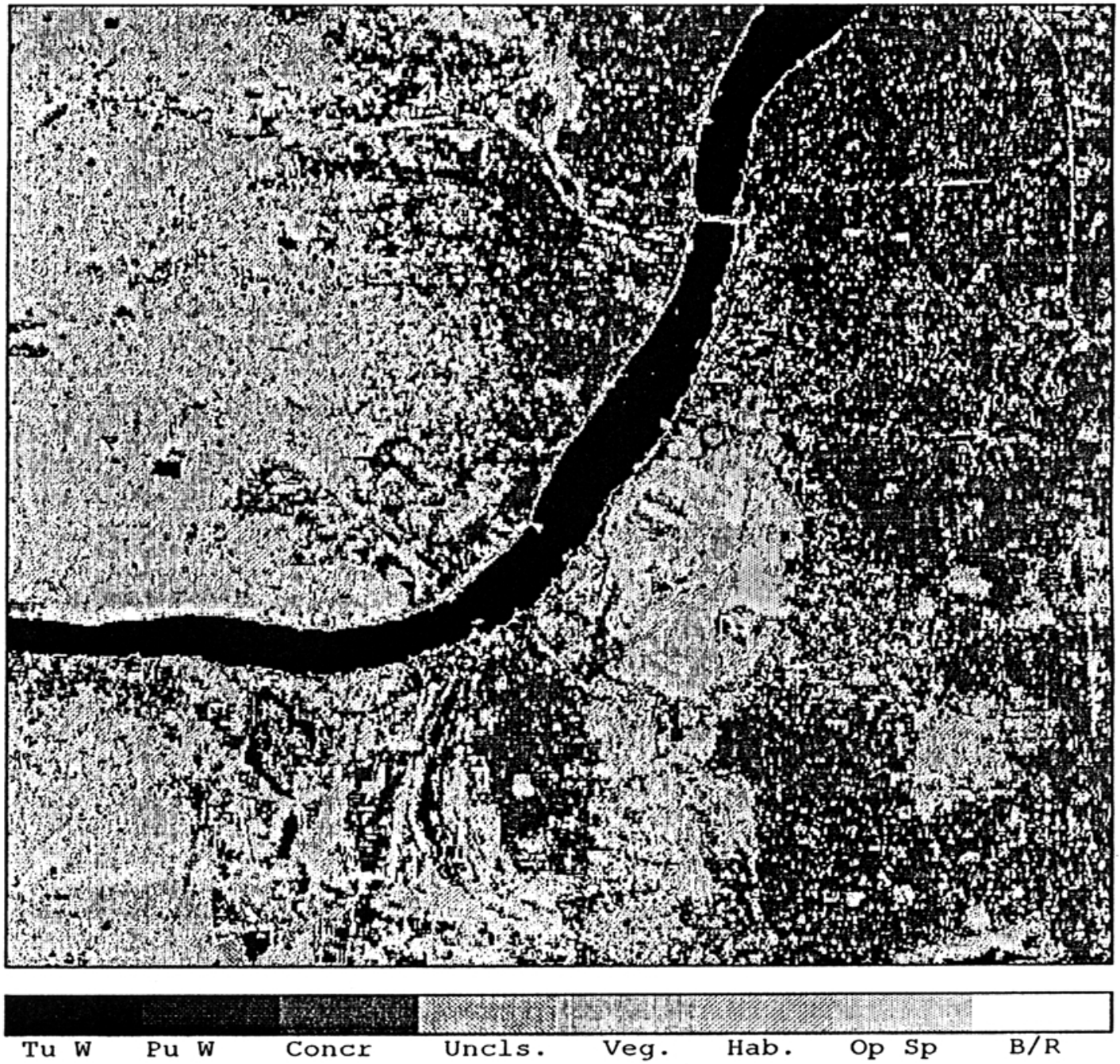


Figure 7.12: Classified image using MVRS

Chapter 8

Conclusions and Scope for Further Research

8.1 Conclusions

The thesis deals with the development of several pattern classifiers, along with their theoretical and practical aspects, using both conventional genetic algorithms and some of their modifications/enhancements. The search and optimization capability of genetic algorithms has been exploited for the placement of an appropriate number of surfaces in the feature space, such that the associated number of misclassified points is minimized. The classifiers store the parameters of the surfaces constituting the final decision boundary, and the region-class associations. These are later used for determining the region and hence the class of an unknown pattern. Utility of both linear and spherical surfaces for modeling the class boundaries is investigated. Various versions of the genetic classifier e.g., *GA-classifier* using fixed number of surfaces, *VGA-classifier* using variable number of surfaces, *GACD-classifier* incorporating the concept of chromosome differentiation, have been formulated, along with their theoretical analyses.

The effectiveness of these genetic classifiers and their comparison with the Bayes maximum likelihood classifier (which is well known for discriminating overlapping classes), k-NN classifier and MLP (where it is known that k-NN classifier and MLP with hard limiters can discriminate well non-overlapping, non-convex regions by generating piecewise linear boundaries) are extensively demonstrated. The way of incorporating the concept of variable string length in *VGA-classifier* is also compared with that of Srikanth et al. [204] in a part of the experiment. Two sets of artificial data, *Iris* data, a speech data, a cancer data and a *LANDSAT* data, with number of features ranging from two to nine, are considered for these purposes.

Besides these, the problem of pixel classification from *SPOT* satellite image for partitioning various landcover types with ill-defined boundaries is considered as another real life application. In a different investigation, a method for the automatic determination of the architecture and connection weights of MLP (with hard limiters) is described, based on its analogy with the VGA based classification methodology.

The genetic classifiers described here are non parametric in nature. They explicitly use the decision boundary in order to classify unknown patterns. This is in contrast to other conventional techniques where the decision boundaries are formed as a consequence of the decision making process.

A method of generating class boundaries in $I\!R^N$, $N \geq 2$, by exploiting the characteristics of GAs for appropriately placing a fixed number of segments is described in Chapter 2. The classifier is first developed for two dimensions, and then generalized to N dimensions. Moreover, utilization of hyperspherical segments for constituting the class boundaries is also investigated. Since an exact value of the number of hyperplanes (H) required for modeling the decision boundary of a given data set is very difficult to find *a priori*, a conservative value of H is used. This may lead to the presence of redundant hyperplanes in the final decision boundary. The method, therefore, includes a scheme for the automatic deletion of such redundant hyperplanes.

Besides leading to the presence of redundant hyperplanes in the decision boundary, a conservative value of H also leads to overfitting of the data, with an associated increase in training performance but reducing the generalization capability of the classifier. These problems are overcome to a large extent by the concept of variable string length in *VGA-classifier* (Chapter 4), where the parameters of a variable number of surfaces are encoded in the chromosomes. The value of H is allowed to vary in a sufficiently large range; the appropriate value is automatically determined by the algorithm. New genetic operators are defined to deal with the concept of variable string lengths for formulating the classifier. The fitness function is defined so that its maximization indicates minimization of the number of misclassified samples as well as the required number of hyperplanes.

Motivated by the various forms of differentiation found in nature, the effect of differentiating the chromosomes of GA into two distinct classes is studied in Chapter 5. The new methodology is called GACD, in which two separate populations are maintained and crossover is allowed only between two chromosomes belonging to these two separate classes. The incorporation of the concept of chromosome differentiation in the aforesaid *GA-classifier* results in the *GACD-classifier*.

Theoretical investigations to find a relationship of the Bayes classifier with the *GA-classifier* and *VGA-classifier* are provided in Chapters 3 and 4 respectively. It is established that for sufficiently large number of iterations, as the size of the training data set increases to infinity, the *error rate* (defined as the ratio of the number of misclassified points to the total number of training samples) provided by these genetic classifiers during training approaches the *Bayes error probability*. In other words, under limiting conditions, the class boundaries, and hence the overall performance,

provided by the genetic classifiers will approach that of the Bayes classifier. Since the Bayes classifier is known to provide an optimal performance when the *a priori* probabilities and the class conditional densities are known, it is desirable to show that the performance of the classifiers developed here approaches that of the Bayes under limiting conditions. These theoretical findings are also verified experimentally for a number of artificially generated data sets following uniform, triangular and normal distributions with known Bayes boundary (linear, non-linear). It is observed that the approximation to the Bayes decision boundary made by these genetic classifiers improves as the size of the training data set increases.

In addition to providing an error rate that approaches the Bayes error probability during training, it has been proved in Chapter 4, that the number of hyperplanes will also be the minimum for the *VGA-classifier* when the size of the training data and the number of iterations go to infinity. Therefore, in the *VGA-classifier*, not only will the minimum value of H be evolved automatically in order to model any kind of decision boundary, but also any value of $H_{max} \geq H_{opt}$ is assured to provide the same performance of the classifier for sufficiently large data set and sufficiently large number of iterations. (Note that the requirement of H_{max} being greater than or equal to H_{opt} can be satisfied easily by selecting a large value of it.) In this context one may note the observation made in Section 4.6 regarding the approximation of Bayes decision boundary by k-NN classifier and MLP. The approximation and hence the performance of these classifiers depend on k and the network architecture respectively.

Regarding the schema analysis of GACD, it has been shown theoretically (Chapter 5) that the lower bound of the number of instances of an above average schema sampled by GACD is greater than or equal to that of the conventional GA in many situations. This claim has also been experimentally verified for a function optimization problem where it is found that the growth rate and the decay rate of the above and below average schemata respectively are higher than those for the conventional GA. GACD is found to perform well for a number of function optimization problems (see Appendix D). It appears that GACD is able to strike a better balance between exploration and exploitation of the search space than its conventional counterpart.

Classification performance of the *GA-classifier* is seen to be comparable to, often better than, those of Bayes maximum likelihood classifier, k-NN classifier and MLP based classifier in discriminating both overlapping and non-overlapping, non-convex

regions. Besides being able to evolve an appropriate number of hyperplanes from a large value, the *VGA-classifier* is found to improve the recognition performance of the *GA-classifier*. Among the genetic classifiers, *GACD-classifier* provided the best performance both in terms of the recognition scores and the number of iterations required for termination when two dimensional artificial data sets *ADS 1* & *ADS 2* and a real life *LANDSAT* data, three dimensional *Vowel* data, four dimensional *Iris* data and nine dimensional *Cancer* data are considered as input.

Since the algorithm of Srikanth et al. [204] does not include a factor for reducing the number of surfaces, it is found to use more hyperplanes than that required by the *VGA-classifier* for constituting the decision boundary. As mentioned in Section 4.5.2, this usually results in a better performance during training, but with poorer generalization capability. Additionally, the execution time is also more since no explicit effort is made to decrease the number of hyperplanes. Note that their method does not have any constraint like the specification of H_{max} in *VGA-classifier*.

SPOT image (three bands) of a part of the city Calcutta is used in Chapter 7 for demonstrating a different real life application of the *GA-classifier* for segmenting various regions (e.g., river, bridge, race course, roads, canal etc.) from satellite imagery. The segmented image outputs of the *GA-classifier* with $H = 10$, 15 and 20 are compared with those of a fuzzy set theoretic recognition system [121] (which is recently developed for detecting ill-defined image regions well), Bayes maximum likelihood classifier and k-NN rule for k = 1, 3 and $\sqrt{n}$. Using large values of H (= 15 and 20) for the *GA-classifier* led to the problem of considering 2^H, a significantly large value, for the number of regions to be labeled. Since the maximum number of regions that may have to be labeled is bounded by the number of training data points (n) (which is much smaller than 2^H, when $H = 15$ and 20), this problem is resolved by considering only n regions in practice.

