CERTAIN PATTERN RECOGNITION TASKS USING GENETIC PROGRAMMING

Durga Prasad Muni
Electronics and Communication Sciences Unit
Indian Statistical Institute
Kolkata - 700108
India.

A thesis submitted to the *Indian Statistical Institute*
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
April 2008

ACKNOWLEDGEMENTS

This work would not have been possible without the active co-operation of my supervisor Prof. J.Das. I would also like to extend my heartfelt thanks to Prof. Nikhil R.Pal who had been not only my valued guide but also a source of inspiration. Like a true mentor, they have guided me in my effort to do the research work and develop the thesis and no word is enough to acknowledge their co-operation and help. I am grateful to them for the time they have spent in discussing our research and scrutinizing it critically and with painstaking care. They have also been kind enough to allow our joint works to be included in this thesis.

I would also like to thank the members of the Research Fellow Advisory Committee of the Computer and Communication Sciences Division for their critical comments during my annual reviews without which this herculian task could not have been possible.

I would also take this opportunity to thank the faculties of Electronics and Communication Sciences Unit for their constant encouragement to speed up the research work.

I would like to thank Mr. Dibyendu Banerjee and my wife Sagarika for giving me the coveted support in time of need.

I also extend my sincere gratitude to Prof. P. K. Nanda, Mr. P.P. Mohanta and my seniors Dr. Debrup Chakraborty and Dr. Arijit Laha who have extended to me their incessant and unconditional support to help me go ahead with my research work. I would never forget the company I had from my friends such as Lingaraj, Kanhu, Sarif, Sanjaya, BLN, Pradip, Bibhas, Sitansu, Arindam, Brojeshwar, and Samrat.

I am very much grateful to my family especially to my parents for providing me the opportunity to continue my research work unhampered and for being a source of continuous mental support and encouragement. Without them, this enormous work wouldn't have seen the light.

The Institute's library staff and everyone in the Electronics and Communication Sciences Unit (ECSU) have always been co-operative and friendly, which helped a lot. The Indian Statistical Institute has been like a mother tree to me helping a scapling as like me to thrive and prosper to my present self by giving me support in every sphere.

ISI, Kolkata
April 2008. (Durga Prasad Muni)

Contents

List of Figures

List of Tables

List of Important Abbreviations

ANN	Artificial Neural Networks
CI	Computational Intelligence
EAs	Evolutionary Algorithms
EC	Evolutionary Computation
EP	Evolutionary Programming
ES	Evolutionary Strategies
FCM	Fuzzy K Means
FS	Feature Selection
GAs	Genetic Algorithm
GP	Genetic Programming

Chapter 1

Introduction and scope of the thesis

1.1 Introduction

In this thesis, we focus on developing methodologies to solve certain pattern recognition tasks using evolutionary algorithms. Before explaining our methodologies, we give a very brief introduction to pattern recognition and evolutionary algorithms for a better understanding of the thesis.

Modern man is over flooded with myriad of information each distinct and complex in its own nature. Extraction of useful information from such data often reduces to recognition of various patterns present in the data. Thus, one needs to do pattern recognition. But what is pattern recognition (PR)?

According to Duda and Hart [36], pattern recognition is a field concerned with machine recognition of meaningful regularities in noisy or complex environments. In [115], pattern recognition is defined as the categorization of input data into identifiable classes via extraction of significant features or attributes of the data from a background of irrelevant detail.

Recognition is a basic feature of every living organism and pattern is the "description" of an object that affords its recognition from amongst other objects. For example, when we spot a known person among a host of people or recognize a voice among a cacophony, we are using our pattern recognition capability. We can read handwriting and recognize music. A human being is a very sophisticated information processing system, partly because he possesses a superior pattern recognition capability [115]. Though human beings have very good pattern recognition ability, human sensory system has

certain limitations. Some undetectable patterns or otherwise patterns which are more complex in nature become difficult for human beings to recognize unaided. For instance, detecting the sound of a submarine, in the presence of other marine signals and noise. There are also some monotonous routine pattern recognition jobs. These tedious tasks require huge manpower. One such case is detection of defects in objects in factory outlet. Information can be the most valuable asset to the human being only if he can extract potential or valuable knowledge from it. Human analysts can no longer keep track of or make sense of the enormous volume of information in this era of rapidly growing and advanced technology. Thus, it becomes essential to automatically process this plethora of information efficiently and automatically. Information may be in the form of text, image, voice and raw data. To store, manage and process oodles of information in real time, and accurately we need automatic techniques like *automatic pattern recognition*. Speech recognition, fingerprint identification, optical character recognition, DNA sequence identification, medical diagnosis, and remote sensing data classification are some examples of pattern recognition.

A typical Pattern Recognition system (PRS) consists of three tasks namely, *data acquisition and/or preprocessing, feature analysis, classification and/or clustering*. In the first step, data are collected by using some sensors or other means. And then these raw data may be preprocessed. Preprocessing may involve noise reduction, normalization and conversion of raw data into suitable form for pattern recognition. After obtaining the data, good features are extracted by mapping data to other domain or a subset of good features is selected from the available features. This process finds useful features to obtain an efficient and improved solution to a given problem. Success of pattern recognition depends on the features used. Finally, in the classification and/or clustering phase, the actual task of PRS is performed. Classification involves assigning a class label to a given pattern while clustering finds homogeneous subgroups in data. However, all these components may not be essential in a PRS. Also, consecutive phases may be combined together. The design of PRS depends on the problem at hand. If data are available for pattern recognition, then we may require schemes for feature extraction/selection and for classification/clustering. The tool that performs classification is called classifier. In this thesis, we have primarily focused on two important tasks: classifier design and feature selection.

Since the very beginning, statistical approaches [44] have been used for pattern recog-

nition. However, the classical statistical methods are not always well suited for difficult pattern recognition problems. Many alternative approaches have been introduced which can address these complex pattern recognition problems. A collective approach called *Computational Intelligence* [70] is one of them. These methods are quite useful and popular to build "intelligent" systems.

Computational intelligence includes mainly three tools: Artificial Neural Networks (ANNs), Fuzzy logic and Evolutionary algorithms (EAs). ANNs [57] are most popular among computational intelligence methods. They have good learning and generalization abilities but sometimes lack interpretability and may work as a black box.

It is desired to have decision-making systems with reasoning ability as human beings. It would be also better if linguistic rules can be used to describe a decision making system. Fuzzy logic [68, 12] can fulfill these requirements up to a great extent. It can handle uncertainty and vagueness in the real-world problems.

Evolutionary algorithms (EAs) [6, 46, 72] evolve desired solution to a given problem using biologically inspired operations like crossover, mutation and the Darwinian *Principle of the Survival of the Fittest*. At first, typically a "population" of representation of possible solutions is (randomly) generated. Each representation is called a chromosome. Then, variation (genetic operation(s)) and selection operations are implemented on the current population to create the next population. This is motivated by the hope that new generation will be better than previous generation. The process of evolution is continued till the desired solution is obtained or till the termination criteria are satisfied. The computation using Evolutionary algorithms is called *Evolutionary Computation*.

Like pattern recognition, an urge to find an appropriate/optimal solution is a basic feature of human being. *Optimization* is a process to find the optimal solution for a given problem. We can *search* the solution space to find the optimal/appropriate solution. Many pattern recognition tasks can be viewed as search and optimization problems. For example, in case of classifier design, we search the feature space to obtain the appropriate classifier(s). In most cases, we assume a structure of the classifier and try to optimize the parameters involved in the classifier. During optimization process, we either minimize or maximize a performance index. So, when we design classifier, we may attempt to minimize the classification error of the classifier over a set of known

samples called training set.

EAs are basically randomized search and optimization techniques [46, 18]. Most traditional search and optimization techniques are appropriate only for certain types of problems. For example, *Calculus-based* optimization methods are suitable for the problems having smooth, continuous, unimodal and differentiable (search) space. *Enumerate* search techniques are used when the number of possible solutions is finite and small. But EAs are robust, efficient and adaptive. They have ability to find near-optima solutions in acceptable time for a wide range of optimization problems. EAs are in great demand where traditional methods fail. EAs do not require that the search surface should have continuous, unimodal and differentiable. EAs can be used for the problems where search space is vast. Moreover, unlike traditional methods, these are population based search techniques. This helps to reach to or near to global optima by avoiding local optima. This is a great advantage of EAs. Due to all these advantages, EAs are widely used to solve complex real-world optimization problems.

Genetic Algorithms (GAs) [46, 18] are most popular EAs. In GAs, each solution is represented by a finite-length string. GAs are usually used to optimize the parameters of a model such as classifier or to find configuration (subset) of certain set of variables.

Genetic Programming (GP) [72, 7] is another EA. GP is a variation of GAs. In GP, each solution is typically represented by a tree or a program. This difference makes it a tool suitable for structural optimization in addition to inherent parameter optimization of a model. And hence, we don't require assuming a particular structure of the model if we use GP. It can find both (near-optimal) structure and values of parameters involved of the model and it provides the expression of the model. We can not only employ the model of the system to solve the problem but also can analyze the expression of the model. For this reason, GP is rapidly becoming popular. This distinct advantage of GP also *motivated* us to adopt it for different pattern recognition tasks. Before describing the scope of the thesis, we will attempt to describe motivation for each of my proposed methods in the next section.

1.2 Motivation

Classifier design is an important pattern recognition task. GP can learn the hidden relationship in the data and can provide the desired classifier including its expression. This helps to analyze the classifier in addition to employing it to classify unknown data points. This feature of GP has attracted many researchers to use GP for classifier design. Most available GP based classifier design methods deal with two class classification problems. Only few researchers have developed classifiers for multi-class problems using GP [81, 24, 84, 67, 123]. These methods are interesting but usually require more than one GP run to develop classifiers for multi-class problems. So, that motivated me to propose an algorithm that can develop classifier in a *single* GP run.

Success of a Pattern Recognition Systems depends on features we use. In literature, very few works are available where GP has been used for feature selection/extraction. The suitability of a feature subset depends not only on the problem being solved but also on the solution (classifier) that is used to solve the problem. So, if we design classifier and select features for that classifier simultaneously, then we can obtain better classifier. As the available GP based feature selection methods do not select features and design classifiers at the same time, we propose an algorithm for it. We have explained the previous line as follow. Typically when GP is used to design classifiers, different classifiers may use different sets of features and they may not use all features. Hence, one can argue that a GP- based scheme for classifier design does an implicit feature selection. However, since such a design does not explicitly consider the task of feature selection, in the worst case some classifier may even use all features, and some may involve derogatory features. Such a design process does not penalize the use of larger feature set because it considers only the classification accuracy. For example, if a classifier using all features performs slightly better (may be either on the training data or on a validation set) than another classifier that uses a very small number of features then we consider the former one better as our objective is to improve the classification accuracy. However, a better objective would be to obtain good accuracy using a small number of features. Such a classifier can lead to better interpretability and usually since such a system will have less degrees of freedom, it is likely to yield better generalization. So, classifier design should be formulated as a multi objective task giving explicit importance on the cardinality of the used feature set. Our approach

considers both classification accuracy and size of the used feature subset explicitly while evaluating a classifier.

While designing models, we may incorporate fuzzy logic concept to address vagueness and uncertainty in data. So, instead of evolving crisp classifiers, we can evolve fuzzy classifier rules using GP. In this case too, few attempts have been made to generate fuzzy classifier rules using GP. We propose a simple but effective scheme to evolve fuzzy classier rules for multi class classification problem. It optimizes both structure and values of involved parameters of the rules.

So far we have considered development of general methodologies. Next we consider a specific application area. Art is being considered as the most creative and innovative discipline. Computer is being used in artistic application too. Generation of interesting images and textures using computer for various purposes including fashion/textile design, animation is a recent trend. Texture [113] is a property of the surface or structure of an object. A texture produced by an algorithm or a procedure is called a procedural texture [37]. For each point of the procedural texture, the procedure gives the corresponding gray level or color value. Although the procedural representation is extremely compact compared to the image representation, it is difficult to find/write procedures that will generate textures for some target application. Evolutionary algorithm [46, 72] is a possible solution to this major problem. For evolution towards better solutions (textures), we need to evaluate each solution (texture) that indicates how good the solution is. Despite its wide use, texture has no precise definition. So an automatic evaluation of textures is not an easy task. In comparison, a human being can easily identify and assess a texture. If we allow computer to generate textures and user to determine which textures are good according to his/her choice in the process of generation, then interesting textures can be created. Generation of textures based on this principle is called interactive texture generation. However, interactive texture generation could be a tedious process if a user needs to assign a fitness value to every generated texture. Consequently some methods generate textures similar to a (or a set of) given reference texture(s). This type of GP based schemes are presented in [120] which evolve procedures to produce textures similar to a given reference texture. This is an interesting approach but requires reference texture(s) and also produces only similar textures with respect to the given reference texture(s). But, the most optimal texture will be identical to the given reference texture.

Both of these approaches have their limitations. Hence, we need to device a hybrid approach that can combine the advantages of both approaches. This motivated us to propose a new approach using GP to generate textures. GP can generate procedures to produce interesting textures. The proposed GP system acts as a sort of pattern recognition system.

Texture Classification is another pattern recognition task. We have already devised GP classifier systems. Hence, we were curious to implement our GP classifier schemes for texture classification.

1.3 Scope of the thesis

In this thesis, I address certain pattern recognition tasks using Genetic Programming. These tasks are classifier design, simultaneous feature selection and classifier design, fuzzy rule based classifier evolution, and texture generation and classification. In this context, we have proposed various methodologies. The proposed schemes are validated on a set of benchmark real data sets. The performances of the methods have been compared with some existing ones.