Exploiting the analogy between the classifiers based on *VGA* and MLP (with hard limiters), a network construction algorithm (NCA) for automatic determination of the MLP architecture and the connection weights is described in Chapter 6 as an application of the *VGA-classifier*. The algorithm guarantees that the architecture will involve atmost two layers, with the neurons in the first and second hidden layers being responsible for hyperplane and region generation, and those in the output providing a combination of regions for the classes. Since the MLP, thus derived using

NCA, is obtained from the principle of *VGA-classifier*, the theoretical results proved for *VGA-classifier* will also hold for the derived MLP. Being based on the principle of *VGA-classifier* makes it mandatory to use hard limiting neurons in the derived MLP. Although this makes the network rigid and susceptible to noise and corruption in the data, NCA may be used for providing a possible appropriate structure of conventional MLPs.

It may be mentioned that the application of GAs to pattern recognition is relatively new, and consequently the literature is not rich. Because of this we have compared our classifiers mostly with some of the widely used ones (e.g., Bayes maximum likelihood classifier, k-NN rule, MLP). Only in one case (Chapter 4) we have implemented the genetic operators of Srikanth et al. [204] in our *VGA-classifier* for comparison.

8.2 Scope for Further Research

In the present thesis, binary representation of chromosomes in linear form has been used, primarily because it is well studied in the literature, and it maximizes the number of schemata sampled by each member of the population; thereby making the implicit parallelism of GAs to be used to the fullest. However, in many practical situations, this may not be a natural choice. Thus, other kinds of representation, like floating point, tree, matrix representation [91, 126] may be studied. Obviously, modifying the representation scheme entails modification of the associated genetic operators and development of new ones. Some such operators are *switch operator*, *translocation operator*, *straight crossover* and *corner crossover* [91].

Since proper selection of genetic parameters viz., probabilities of crossover, mutation, population size, is still an open issue, a study may be made where these values are kept variable (like the mutation probability in this thesis) or are allowed to evolve adaptively. Alternatively, a meta level GA may be used in this regard. Moreover, application of other, more advanced, operators and techniques, like linear inversion, reordering, dominance, niching, segregation, migration, directed mutation, incorporation of ancestors' influence in fitness computation, effect of aging [29, 49, 71, 73] may be investigated.

A modification of the fitness function by incorporating the information on relative

position of the boundary from the training data may constitute another part of further investigation. The classification methodology presented in this thesis, considers only the total number of misclassified points as the optimizing criterion. It does not take the classwise recognition scores into account. This may sometimes lead to an undesirable situation where the overall recognition score is high, but some classes are totally ignored. In order to tackle this, an investigation may be undertaken where the classwise weighted scores constitute the fitness criterion of the chromosomes. The weighting factor may take some *a priori* class information, like the class probabilities, into account.

The present system of fitness computation in *VGA-classifier* first of all attempts to reduce the number of misclassified points and then the number of hyperplanes. This sometimes results in slow removal of redundant hyperplanes. As a part of future work, different weighting coefficients may be used with the number of misclassified points and the number of hyperplanes; thus enabling one to adjust the relative importance of these two factors depending on the state of the process.

A sensitivity analysis of the methodologies for different discretization parameters, like angle and perpendicular distance values, and their relationships to other genetic parameters can form a part of further research work. As mentioned in Chapter 2, due to practical limitations, we have assumed a relatively small population size. More investigation is needed using larger population sizes and allowing more number of iterations before the algorithms are terminated.

In GACD, the chromosomes have been differentiated into two classes and a form of restricted mating has been applied. As a part of future work, the effect of differentiating the chromosomes into more than two categories may be investigated. Incorporation of the concept of variable string lengths in GACD may also be done in the future, thereby yielding a methodology and classifier that may be called VGACD and *VGACD-classifier* respectively. Note that the theoretical results of the classifiers based on both VGA and GACD would hold for the *VGACD-classifier* as well.

An investigation needs to be performed for the selection of appropriate kinds of surfaces (i.e., linear, higher order) for modeling the class boundaries. Although application of hyperspherical segments provided better results for certain problems

in Chapter 2, a proper guideline needs to be developed in this regard.

The classification scheme developed in this thesis is a supervised one, necessitating the presence of training data. An unsupervised scheme, based on the similar principle of placement of hyperplanes, may be developed for partitioning of unlabeled data sets.

Theorems 3.1 (Chapter 3) and 4.3 (Chapter 4) were proved under limiting conditions i.e., when the size (n) of the training data set and the number of iterations go to infinity. As mentioned in Section 4.6, one may use the concept of ϵ-optimal stopping time [135] regarding the termination of GAs in practice. A proper choice of n, which would assure that the performance of the developed classifier is within a factor of the optimum performance, is an open issue and requires further investigation.

Parallel GAs [114, 124, 131, 132], which are different from the conventional GAs, have been drawing the attention of the researchers recently. Classification methodology developed in the thesis may be formulated in the framework of parallel GAs for investigating its merits. Similarly, the complexity analyses of the methodologies, in terms of space and time requirements, for both the serial and parallel implementations, may be performed.

Finally the question that may come to mind is why not use simulated annealing, another well known search and optimization strategy, instead of genetic algorithms for designing the classifier. Simulated annealing also deals with an encoding of the parameters of the search space in strings, each representing a potential candidate solution. It attempts to adaptively evolve the strings, such that the energy value is minimized. This is analogous to the case of fitness function in genetic algorithms. As mentioned in Chapter 2, such a preliminary comparative study between simulated annealing and genetic algorithms has been made in [20], where it is found that the two perform comparably when the criterion to be minimized is the number of misclassified training data points. However, if the error rate (obtained by dividing the number of misclassified points by the size of the training data) is minimized, then the performance of the simulated annealing based classifier degrades considerably, while that of *GA-classifier* remains unchanged.

Appendix A

An Issue Related to Classification by Bayes Rule

Here we explain how more than one decision boundary can provide the same Bayes error probability. Let us consider a three class problem. Let the *a priori* probabilities for the three classes be P_1, P_2 and P_3 and the class conditional densities be $p_1(\mathbf{x}), p_2(\mathbf{x})$ and $p_3(\mathbf{x})$. Let the regions associated with the three classes be Ω_1, Ω_2 and Ω_3 such that $\Omega_i \bigcap \Omega_j = \emptyset$, $\forall i \neq j$ and $\Omega_1 \bigcup \Omega_2 \bigcup \Omega_3 = \Omega$. Then the error probability e is given by

$$
\begin{aligned}
e &= \sum_{i=1}^{3} \int_{\Omega_i^c} P_i p_i(\mathbf{x}) d\mathbf{x} \\
&= \int_{\Omega_2 \bigcup \Omega_3} P_1 p_1(\mathbf{x}) d\mathbf{x} + \int_{\Omega_1 \bigcup \Omega_3} P_2 p_2(\mathbf{x}) d\mathbf{x} + \int_{\Omega_1 \bigcup \Omega_2} P_3 p_3(\mathbf{x}) d\mathbf{x} \\
&= \int_{\Omega_1} P_1 p_1(\mathbf{x}) d\mathbf{x} - \int_{\Omega_1} P_1 p_1(\mathbf{x}) d\mathbf{x} + \int_{\Omega_2 \bigcup \Omega_3} P_1 p_1(\mathbf{x}) d\mathbf{x} + \int_{\Omega_1 \bigcup \Omega_3} P_2 p_2(\mathbf{x}) d\mathbf{x} + \\
&\quad \int_{\Omega_1 \bigcup \Omega_2} P_3 p_3(\mathbf{x}) d\mathbf{x} \\
&= P_1 - \int_{\Omega_1} P_1 p_1(\mathbf{x}) d\mathbf{x} + \int_{\Omega_1 \bigcup \Omega_3} P_2 p_2(\mathbf{x}) d\mathbf{x} + \int_{\Omega_1 \bigcup \Omega_2} P_3 p_3(\mathbf{x}) d\mathbf{x} \\
&= P_1 + P_2 - \int_{\Omega_1} P_1 p_1(\mathbf{x}) d\mathbf{x} - \int_{\Omega_2} P_2 p_2(\mathbf{x}) d\mathbf{x} + \int_{\Omega_1 \bigcup \Omega_2} P_3 p_3(\mathbf{x}) d\mathbf{x} \\
&= P_1 + P_2 + \textstyle\int_{\Omega_1} (P_3 p_3(\mathbf{x}) - P_1 p_1(\mathbf{x})) d\mathbf{x} + \int_{\Omega_2} (P_3 p_3(\mathbf{x}) - P_2 p_2(\mathbf{x})) d\mathbf{x}.
\end{aligned}
$$

Similarly

$$e = P_2 + P_3 + \int_{\Omega_2} (P_1 p_1(\mathbf{x}) - P_2 p_2(\mathbf{x})) d\mathbf{x} + \int_{\Omega_3} (P_1 p_1(\mathbf{x}) - P_3 p_3(\mathbf{x})) d\mathbf{x},$$

and

$$e = P_3 + P_1 + \int_{\Omega_3} (P_2 p_2(\mathbf{x}) - P_3 p_3(\mathbf{x})) d\mathbf{x} + \int_{\Omega_1} (P_2 p_2(\mathbf{x}) - P_1 p_1(\mathbf{x})) d\mathbf{x}.$$

Summing up, we get

$$
\begin{aligned}
3e \;\; = 2 + \; & \Big[\int_{\Omega_1} ((P_3 p_3(\mathbf{x}) - P_1 p_1(\mathbf{x})) d\mathbf{x} + (P_2 p_2(\mathbf{x}) - P_1 p_1(\mathbf{x}))) + \\
& \int_{\Omega_2} ((P_3 p_3(\mathbf{x}) - P_2 p_2(\mathbf{x})) + (P_1 p_1(\mathbf{x}) - P_2 p_2(\mathbf{x}))) d\mathbf{x} + \\
& \int_{\Omega_3} ((P_1 p_1(\mathbf{x}) - P_3 p_3(\mathbf{x})) + (P_2 p_2(\mathbf{x}) - P_3 p_3(\mathbf{x}))) d\mathbf{x} \Big].
\end{aligned}
$$

Let γ be used to represent the term in square brackets. Therefore $3e = 2 + \gamma$.

Bayes classifier classifies a point $\mathbf{x}$ to the class which minimizes e i.e., which in effect

minimizes γ. Accordingly in order to classify a point $\mathbf{x}$ to one of the three classes the following cases may arise :

1. $\{\mathbf{x} : P_1 p_1(\mathbf{x}) > P_2 p_2(\mathbf{x}) > P_3 p_3(\mathbf{x})\}$: classify to class 1.
2. $\{\mathbf{x} : P_1 p_1(\mathbf{x}) > P_2 p_2(\mathbf{x}) = P_3 p_3(\mathbf{x})\}$: classify to class 1.
3. $\{\mathbf{x} : P_1 p_1(\mathbf{x}) > P_3 p_3(\mathbf{x}) > P_2 p_2(\mathbf{x})\}$: classify to class 1.
4. $\{\mathbf{x} : P_1 p_1(\mathbf{x}) = P_3 p_3(\mathbf{x}) > P_2 p_2(\mathbf{x})\}$: classify to class 1 or 3.
5. $\{\mathbf{x} : P_3 p_3(\mathbf{x}) > P_1 p_1(\mathbf{x}) > P_2 p_2(\mathbf{x})\}$: classify to class 3.
6. $\{\mathbf{x} : P_3 p_3(\mathbf{x}) > P_1 p_1(\mathbf{x}) = P_2 p_2(\mathbf{x})\}$: classify to class 3.
7. $\{\mathbf{x} : P_3 p_3(\mathbf{x}) > P_2 p_2(\mathbf{x}) > P_1 p_1(\mathbf{x})\}$: classify to class 3.
8. $\{\mathbf{x} : P_3 p_3(\mathbf{x}) = P_2 p_2(\mathbf{x}) > P_1 p_1(\mathbf{x})\}$: classify to class 3 or 2.
9. $\{\mathbf{x} : P_2 p_2(\mathbf{x}) > P_3 p_3(\mathbf{x}) > P_1 p_1(\mathbf{x})\}$: classify to class 2.
10. $\{\mathbf{x} : P_2 p_2(\mathbf{x}) > P_3 p_3(\mathbf{x}) = P_1 p_1(\mathbf{x})\}$: classify to class 2.
11. $\{\mathbf{x} : P_2 p_2(\mathbf{x}) > P_1 p_1(\mathbf{x}) > P_3 p_3(\mathbf{x})\}$: classify to class 2.
12. $\{\mathbf{x} : P_2 p_2(\mathbf{x}) = P_1 p_1(\mathbf{x}) > P_3 p_3(\mathbf{x})\}$: classify to class 2 or 1.
13. $\{\mathbf{x} : P_1 p_1(\mathbf{x}) = P_2 p_2(\mathbf{x}) = P_3 p_3(\mathbf{x})\}$: classify to any of the three classes.