Chapter 1 and Chapter 2 act as the prologue to the thesis where we have introduced the reader briefly to the scope and nature of the work. The following chapters, however, are the heart of the thesis that contain the detail studies of our research and methodologies. The introductory sections of these chapters begin with a brief literature survey of the works that are to follow. Contents and contribution of the chapters are summarized in the subsequent sections.

1.3.1 An Introduction to Pattern Recognition and Evolutionary Algorithms

In Chapter 2, we provide an introduction to Evolutionary algorithms (EAs) and pattern recognition. We especially emphasis on Genetic Programming, the tool we use for solving the tasks. Also how EAs excel in optimization has been briefed.

1.3.2 Classifier Design using GP

In Chapter 3, we begin with a short survey of previous works on classifier design using GP and then we present our proposed GP based scheme for classifier design for multi-category classification problems. The proposed approach takes an integrated view of all classes when the GP evolves. A multi-tree representation of chromosomes is used. For c-class problem, a chromosome (possible classifier) consists of c trees, each representing a classifier for a particular class. In this context, we propose a modified crossover operation and a new mutation operation that reduces the destructive nature of conventional genetic operations. We use a new concept of *unfitness* of a tree to select trees for genetic operations. This gives more opportunity to unfit trees to become fit. Further a new concept of OR-ing chromosomes in the terminal population is also introduced, which enables us to get a classifier with better performance. Finally, a weight based scheme and some heuristic rules characterizing typical ambiguous situations are used for conflict resolution. The classifier is capable of saying "don't know" when faced with unfamiliar examples. The effectiveness of our scheme is demonstrated on several real data sets.

1.3.3 Simultaneous Feature Selection and Classifier Design using GP

In Chapter 4, we propose a GP system that performs feature selection and classifier design simultaneously. Although, one can view that GP does an implicit feature analysis, as explained earlier, if we do not explicitly add a penalty to the fitness function when more features are used, then the classifiers can use more features than what is required. Some features may be redundant or irrelevant, even some may be derogatory. Therefore, we should consider both classification accuracy and the number of features used while evaluating a classifier. The goal is to design a better classifier using small number of features. Here we propose this multi-objective approach that can simultaneously select features and design a classifier.

At the beginning of the chapter, we give a brief literature survey on feature selection. Our GP scheme selects a good subset of features and constructs a classifier using the selected features simultaneously. For a c-class problem, it provides a classifier having

c trees. In this context, we introduce two new crossover operations to suit the feature selection process. As a byproduct, our algorithm produces a feature-ranking scheme. We tested our method on several data sets having dimensions varying from 4 to 7129. We compared the performance of our method with results available in the literature and found that the proposed method produces consistently good results. To demonstrate the robustness of the scheme, we studied its effectiveness on data sets with known (synthetically added) redundant/bad features.

1.3.4 Evolution of Fuzzy Classifier Rules using GP

In Chapter 5, we propose a Genetic Programming (GP) based approach to evolve fuzzy rule based classifiers. For a c-class problem, a classifier consists of c trees. Each tree, T_i, of the multi-tree classifier comprised a set of rules for class i. During the evolutionary process, the inaccurate/inactive rules of the initial set of rules are removed by a cleaning scheme. This allows good rules to sustain and that eventually determines the number of rules. In stead of using all features in a rule, in the beginning, our GP scheme uses only a randomly selected subset of features and then evolves the features to be used in each rule. The initial rules are constructed using prototypes. The prototypes are generated randomly as well as by the fuzzy K-means (FKM) algorithm. Experiments are conducted in three different ways: using only randomly generated rules, using a mixture of randomly generated rules and FKM prototype based rules, and with exclusively FKM prototype based rules. Contrary to expectation, randomly generated rules work better than FKM based rules in most cases and this emphasizes the novelty of the proposed scheme. In this context, we propose a new mutation operation to alter the rule parameters. Hence, the GP scheme not only optimizes the structure of rules but also optimizes the parameters involved. This results in good fuzzy rule based classifiers. Moreover, the resultant fuzzy rules can be analyzed. The performance of the proposed scheme is quite satisfactory.

1.3.5 Texture Generation and Classification using GP

So far, we have developed general methodologies. In Chapter 6, we consider a specific application area. We present a new method to generate textures using GP and also

we use our GP classifier schemes for texture classification. Genetic Programming can evolve suitable procedures or mathematical functions to solve a problem. This advantage has been utilized to generate procedures that can produce interesting/desired textures. Our GP based texture generation scheme acts as a sort of a pattern recognition system.

The initialization process of GP generates tree representation of procedures. Each generated procedure T_i is activated to produce an image S_i (2-dimensional gray value matrix). We use *contrast* of the generated images/textures to filter out poor textures. This phase resembles Data acquisition/Preprocessing phase.

If S_i could able to pass through the filtering process, a vector of statistical features $\mathbf{v}_i$ is extracted from S_i to represent it.

Then the pattern is passed through the classification/clustering phase. *Similarity* of the texture S_i with respect to the already generated textures is computed using the feature vector $\mathbf{v}_i$. If the texture is *more* similar to a cluster of already generated textures then the texture is placed in that cluster and the fitness value of that cluster is assigned to the procedure S_i. Otherwise, if the generated texture is *quite dissimilar* with the existing textures then it is displayed for the user. The user according to his/her choice assigns a fitness value by visually inspecting it. The texture is placed in a new cluster and the fitness value of the texture is assigned to that cluster.

After assigning fitness value to each procedure T_i, genetic operations are applied and the evolutionary process is carried on. This produces many interesting textures according to the user's choice by *occasionally* seeking user's discretion.

For texture classification, we extract statistical features to represent them. These features are carefully chosen because success of the process (pattern classification) depends on the features we use. After representing textures by the corresponding statistical feature vectors, we use our GP classifier schemes to design classifier for texture classification. We have experimented the texture classification task on a set of natural textures.

1.3.6 Conclusion and Scope of the future works

We conclude the thesis in chapter 7. The hitherto detailed discussion is represented in a nutshell with comments on its respective merits and drawbacks. The chapter also includes a discussion on the future scope of the research work.

Chapter 2

An Introduction to Pattern Recognition and Evolutionary Algorithms

2.1 An Introduction to Pattern Recognition

According to Duda and Hart [36], pattern recognition is a field concerned with machine recognition of meaningful regularities in noisy or complex environments. As mentioned earlier typical Pattern Recognition system consists of three components namely, *data acquisition and/or preprocessing, feature analysis, classification and/or clustering.* Before going into detail about different pattern recognition tasks, I am giving the following example to explain pattern recognition.

Suppose a girl has never seen a cow or a goat. She also does not know how cows and goats look like. Now a teacher takes the girl to a playground where a host of cows and goats are grazing. Next, the teacher points to each animal and tells whether that is a cow or a goat. In this case each appearance of cow or goat is called a *pattern*, or more particularly a training sample. The girl tries to learn the characteristics of cows and goats and may make the following observations on each animal.

1. Approximate *height*
2. Approximate *length*
3. Approximate length of *tail*
4. Shape of the head

5. Number of *legs*
6. Number of *eyes*
7. Number of *ears*

Such observations are called *attributes* or *features*. A pattern is represented by a set of features or attributes. Collection of data by sensing or measuring the features is called data acquisition.

Now the girl has been told that she may have to recognize cows and goats later. Then the girl analyzes the above mentioned observations and finds that:

(i) Typically the height of a cow is *larger* then the height of a goat.

(ii) Length of a cow is usually *larger* than the length of a goat.

(iii) Tail of a cow is *longer* than the tail of a goat.

(iv) Shapes of heads of cow and goat are different, usually the head of a cow is bigger than that of a goat.

(v) Both cow and goat have the *same* number of legs.

(vi) Both cow and goat have the *same* number of eyes.

(vii) Both cow and goat have the *same* number of ears.

Now, the girl gets an idea that for discrimination between a cow and a goat, the first four observations are important to classify an animal whether it is a cow or a goat and the last three features are not useful. The above analysis of observations/features to find the good features (for classification) is called **feature analysis**. The selection of good features from a given set of features is called *feature selection*. In this case, the first four good features are being selected from the set of seven features.

On the way back, the teacher notices an animal (cow/goat) and asks the girl to identify whether it is a cow or a goat. With her previous experience, the girl only takes interest to observe the above four important features. Then she tries to estimate *how close* are the given features of the shown animal to the corresponding features of a cow and a

goat. That means, she may observe *how big* is the animal is, the length of its tail, and the shape of its head. With her previous learning, she might be able to say whether the animal is a *cow* or a *goat*. Here *cow* and *goat* are called the *classes* to which the pattern may belong to. This task of categorization of a given pattern to a known class is called **classification**. This is also called supervised classification because there is a need to supervise or to teach the learning algorithm(girl) prior to classification of unknown patterns. If c stands for the number of classes then classification can be defined as follows: Classification is the partition of the feature space into c subsets, one for each class and mapping a given unknown pattern (belongs to one of these classes) or point to one of the subsets of the feature space.

In case of **clustering**, there is no need of a supervisor (or a teacher). To explain *clustering* the above example can be modified as follow. A boy who has never seen a cow or a goat is taken to a playground where the cows and goats are grazing. The boy is asked to group the similar animals. He may observe the above mentioned features of the animals and may analyze the features to find good features to discriminate the two types of animals. Further, he may choose the above mentioned four good features. Using these four features of animals, he may be able to cluster the cows into one group and goats into another group. This task is called *clustering*. Clustering may be defined as the grouping of similar given patterns into groups- it is a partitioning of a given data set.

In the following subsections, different phases of a typical pattern recognition system are described.

2.1.1 Data Acquisition and/or Preprocessing

In this phase data is collected by a set of sensors or by other means. Data may be numerical, linguistic, pictorial, signal/waveform, or any combination of them. After obtaining, the raw data is preprocessed. It may involve noise reduction, normalization and conversion of raw data suitable for the task i.e. for pattern recognition. It typically represents a pattern by a vector of features. This type of data representation is called *object data* type and it is most common in pattern recognition. There is another type of data structure that is rarely used in pattern recognition. It is called *relational data* and consists of the pairwise relations (similarities or dissimilarities) between each pair

of objects.

2.1.2 Feature Analysis

Feature analysis (FA) is a process to find useful features to obtain an efficient and improved solution to a given problem. All available features are not useful for the task at hand. Some of the features may be redundant while some others may bad too. These features may cause confusion during the process of model(e.g. classifier) development. These features unnecessarily increase the complexity of the feature space which in turn demands more computation time to find a solution to the given problem. FA is a process of finding a map $\Phi : \mathcal{R}^p \rightarrow \mathcal{R}^q$ using a criterion J on the given (training) data set. Typically, $q < p$ and it is called dimensionality reduction.

Depending upon the type of process, it may be categorized into two basic types: *feature extraction (FE)* and *feature selection (FS)*. Feature extraction is a method to generate a q dimensional feature vector from a given p dimensional input vector. Although, it is not necessary that $q < p$; however, for pattern recognition $q < p$ is preferred. In other words, the original features are projected in a different space of lower dimensionality by using some criteria. The extracted features are the linear/nonlinear combination of the given set of features that may not bear the meanings of the original ones. Principal component Analysis (PCA) [44] is a popular feature extraction method for pattern recognition.

Feature selection(FS) selects a subset of good features from the set of available features. Ideally, the feature selection process should select an optimum subset of features from the set of available features which is necessary and sufficient for solving the problem.

2.1.3 Classification and/or clustering

Classification and/or clustering is the actual task of a pattern recognition system. In pattern classification task, we assume that there exist c groups or classes, denoted by $\omega_1, \omega_2, ..., \omega_c$. For a given pattern $\mathbf{x}$, we assign a class label $i \in \{1, 2, ..., c\}$ denoting $\mathbf{x}$ belongs to class ω_i. The abstract space that represent all possible data points is called feature space. Basically, for c-class classification task, the feature space is partitioned into c partitions, each representing a particular class. The model that defines the

partition is called classifier. Unfortunately, in real world we don't have all possible data points to obtain the exact partition or classifier. In stead, we have a finite and usually a small set of data points that provides partial information for optimal design of models such as classifier, feature selector/extractor. Hence, it is assumed that these data points are representative of (distribution of) the classes. This data set is typically called *training set*. Each data point of the training set is associated with its corresponding class label. On the basis of the training set, we extract/select features and design the classifier. The algorithm or the methodology learns from the information including class membership available in the training set and provides the model. Here, the algorithm is supervised by providing the class membership of training samples while learning. This type of learning is called *supervised learning*. After obtaining the classifier, it is used to classify unknown data points i.e. patterns without knowing their class labels. In practice, before classifying unknown patterns, the classifier is tested on a set of data points called *test set*. Although class label is associated to each data point of test set, while classifying the test point by the classifier we don't use the class label. After classification of test point, the estimated class label of the test point by the classifier and its actual class label is compared. On the basis of the comparison, the error/accuracy of the designed classifier is estimated. If the performance is satisfactory, then it is used to classify unknown data points.

A classifier can be defined as follow. A classifier D is a mapping, $D : \mathcal{R}^p \rightarrow N_{hc}$, where $\mathcal{R}^p$ is the p-dimensional real space and N_{hc} is the set of label vectors for a c-class problem and is defined as $N_{hc} = \{\mathbf{y} \in \mathcal{R}^c : y_i \in \{0, 1\} \forall i, \sum_{i=1}^{c} y_i = 1\}$. For any vector $\mathbf{x} \in \mathcal{R}^p$, D($\mathbf{x}$) is a vector in c-dimension with only one component as 1 and all others as 0. In other words, a classifier is a function which takes a feature vector in p dimension as input and assigns a class label to it.