As is obvious from the previous discussion, regions represented by cases 4, 8, 12 and 13 do not have unique classification associated with them. These regions may be included in more than one class provided they are non-empty, while still providing the least error probability.

Appendix B

Variation of Error Probability with P_1

In Chapter 3, we mentioned (Eq. (3.6)) that the Bayes error probability a is given by

$$a = \sum_{i=1}^{k} P_i \int_{S_{0i}^c} p_i(\mathbf{x}) d\mathbf{x},$$

where S_{0i} is the region for class i. For a two class problem a may be written as

$$a = P_1 \int_{S_{01}^c} p_1(\mathbf{x}) d\mathbf{x} + P_2 \int_{S_{02}^c} p_2(\mathbf{x}) d\mathbf{x}.$$

Since $P_2 = 1 - P_1$, we get

$$a = P_1 \int_{S_{01}^c} p_1(\mathbf{x}) d\mathbf{x} + (1 - P_1) \int_{S_{02}^c} p_2(\mathbf{x}) d\mathbf{x}. \tag{B.1}$$

For the triangular distribution mentioned in Section 3.4.1 (for generating *Data Set 1*), using Eq. (3.13) we may write

$$a = P_1 \int_{1+P_1}^{2} (2 - \mathbf{x}) d\mathbf{x} + (1 - P_1) \int_{1}^{1+P_1} (\mathbf{x} - 1) d\mathbf{x}.$$

Solving for a we get

$$a = P_1 \frac{(1 - P_1)}{2}.$$

Obviously this is a symmetric function with minimum values at $P_1 = 0$ or 1, and maximum value at $P_1 = 0.5$. Thus the recognition score of the Bayes classifier should be minimum for $P_1 = 0.5$, increasing symmetrically on both sides.

For normal distribution, it is very difficult to obtain a closed form expression for a in terms of P_1. An analysis presented in [67] indicates that the risk r associated with a particular decision is maximum for some value of $P_1 = P_1^*$, decreasing on both sides of this value when the regions associated with each class change with the class *a priori* probabilities.

Alternatively, one can also derive bounds on the error probabilities. One such bound for normal distribution is given by [67]

$$a \leq \sqrt{P_1 P_2}\ \exp^{-\mu(1/2)}, \tag{B.2}$$

where $\mu(1/2)$ is called the *Bhattacharyya distance*. Let us define the upper bound of a by a', i.e., $a' = \sqrt{P_1 P_2}\ \exp^{-\mu(1/2)}$. Or,

$$\begin{aligned} a' &= \sqrt{P_1(1 - P_1)}\ \exp^{-\mu(1/2)} \\ \frac{da'}{dP_1} &= \frac{1}{2} \frac{1 - 2P_1}{\sqrt{P_1(1 - P_1)}} \exp^{-\mu(1/2)}. \end{aligned}$$

This shows that the error bound is maximum when $P_1 = P_2 = 0.5$.

Appendix C

Merits of Cooperation and Specialization [73]

Here we discuss intuitively how the survival probability of an individual increases with increase in cooperation. Let an individual spend the available time in two different activities, say nurturing and hunting. If the proportion of the time spent on nurturing and hunting are n and h respectively, then the survival probability of the offspring, $s(n,h)$, is postulated to be equal to $n * h$. If the loss of time available for either activity is proportional to the product of the activity proportions, which is termed jack-of-all-trade loss, then the constraint equation obtained is

$$n + h + anh = 1. \tag{C.1}$$

where a is the loss coefficient. Elementary analysis shows that $s(n,h)$ attains the maximum value of 0.25 when $n = h = 0.5$ and a=0.

On the contrary, if two individuals cooperate to act as one unit, then the survival probability (given by $s = \frac{1}{2}(n_1 + n_2)(h_1 + h_2)$) immediately increases to 0.5 for $a = 0$. In this case $n_1 + n_2 = 1$ and $h_1 + h_2 = 1$, where n_1, n_2, h_1 and h_2 are defined analogously for the two individuals. The constraint Eq. (C.1) holds as follows :

$$n_i + h_i + an_ih_i = 1 \qquad i = 1, 2.$$

Hence the individuals must cooperate but need not specialize. For the case when $a \neq 0$, the maximum survival probability is obtained when either (n_1, n_2) =(1,0) or (0,1) while (h_1, h_2) = (0,1) or (1,0). This indicates full specialization and cooperation within the unit. In either case, the survival probability is larger compared to the uncooperative individual [73].

Appendix D

Application of GACD for function optimization

Here we present some results on the comparison of the performance of GACD with that of CGA for a variety of function optimization problems. Functions of 1, 2 and 3 variables with varying degrees of complexity have been chosen.

The experimental parameters chosen for the function optimization problems are as follows :

Population size	=	40 (CGA)
Initial male and female population sizes	=	20 (GACD)
μ_c	=	0.8
μ_m	=	0.01
Max. no. of generations	=	100
No. of simulations	=	50

In order to bring CGA and GACD logically closer for more effective comparison, a modified version of the CGA called CGAP (CGA with a different mode of population initialization) is also developed. CGAP differs from CGA only in the construction of the initial population. In CGAP, half of the initial population is generated randomly while the other half is generated in such a way that its hamming distance from the first half is maximized. The algorithm similar to the one described in Fig. 5.2 is used for generating the second half of the population in CGAP. The function descriptions, results and associated discussions are now presented in details.

Function 1 : Sparse One Max

This function is similar to the One Max function (which computes the number of 1s in a bit string, the aim being to maximize this value) except that some fake bits are included in the string which do not contribute anything towards the objective function. Strings of length 60 are chosen, where the initial and the final 10 bits are fake. The objective function to be maximized is the number of 1's in bits 11 through 50. Maximum value of the objective function is therefore 40.

Function 2 : Two Max

This function has one local and one global maxima [1]. The function is of the form

$$f(x) = |18\ n - 8\ l|,$$

where n is the number of 1's in the l bit string representing x. There is one global maxima with value 10 l (when x is composed of all 1s i.e., $n = l$), and one local maxima with value 8 l (when x is composed of all 0s i.e., $n = 0$). The boundary

between the two peaks occurs at $\frac{4}{9}\ l$. Any $n > \frac{4}{9}\ l$ leads to the global maxima while $n < \frac{4}{9}\ l$ leads to the local maxima. For $l = 30$ the global maxima has value 300 while the local maxima has value 240.

Function 3 : Trap

This function with one global and one local maxima deals with a situation where the collecting area of the local maxima is much larger than the collecting area of the global maxima. The function [1] is defined as follows :

Let $z = \left\lfloor \frac{3}{4}\ l \right\rfloor$.

Then

$$\begin{aligned} f(x) &= \tfrac{8l}{z}(z-n) && \text{for } n \leq z \\ &= \tfrac{10\ l}{(l-z)}(n-z) && \text{for } n > z. \end{aligned}$$

This function has a global maxima with value 10 l when x is composed of all 1s and a local maxima with value 8 l when x is composed of all 0s. For $l = 30$, the global and local maximas have values 300 and 240 respectively.

Function 4 : Plateau

This function contains large plateau regions which are areas in the solution space with same objective function value providing no uphill direction [1]. The function is described as follows : The l bits are divided into four equal sized groups. Each group provides a score of 2.5 l if it contains all 1s or 0.5 l if it contains atleast one 0. The objective function for a chromosome is the sum of the scores of the four groups. Note that the only possible values of the objective function are $2l$, $4l$, $6l$, $8l$, $10l$. For this function l is 20.

Function 5 : Exp function

The function is of the form

$$\begin{aligned} f(x) &= 2 + \exp^{(x-10)} \cos(10-x) && x \leq 10.0 \\ &= 2 + \exp^{(10-x)} \cos(x-10) && x > 10.0. \end{aligned}$$

$l = 22$ and x is allowed to vary in the range [0,20]. Global maxima exists at $x = 10.0$ where $f(x) = 3.0$.

Function 6 : Sine square function

This is a function of two variables, (x_1, x_2), [190] of the following form

$$f(x_1, x_2) = 0.5 - \frac{(\sin\sqrt{{x_1}^2 + {x_2}^2})^2 - 0.5}{[1 + 0.001({x_1}^2 + {x_2}^2)]^2}.$$

Each variable is encoded using 22 bits ($l = 44$) and is allowed to vary in the range [-100,100]. Global maxima with value 1 occurs when $x_1 = x_2 = 0.0$.

Function 7 : De Jong 1 function

This is a minimization problem of 3 variables [73], where the function to be minimized is

$$f(x_1, x_2, x_3) = \sum_{i=1}^{3} x_i^2.$$

The range for each variable is [-5.12,5.12] and 22 bits are used to encode each variable. Hence $l = 66$. The minimum value is 0.0 when $x_1 = x_2 = x_3 = 0.0$.

Table D.1 presents the comparative results of the best values obtained after 100 iterations for GACD, CGA and CGAP. The optimal values for the corresponding functions are also included in the table. It is seen from the Table that GACD

Table D.1: Comparative results for the best value obtained after 100 iterations.

function	optimal value	Best value obtained		
		GACD	CGA	CGAP
1	40	37.89	37.42	37.60
2	300	298.80	274.08	274.08
3	300	240.00	220.15	222.11
4	200	178.00	182.99	175.00
5	3.0	2.999997	2.987733	2.987977
6	1.0	0.970266	0.911959	0.914001
7	0.0	0.000118	0.208958	0.224651

outperforms both CGA and CGAP for almost all the functions. Only for function 4, the result for GACD is inferior to that of CGA. For function 3, it is found that none of the algorithms can attain values near the global optima. GACD attains the local maxima in all the 50 simulations. CGA and CGAP fail to attain even this value consistently. In fact, results presented later show that all the three algorithms get stuck at the local maxima for this function. Graphical demonstration of the variation of the average and best objective values are shown in Figs. D.1 and D.2 for function 3 respectively, Figs. D.3 and D.4 for function 4 respectively, and Figs. D.5 and

D.6 for function 6 respectively. For function 4, although GACD attains an objective function value that is lower than that of CGA (Fig. D.4), the variation of average value is superior (Fig. D.3). Functions 3 and 6 show a marked superior performance of GACD (Figs. D.1, D.2 and D.5, D.6 respectively). Results for the remaining six functions are similar to that of function 6 and are omitted.

The ability of the algorithms in attaining a user specified objective function value *thresh* is shown in Table D.2. It shows the average number of generations required by GACD, CGA and CGAP to attain *thresh* as well as the number of times (of a total of 50) in which this is possible. A maximum of 2000 generations are executed.

Table D.2: Comparative results for the average number of generations required to attain an objective function value *thresh*.

function	*thresh*	No. of times *thresh* attained			Avg. no. of generations		
		GACD	CGA	CGAP	GACD	CGA	CGAP
1	40	50	0	0	81.66	-	-
2	300	49	49	47	39.16	610.04	591.72
3	300	0	0	0	-	-	-
4	200	50	50	50	81.5	108.42	97.96
5	2.9999	50	45	45	31.84	665.99	755.31
6	0.999	18	7	12	218.78	898.29	986.67
7	0.003	50	16	11	42.22	1108.42	811.00

In most of the cases, it is found that GACD far outperforms both CGA and CGAP in terms of the average number of generations required to attain *thresh* and the number of times that *thresh* is attained. The value of *thresh* is chosen to be sufficiently close to the global optimum value (since the coding of the parameters itself, i.e., the number of bits used to code the variables, may eliminate the attainment of the exact global optimum value). Note that the average number of generations is computed for only those simulations in which *thresh* is attained.