In clustering, we cluster the given data points into homogeneous subgroups. The difference between classification and clustering is that in classification we partition the feature space while in clustering we partition the given data points into groups. Unlike classification, we do not require the class label of the data points in clustering. And hence the process is also called unsupervised learning.

These are the three basic components of a typical pattern recognition system. However, all these above mentioned processes may not be required for a pattern recognition system. For example, we can directly provide the measurements or values of required

features to the classifier. Here, the user intuitively decide the features and hence acts as a feature selector. Also, some phases may be combined together. In the above example, data acquisition and feature analysis are considered together. A pattern recognition system may have both classification and clustering schemes. In some cases, feature analysis and classification/clustering are performed simultaneously. We have proposed an algorithm where feature selection and classifier design are simultaneously performed.

With the introduction to pattern recognition, we now discuss the main tool, EA, that we have used in this thesis to solve different pattern recognition tasks.

2.2 An Introduction to Evolutionary Algorithms

Evolution is the natural developmental process by which different kinds of living organism develop from the earlier forms. From the very dawn of human civilization man has been engaged incessantly in his onward quest for perfection by closely observing the various traits of nature and consequently analyzing and deducing the logic behind such natural occurances. He has observed the flights of birds and with an urge to defy gravity like them, he developed the flying machine but without being self complacement, he has been constantly improving on it till now. Fascinated by the way the brain neurons function, he developed artificial neural networks (ANNs). With this similar urge, he developed algorithms which mimic the principle of *natural evolution*. These algorithms are called *evolutionary algorithms*.

Evolutionary algorithms(EAs) evolve desired solution to a given problem using biologically inspired operations like crossover, mutation and the Darwinian principle of the *survival of the fittest*. Typically, initially a "population" of possible solutions is taken. Then, variation (genetic operation(s)) and selection operations are implemented on the the current population to create the next population. This is motivated by the hope that new generation will be better then previous generation. The computation using Evolutionary algorithms is called *Evolutionary Computation*. Although the origin of evolutionary computation can be traced back to the late 1950's, it slowly became popular during the 1970's with major contributions from pioneer researchers like Holland, Rechenberg, Schwefel and Fogel [5]. Now, with powerful computers, use of EAs is increasing rapidly.

Genetic Algorithms [46], Genetic Programming [72], Evolutionary Programming [6] and Evolutionary Strategies [6] are four major Evolutionary Algorithms. These algorithms have been discussed in the following sections.

2.2.1 Genetic Algorithms

Genetic Algorithms(GAs) [46, 18], introduced by Holland and subsequently studied by researchers like De Jong and Goldberg are most popular evolutionary algorithms. GAs are mainly used for optimization problems.

In GAs, the solutions are encoded into finite-length strings of alphabets of certain cardinality. These strings are called chromosomes, the alphabets are referred to as genes and the values of genes are called alleles. In most cases, chromosomes are binary strings consisting of alphabets "1" and "0". In a problem such as the traveling sales man problem, a chromosome represents a route, and a gene may represent a city.

Initially, a population (set) of solutions are randomly generated for evolution. In the process of evolution and natural selection, the good solutions survive and produce offspring while the bad solutions die out (removed). To determine the goodness of the solution, each solution is evaluated using an objective function. The evaluated value is assigned as the fitness value to that solution. Instead of using an objective function, a subjective function may be considered where the user choose better solutions over worse one.

The following steps are carried out in Genetic Algorithm.

1. Initialization: A population of solutions are randomly generated across the search space. However, if domain-specific knowledge is available, then it can be incorporated.

2.Evaluation: Each solution of the population is evaluated and the evaluated fitness value is assigned to the solution.

3. Based on the fitness values, better solutions are selected for genetic operations.

4. Usually two chromosomes are taken for crossover with probability p_c. That means, a random number $r \in [0, 1]$ is generated. If $r \leq p_c$, then these two chromosomes are allowed for crossover. In crossover operation, sub-strings of both chromosomes (parents) are randomly taken and swapped among them. Usually, p_c is very high

(≈ 0.95).

5. After crossover operation, each alphabet of each chromosome is allowed for mutation with mutation probability p_m. Usually, p_m is very low (≈ 0.001). In mutation operation, the alphabet is altered. In binary string, "1" is inverted to "0" and vice versa.

6. During evolution, there is chance that the best chromosome of the GP population may be destructed by genetic operations. Hence to avoid this, a copy of best chromosome is always preserved during evolution.

7. Continue steps 2-6 till the termination criteria are not satisfied.

2.2.2 Evolutionary Programming

Evolutionary Programming(EP) [5, 63] was introduced by Fogel in 1961. The purpose was to create artificial intelligence in the sense of developing ability to predict changes in an environment. EP uses the concepts of natural evolution, selection and stochastic mutation. It does not use crossover operation. EP was originally meant for evolving finite state machine(FSM) to predict events on the basis of former observations. An FSM transforms a sequence of input symbols into a sequence of output symbols based on a finite set of states and state transition rules.

Now-a-days EP methodologies are implemented in many discrete and continuous parameter optimization problems [6]. Like other EAs to evolve the desired (optimum) solution (parameter), a population of μ possible solutions are generated. Depending on the problem domain, the representation of solutions varies. Each solution of the population was evaluated. Mutation operation is then implemented on the solutions to produce μ offspring. In case of FSM, there are five possible mutation operators: change of an output symbol, change of a state transition, addition of a state, deletion of a state, and change of the initial state. After evaluating the μ offspring, a selection of the μ best out of parents and offspring, i.e. a $(\mu + \mu)$-selection, was performed. EP implements a probabilistically selection method to select individuals from the $(\mu+\mu)$ individuals for the next generation: Each individual is compared with $q > 1$ other randomly chosen individuals from the union of parents and offspring. For each comparison, a "win" is assigned if the individual's score is better or equal to that of

opponent. Then μ individuals with the greatest number of wins are retained to be parents of the next generation [6].

2.2.3 Evolutionary Strategies

Evolutionary Strategies (ES) [98, 6, 63] were proposed by Rechenberg and Schwefel in 1960s as a method to solve parameter optimization problems.

Initially ESs was based on population consisting of only one individual and only one genetic operator: mutation. The individual was represented by a pair of float-valued vectors,i.e., $\vec{v} = (\vec{x}, \vec{\sigma})$. $\vec{x}$ represents a point in the search space and $\vec{\sigma}$ is a vector of standard deviations. Mutation is implemented by replacing $\vec{x}$ by
$\vec{x}^{t+1} = \vec{x}^{t} + N(0, \vec{\sigma})$
where $N(0, \vec{\sigma})$ is a vector of independent random Gaussian numbers with a mean of zero and standard deviations $\vec{\sigma}$. If the offspring (mutated individual) is better (higher fitness) than parent, then it replaces the parent. Otherwise, parent is retained and the offspring is discarded.

Later on, population (multi membered) based ESs were introduced. Here μ parents produce λ offspring. In (μ, λ)-ESs, μ individuals are selected from the λ offspring for the next generation. This method is not elitist and we may get worse individuals in the next generation. It has been given in [6] that this may help to leave the region of attraction of a local optimum and reach a better optimum. In contrast, the $(\mu+\lambda)$-ESs select μ offspring from the combined $(\mu + \lambda)$ individuals for the next generation. This retains best individuals of the previous generation and hence it does not loose the best individuals during the process of evolution (variation).

In these population based ESs, the strategy parameter $\vec{\sigma}$ is no longer constant. Rather, it is incorporated in the structure of the individual itself and undergoes the evolutionary process. This facilitates the *self-adaptation* of the parameters. To produce an offspring, usually two parents are selected and crossover operation is applied on them. Suppose two selected parents are:
$(\vec{x}^1, \vec{\sigma}^1) = ((x_1{}^1, ..., x_n{}^1), (\sigma_1{}^1, ..., \sigma_n{}^1))$ and
$(\vec{x}^2, \vec{\sigma}^2) = ((x_1{}^2, ..., x_n{}^2), (\sigma_1{}^2, ..., \sigma_n{}^2))$.

The crossover operation blends the two parents. The crossover operation may be

different types. In one type, components from the first or second parent are selected to produce the offspring $(\vec{x}, \vec{\sigma})$ as below:
$(\vec{x}, \vec{\sigma}) = ((x_1{}^{l_1}, ..., x_n{}^{l_n}), (\sigma_1{}^{l_1}, ..., \sigma_n{}^{l_n}))$,
where $l_1 = 1$ or 2.
In other type, average of components of parents are taken:
$(\vec{x}, \vec{\sigma}) = (((x_1{}^1 + x_1{}^2)/2, ..., (x_n{}^1 + x_n{}^2)/2), ((\sigma_1{}^1 + \sigma_1{}^2)/2, ..., (\sigma_n{}^1 + \sigma_n{}^2)/2))$

After producing offspring $(\vec{x}, \vec{\sigma})$ by crossover operation, mutation operation is applied on it. The resultant offspring is $(\acute{\vec{x}}, \acute{\vec{\sigma}})$ where
$\acute{\vec{\sigma}} = \vec{\sigma}.e^{N(0,\Delta\vec{\sigma})}$
$\acute{\vec{x}} = \vec{x} + N(0, \acute{\vec{\sigma}})$
where $\Delta\vec{\sigma}$ is a parameter of the method.

2.2.4 Genetic Programming

In 1980s, researchers like S.F. Smith (in 1980) and N. Cramer (in 1985) proposed variation of genetic algorithms that can evolve computer codes. However, with Koza's contribution in 1990s, it become rapidly popular. This algorithm has been given name Genetic Programming.

Since the thesis is mainly based on GP, we provide a detailed discussion on it.

Genetic Programming [72, 7, 73] evolves a population of *computer programs*, which are possible solutions to a given optimization problem, using the Darwinian principle of *Survival of the Fittest*. It uses biologically inspired operations like reproduction, crossover and mutation. This is a variation of Genetic Algorithm (GA). The main difference between GP and GA is representation. In GP, each solution is typically represented by a *tree*. In few research papers, different structures such as linear structure and graph structure are also used. I focus on tree representation of GP solution for three reasons: 1. It is the standard representation 2. It is most popular 3. I use this representation in my thesis.

However, the concept of GP is same for all representations.

I give a schematic overview of GP algorithm in the next section and details of GP in subsequent sections.

Steps of GP:

A typical implementation of GP involves the following steps.
Step 1) *Initialization:* GP begins with a randomly generated population of solutions of size P.
Step 2) *Termination:* GP is terminated when termination criteria are satisfied. Unlike GA , GP will not converge. So, GP is terminated when a desired solution (may be with fitness value 1) is achieved. Otherwise, it is terminated after a predefined number of generations.
Step 3) *Evaluation:* A fitness value is assigned to each solution of the population.
Step 4) *Next Generation:* The current population is replaced by a new population by means of applying genetic operations probabilistically.
Step 5) This completes one generation. Go to step 2 and repeat if termination criteria are not satisfied.

Initialization:

Each individual or solution in the GP population is generally represented by a hierarchically structured program or a tree composed of functions and data/terminals appropriate to the problem domain. The function set may contain

- standard arithmetic operators: +,-,*,/,...
- mathematical functions: Sin,Cos,Exp,Log,...
- Boolean Operators: AND,OR,NOT,...
- Conditional: If-Then-Else,...
- Relations: $<, =, >, ...$
- Iterations and Loops: Do-Until, While-Do,For-Do
- Control Transfer Statements: Go To, Call, Jump
- Domain specific functions: Move-Random,If-Food-Ahead,...

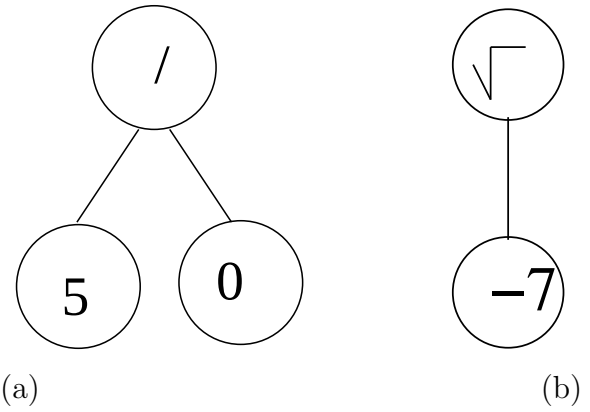


Figure 2.1: (a) Division by 0 and (b) square root of a negative number

The terminal set usually consists of arguments and constants for the functions. The set of functions $\mathcal{F}$ and set of terminals/inputs $\mathcal{S}$ must satisfy the closure and sufficiency properties.

Closure Property: The closure property demands that the function set is well defined and closed for any combination of arguments that it may encounter. As a tree (or expression) is generated randomly and afterward it is also randomly changed, a function (parent node) of a tree or expression may encounter different types of arguments (children nodes). For example, when we use *division* function to generate tree in random manner, then *division by 0* situation may occur (e.g. Fig. 2.1 (a)). To cope up with this type of undefined case, we have to define the *division* function properly. Similarly, the square root can encounter a negative argument (e.g. Fig. 2.1 (b)) and the logarithm function can encounter non-positive argument (in a randomly generated expression or tree). In these cases, we must satisfy the closure property by using protected functions which can handle these type of situations.