Interestingly, maximizing function 1 appears to be difficult to solve for both CGA and CGAP. However, GACD provides the optimal result in all the 50 simulations in a reasonably small number of generations. Function 3 presents an interesting

finding. None of the three algorithms could attain the global maxima in even one of the 50 simulations, thereby indicating that like CGA, GACD and CGAP get stuck at a local optima when the region from which the global optima can be reached is comparatively quite small. The results for the remaining functions show a markedly superior performance of GACD. Note that the comparative results for CGA and CGAP are not very conclusive since neither one consistently outperforms the other for the cases considered.

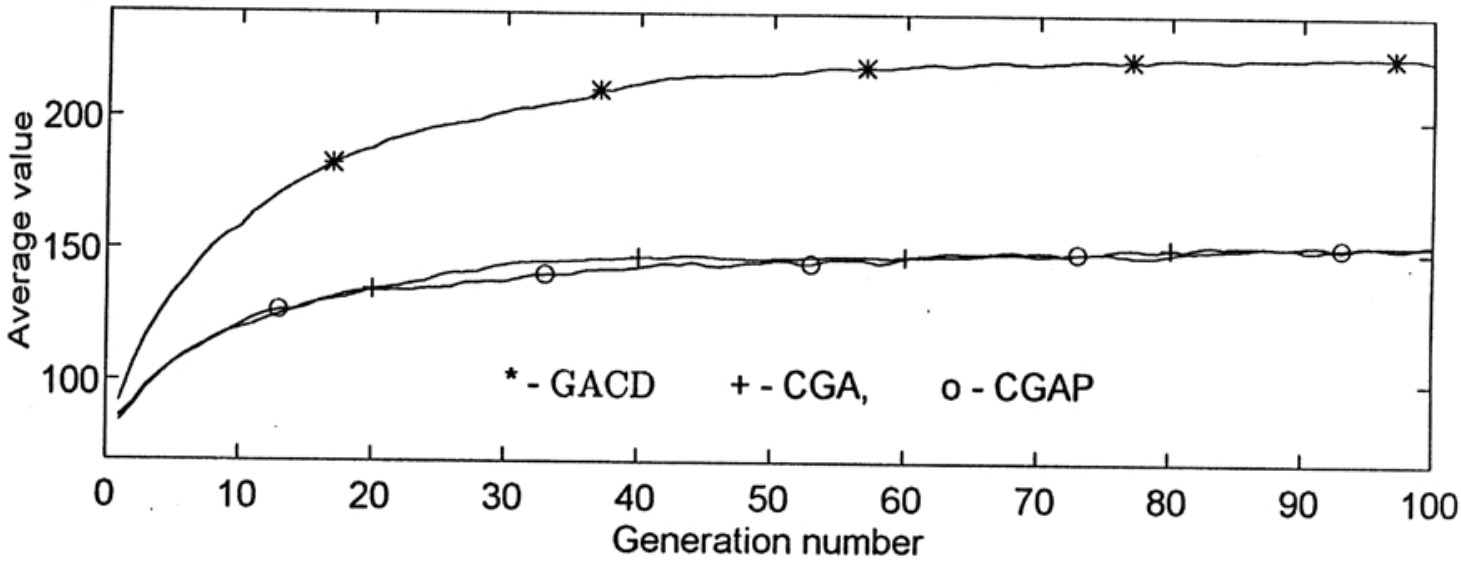


Figure D.1: Variation of average value of objective function with generations for function 3 (Trap function).

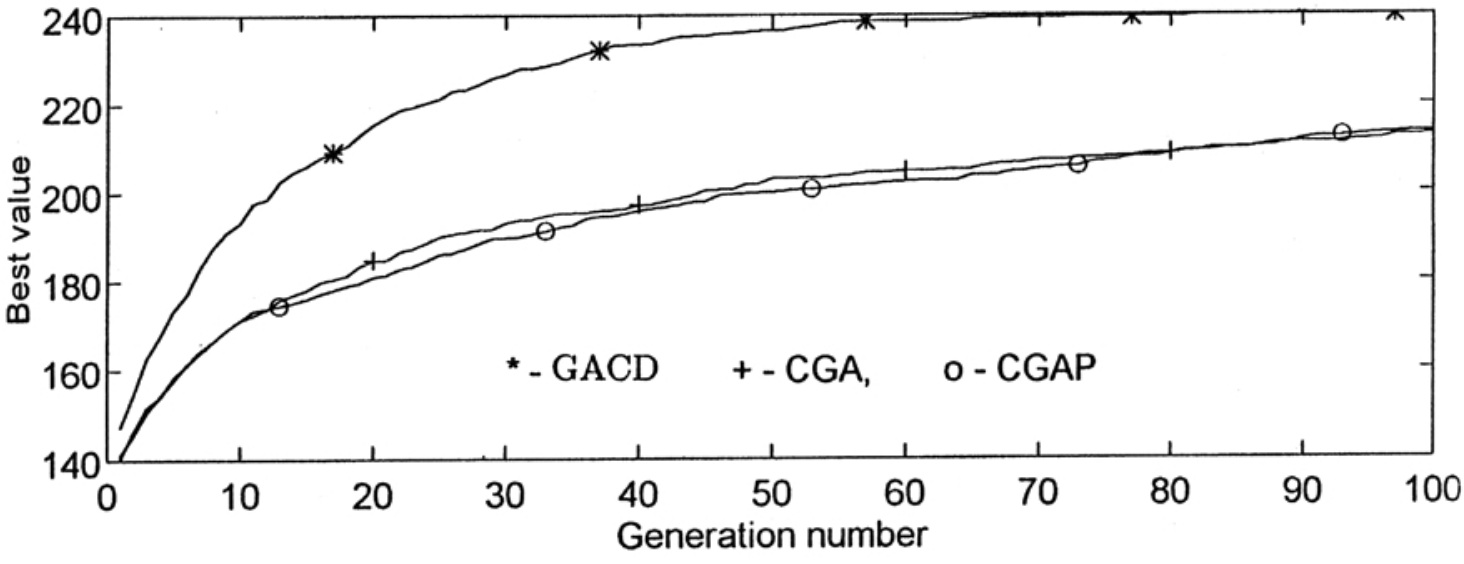


Figure D.2: Variation of best value of objective function with generations for function 3 (Trap function).

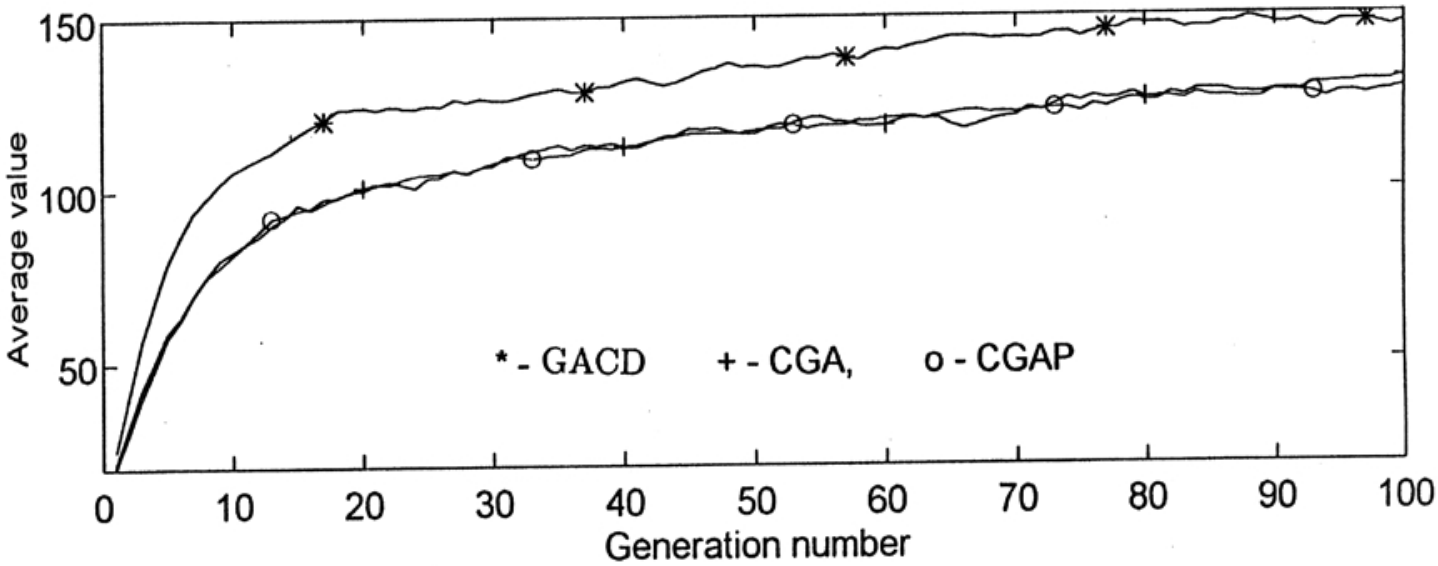


Figure D.3: Variation of average value of objective function with generations for function 4 (Plateau function).

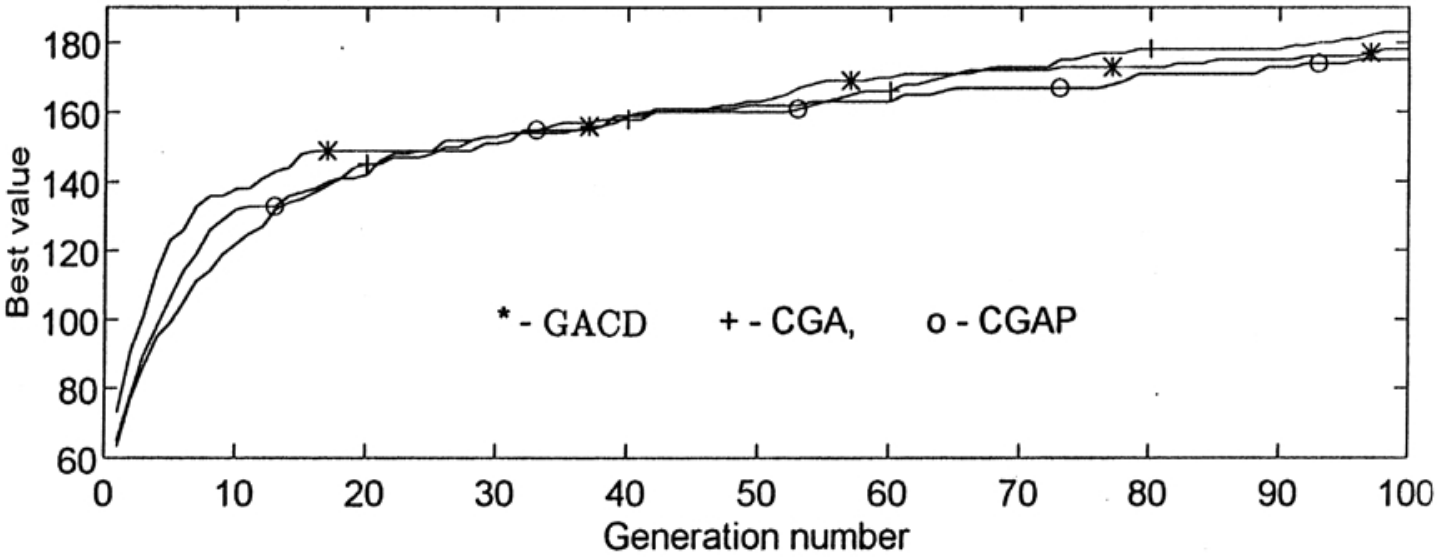


Figure D.4: Variation of best value of objective function with generations for function 4 (Plateau function).

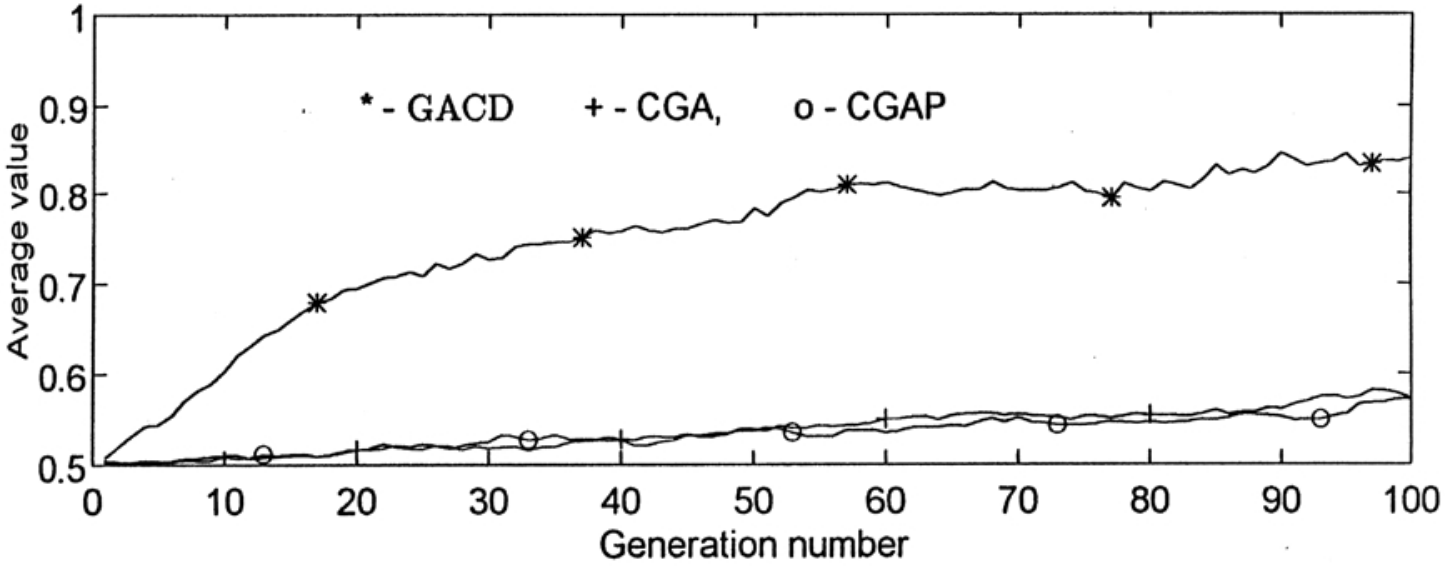


Figure D.5: Variation of average value of objective function with generations for function 6 (Sine square function).

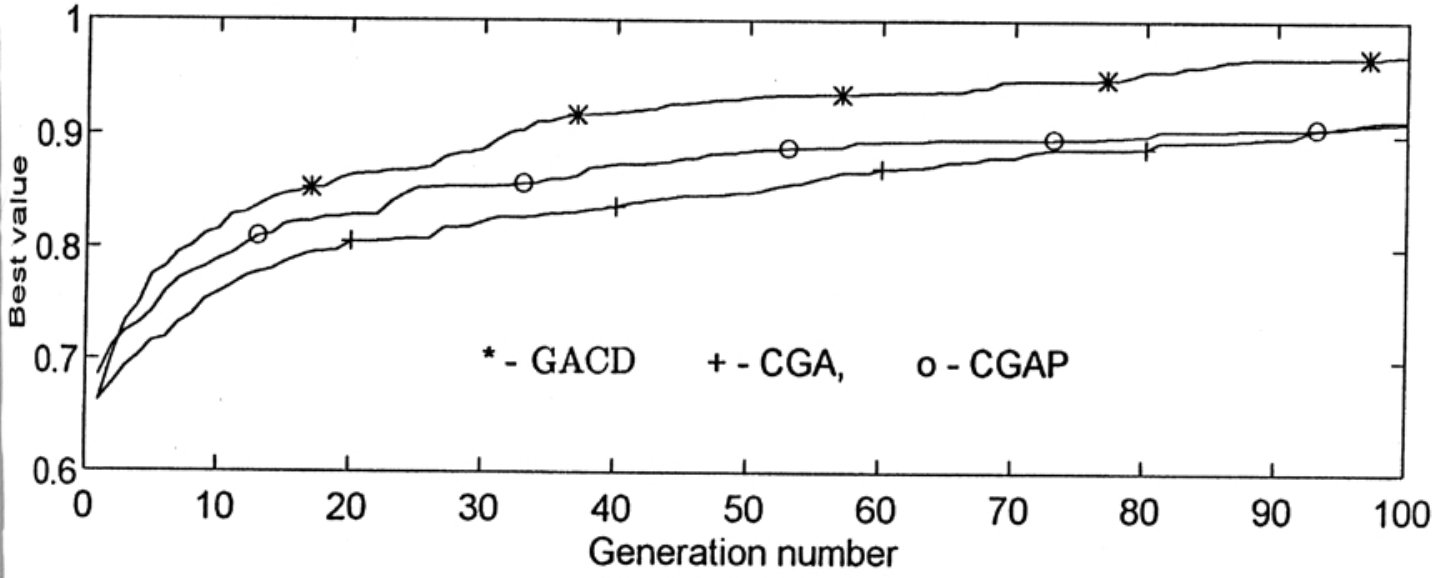


Figure D.6: Variation of best value of objective function with generations for function 6 (Sine square function).

Bibliography

[1] D. H. Ackley, "An empirical study of bit vector function optimization," in *Genetic Algorithms and Simulated Annealing* (L. Davis, ed.), pp. 170–204, London: Pitman, 1987.

[2] T. W. Anderson, *An Introduction to Multivariate Statistical Analysis.* New York: Wiley, 1958.

[3] H. C. Andrews, *Mathematical Techniques in Pattern Recognition.* New York: Wiley Interscience, 1972.

[4] C. A. Ankenbrandt, B. P. Buckles, and F. E. Petry, "Scene recognition using genetic algorithms with semantic nets," *Pattern Recog. Lett.*, vol. 11, pp. 285–293, 1990.

[5] T. M. Apostol, *Mathematical Analysis.* New Delhi: Narosa Publishing House, 1985.

[6] T. Back, "The interaction of mutation rate, selection and self adaptation within a genetic algorithm," in *Proc. Parallel Problem Solving from Nature* (R. Manner and B. Manderick, eds.), pp. 85–94, Amsterdam: North Holland, 1992.

[7] T. Back, "Optimal mutation rates in genetic algorithms," in *Proc. 5th Int. Conf. Genetic Algorithms* (S. Forrest, ed.), pp. 2–8, San Mateo: Morgan Kaufmann, 1993.

[8] J. D. Bagley, *The Behaviour of Adaptive Systems which Employ Genetic and Correlation Algorithms.* PhD thesis, University of Michigan, Ann Arbor, 1967.

[9] J. E. Baker, "Adaptive selection methods for genetic algorithms," in *Proc. 1st Int. Conf. Genetic Algorithms* (J. J. Grefenstette, ed.), pp. 101–111, Hillsdale: Lawrence Erlbaum Associates, 1985.

[10] S. Bandyopadhyay, H. Kargupta, and G. Wang, "Similarity aided linkage learning in GEMGA," *Proc. IEEE Int. Conf. Neural Networks*, Alaska, 1998 (accepted).

[11] S. Bandyopadhyay, C. A. Murthy, and S. K. Pal, "Pattern classification using genetic algorithms," *Pattern Recog. Lett.*, vol. 16, pp. 801–808, 1995.

[12] S. Bandyopadhyay, C. A. Murthy, and S. K. Pal, "GA based pattern classification : Theoretical and experimental studies," in *Proc. 13th Int. Conf. Pattern Recog.*, (Vienna), pp. 758–762, 1996.

[13] S. Bandyopadhyay, C. A. Murthy, and S. K. Pal, "Genetically searched class boundaries with removal of redundancy," in *Proc. 6th Nat. Sem. Theo. Comp. Sci.*, (Rajasthan, India), pp. 345–356, 1996.

[14] S. Bandyopadhyay, C. A. Murthy, and S. K. Pal, "Theoretical performance of genetic pattern classifier," *J. Franklin Institute* (accepted).

[15] S. Bandyopadhyay, C. A. Murthy, and S. K. Pal, "Pattern classification using genetic algorithms : Determination of H," *Pattern Recog. Lett.* (communicated).

[16] S. Bandyopadhyay and S. K. Pal, "Pattern classification with genetic algorithms : Incorporation of chromosome differentiation," *Pattern Recog. Lett.*, vol. 18, pp. 119–131, 1997.

[17] S. Bandyopadhyay and S. K. Pal, "Relation between *VGA-classifier* and MLP : Determination of network architecture," *Fundamenta Informaticae* (communicated).

[18] S. Bandyopadhyay, S. K. Pal, and U. Maulik, "Incorporating chromosome differentiation in genetic algorithms," *Inform. Sci.* (accepted).

[19] S. Bandyopadhyay, S. K. Pal, and C. A. Murthy, "Variable length chromosomes in genetic algorithms for modeling the class boundaries," *3rd Asian Fuzzy Sets Symp.*, Korea, 1998, (accepted).

[20] S. Bandyopadhyay, S. K. Pal, and C. A. Murthy, "Simulated annealing based pattern classification," *Inform. Sci.* (accepted).

[21] R. Battiti, "Using mutual information for selecting features in supervised neural net learning," *IEEE Trans. Neural Networks*, vol. 5, pp. 537–550, 1994.

[22] R. K. Belew and J. B. Booker, eds., *Proc. 4th Int. Conf. Genetic Algorithms.* San Mateo: Morgan Kaufmann, 1991.

[23] M. Ben-Bassat, "Use of distance measures, information measures and error bounds in feature evaluation," in *Handbook of Statistics 2. Classification, Pattern Recognition and Reduction of Dimensionality* (P. R. Krishnaiah and L. Kanal, eds.), pp. 773–792, Amsterdam: North-Holland, 1982.

[24] A. D. Bethke, *Genetic Algorithms as Function Optimizers.* PhD thesis, University of Michigan, Ann Arbor, 1981.

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

[26] J. C. Bezdek, "On the relationship between neural networks, pattern recognition, and intelligence," *Int. J. Approx. Reason.*, vol. 6, pp. 85–107, 1992.

[27] J. C. Bezdek and S. K. Pal, eds., *Fuzzy Models for Pattern Recognition : Methods that Search for Structures in Data.* New York: IEEE Press, 1992.

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

[29] D. Bhandari, N. R. Pal, and S. K. Pal, "Directed mutation in genetic algorithms," *Inform. Sci.*, vol. 79, pp. 251–270, 1994.

[30] L. B. Booker, D. E. Goldberg, and J. H. Holland, "Classifier systems and genetic algorithms," *Art. Intell.*, vol. 40, pp. 235–282, 1989.

[31] S. Bornholdt and D. Graudenz, "General asymmetric neural networks and structure design by genetic algorithms," *Neural Networks*, vol. 5, pp. 327–334, 1992.

[32] N. K. Bose and A. K. Garga, "Neural network design using voronoi diagrams," *IEEE Trans. Neural Networks*, vol. 4, pp. 778–787, 1993.

[33] M. F. Bramlette, "Initialization, mutation and selection methods in genetic algorithms for function optimization," in *Proc. 4th Int. Conf. Genetic Algorithms* (R. K. Belew and L. B. Booker, eds.), pp. 100–107, San Mateo: Morgan Kaufmann, 1991.