Sufficiency Property: The sufficiency property requires that the set of functions in $\mathcal{F}$ and the set of terminals in $\mathcal{S}$ be able to express a solution to the problem [72]. For example, the function set $\mathcal{F} = \{\text{AND, OR, NOT}\}$ is sufficient to represent any boolean expression. However, $\mathcal{F} = \{$ AND, OR$\}$ is not sufficient to build all possible boolean expressions.

After determining function set $\mathcal{F}$ and terminal set $\mathcal{S}$, a set of tree structures are (randomly) generated as an initial population. To prevent generation of large tree structures, restriction in size of tree is imposed. That means, depth of tree is restricted

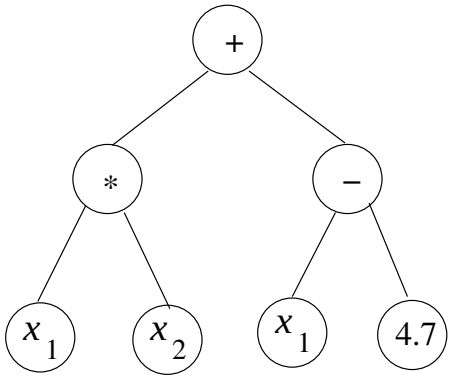


Figure 2.2: A tree generated by Full Method

to a maximum depth m_h and number of nodes of an individual is not allowed to exceed a limit m_n.

There are basically two approaches used to generate trees, namely, *Full* method and *grow* method.

Full method: Trees are generated by randomly choosing nodes *only* from function set till the last level(maximum depth). At the last level (level m_h), nodes are chosen randomly *only* from terminal set. This produces trees of same height m_h. Fig. 2.2 shows a typical tree generated by the *Full* method.

Grow method: It generates trees of irregular shapes. The root node is randomly chosen from the function set $\mathcal{F}$. After that, for each node at depths less than the maximum depth, each node is randomly selected from the *union* of function set and terminal set $(F \cup S)$. As the growth of a tree is restricted to maximum height m_h, so if a branch goes up to the maximum depth (m_h), then it is terminated by selecting a node only from the terminal set $\mathcal{S}$.

Although *grow* method generates a variety of tree structures, but still it is preferred to have diverse tree structures of different heights and shapes using both the above mentioned methods. To facilitate this a mixed approach called *ramped half-and-half* is widely used.

Ramped half-and-half method: It incorporates both full method and grow method with equal importance. It generates equal number of trees of each height from 2 to maximum height m_{rh}. For example, if $m_{rh} = 5$, then 25% of the trees will have depth 2, 25% will

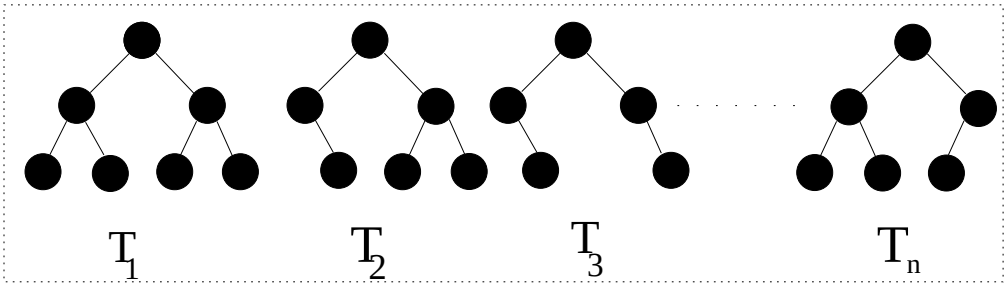


Figure 2.3: A multi-tree representation of an individual

have depth 3, 25% will have depth 4 and 25% will have depth 5. Then, for each depth, 50% of the trees are generated using *Full* method and remaining 50% of the trees are created using *Grow* method. This ramped half-and-half produces trees of wide variety having various sizes and shapes.

Please note that, in all the above three mentioned methods, if number of nodes of the tree exceeds the maximum number of allowed nodes m_n then the tree is abandoned and it is regenerated.

Multi-tree Representation

Instead of representing a solution by a single tree, it can be represented by a set of trees. For example, consider the classifier design task for multi-class classification problem. If $c(\geq 2)$ is the number of classes, then the classifier can be represented by c number of trees $T_1, T_2, ..., T_c$. Tree T_i will represent a classifier for the i_{th} class. Fig 2.3 shows a multi-tree representation of a chromosome or an individual. I have used this multi-tree representation to represent classifiers in my thesis. However, as I am discussing standard Genetic Programming in this introduction section, so I am considering single tree representation here.

Evaluation

Each individual in the population is assigned a fitness value, which quantifies how well it performs in the problem environment. The fitness value is computed by a problem-dependent fitness function.

We can consider the raw fitness value [72] of the solution. However, it is helpful [72] to have fitness value in the range 0 to 1, where 0 denotes fitness value of possible worst solution and 1 denotes fitness of best solution. For example, in classifier design problem, the raw fitness of a l^{th} individual is the total number of correctly classified training samples (${n_c}^l$) by the classifier. This fitness value can be adjusted to have a fitness value in the range 0 to 1 as follow:

$$fitness, f^l = \frac{{n_c}^l}{N_{tr}} \tag{2.1}$$

where N_{tr} is the total number of training samples.

In my thesis, I use the adjusted fitness that lies in [0,1].

There is a recent paper [28] to improve the GP. In addition to the evaluation of the individual for its fitness value, the authors evaluate individual's relative strengths and weakness and represent these in the form of Binary String Fitness Characterization. Then they use this characterization for both population evaluation and for a pairwise mate selection strategy.

Next Generation

The current GP population is passed through the selection and genetic operations to create a new population as the next generation. There are mainly three genetic operations: Reproduction, Crossover and Mutation. The genetic operators reproduction, crossover and mutation are chosen probabilistically with pre-defined probability values p_r, p_c and p_m respectively. Usually p_c, is high (say, 0.8) and p_r, p_m are low (say, 0.1 each). The algorithm to create new population is given below:

Algorithm:

Step 1. $P' = 0$.

Step 2. A genetic operator is selected probabilistically.

1. If it is the reproduction operator, then an individual is selected from the current population and it is copied into the new population. $P' = P' + 1$.
2. If it is the crossover operator, then two individuals are selected. After crossover these two individuals the resultant two offspring are included in the new population. $P' = P' + 2$.
3. If the selected operator is mutation, then a solution or individual is selected for mutation. This mutated solution is allowed to survive in the new population. $P' = P' + 1$.

Step 3. Continue *Step 2*, until $P' = P$.

Selection

The individuals of the current population are *selected* for the genetic operations to produce a new population. Now, question arises how we will select the individuals. There are many selection methods such as Roulette wheel selection, tournament selection, rank selection, steady state selection, random selection. I present only three selection methods that I have used in my thesis.

Roulette Wheel Selection: It is the fitness-proportional selection method. The chance of an individual to be selected is proportional to its fitness value. The better the individual is, the more chance it possesses to be selected. This is synonymous to a biased roulette wheel where each individual of the population has a roulette wheel slot sized in proportion to its fitness. Fig. 2.4 shows a roulette wheel representation for individual selection. When we give a force to rotate, the wheel rotates and then stops. When it stops, the individual corresponding to the slot that touches the marker becomes the winner. The winner individual or solution is selected for genetic operation. There is high chance that the wheel will stop with a slot of larger area (individual with larger fitness value) touching the marker. If we repeat the rotating of the wheel, then the slots with larger areas will touch marker more often and hence the individuals with higher fitness values will be selected more number of times.

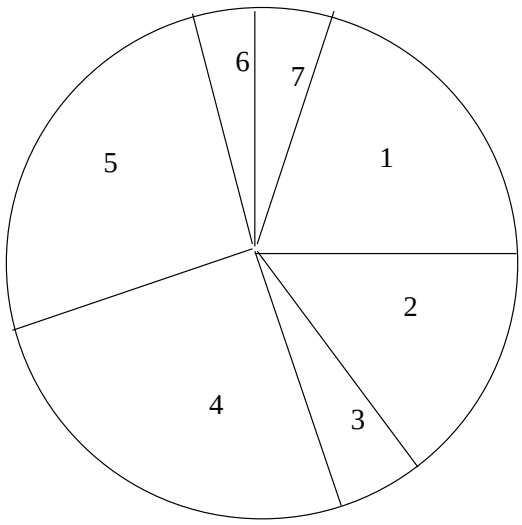


Figure 2.4: Roulette wheel representation for the selection scheme

Tournament Selection: A tournament of τ chromosomes are randomly taken from the population. The best chromosome(s) among the tournament is(are) selected for the genetic operation.

Random Selection: A solution is randomly taken from the population.

Genetic Operations

The three main genetic operations have been discussed below:

Reproduction This copies good solutions of the GP population to the next generation. This allows us to retain few good solutions of the current GP population in the next generation and hence prevents losing of them in the variation process (e.g. crossover, mutation). It replicates the principle of natural selection and survival of the fittest. We may use *roulette wheel* or *rank* selection methods to select good solutions for this copying operation. In our thesis, we have used roulette wheel selection.

This operation has one more advantage. As it does not alter the solutions, we do not require to evaluate the solutions in the new population. Evaluation of solutions consumes almost all of the time of GP evolution. Hence, it saves a considerable amount of computational time.

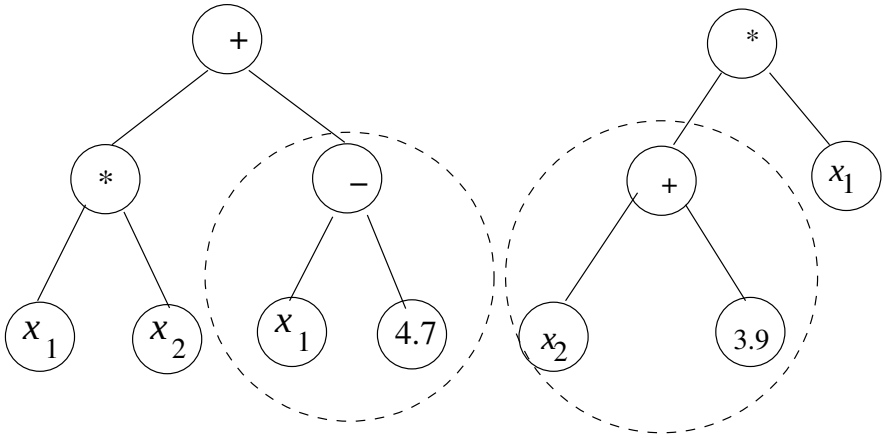


Figure 2.5: Two parents for Crossover Operation

Crossover The good solutions are recombined to produce new solutions. It is expected that the new solutions will drive toward better and desired solutions. This recombination or crossover operation plays a vital role in the evolutionary process. Two individuals are selected for this operation. We use tournament selection method to select individuals for this operation. Then a subtree is randomly selected from each of the selected individuals and these two subtrees are interchanged. Figs. 2.5 and 2.6 show two individuals before and after crossover operation respectively. If the offspring exceed size limit, then the crossover operation is usually abandoned and it is repeated with different randomly chosen subtrees. This is the important genetic operation among all genetic operations.

Mutation Mutation is a process to alter a single solution. After choosing a solution for the mutation, a subtree of the selected individual is randomly selected and is replaced by a new randomly generated subtree. If the mutated tree exceeds the size limit, then it is rejected and the operation is repeated. Mutation maintains diversity and can provide significant variation. Fig. 2.7 illustrates a mutation operation.

Both crossover and mutation are not always constructive. They can be destructive too [7]. That means, the resultant solutions after genetic operations may be worse than parents. To reduce the destructive nature of these genetic operations, special approaches can be considered.