[34] B. G. Buchanan, "Can machine learning offer anything to expert systems ?," *Machine Learning*, vol. 4, pp. 251–254, 1989.

[35] B. P. Buckles and F. E. Petry, eds., *Genetic Algorithms. Los Alamitos: IEEE* Computer Society Press, 1994.

[36] B. P. Buckles, F. E. Petry, D. Prabhu, and M. Lybanon, "Mesoscale feature labeling from satellite images," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 167–177, Boca Raton: CRC Press, 1996.

[37] D. J. Cavicchio, "Reproductive adaptive plans," in *Proc. of the ACM 1972 Annual Conf.*, pp. 1–11, 1972.

[38] C. H. Chen, "On information and distance measures, error bounds and feature selection," *Inform. Sci.*, vol. 10, pp. 159–173, 1976.

[39] T. Cleghorn, P. Baffes, and L. Wang, "Robot path planning using a genetic algorithm," in *Proc. SOAR*, (Houston), pp. 81–87, 1988.

[40] F. A. Cleveland and S. F. Smith, "Using genetic algorithms to schedule flow shop releases," in *Proc. 3rd Int. Conf. Genetic Algorithms* (J. D. Schaffer, ed.), pp. 160–169, San Mateo: Morgan Kaufmann, 1989.

[41] M. G. Cooper and J. J. Vidal, "Genetic design of fuzzy controllers," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 283–298, Boca Raton: CRC Press, 1996.

[42] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 21–27, 1967.

[43] 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* (R. K. Belew and J. B. Booker, eds.), pp. 166–173, San Mateo: Morgan Kaufmann, 1991.

[44] L. Davis, "Job shop scheduling with genetic algorithms," in *Proc. 1st Int. Conf. Genetic Algorithms* (J. J. Grefenstette, ed.), pp. 136–140, Hillsdale: Lawrence Erlbaum Associates, 1985.

[45] L. Davis, ed., *Genetic Algorithms and Simulated Annealing.* London: Pitman, 1987.

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

[47] T. E. Davis and J. C. Principe, "A simulated annealing-like convergence theory for the simple genetic algorithm," in *Proc. 4th Int. Conf. Genetic Algorithms* (R. K. Belew and J. B. Booker, eds.), pp. 174–181, San Mateo: Morgan Kaufmann, 1991.

[48] J. E. Dayhoff, *Neural Network Architectures An Introduction.* New York: Van Nostrand Reinhold, 1990.

[49] S. De, A. Ghosh, and S. K. Pal, "Fitness evaluation in genetic algorithms with ancestors' influence," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 1–23, Boca Raton: CRC Press, 1996.

[50] K. A. DeJong, *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* PhD thesis, Dept. of Computer and Communication Science, Univ. of Michigan, Ann Arbor, 1975.

[51] K. A. DeJong, "Learning with genetic algorithms : An overview," *Machine Learning*, vol. 3, pp. 121–138, 1988.

[52] K. A. DeJong and W. M. Spears, "An analysis of the interacting roles of population size and crossover in genetic algorithms," in *Proc. Parallel Problem Solving from Nature*, pp. 38–47, Berlin: Springer-Verlag, 1990.

[53] T. Denoeux, "A k-nearest neighbor classification rule based on Dempster-Shafer theory," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, pp. 804–813, 1995.

[54] K. Dev and C. R. Murthy, "A genetic algorithm for the knowledge base partitioning problem," *Pattern Recog. Lett.*, vol. 16, pp. 873–879, 1995.

[55] P. A. Devijver and J. Kittler, *Pattern Recognition : A Statistical Approach.* London: Prentice-Hall, 1982.

[56] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.

[57] L. J. Eshelman, "The CHC adaptive search algorithm : How to have safe search when engaging in nontraditional genetic recombination," in *Foundations of Genetic Algorithms* (G. J. E. Rawlins, ed.), pp. 265–283, San Mateo: Morgan Kaufman, 1991.

[58] L. J. Eshelman, ed., *Proc. 6th Int. Conf. Genetic Algorithms.* San Mateo: Morgan Kaufmann, 1995.

[59] L. J. Eshelman and D. Schaffer, "Preventing premature convergence by preventing incest," in *Proc. 4th Int. Conf. Genetic Algorithms*, pp. 115–122, San Mateo: Morgan Kaufmann, 1991.

[60] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems* (D. S. Touretzky, ed.), pp. 524–532, Los Altos: Morgan-Kaufmann, 1990.

[61] J. L. R. Filho, P. C. Treleaven, and C. Alippi, "Genetic algorithm programming environments," *IEEE Computer*, pp. 28–43, June 1994.

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

[63] D. B. Fogel, *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence.* New York: IEEE Press, 1995.

[64] S. Forrest, ed., *Proc. 5th Int. Conf. Genetic Algorithms.* San Mateo: Morgan Kaufmann, 1993.

[65] K. S. Fu, *Syntactic Pattern Recognition and Applications.* London: Academic Press, 1982.

[66] T. Fukuda, Y. Komata, and T. Arakawa, "Recurrent neural network with self-adaptive GAs for biped locomotion robot," in *Proc. IEEE Int. Conf. Neural Networks*, (Houston), pp. 1710–1715, 1997.

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

[68] E. S. Gelsema, ed., *Special Issue on Genetic Algorithms, Pattern Recognition Letters*, vol. 16, no. 8. Elsevier Sciences, Inc., 1995.

[69] E. S. Gelsema and L. Kanal, eds., *Pattern Recognition in Practice II.* Amsterdam: North Holland, 1986.

[70] F. Ghannadian, C. Alford, and R. Shonkwiler, "Application of random restart to genetic algorithms," *Inform. Sci.*, vol. 95, pp. 81–102, 1996.

[71] A. Ghosh, S. Tsutsui, and H. Tanaka, "Genetic search with aging of individuals," *Int. J. Knowledge-based Intell. Eng. Syst.*, vol. 1, pp. 86–103, 1997.

[72] D. E. Goldberg, "Simple genetic algorithms, and the minimal deceptive problem," in *Genetic Algorithms and Simulated Annealing* (L. Davis, ed.), pp. 74–88, London: Pitman, 1987.

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

[74] D. E. Goldberg, "Sizing populations for serial and parallel genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms* (J. D. Schaffer, ed.), pp. 70–79, San Mateo: Morgan Kaufmann, 1989.

[75] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik, "Rapid, accurate optimization of difficult problems using fast messy genetic algorithms," in *Proc. 5th Int. Conf. Genetic Algorithms* (S. Forrest, ed.), pp. 56–64, San Mateo: Morgan Kaufmann, 1993.

[76] D. E. Goldberg, K. Deb, and B. Korb, "Messy genetic algorithms : Motivation, analysis, and first results," *Complex Systs.*, vol. 3, pp. 493–530, 1989.

[77] 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.), pp. 24–30, San Mateo: Morgan Kaufmann, 1991.

[78] D. E. Goldberg and J. J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proc. 2nd Int. Conf. Genetic Algorithms*, pp. 41–49, Hillsdale: Lawrence Erlbaum Associates, 1987.

[79] D. E. Goldberg and P. Segrest, "Finite markov chain analysis of genetic algorithms," in *Proc. 2nd Int. Conf. Genetic Algorithms*, pp. 1–8, Hillsdale: Lawrence Erlbaum Associates, 1987.

[80] R. C. Gonzalez and M. G. Thomason, *Syntactic Pattern Recognition : An Introduction.* Reading: Addison-Wesley, 1978.

[81] J. J. Grefenstette, ed., *Proc. 1st Int. Conf. Genetic Algorithms.* Hillsdale: Lawrence Erlbaum Associates, 1985.

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

[83] J. J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht, "Genetic algorithms for the traveling salesman problem," in *Proc. 1st Int. Conf. Genetic Algorithms* (J. J. Grefenstette, ed.), pp. 160–168, Hillsdale: Lawrence Erlbaum Associates, 1985.

[84] S. A. Harp and T. Samad, "Genetic synthesis of neural network architecture," in *Handbook of Genetic Algorithms* (L. Davis, ed.), pp. 202 – 221, New York: Van Nostrand Reinhold, 1991.

[85] S. Haykin, *Neural Networks, A Comprehensive Foundation.* New York: McMillan College Publishing Company, 1994.

[86] D. O. Hebb, *The Organization of Behaviour.* New York: Wiley, 1949.

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

[88] A. Hill and C. J. Taylor, "Model-based image interpretation using genetic algorithms," *Image and Vision Comput.*, vol. 10, pp. 295–300, 1992.

[89] J. H. Holland, *Adaptation in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press, 1975.

[90] J. Horn, "Finite markov chain analysis of genetic algorithms with niching," in *Proc. 5th Int. Conf. Genetic Algorithms* (S. Forrest, ed.), pp. 110–117, San Mateo: Morgan Kaufmann, 1993.

[91] H. V. Hove and A. Verschoren, "Genetic algorithms and recognition problems," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 145–166, Boca Raton: CRC Press, 1996.

[92] H. Ishibuchi, T. Murata, and H. Tanaka, "Construction of fuzzy classification systems with linguistic if-then rules using genetic algorithms," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 227–251, Boca Raton: CRC Press, 1996.

[93] H. Ishibuchi, M. Nii, and T. Murata, "Linguistic rule extraction from neural networks and genetic algorithm based rule selection," in *Proc. IEEE Int. Conf. Neural Networks*, (Houston), pp. 2390–2395, 1997.

[94] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, "Acquisition of fuzzy classification knowledge using genetic algorithms," in *Proc. 3rd IEEE Int. Conf. Fuzzy Systs.*, (Orlando), pp. 1963–1968, 1994.

[95] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, "Selecting fuzzy if-then rules for classification problems using genetic algorithms," *IEEE Trans. Fuzzy Systs.*, vol. 3, pp. 260–270, 1995.

[96] H. Ishigami, T. Fukuda, T. Shibata, and F. Arai, "Structure optimization of fuzzy neural network by genetic algorithm," *Fuzzy Sets Systs.*, vol. 71, pp. 257–264, 1995.

[97] C. J. Janikow, "A genetic algorithm method for optimizing the fuzzy component of a fuzzy decision tree," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 253–281, Boca Raton: CRC Press, 1996.

[98] P. Jog, J. Y. Suh, and D. V. Gucht, "The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem," in *Proc. 3rd Int. Conf. Genetic Algorithms* (J. D. Schaffer, ed.), pp. 110–115, San Mateo: Morgan Kaufmann, 1989.

[99] L. Kanal, "Patterns in pattern recognition," *IEEE. Trans. Inform. Theory*, vol. IT-20, pp. 697–722, 1974.

[100] A. Kandel, *Fuzzy Techniques in Pattern Recognition.* New York: Wiley Interscience, 1982.

[101] A. Kandel, *Fuzzy Mathematical Techniques with Applications.* New York: Addison-Wesley, 1986.

[102] H. Kargupta, "The gene expression messy genetic algorithm," in *Proc. IEEE Int. Conf. Evolutionary Computation*, pp. 814–819, New Jersey: IEEE Press, 1996.

[103] H. Kargupta, K. Deb, and D. E. Goldberg, "Ordering genetic algorithms and deception," in *Proc. Parallel Problem Solving from Nature* (R. Manner and B. Manderick, eds.), pp. 47–56, Amsterdam: North-Holland, 1992.

[104] N. Kasabov and M. Watts, "Genetic algorithms for structural optimization, dynamic adaptation and automated design of fuzzy neural networks," in *Proc. IEEE Int. Conf. Neural Networks*, (Houston), pp. 2546–2549, 1997.

[105] J. D. Kelly, Jr. and L. Davis, "A hybrid genetic algorithm for classification," in *Proc. 12th Int. Joint Conf. Art. Intell.*, (Sydney), 1991.

[106] J. D. Kelly, Jr. and L. Davis, "Hybridizing the genetic algorithm and the K nearest neighbors classification algorithm," in *Proc. 4th Int. Conf. Genetic*

Algorithms (R. K. Belew and J. B. Booker, eds.), pp. 377–383, San Mateo: Morgan Kaufmann, 1991.

[107] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vechhi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[108] T. Kohonen, *Self-Organization and Associative Memory.* Berlin: Springer-Verlag, 1989.

[109] J. R. Koza, *Genetic Programming : On the Programming of Computers by Means of Natural Selection.* Cambridge: MIT Press, 1992.

[110] A. Krone and H. Kiendl, "An evolutionary concept for generating relevant fuzzy rules from data," *Int. J. Knowledge-based Intell. Eng. Syst.*, vol. 1, pp. 207–213, 1997.

[111] L. I. Kuncheva, "Editing the k-nearest neighbor rule by a genetic algorithm," *Pattern Recog. Lett.*, vol. 16, pp. 809–814, 1995.

[112] L. I. Kuncheva, "Initializing of an RBF network by a genetic algorithm," *Neurocomputing*, vol. 14, pp. 273–288, 1997.

[113] T. Kuo and S. Hwang, "Using disruptive selection to maintain diversity in genetic algorithms," *App. Intell.*, vol. 7, pp. 257–267, 1997.

[114] G. E. Liepins and S. Baluja, "apGA : An adaptive parallel genetic algorithm," in *Computer Science and Operations Research, New Development in Their Interfaces* (Balci, Sharda, and Zenios, eds.), pp. 399–409, Pergamon Press, 1992.

[115] G. E. Liepins, M. R. Hilliard, M. Palmer, and G. Rangarajan, "Credit assignment and discovery in classifier systems," *Int. J. Intell. Syst.*, vol. 6, pp. 55–69, 1991.

[116] G. E. Liepins and M. D. Vose, "Deceptiveness and genetic algorithm dynamics," in *Foundations of Genetic Algorithms* (G. J. E. Rawlins, ed.), pp. 36–50, San Mateo: Morgan Kaufmann, 1991.

[117] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, vol. 4, no. 2, pp. 4–22, 1987.

[118] H. Maini, K. Mehrotra, C. Mohan, and S. Ranka, "Knowledge-based nonuniform crossover," *Complex Systs.*, vol. 8, pp. 257–293, 1994.

[119] D. P. Mandal, *A Multivalued Approach for Uncertainty Management in Pattern Recognition Problems using Fuzzy Sets.* PhD thesis, Electronics and Communication Sciences Unit, Indian Statistical Institute, Calcutta, India, 1992.

[120] D. P. Mandal, C. A. Murthy, and S. K. Pal, "Formulation of a multivalued recognition system," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, pp. 607–620, 1992.

[121] D. P. Mandal, C. A. Murthy, and S. K. Pal, "Analysis of IRS imagery for detecting man-made objects with a multivalued recognition system" *IEEE Trans. Syst., Man, Cybern., Part A*, vol. 26, pp. 241–247, 1996.

[122] 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.), pp. 22–30, Philadelphia: SIAM Publications, 1990.

[123] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 39–53, 1994.

[124] T. Maruyama, A. Konagaya, and K. Konishi, "An asynchronous fine grained parallel genetic algorithm," in *Proc. Parallel Problem Solving from Nature*, pp. 563–572, 1992.

[125] K. Mathias, D. Whitley, A. Kusuma, and C. Stork, "An empirical evaluation of genetic algorithms on noisy objective functions," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 65–86, Boca Raton: CRC Press, 1996.

[126] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs.* New York: Springer-Verlag, 1992.

[127] Z. Michalewicz and C. Z. Janikow, "Genetic algorithms for numerical optimization," *Statistics and Computing*, vol. 1, pp. 75–91, 1991.

[128] M. Minsky and S. Papert, *Perceptrons : An Introduction to Computational Geometry.* Cambridge: MIT Press, 1969.

[129] S. Mitra and S. K. Pal, "Self-organizing neural network as a fuzzy classifier," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, pp. 385–399, 1994.

[130] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proc. 11th Int. Joint Conf. Art. Intell.* (N. S. Sridharan, ed.), pp. 762–767, San Mateo: Morgan Kaufmann, 1989.

[131] H. Muhlenbein, "Evolution in time and space - the parallel genetic algorithm," in *Foundations of Genetic Algorithms* (G. J. E. Rawlins, ed.), pp. 316–337, San Mateo: Morgan Kaufmann, 1991.

[132] H. Muhlenbein, M. Schomish, and J. Born, "The parallel genetic algorithm as function optimizer," in *Proc. 4th Int. Conf. Genetic Algorithms* (R. K. Belew and J. B. Booker, eds.), pp. 271–278, San Mateo: Morgan Kaufmann, 1991.

[133] H. J. Muller, ed., *Studies in Genetics - selected papers.* Bloomington: Indiana University Press, 1962.

[134] C. A. Murthy, S. Bandyopadhyay, and S. K. Pal, "Genetic algorithm based pattern classification : Relationship with Bayes classifier," in *Genetic Algorithms for Pattern Classification* (S. K. Pal and P. P. Wang, eds.), pp. 127–144, Boca Raton: CRC Press, 1996.

[135] C. A. Murthy, D. Bhandari, and S. K. Pal, "Optimal stopping time for genetic algorithms with elitist model," *Fundamenta Informaticae* (revised).

[136] C. A. Murthy and N. Chowdhury, "In search of optimal clusters using genetic algorithms," *Pattern Recog. Lett.*, vol. 17, pp. 825–832, 1996.

[137] A. Nafarieh and J. Keller, "A fuzzy logic rule-based automatic target recognition," *Int. J. Intell. Syst.*, vol. 6, pp. 295–312, 1991.

[138] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A study of permutation crossover operators on the traveling salesman problem," in *Proc. 2nd Int. Conf. Genetic Algorithms*, pp. 224–230, Hillsdale: Lawrence Erlbaum Associates, 1987.

[139] E. Ozcan and C. K. Mohan, "Partial shape matching using genetic algorithms," *Pattern Recog. Lett.*, vol. 18, pp. 987–992, 1997.

[140] A. Pal, *On a Class of Stochastic Approximation-Type Parameter-Learning Algorithms for Pattern Recognition.* PhD thesis, Electronics and Communication Sciences Unit, Indian Statistical Institute, Calcutta, India, 1990.

[141] A. Pal, "Some applications of GGA for automatic learning of class parameters in the presence of wrong samples," *Inform. Sci.*, vol. 67, pp. 189–208, 1993.

[142] S. K. Pal, "Fuzzy set theoretic measures for automatic feature evaluation - II," *Inform. Sci.*, vol. 64, pp. 165–179, 1992.

[143] S. K. Pal, S. Bandyopadhyay, and C. A. Murthy, "Genetic algorithms for generation of class boundaries," *IEEE Trans. Syst., Man, Cybern.* (accepted).

[144] 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.

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

[146] S. K. Pal, S. De, and A. Ghosh, "Designing Hopfield type networks using genetic algorithms and its comparison with simulated annealing," *Int. J. Pattern Recog. Art. Intell.*, vol. 11, pp. 447–461, 1997.

[147] S. K. Pal and A. Ghosh, "Neuro-fuzzy computing for image processing and pattern recognition," *Int. J. Syst. Sci.*, vol. 27, pp. 1179–1193, 1996.

[148] 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.

[149] S. K. Pal and D. D. Majumder, *Fuzzy Mathematical Approach to Pattern Recognition.* New York: John Wiley, 1986.

[150] S. K. Pal and D. P. Mandal, "Linguistic recognition system based on approximate reasoning," *Inform. Sci.*, vol. 61, pp. 135–161, 1992.

[151] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets and classification," *IEEE Trans. Neural Networks*, vol. 3, pp. 683–697, 1992.

[152] S. K. Pal, C. A. Murthy, and B. Uma Shankar, "Development of software package - handling uncertainties for machine interpretation of ill-defined structures present in gray-level images : Phase II," 1995. A project sponsored by Defence Electronics Application Laboratory, Dehradun, (Contract no. DEAL/66/93-94/DC-01, dated August 20, 1993).

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

[154] Y. H. Pao, *Adaptive Pattern Recognition and Neural Networks.* New York: Addison-Wesley, 1989.

[155] T. Pavlidis, *Structural Pattern Recognition.* New York: Springer-Verlag, 1977.

[156] W. Pedrycz, "A fuzzy cognitive structure for pattern recognition," *Pattern Recog. Lett.*, vol. 9, pp. 305–313, 1989.

[157] W. Pedrycz, "Fuzzy sets in pattern recognition : Methodology and methods," *Pattern Recognition*, vol. 23, pp. 121–146, 1990.

[158] W. Pedrycz, "Fuzzy neural network with reference neurons as pattern classifiers," *IEEE Trans. Neural Networks*, vol. 3, pp. 770–775, 1992.

[159] W. Pedrycz, "Genetic algorithms for learning in fuzzy relational structures," *Fuzzy Sets Systs.*, vol. 69, pp. 37–52, 1995.

[160] W. Pedrycz, ed., *Fuzzy Evolutionary Computation.* Boston: Kluwer Academic Publisher, 1997.

[161] J. Piper, "Genetic algorithm for applying constraints in chromosome classification," *Pattern Recog. Lett.*, vol. 16, pp. 857–864, 1995.

[162] G. Pitney, T. R. Smith, and D. Greenwood, "Genetic design of processing elements for path planning networks," in *Proc. Int. Joint Conf. Neural Networks*, vol. 3, pp. 925–932, Piscataway: IEEE Press, 1990.

[163] J. Potvin, D. Dube, and C. Robillard, "A hybrid approach to vehicle routing using neural networks and genetic algorithms," *App. Intell.*, vol. 6, pp. 241–252, 1996.

[164] M. Prakash and M. N. Murty, "A genetic approach for selection of (near-) optimal subsets of principal components for discrimination," *Pattern Recog. Lett.*, vol. 16, pp. 781–787, 1995.

[165] *Proc. 1st IEEE Int. Conf. Fuzzy Syst.*, (San Diego, USA), March 1992.

[166] *Proc. 2nd Int. Conf. Fuzzy Logic and Neural Networks (IIZUKA94)*, (Iizuka, Japan), July 1992.

[167] *Proc. 2nd IEEE Int. Conf. Fuzzy Syst.*, (California, USA), March 1993.

[168] *Proc. 3rd IEEE Int. Conf. Fuzzy Syst.*, (Florida, USA), June 1994.

[169] *Proc. 3rd Int. Conf. Fuzzy Logic and Neural Networks (IIZUKA94)*, (Iizuka, Japan), July 1994.

[170] *Proc. 1995 IEEE Int. Conf. Evolutionary Computation*, (Perth, Australia), 1995.

[171] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 3, pp. 81–106, 1986.

[172] N. J. Radcliffe, "Genetic set recombination," in *Foundations of Genetic Algorithms 2* (L. D. Whitley, ed.), pp. 203–219, San Mateo: Morgan Kaufmann, 1993.

[173] R. Reed, "Pruning algorithms - a survey," *IEEE Trans. Neural Networks*, vol. 4, pp. 740–747, 1993.

[174] L. A. Rendell, "A doubly layered genetic penetrance learning system," in *Proc. Nat. Conf. Art. Intell.*, (Washington DC), pp. 343–347, 1983.

[175] J. A. Richards, *Remote Sensing Digital Image Analysis : An Introduction.* New York: Springer-Verlag, 1993.

[176] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard, "Some guidelines for genetic algorithms with penalty functions," in *Proc. 3rd Int. Conf. Genetic Algorithms* (J. D. Schaffer, ed.), pp. 191–197, San Mateo: Morgan Kaufmann, 1989.