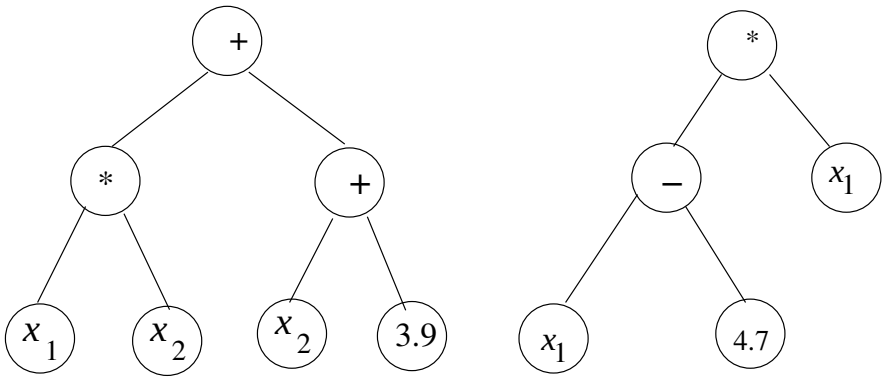


Figure 2.6: Two Offspring after the Crossover Operation

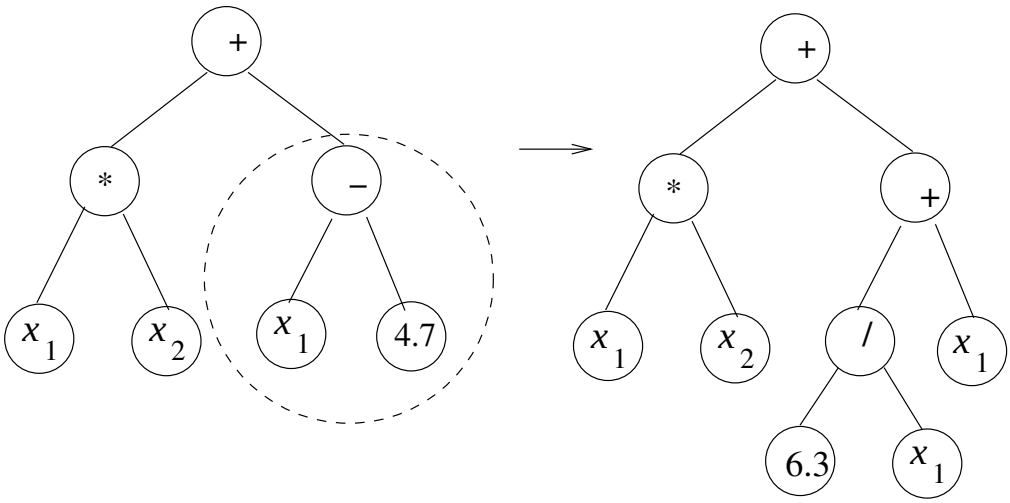


Figure 2.7: A typical Mutation Operation

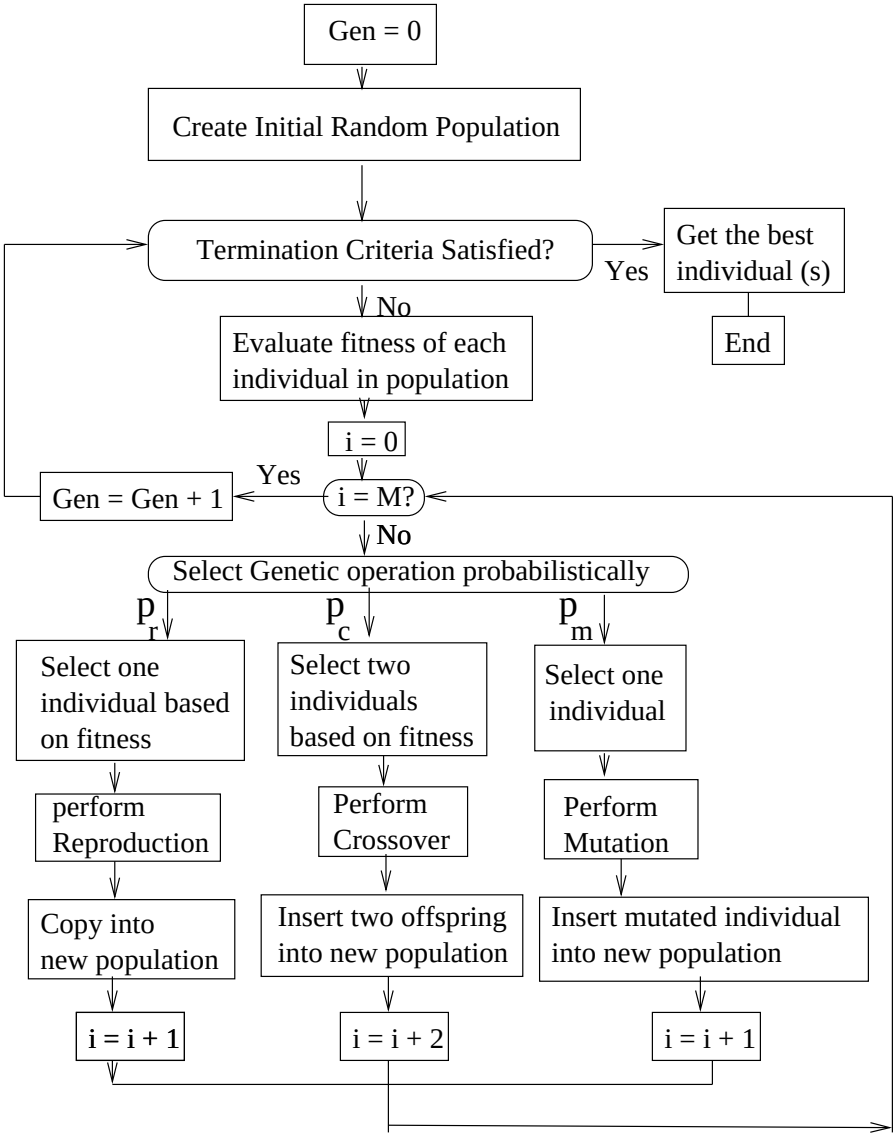


Figure 2.8: Flow Chart of Genetic Programming

The entire procedure is summarized in Fig. 2.8 [72].

2.3 Relevance of EA in Pattern Recognition

Most pattern recognition tasks can be viewed as optimization problems. There are many tools available that can be used for optimization. The suitability of the techniques depend on the objective function, the constraints and the control variables.

If the objective and the constraints are linear functions of the variables, then such a problem is called a *linear programming* problem and it can be solved using methods like *Simplex Method*.

Calculus-based optimization methods are suitable for problems having smooth, continuous, and differentiable (search) surfaces. These methods use gradient information to find the optimal solution. *Hill climbing* is a popular calculus-based method. It climbs the surface in the steepest permissible direction to find local optima. These methods fail to give global optima which is desired. Unfortunately, many real world problems are complex having discontinuous, multi modal surfaces without existence of derivatives.

If the search space is small, then *enumerate search* techniques can be used to find the optimal solution. In this approach, the objective function at every point of the search space is evaluated to determine the optimal one. This is only suitable for very small (search space) problems. Even for moderate size, the computational cost is so high that it is not practically possible to be implemented. To reduce computational time, we may implement techniques like "branch and bound" and "Dynamic Programming". These repeatedly partition the problem into a set of smaller sub-problems to reduce search time. But these too fail when search space increases. Consider popular traveling salesman problem (TSP). In this problem, we attempt to minimize the distance taken by a traveling salesman who has to visit a certain number of cities exactly once and return home. When the number of cities is very small, then the number of possible solutions is relatively small and hence we can easily find the optimal solution using exhaustive search. For example, with 5 cities, all possible routes can be easily checked. However, for a 50-city problem, the number of solutions rises to 10^{60}. This is very vast search space where it is almost impossible to find the optimal solution.

EAs are randomized techniques that are mainly used for optimization problems. Evolutionary algorithms can be used to solve optimization problems where the search surface is discontinuous, uni-/multi- modal and without derivative and the search space is vast. These are typically population based search algorithms. Such randomized search techniques operate on multiple solutions at a time hence can reach (near-to) global optima without getting stuck in a local optima. These can be used to solve complex real-world optimization problems. They can find global or near-to-global optima solutions. Genetic Programming, particularly, can find the model and its structure that are good enough to solve a given problem.

Classifier design is an optimization problem. Unlike many methods, GP doesn't assume the structure of the classifier (model). Rather it attempts to find an appropriate structure of the classifier in addition to appropriate values of involved parameters.

In the following chapters, we have proposed methodologies for different pattern recognition tasks using Genetic Programming. At the beginning of each chapter, we have provided a brief summary of the state of the art in the respective pattern recognition tasks. In addition to other techniques, we have discussed the use of GP for the pattern recognition under consideration.

Chapter 3

Classifier Design using Genetic Programming [A1, A7]

3.1 Introduction

Classifier design is a major pattern recognition task. GP has been used by many authors [102, 1, 114, 39, 35, 16] to design classifiers or to generate rules for *two class* problems. Rauss [102] et al. used GP to evolve binary trees (equations) involving simple arithmetic operators and feature variables for hyper-spectral image classification. A data point is assigned a class if the response for that class is positive and responses for all other classes are negative. Agnelli et al. [1] also applied GP for image classification problems. In addition to simple arithmetic operations, they considered exponential function, conditional function and constants to construct binary trees. The generation of rules using GP for two class problems has been addressed by Stanhope and Daida [114] and Falco et al. [39]. Binary trees consisting of logical functions, comparators, feature variables and constants have been generated to represent possible classification rules. During the construction of binary trees, some restrictions are imposed to enforce a particular structure, so that they can represent logical statements or rules. Dounias et al. [35] implemented GP to generate both crisp and fuzzy rules for classification of medical data. Day and Nandi used GP for speaker verification [29]. In [122], GP is used to detect faults in rotating machinery. GP is used to classify breast masses in mammograms in [89].

Multicategory pattern classification using GP has been attempted by a few researchers [81, 24, 84, 67, 123]. Loveard et al. [81] proposed five methodologies for multi-category

classification problems. Of these five methodologies, they have shown that dynamic range selection method is more suitable for multi-class problems. In this dynamic range selection scheme, they record the real valued output returned by a classifier (tree or program) for a subset of training samples. The range of the recorded values is then segmented into c regions $(R_1, R_2, ...R_c)$ to represent c class boundaries. If the output of the classifier for a pattern $\mathbf{x}$ falls in the region R_i, then the i_{th} class is assigned to $\mathbf{x}$. Once the segmentation of the output range has been performed, the remaining training samples can then be used to determine the fitness of an individual (or classifier). Chien et al. [24] used GP to generate discriminant functions using arithmetic operations with fuzzy attributes for a classification problem. In [84] Mendes et al. used GP to evolve a population of fuzzy rule sets and a simple evolutionary algorithm to evolve the membership function definitions. These two populations are allowed to co-evolve so that both rule sets and membership functions can adapt to each other. For a c-class problem, the system is run c times. Kishore et al. [67] proposed an interesting method which considers a c class problem as a set of c two-class problems. When a GP classifier expression (GPCE) is designed for a particular class, that class is viewed as the desired class and the remaining classes taken together are treated as a single undesired class. So, with c *GP runs*, all GPCEs are evolved and can be used together to get the final classifier for the c-class problem. They have experimented with different function sets and incremental learning. For conflict resolution (where a pattern is classified by more than one GPCE) each GPCE is assigned a "strength of association" (SA). In case of a conflicting situation, a pattern is assigned the class of the GPCE having the largest SA. They have also used heuristic rules to further reduce the misclassification. Zhang and Nandi has used GP for muli-class classification problems in roller bearing fault detection [123].

Lim et al. presented an excellent comparison of thirty-three classification algorithms in [80]. They used a large number of benchmark data sets for comparison. None of these 33 algorithms use GP. The set of algorithms includes twenty-two decision tree/rule based algorithms, nine statistical algorithms and two neural network based algorithms. We shall use the results reported in [80] for comparison of our results.

We have proposed a method to design classifiers for a c-class pattern classification problem using a *single* run of GP. For a c class problem, a multi-tree classifier consisting of c trees is evolved where each tree represents a classifier for a particular class. The

performance of a multi-tree classifier depends on the performance of its constituent trees. A new concept of *unfitness* of a tree is exploited in order to improve genetic evolution. Weak trees having poor performance are given more chance to participate in the genetic operations so that they get more chance to improve themselves.

In this context, a new mutation operation called non-destructive directed point mutation is proposed which reduces the destructive nature of mutation operation. During crossover, not only is swapping of subtrees among partners performed but also swapping of trees is allowed. As a result, more fit trees may replace the corresponding less fit trees in the classifier. Multiple classifiers from the terminal population are then combined together by a suitable OR-ing operation in order to further improve the classification result. A conflict situation occurs when a pattern $\mathbf{x}$ is recognized by more than one tree to their respective classes. Each tree of the classifier is assigned a weight to help conflict resolution. In addition, heuristic rules, that characterize typical situations when the classifier fails to make unambiguous decisions, are used to further enhance the classifier performance. For a reasonably large number of training points, if the classifier fails to make unambiguous decision, and the response of the classifier for each such data point is the same, then it is likely that those training points come from some particular area of the input space. Heuristic rules exploit this information and try to resolve situations when more than one tree of the classifier produce positive responses. Combination of all these results in a good classifier.

3.2 Proposed Multi-tree GP based classifier

A classifier D is a mapping, $D : \mathcal{R}^p \rightarrow N_{hc}$, where $\mathcal{R}^p$ is the p-dimensional real space and N_{hc} is the set of label vectors for a c- class problem and is defined as $N_{hc} = \{\mathbf{y} \in \mathcal{R}^c : y_i \in \{0, 1\} \forall i, \sum_{i=1}^{c} y_i = 1\}$. For any vector $\mathbf{x} \in \mathcal{R}^p$, D($\mathbf{x}$) is a vector in c-dimension with only one component as 1 and all others as 0. In other words, a classifier is a function which takes a feature vector in p dimension as input and assigns a class label to it.

In this paper our objective is to find a D using GP. We shall use a multi-tree concept for designing classifiers. The beauty of using this concept is that we can get a classifier for the multi-class problem in a single run of GP.

Given a set of training data X = $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\} \subset \mathcal{R}^p$ and its associated set of label vectors Y = $\{\mathbf{y}_1,\mathbf{y}_2,...,\mathbf{y}_N\} \subset N_{hc}$, our objective is to find a "good" D using GP.

For a two class problem, a possible classifier or an individual is generally represented by a *single* tree(T). For a pattern $\mathbf{x}$,

if $T(\mathbf{x}) \geq 0$, $\mathbf{x} \in$ *class* 1

else $\mathbf{x} \in$ *class* 2.

The single tree representation of the classifier is sufficient for a two-class problem. This scheme can be extended to a multi-category classification problem. In our design every chromosome or individual will have a tree for every class. So the l_{th} chromosome will have c trees, and these will be denoted by ${T_k}^l, k = 1, 2, ..., c$. If the identity of the chromosome is not important, then for clarity we will ignore the superscript and use only the class index, i.e., the subscript. So a possible solution or an individual for the GP is represented by c trees $(T_1, T_2, ..., T_c)$. For a pattern $\mathbf{x}$,

if $T_i(\mathbf{x}) \geq 0$ *and* $T_j(\mathbf{x}) < 0$ for all $j \neq i$, $i, j \in \{1, 2, ..., c\}$ then $\mathbf{x} \in$ *class* i.