[177] R. Riolo, "Modeling simple human category learning with a classifier system," in *Proc. 4th Int. Conf. Genetic Algorithms* (R. K. Belew and L. B. Booker, eds.), pp. 324–333, San Mateo: Morgan Kaufmann, 1991.

[178] S. G. Romaniuk, "Learning to learn with evolutionary growth perceptrons," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 179–211, Boca Raton: CRC Press, 1996.

[179] D. W. Ruck, S. K. Rogers, and M. Kabrisky, "Feature selection using a multilayer perceptron," *J. Neural Network Computing*, pp. 40–48, 1990.

[180] D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley, and B. W. Suter, "The multilayer perceptron as an approximation to a Bayes optimal discriminant function," *IEEE Trans. Neural. Networks*, vol. 1, pp. 436–438, 1990.

[181] G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Trans. Neural Networks*, vol. 5, pp. 96–101, 1994.

[182] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representation by error propagation," in *Parallel Distributed Processing : Explorations in the Microstructures of Cognition* (D. E. Rumelhart and J. L. McClelland, eds.), vol. 1, pp. 318–362, Cambridge: MIT Press, 1986.

[183] D. E. Rumelhart, J. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. Cambridge: MIT Press, 1986.

[184] D. E. Rumelhart, J. McClelland, and the PDP Research Group, *Parallel Distributed Processing*, vol. 2. Cambridge: MIT Press, 1986.

[185] S. Saha and J. P. Christensen, "Genetic design of sparse feedforward neural networks," *Inform. Sci.*, vol. 79, pp. 191–200, 1994.

[186] M. Sarkar and B. Yegnanarayana, "An evolutionary programming-based probabilistic neural networks construction technique," in *Proc. IEEE Int. Conf. Neural Networks*, (Houston), pp. 456–461, 1997.

[187] M. Sarkar and B. Yegnanarayana, "Feedforward neural networks configuration using evolutionary programming," in *Proc. IEEE Int. Conf. Neural Networks*, (Houston), pp. 438–443, 1997.

[188] M. Sarkar, B. Yegnanarayana, and D. Khemani, "A clustering algorithm using an evolutionary programming-based approach," *Pattern Recog. Lett.*, vol. 18, pp. 975–986, 1997.

[189] J. D. Schaffer, R. Caruana, L. J. Eshelman, and R. Das, "A study of control parameters affecting online performance of genetic algorithms," in *Genetic Algorithms and Simulated Annealing* (L. Davis, ed.), pp. 89–103, London: Pitman, 1987.

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

[191] J. D. Schaffer, R. A. Caruana, and L. J. Eshelman, "Using genetic search to exploit the emergent behavior of neural networks," *Physica D*, vol. 42, pp. 244–248, 1990.

[192] J. D. Schaffer and L. J. Eshelman, "On crossover as an evolutionarily viable strategy," in *Proc. 4th Int. Conf. Genetic Algorithms* (R. K. Belew and L. B. Booker, eds.), pp. 61–68, San Mateo: Morgan Kaufmann, 1991.

[193] N. N. Schraudolph and R. K. Belew, "Dynamic parameter encoding for genetic algorithms," *Machine Learning*, vol. 9, pp. 9–21, 1992.

[194] G. Seetharaman, A. Narasimahan, and L. Stor, "Image segmentation with genetic algorithms : A formulation and implementation," in *Proc. SPIE Conf. Stochastics and Neural Methods in Signal Processing and Computer Vision*, vol. 1569, (San Diego), 1991.

[195] G. Shafer, *A Mathematical Theory of Evidence.* Princeton: Princeton University Press, 1976.

[196] K. Shimojima, T. Fukuda, and Y. Hasegawa, "Self-tuning fuzzy modeling with adaptive membership function, rules and hierarchical structure based on genetic algorithm," *Fuzzy Sets Systs.*, vol. 71, pp. 295–309, 1995.

[197] W. Siedlecki and J. Sklansky, "A note on genetic algorithms for large-scale feature selection," *Pattern Recog. Lett.*, vol. 10, pp. 335–347, 1989.

[198] R. Sikora and M. Shaw, "A double layered genetic approach to acquiring rules for classification : Integrating genetic algorithms with similarity based learning," *ORSA J. Computing*, vol. 6, pp. 174–187, 1994.

[199] S. F. Smith, *A Learning System Based on Genetic Algorithms.* PhD thesis, University of Pittsburg, PA, 1980.

[200] W. M. Spears, "Crossover or mutation ?," in *Foundations of Genetic Algorithms 2* (L. D. Whitley, ed.), pp. 221–237, San Mateo: Morgan Kaufmann, 1993.

[201] W. M. Spears and K. A. DeJong, "An analysis of multi-point crossover," in *Foundations of Genetic Algorithms* (G. J. E. Rawlins, ed.), pp. 301–315, San Mateo: Morgan Kaufmann, 1991.

[202] W. M. Spears and K. A. DeJong, "On the virtues of parameterized uniform crossover," in *Proc. 4th Int. Conf. Genetic Algorithms* (R. K. Belew and L. B. Booker, eds.), pp. 230–236, San Mateo: Morgan Kaufmann, 1991.

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

[204] R. Srikanth, R. George, N. Warsi, D. Prabhu, F. E. Petry, and B. P. Buckles, "A variable-length genetic algorithm for clustering and classification," *Pattern Recog. Lett.*, vol. 16, pp. 789–800, 1995.

[205] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, pp. 656–667, 1994.

[206] M. Srinivas and L. M. Patnaik, "Genetic algorithms : A survey," *IEEE Computer*, pp. 17–26, June 1994.

[207] J. Suzuki, "A markov chain analysis on simple genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, pp. 655–659, 1995.

[208] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms* (J. D. Schaffer, ed.), pp. 2–9, San Mateo: Morgan Kaufmann, 1989.

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

[210] W. H. Tsai and S. S. Yu, "Attributed string matching with merging for shape recognition," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. 7, pp. 453–462, 1985.

[211] L. Tseng and S. Yang, "Genetic algorithms for clustering, feature selection, and classification," in *Proc. IEEE Int. Conf. Neural Networks*, (Houston), pp. 1612–1616, 1997.

[212] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing : Theory and Applications.* Holland: D. Reidel Publishing Company, 1987.

[213] V. Vemuri and W. Cedeno, "Industrial applications of genetic algorithms," *Int. J. Knowledge-based Intell. Eng. Systs.*, vol. 1, pp. 1–12, 1997.

[214] M. D. Vose and G. E. Liepins, "Punctuated equilibria in genetic search," *Complex Systs.*, vol. 5, pp. 31–44, 1991.

[215] M. D. Vose and A. H. Wright, "The walsh transform and the theory of simple genetic algorithm," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 25–44, Boca Raton: CRC Press, 1996.

[216] B. W. Wah, A. Ieumwananonthachai, and Y. Li, "Generalization of heuristics learned in genetics-based learning," in *Genetic Algorithms for Pattern Recognition* (S. K. Pal and P. P. Wang, eds.), pp. 87–126, Boca Raton: CRC Press, 1996.

[217] P. P. Wang, ed., *Advances in Machine Intelligence and Soft Computing*, vol. 4. 1997.

[218] Y. Wang, K. Fan, and J. Horng, "Genetic-based search for error-correcting graph isomorphism," *IEEE Trans. Syst., Man, Cybern.*, vol. 27, pp. 588–597, 1997.

[219] D. Whitley and T. Starkweather, "GENITOR-II : A distributed genetic algorithm," *J. Experimental Theoretical Art. Intell.*, vol. 2, pp. 189–214, 1990.

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

[221] L. D. Whitley, "Fundamental principles of deception in genetic search," in *Foundations of Genetic Algorithms* (G. J. E. Rawlins, ed.), pp. 221–241, San Mateo: Morgan Kaufmann, 1991.

[222] L. D. Whitley, K. Mathias, and P. Fitzhorn, "Delta coding : An iterative strategy for genetic algorithms," in *Proc. 4th Int. Conf. Genetic Algorithms*, pp. 77–84, San Mateo: Morgan Kaufmann, 1991.

[223] G. Winter, J. Periaux, M. Galan, and P. Cuesta, eds., *Genetic Algorithms in Engineering and Computer Science*. Chichester: John Wiley and Sons, 1995.

[224] H. Xue, M. Jamshidi, and E. Tunstel, "Genetic algorithms for optimization of fuzzy systems in prediction and control," *Int. J. Knowledge-based Intell. Eng. Syst.*, vol. 1, pp. 13–21, 1997.

[225] L. A. Zadeh, "Fuzzy sets," *Inform. and Control*, vol. 8, pp. 338–353, 1965.

[226] Q. Zhao and T. Higuchi, "Efficient learning of NN-MLP based on individual evolutionary algorithm," *Neurocomputing*, vol. 13, pp. 201–215, 1996.

[227] Q. Zhao and T. Higuchi, "Minimization of nearest neighbor classifiers based on individual evolutionary algorithm," *Pattern Recog. Lett.*, vol. 17, pp. 125–131, 1996.

LIST OF PUBLICATIONS OF THE AUTHOR

1. S. Bandyopadhyay, C. A. Murthy and S. K. Pal, "Pattern classification using genetic algorithms," *Pattern Recog. Lett.*, vol. 16, pp. 801–808, 1995.

2. C. A. Murthy, S. Bandyopadhyay, and S. K. Pal, "Genetic algorithm based pattern classification : Relationship with Bayes classifier," in *Genetic Algorithms for Pattern Classification* (S. K. Pal and P. P. Wang, eds.), pp. 127–144, Boca Raton: CRC Press, 1996.

3. S. Bandyopadhyay, C. A. Murthy and S. K. Pal, "Genetically searched class boundaries with removal of redundancy," in *Proc. 6th Nat. Sem. Theo. Comp. Sci.*, (Rajasthan, India), pp. 345–356, 1996.

4. S. Bandyopadhyay, C. A. Murthy and S. K. Pal, "GA based pattern classification : Theoretical and experimental studies," in *Proc. 13th Int. Conf. Pattern Recog.*, (Vienna), pp. 758–762, 1996.

5. S. Bandyopadhyay and S. K. Pal, "Pattern classification with genetic algorithms : Incorporation of chromosome differentiation," *Pattern Recog. Lett.*, vol. 18, pp. 119–131, 1997.

6. S. K. Pal, S. Bandyopadhyay, and C. A. Murthy, "Genetic algorithms for generation of class boundaries," *IEEE Trans. Syst., Man, Cybern.* (accepted).

7. S. Bandyopadhyay, S. K. Pal and U. Maulik, "Incorporating chromosome differentiation in genetic algorithms," *Inform. Sci.* (accepted).

8. S. Bandyopadhyay, C. A. Murthy and S. K. Pal, "Theoretical performance of genetic pattern classifier," *J. Franklin Institute* (accepted).

9. S. Bandyopadhyay, S. K. Pal and C. A. Murthy, "Simulated annealing based pattern classification," *Inform. Sci.* (accepted).

10. S. Bandyopadhyay, S. K. Pal and C. A. Murthy, "Variable length chromosomes in genetic algorithms for modeling the class boundaries," *3rd Asian Fuzzy Sets Symp.*, Korea, 1998, (accepted).

11. S. Bandyopadhyay, H. Kargupta, and G. Wang, "Similarity aided linkage learning in GEMGA," *Proc. IEEE Int. Conf. Neural Networks*, Alaska, 1998 (accepted).

12. S. Bandyopadhyay and S. K. Pal, "Genetic classifier for pattern classification and determination of MLP architecture," *SCI*, Hongkong, 1998, (communicated).

13. S. Bandyopadhyay, C. A. Murthy and S. K. Pal, "Pattern classification using genetic algorithms : Determination of H," *Pattern Recog. Lett.* (communicated).

14. S. Bandyopadhyay and S. K. Pal, "Relation between *VGA-classifier* and MLP : Determination of network architecture," *Fundamenta Informaticae* (communicated).