If more than one tree show positive responses for the pattern $\mathbf{x}$ then we require additional methodologies for assigning a class to $\mathbf{x}$. The steps followed to achieve our goal are summarized in the following sections.

3.2.1 Initialization

Each of the c trees for each individual is initialized randomly using the function set $\mathcal{F}$ which consists of arithmetic functions and the terminal set $\mathcal{S}$ containing feature variables and constants. The function set $\mathcal{F}$ and terminal set $\mathcal{S}$ used here are:

$\mathcal{F}$ = {+,-,*,/} and $\mathcal{S}$ = {feature variables, R}, where R contains randomly generated constants in [0.0,10.0]. We have initialized trees using the ramped half-and-half method [72].

3.2.2 Training and Fitness measure

The GP is trained with a set of N training samples, $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$. Instead of training the GP with all training samples at a time, it is accomplished in a step-wise

manner increasing the number of training samples in steps. The step-wise increment of training samples is accomplished by a preset number of generations M_s. The step-wise learning can reduce the computational overhead significantly [41], [32] and also can improve the performance [67]. If we use s_1 steps, then each step size will be $\frac{M_s}{s_1}$ generations. And the incremental change in the size of training subset in each step will be $N_{sub} = \frac{N}{s_1}$. Let X_s be the set of the training samples at step s, $|X_s| = N_s$. At the first step $|X_s| = \frac{N}{s_1}$ and at the last step, $X_s = X_{tr}$. After step-wise learning GP is continued with all N training samples up to the maximum number of generation M, where $M \geq M_s$. Note that, we have not used any validation set as many GP approaches have not used it [67, 16, 24, 35, 39, 41, 50].

While training, the response of a tree T_i for a pattern $\mathbf{x}$ is expected to be as follows:

$T_i(\mathbf{x}) \geq 0 \; if \; \mathbf{x} \in class \; i$

$T_i(\mathbf{x}) < 0 \; if \; \mathbf{x} \notin class \; i$

In other words, a *classifier* with c trees is said to correctly classify a sample $\mathbf{x}$, if and only if all of its trees correctly classify that sample. We emphasize that if a training sample $\mathbf{x} \in R^p$ is from class k, then we say that *tree* T_k correctly classifies $\mathbf{x}$, if $T_k(\mathbf{x}) \geq 0$. On the other hand, the tree $T_{j,j\neq k}$ is said to correctly classify $\mathbf{x}$, if $T_j(\mathbf{x}) < 0$. For each correct classification of a training sample by a classifier, its raw fitness is increased by 1.

At the initial stage of learning (evolution), if some but not all of the trees of a classifier are able to do a good job, then that should not necessarily be considered a bad classifier. Because by giving more chance to unfit trees to take part in subsequent genetic operations, unfit trees may be made to converge to more fit trees and hence may result in a better overall classifier. We will take into account this factor while computing the fitness value. Let g_i trees correctly classify a training sample $\mathbf{x}_i$. Then we increase the raw fitness by $\frac{g_i}{c}$ irrespective of the class label of $\mathbf{x}_i$. In other words, we give equal importance to all trees. So if all trees correctly classify a data point, then the raw fitness is increased by 1 as mentioned above. This partial increment of raw fitness function by $\frac{g_i}{c}$ (for $g_i < c$) is considered only during the stepwise learning, i.e., only up to M_s generations. This helps to refine the initial population. After completion of the step-wise learning, the fitness function considers only the correctly classified samples. We again emphasize the definition of correct classification by a *tree*. If $\mathbf{x}_i \in R^p$ is from

class k, then for a chromosome, if $T_k(\mathbf{x}_i) \geq 0$, then T_k's classification is correct and also if $T_{j\neq k}(\mathbf{x}_i) < 0$, then T_j also classifies $\mathbf{x}_i$ correctly.

Let g_i be the number of trees that correctly classify $\mathbf{x}_i \in X_s$. So the fitness function f_s at step s of the step-wise learning task is defined as,

$$f_s = \frac{\sum_{\mathbf{x}_i \in X_s} \frac{g_i}{c}}{N_s} \tag{3.1}$$

In (1), $N_s = |X_s|$ = Number of training samples used at step s.

After step-wise learning we consider the samples which are correctly classified by all trees($g_i = c$) of the classifier. So, the fitness function after step-wise learning is defined as,

$$f = \frac{Number\ of\ training\ samples\ correctly\ classified\ (for\ which\ g_i = c)}{N}. \tag{3.2}$$

Thus, during initial evolution, individuals with potential (partially good) trees are given some extra importance in the fitness calculation. Note that, fitness function (1) or (2) can be used for selection of individuals. *Algorithm fitness* shows the procedure for evaluating the fitness of an individual during the step-wise learning process.

Algorithm *fitness*

BEGIN

$f_{raw} = 0$;
  for all i = 1,2,...,N_s
    for all j = 1,2,...,c
      count = 0;
      if(($\mathbf{x}_i \in class\ j$) AND ($T_j(\mathbf{x}_i) \geq 0$)) count = count + 1;
      end if /* if T_j classifies $\mathbf{x}_i$ correctly */
      if(($\mathbf{x}_i \notin class\ j$) AND ($T_j(\mathbf{x}_i) < 0$)) count = count + 1;
      end if /* if T_j classifies $\mathbf{x}_i$ correctly */
      /* For other cases the trees make wrong decisions and hence *count* is not incremented */
    end for

$f_{raw} = f_{raw} + \frac{count}{c}$;
end for
$f_s = \frac{f_{raw}}{N_s}$;
END

3.2.3 Unfitness of trees

When all trees are able to classify a pattern correctly then the said classifier will recognize the pattern correctly. On the other hand, if there are some unfit trees in the classifier, they should be given more chance to evolve through genetic operations in order to improve their performance. In addition to the fitness functions, we need another *unfitness* function, to *select a tree* after an individual is selected (using (1) or (2)) for genetic operations.

So after an individual is selected for genetic operation, we compute the degree of unfitness of its each constituent tree $T_i, i = 1, 2, ..., c$. The total number of training samples for which T_i is *unable* to classify correctly is counted. Let q_i be the total number of training samples not correctly classified by T_i. To compute $q_i, i = 1, 2, \cdots, c$, we proceed as follow: If a training sample $\mathbf{x}$ is from class k and $T_k(\mathbf{x}) < 0$, then q_k is increased by 1 and also if $T_j(\mathbf{x}) > 0, j \neq k$, then q_j is increased by 1.

If $q_j > q_k$, then T_j is more unfit then T_k.

$$Let\ p_i = \frac{q_i}{\sum_{j=1}^{c} q_j}. \tag{3.3}$$

This p_i is used as the probability of selection of the i_{th} tree by the Roulette wheel selection as an unfit tree for *genetic operations* (crossover and mutation). In this way, the unfit trees are given more chance to take part in the genetic operations to rectify themselves.

3.2.4 A modified Crossover operation

Crossover plays a vital role in GP for evolution. To select *trees* (within a chromosome) for crossover we use p_i as the probability of selection and this gives more preference

to unfit trees for the crossover operation. We use the tournament selection scheme for selecting chromosomes for the crossover operation. The fitness function defined in (1) or (2) is used for the selection of a pair of chromosomes. Let the selected chromosomes be C_1 and C_2. Each of C_1 and C_2 has c trees ${T_i}^1, {T_i}^2, i = 1, 2, \cdots, c$.

Now we select a tree ${T_k}^1$ from chromosome C_1 using Roulette wheel selection based on the probability ${p_i}^1, i = 1, 2, ..., c$. ${p_i}^1$ is computed using Eq. (3.3). We now randomly select a node from each of ${T_k}^1$ and ${T_k}^2$ where the probability of selecting a function node type is q_f and that of a terminal node type is p_t , $p_f + p_t = 1$. After selecting one node from each of the two trees ${T_k}^1$ and ${T_k}^2$, we swap the subtrees rooted at the selected nodes. In addition to this, we also swap the trees ${T_j}^1$ and ${T_j}^2$ for all $j = k + 1, ..., c$. That means we swap trees ${T_j}^1$ of chromosome C_1 with ${T_j}^2$ of chromosome C_2 for all $j > k$.

This crossover operation has *three* interesting aspects. First, we swap subtrees between classifiers from the same class. The motivation is that, good features from one classifier (for a particular class) may get combined with good features of another classifier of the *same* class. Note that, a subtree of the good classifier for class k, may not be very useful for a class j, $j \neq k$ of a different classifier. The second interesting aspect of this crossover operation is that it also exchanges classifier trees as a whole between two chromosomes. The third point is that, by selecting trees for the crossover operation according to their *unfitness*, the chance of unwanted disruption of already fit trees is reduced and the chance of evolution of weak trees is increased. Thus, we not only try to change weak trees but also try to protect good trees from the destructive nature of the crossover operation.

Figure 1 illustrates the crossover operation. Figures 1(a) and 1(b) are two chromosomes selected for crossover. Suppose ${T_2}^1$ of C_1 is selected using Eq.(3.3). A node of ${T_2}^1$ and a node of ${T_2}^2$ (shown by the dotted circles) are then randomly selected (the probability of selecting a function node is p_f and that of a terminal node is p_t) for crossover operation. Now the subtrees rooted at the selected nodes are swapped. And also ${T_3}^1, \cdots, {T_c}^1$ are swapped respectively with ${T_3}^2, \cdots, {T_c}^2$. The resultant chromosomes obtained after crossover are shown in panels (c) and (d) in Fig. 1.

A schematic description of the crossover operation is given next.

Algorithm *Crossover*

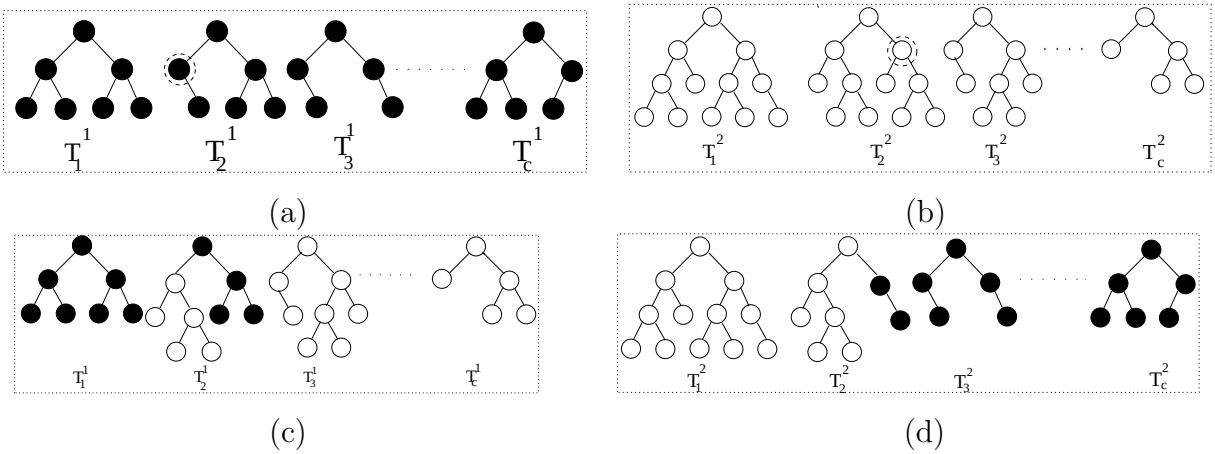


Figure 3.1: (a) and (b) Chromosomes C1 and C2 before Crossover operation; (c) and (d) Chromosomes C1 and C2 after Crossover operation

Step 1. Randomly select τ (tournament size) individuals from the population for tournament selection.

Step 2. Select the best two individuals (C_1 *and* C_2 , say) of the tournament for the crossover operation.

Step 3. Compute ${p_i}^1, i = 1, 2, \cdots, c$ of C_1 using Eq. (3.3).

Step 4. Select a tree ${T_k}^1$ of C_1, by the Roulette-wheel selection using the unfitness probability, ${p_i}^1$.

Step 5. Choose a node type - a function (internal) node type is chosen with probability p_f and a terminal (leaf) node type is selected with probability p_t. Randomly select a node of the chosen type from each of the trees ${T_k}^1$ and ${T_k}^2$.

Step 6. Swap the subtrees rooted at the selected nodes of ${T_k}^1$ and ${T_k}^2$.

Step 7. Swap ${T_j}^1$ with ${T_j}^2$, for all $j = k + 1, ..., c$.

3.2.5 A modified point mutation operation

The conventional GP mutation mostly is a destructive process, because it swaps a subtree for a randomly generated tree. For this reason, we have utilized point mutation with some additional precautions. It is just like the fine-tuning of a solution.

In case of point mutation, a node is randomly picked. If it is a function (terminal) node then it is replaced by a randomly chosen new function (terminal) node (having the same arity). Thus, it causes a very small change. To make a considerable change, it is repeated a number of times. Although, usually it is not expected to severely affect the tree, sometimes its effect could be significant. So we introduce a kind of directed mutation, which always accepts mutations that improve the solution, but also occasionally, with some probability, accepts a mutation that does not improve the solution. This involves comparison of the fitness of the mutated tree with that of the original tree. To reduce computation time, we evaluate fitness f_m of the mutated tree $T_{i(m)}$ and fitness f_o of original tree T_i using only 50% of the training samples of the i^{th} class. If f_m equals f_o, then we use the remaining 50% of training samples of the i_{th} class to find f_m. If the mutated tree is equal or more fit than the original tree, then the mutated tree is retained. Otherwise, it is ignored with probability p_{th}. We have taken $p_{th} = 0.5$.

Mutation causes random variation and unfit trees need such variation more than fit ones. Consequently, after selecting an individual for mutation, the more unfit trees are given more chances to mutate. In other words, more fit trees are given more opportunities to protect themselves from the destructive nature of mutation. To achieve this we proceed as follow:

We randomly select a chromosome and then select a tree T_i from the chromosome. The tree is selected using Roulette wheel selection criterion with p_i in (3.3). This is tree selection with a probability proportional to the unfitness of the tree. And then we select m% of nodes from the selected tree for point mutation. A function node is selected with probability p_{mf} and a terminal node is selected with probability p_{mt}, $p_{mf} + p_{mt} = 1$.

After mutation, a decision is made as to whether the mutated tree will be retained or ignored. This procedure is repeated c times for the selected individual. The basic steps of the point mutation are schematized in *Algorithm Mutation*.

Algorithm *Mutation*

Step 1. Randomly pick an individual(C) for mutation (from the old population).

Step 2. Compute unfitness probability $p_i, i = 1, 2, \cdots, c$ of C using (3.3).

Step 3. Select a tree (T_i) of C with Roulette wheel selection using $p_i, i = 1, 2, ..c$.

Step 4. Randomly select a node from T_i with probability of selecting a function node p_{mf} and that of a terminal node p_{mt} for point mutation.

Step 5. If the selected node is a function node, replace it with a randomly chosen function node (having the same arity). Otherwise, replace it with a randomly chosen terminal node.

Step 6. Repeat *Step 4* and *Step 5* for m% of the total nodes in T_i.

Step 7. Evaluate the fitness f_m of the mutated tree and fitness f_o of the original tree using 50% of training samples of the i_{th} class.

Step 8. If $f_m = f_o$ then evaluate again f_m and f_o using the remaining 50% of training samples of the i_{th} class.

Step 9. If $f_m \geq f_o$ then accept the mutated tree, else retain it with probability 0.5.

Step 10. Repeat *Step 3* through *Step 9* c times.

3.2.6 Termination of GP

The GP is terminated when all N training samples are classified correctly by a classifier (an individual of the GP population) or a predefined number, M, of generations are completed. If all of the N training samples are correctly classified by the GP, the best individual (BCF) of the population is the required optimum classifier, CF. Otherwise, when the GP is terminated after completion of M generations, the best individual (BCF) of the population is passed through an OR-ing operation to obtain the optimal classifier CF.

The best individual (BCF) is selected as follows,

Let *hit(I)* = Number of correctly classified training samples by the classifier I. The individual which scores the maximum *hit* is considered the best classifier BCF. If more than one individual score the same *hit* (maximum), then the individual which has the smallest number of nodes is chosen as the best individual BCF.

3.2.7 Improving performance of classifier with OR-ing

It is possible that the terminal GP population contains two chromosomes $C_1 = \{T_1^1, T_2^1, \cdots, T_k^1, \cdots, T_c^1\}$ and $C_2 = \{T_1^2, T_2^2, \cdots, T_k^2, \cdots, T_c^2\}$ such that ${T_k}^1$ is good for a particular segment of the feature space; while ${T_k}^2$ models well another segment of the feature space. The overall performance, in terms of misclassification, of C_1 and C_2 could be comparable or different. In this case, combining ${T_k}^1$ and ${T_k}^2$ by OR-ing may result in a much better classification tree. This is the principle behind this OR-ing operation. The best individual BCF of the GP run, which is unable to classify correctly all training samples is further strengthened by this operation. BCF is OR-ed in a consistent fashion with every individual of the terminal population. From this set of (OR-ed) pairs, the best performing (in terms of number of misclassification) pair is taken as the optimum classifier CF. However, OR-ing is to be done carefully. Next we explain how the OR-ing is done. We introduce a set of indicator variables b_i, $i = 1, 2, .., c$ for the best chromosome BCF as

$b_i = 1$ if $T_i^{BCF}(\mathbf{x}) \geq 0$
$= 0$, otherwise, where $i \in \{1, 2, ..., c\}$.

In other words, if the i_{th} tree of BCF shows a positive response for a sample $\mathbf{x}$, then b_i for that sample is 1; otherwise, it is 0. Now to combine the BCF with any other chromosome C_l, we define another set of indicators ${d_i}^l$, $i = 1, 2, .., c$ as

${d_i}^l = 1$ if $T_i^l(\mathbf{x}) \geq 0$
$= 0$, otherwise.

For notational clarity we consider a set of indicators ${a_i}^l$ to represent the combined classifier CF_l using BCF and C_l:

${a_1}^l = (b_1 \wedge \bar{b}_2 \wedge \bar{b}_3 .... \wedge \bar{b}_c) \quad \vee \quad ({d_1}^l \wedge \bar{d}_2^l \wedge \bar{d}_3^l .... \wedge \bar{d}_c^l) \quad \vee \quad (b_1 \wedge {d_1}^l)$
${a_2}^l = (\bar{b}_1 \wedge b_2 \wedge \bar{b}_3 .... \wedge \bar{b}_c) \quad \vee \quad (\bar{d}_1 \wedge {d_2}^l \wedge \bar{d}_3 .... \wedge \bar{d}_c^l) \quad \vee \quad (b_2 \wedge {d_2}^l)$

$\vdots$

${a_i}^l = (\bar{b}_1 \wedge \bar{b}_2 ... \wedge \bar{b}_{i-1} \wedge b_i \wedge \bar{b}_{i+1} ... \wedge \bar{b}_c) \quad \vee \quad (\bar{d}_1^l \wedge \bar{d}_2^l ... \wedge \bar{d}_{i-1}^l \wedge {d_i}^l \wedge \bar{d}_{i+1}^l ... \wedge \bar{d}_c^l) \quad \vee \quad (b_i \wedge {d_i}^l)$
$i = 1, 2, \cdots, c$

The combined classifier CF_l is, thus, defined by $({a_1}^l, {a_2}^l, \cdots, {a_c}^l)$. Given a data point $\mathbf{x}$, CF_l assigns a class label to $\mathbf{x}$ as follows:

If ${a_i}^l = 1$ and ${a_j}^l = 0$, for all j $\neq$ i and i,j $\in$ {1,2,...,c} then $\mathbf{x} \in$ *class i*.

In this way, if the population has N_c chromosomes then we will get CF_l, $l = 1, 2, ...,$ $(N_c - 1)$ combined classifiers. The combined classifier which correctly classifies the maximum number of training samples, is taken as the resultant classifier CF.

3.2.8 Conflict Resolution

The resultant classifier CF, thus obtained, is now ready for validation with the test data set. For a test data point $\mathbf{x}$, to make an unambiguous decision, we need $T_k(\mathbf{x}) \geq 0$ and $T_{j \neq k}(\mathbf{x}) < 0$ (or $a_k = 1$ and $a_{j \neq k} = 0$), $j = 1, 2, \cdots, c$ for some k, $k \in \{1, 2, \cdots, c\}$.

But it can happen that more than one tree of CF show positive responses. In this case we face a conflicting situation. To resolve it, we use a set of heuristic rules followed by a weighting scheme. The heuristic rule based scheme described next is a slightly modified version of the rule proposed by Kishore et al. [67].

Extraction of the rules

The classifier CF is used to classify all N training samples. The objective of the heuristic rules is to identify the *typical* situations when the classifier cannot make unambiguous decisions and exploit that information to make decisions. Let n_j be the number of unclassified training samples of class j for which either two or more trees or none of the trees of CF show positive response. For each such unclassified sample $\mathbf{z}_{ij}, i = 1, 2, \cdots, n_j$, we compute a *response* vector $\mathbf{v}_{ij}$ in c dimension as follows:

$v_{ijk} = 1,\ if\ T_k(\mathbf{z}_{ij}) \geq 0$
$\quad = 0$, otherwise; $k = 1, 2, ..., c$.

As an illustration, consider a 4-class problem, where a training sample $\mathbf{z}_{11}$ from class 1 is not classified by CF, because more than one tree show positive responses. Suppose T_1, T_3 *and* T_4 show positive responses and T_2 shows a negative response for the training example . Then, the corresponding response vector $\mathbf{v}_{11}$ will be $(1\ 0\ 1\ 1)^T$. Similarly, if a training sample $\mathbf{z}_{21}$ of class 1 is unclassified because all trees show negative responses, i.e., $T_i(\mathbf{z}_{21}) < 0\ \forall i = 1, 2, \cdots, c$, then the response vector $\mathbf{v}_{21}$ is $(0\ 0\ 0\ 0)^T$.

If there are c classes then there can be at most $L = (2^c - c)$ possible distinct response vectors when the classifier fails to make a decision. Let these vectors be $V = \{\mathbf{v}_i, i = 1, 2, \cdots, L\}$. Clearly $\mathbf{v}_{ij} \in V$. The classifier may produce a particular vector $\mathbf{v}_l$ for *several* data points from class j. If this is so, then $\mathbf{v}_m$ represents a *typical conflicting or ambiguous situation* for class j when the classifier cannot make a decision. Since several training data points generated the same response vector, $\mathbf{v}_m$, it is likely that those training points came from some particular area of the input space and the classifier failed to model that area correctly. These points are expected (but not necessarily) to form a cluster in the input space. Therefore, for a *test* data point $\mathbf{x}$, if CF fails to classify unambiguously and the response vector corresponding to $\mathbf{x}$ matches $\mathbf{v}_m$, then it may be reasonable to classify $\mathbf{x}$ to the j_{th} class. We call $\mathbf{v}_m$ a rule for conflict resolution. We emphasis that unless $\mathbf{v}_m$ is supported by a sufficient number of instances, we should not use $\mathbf{v}_m$ as a rule for conflict resolution for class j. In other words, the number of cases from class j for which the response vector $\mathbf{v}_m$ is produced should be large. Suppose, of the n_j unclassified samples, $\mathbf{v}_m$ is obtained for r_{mj} times. We may use a threshold h on r_{mj} to decide on the acceptance of $\mathbf{v}_m$ as a rule. Using such a threshold, $\mathbf{v}_m$ may also become a rule for a different class. But, a particular $\mathbf{v}_m$ should be used as a rule for only one class.

So for a $\mathbf{v}_m \in V$, we find the class k for which $r_{mk}(>\text{h})$ is the maximum. Then $\mathbf{v}_m$ is used to represent a rule only for class k. The difference of our rules from those of Kishore et al. [67] is that authors in [67] used a threshold on the *percentage* of training samples to pick up the rules, while we threshold on the number of samples. The reason for such a choice, as explained earlier, is that if any $\mathbf{v}_m$ is supported by a reasonable number of points h, it should be considered a rule and thus h is independent of the size of X.

The heuristic rules are derived from the behavior of the classifier when it fails. So unless there are sufficient number of failure cases, the rules may not be useful. Based on a few experiments with the training data we decided to use $h = 20$. A better strategy would be to use a validation set to decide on the value of h.

If the classifier fails to classify a pattern unambiguously and that pattern produces a response vector which matches a heuristic rule, then the class corresponding to that rule is assigned to that pattern. If there is no heuristic rule for that response vector, then the weight based scheme discussed next is used to assign a class to the pattern.

If there is no heuristic rule at all, then we directly use the weight based scheme.

The weighting scheme

The classifier CF is used to classify all training samples. Now we compute a matrix $\mathcal{A} = [a_{ij}]$ of dimension $c \times c$,where a_{ij} is the total number of cases such that the instance is from class j, but the i_{th} tree of classifier CF shows a positive response, i.e., $T_i(\mathbf{x}) \geq 0, \mathbf{x} \in class\ j$. In other words, a_{ij} gives the number of data points from class j for which the tree T_i gives a positive response. Consequently, diagonal elements of $\mathcal{A}$ represent the number of cases correctly classified and the off-diagonal elements represent the number of misclassified cases by the *trees*. For an ideal classifier, the sum of all diagonal elements will be N and the sum of all off-diagonal elements will be zero.

Now $A_i = \sum_{j=1_{j\neq i}}^{c} a_{ij}$ gives the number of data points from all classes except class i for which $T_i(\mathbf{x}) > 0$. So, A_i gives the number of data points from other classes which are wrongly classified to class i by tree T_i. A_i represents the false positive cases. The classifier CF will make mistake for these A_i cases. If $A_k = Max_i\ A_i$, then the tree T_k can be held most (or at least, significantly) responsible for the misclassification reported by the classifier CF.

For a row i in A, the difference (m_i) between the total number (N_i) of patterns of class i and the diagonal element a_{ii} indicates the number of cases from class i for which tree T_i did not predict the result correctly. In other words, m_i is related to *how* tree T_i failed to represent its own class. m_i represents the false negative cases. Let $m_g = Max_i\ m_i$, then the g_{th} tree is also a determinant of the poor performance of classifier, CF.

Therefore, for a tree T_i, we define a weight w_i that provides the relative importance of the tree in making a correct prediction as follows:

For the i_{th} tree T_i of the CF, we compute

$$w_i = 1 - w'_i\ where,\ w'_i = \frac{A_i + m_i}{Total\ number\ of\ training\ samples(N) = \sum_{j=1}^{c} N_j}. \quad (3.4)$$

$w_i \in [0, 1]$. These weights $\{w_i\}$ can be used for resolution of conflict, when needed.

For a test data point $\mathbf{x}$, if we find that CF identifies a conflict between classes g and k (in other words, both T_g and T_k give positive responses), then $\mathbf{x}$ is assigned to class k, if $w_k > w_g$; otherwise, $\mathbf{x}$ is assigned to class g. Note that, this weight scheme is

different from the one used by Kishore et al.[67].

We recommend using the heuristic rules *first* because each heuristic rule represents a typical mistake, a scenario represented by an adequate number of data points. Moreover, as pointed out by Kishore et al. [67], the weight based scheme has some limitations. It cannot assign a class label, if none of the $T_i(\mathbf{x}) \geq 0, i = 1, 2, \cdots, c$. Also in some cases, it may happen that for $\mathbf{x} \in$ class k, $T_k(\mathbf{x}) \geq 0$ and for another tree $T_i(\mathbf{x}) \geq 0$, $i \neq k$ with $w_i > w_k$. Then $\mathbf{x}$ will be misclassified to class i by the weight scheme. So we suggest to use the heuristic rules first. If the heuristic rule cannot resolve the conflict, the weight scheme should be used.

It is possible that neither the weight based scheme nor the heuristic rule is able to assign a class label to a test data point. This must not be viewed as a disadvantage, but rather a distinct advantage. The classifier is able to say " don't know " when faced with very unfamiliar examples. This is better than giving a wrong class. Moreover, if there are too many such ' don't know' cases, then these indicate that either the training set used was not a sufficient representative of the population that generated the data, or the identified classifier is a poor one. So a redesign of the classifier is required.

Note that, the OR-ing operation, the heuristic rule generation and computation of the weights are done only once after the GP terminates. Consequently, the additional computation involved for the post-processing, as we shall see later, is much less than the time required for the GP to evolve. The complete multi-tree classifier algorithm is now schematized in *Algorithm Classifier* GP_{mtc}.

Algorithm *Classifier* GP_{mtc}

BEGIN

Let gen $= 0$, $hit_g = 0$, $hit_r = 0$ $/* hit_g =$ highest hit of that generation and $*/$
Initialize population of size P for GP $/* hit_r =$highest hit till that generation $*/$
while (gen $<$ M AND $hit_r < N$) /* while the termination condition is not satisfied */
if (gen $<$ M_s)
Evaluate fitness of each individual using Eq.(3.1)
else
Evaluate fitness of each individual using Eq.(3.2).
end if

Find the best individual (I_{gen}),
$hit_g = hit(I_{gen})$.
$if(hit_g > hit_r)$ $hit_r = hit_g$ and BCF = I_{gen}.
end if
Perform *Breeding* /* perform all genetic operations */
gen = gen + 1. /* go to the next generation */
end while
if (hit_r = N) CF = BCF /* if all training samples are classified correctly */
else do OR-ing operation.
end if
Extract Heuristic rules and Compute weights $\{w_i\}$ using Eq.(3.4).

END

Algorithm *Breeding* [124]

BEGIN

Let $P_n = 0$ /* P_n = Population size of the next population */
while($P_n \neq P$) /* P= Population size */
Select one of the operators from reproduction, crossover and mutation
with a probability p_r, p_c and $p_{\mu c}$ respectively.
If (operator = reproduction)
select an individual using fitness-proportion selection method
and perform the reproduction operation. Now $P_n = P_n + 1$.
end if
If (operator = crossover)
select two individuals using tournament selection method
and perform *crossover* . Now $P_n = P_n + 2$.
end if
/* Note that, if P_n becomes $P + 1$, then reject the second offspring
after crossover operation. */
If (operator = mutation)
select an individual randomly and
perform *mutation* . Now $P_n = P_n + 1$.
end if

end while

END

3.2.9 Validation of classifier

After obtaining the classifier CF, it is necessary to validate it against test data sets. Each test data point $\mathbf{x}$ is classified by the classifier CF. For a pattern $\mathbf{x}$, there are three possible cases:

Case 1: For *only one* tree $T_i(\mathbf{x}) \geq 0$ and for all other trees $T_j(\mathbf{x}) < 0$,$j \neq i$.
Or in case of the combined classifier after OR-ing operation, *only one* $a_i = 1$ and all other $a_j = 0$, $j \neq i$. In this case, class i is assigned to the pattern $\mathbf{x}$. This is the simplest case that does not require further operations.

Case 2: For *more than one* tree $T_i(\mathbf{x}) \geq 0$ or *more than one* $a_i s$ are 1. Each tree for which $T_i(\mathbf{x}) \geq 0$ or $a_i = 1$ is called a competitive tree. To resolve this conflict or to assign only one class to the pattern $\mathbf{x}$, we *first* try to use the *heuristic rules*. If it cannot assign a class then we apply the weight based scheme. Also if there is no heuristic rule available, then we directly use the weight based scheme. As an illustration, in a 4-class problem, let $T_1(\mathbf{x}) < 0$, $T_2(\mathbf{x}) < 0$, $T_3(\mathbf{x}) \geq 0$ and $T_4(\mathbf{x}) \geq 0$. In this case T_3 and T_4 are the competitive trees and hence the vector $\mathbf{v}$ will be $(0\ 0\ 1\ 1)^T$. If there exists a heuristic rule $(0\ 0\ 1\ 1)^T$ which assigns class j, j $\in$ {1,2,3,4}, then that class is assigned to the pattern $\mathbf{x}$. Otherwise, the weight based method will be tried. If $w_3 \geq w_4$ then class 3 will be assigned to $\mathbf{x}$, else class 4 will be assigned. Note that, weights and rules will be computed only when the classifier CF makes mistakes on the training data set. So if CF makes 100% correct classification of the training data then weights and rules will not be available.

Case 3: For all trees $T_i(\mathbf{x}) < 0$ or all $a_i = 0$.

In this case *none* of the trees can recognize the pattern $\mathbf{x}$. So only the heuristic rules may resolve this situation. Here all components of the vector $\mathbf{v}$ are zero. If there is a Null (vector whose all components are zero) heuristic rule that maps to class j, then class j is assigned to the pattern $\mathbf{x}$. Otherwise, no class will be assigned to $\mathbf{x}$ and it becomes a "don't know" case.

3.3 Experimental Results

We have used five data sets for training and validating our methodology. These are all real data sets, named, IRIS [4], WBC [14], BUPA [14], Vehicle [14] and RS-DATA [75]. WBC, BUPA and Vehicle data sets are extensively studied in [80]. We shall use these data sets in other chapters.

3.3.1 Data Sets

IRIS Data

This is the well-known Anderson's Iris data set [4]. It contains a set of 150 measurements in four dimensions taken on Iris flowers of 3 different species or classes. The four features are sepal length, sepal width, petal length and petal width. The data set contains 50 instances of each of the three classes. We used ten fold cross validation to estimate the misclassification error.

Wisconsin Breast Cancer (WBC)

It has 2 classes with 699 instances. Sixteen of the instances have missing values and hence we removed them. All reported results are computed on the remaining 683 data points. Each data point is represented by 10 attributes. Misclassification error rates were estimated using ten fold cross validation.

BUPA Liver Disorders (BUPA)

It consists of 345 data points in 6 dimensions distributed into 2 classes on liver disorders. Performance of our methodology on this data is determined using ten fold cross validation.

Vehicle data

This data set has 846 data points distributed in 4 classes. Each data point is represented by 18 attributes. Here also we used ten fold cross validation.

RS-DATA

It is a Landsat-TM3 satellite image of size 512 × 512 (= 262144 pixels) captured by seven sensors operating in different spectral bands [75]. With each spectral band/channel we get an image with intensities varying between 0 to 255. The 512 x 512 ground truth data provide the actual distribution of classes of objects captured in the image. From these images we produced the labeled data set with each pixel represented by a 7-dimensional feature vector and a class label. Each dimension of a feature vector comes from one channel and the class label comes from the ground truth value. The class distribution of the samples is given in Table 3.1. We used only 200 random points from each of the eight classes as training samples. In other words, out of 262144 data points, we considered only 1600 instances for training.

The five data sets are summarized in Table 3.2.

Table 3.1: Different Classes and their frequencies for RS data

Classes	Frequencies
Forest	176987
Water	23070
Agriculture	26986
Bare ground	7400
Grass	12518
Urban area	11636
Shadow	3197
Clouds	350
Total	262144

3.3.2 Results

The experiments have been performed using lilgp [124] on Alpha server DS10. The computational protocols which are common to all data sets are given in Table 3.3 and the data specific parameters are listed in Table 3.4.

In Table 3.4,

Table 3.2: The Five Data sets

Name of Data Set	No of classes	No of Features	Size of the data set and classwise distribution
IRIS	3	4	150 (= 50 + 50 + 50)
WBC	2	9	683 (= 444 + 239)
BUPA	2	6	345 (= 145 + 200)
Vehicle	4	18	846 (= 212+ 217+ 218+ 199)
RS-DATA	8	7	262144 (see Table 3.1)

Table 3.3: Common Parameters for all Data sets

Parameters	Values
Probability of crossover operation, p_c	0.75
Probability of reproduction operation, p_r	0.1
Probability of mutation operation, p_m	0.15
Probability of selecting a function node during crossover operation, p_f	0.8
Probability of selecting a terminal node during crossover operation, p_t	0.2
Probability of selecting a function node during mutation operation, p_{mf}	0.9
Probability of selecting a terminal node during mutation operation, p_{mt}	0.1
% of the total nodes of a tree used for point mutation, m	2%
Tournament size, τ	7
Threshold for defining heuristic rules, h	20

Table 3.4: Different parameters used for different data sets

Data set	P	s_1	M_s	M	m_l	m_n
IRIS	400	5	10	10	10	1200
WBC	300	5	100	120	10	1000
BUPA	500	5	100	120	10	1500
Vehicle	700	5	100	120	12	2500
RS	700	25	500	520	13	5000

P = Population size,
s_1 = Number of steps for step-wise learning,
M_s = Total number of generations for step-wise learning,
M = Total number of generations the GP is evolved,
m_h = Maximum allowed depth of a tree and
m_n = Maximum allowed number of nodes for an individual.

We use following notations to describe the performance:
P_{ts} = Average percentage of correctly classified test samples over 10 runs
m_{err} = Average percentage of misclassification including unclassified points on the test set over 10 runs

For each of the first four data sets we used the following computational protocol: We make a ten-fold random partitioning of the data and make two GP runs on each fold with a different initial population but keeping all other GP parameters the same. This process is then repeated 5 times, each with a different 10-fold partitioning of the data. In this way, we make ten GP runs, each involving a 10-fold validation. (Note that, here each GP run essentially involves 10 runs, each with one of the 10 folds). This helps to compare our results with other reported results.

In the case of the RS data set we do not use 10 fold validation to keep the results consistent with other results reported in the literature [75]. For this data set, as used by other authors, we generate a training-test partition with 200 points from each class for the training set and rest for the test set. Here also we make 5 such random training-test partitions and for each partition we make 2 GP runs.

The average classification accuracy on the test data P_{ts}, average misclassification error

m_{err} and standard deviation sd of errors over 10 runs for IRIS,WBC and BUPA data sets are given in Table 3.5 and for vehicle and RS data these are shown in Table 3.6.

Table 3.5 has four rows R_1 to R_4, where R1 = without any post-processing method, R2 = with only OR-ing operation, R3 = with OR-ing and (then) weighting scheme and R4 = with only weighting scheme.

Table 3.6 has six Rows, where Rw1 = without any post-processing method, Rw2 = with only OR-ing operation, Rw3 = with OR-ing and then applying heuristic rules, Rw4 = with OR-ing, heuristic rules and then weighting scheme, Rw5 = with only heuristic rules and Rw6 = with only weight based scheme.

The results after each post-processing operation (like OR-ing, heuristic rules and weighting scheme) are given in order to provide a clear understanding of the performance of each post-processing method. The mean GP run time for each data set is given in Table 3.8. The GP run time in Table 3.8 includes time taken for partitioning the data sets, evolving the GP, input/output file handling, post processing and validation. As discussed earlier, the post-processing time is insignificant compared to the evolution time of GP. Later we shall report some illustrative values for the vehicle data set.

For IRIS, 96.9% of the test data are classified correctly without using any post-processing methods. After OR-ing operation, we achieved 98% test accuracy. As there was no heuristic rule, we used only the weight based scheme. Combination of both OR-ing and (then) weight based scheme could provided 98.7% test accuracy as shown in Table 3.5. Without OR-ing, the weight based scheme alone could provided the result to 97.3%. In average, a GP run required 8.4 seconds to complete (Table 3.8).

As an illustration, the (preorder) expressions of a classifier (three trees) for the IRIS data are shown below:

TREE 1: (- x_2 x_3)

TREE 2: (/ (- x_3 x_2) (- (/ (- (/ 9.86082 x_4)(- x_4 x_2)) x_3) x_4))

TREE 3: (- (* (- x_3 5.15982) (+ x_1 3.36042))(- (- x_3 5.15982)(* x_4 x_4)))

x_1, x_2, x_3 and x_4 represent four features or attributes of IRIS data.

After simplifying the above expressions, we can write the following rules: For a pattern $\mathbf{x} = (x_1, x_2, x_3, x_4)^T$,

if $x_2 - x_3 \geq 0 \Rightarrow \mathbf{x} \in$ *class 1*

if $\frac{(x_2-x_3)x_3x_4}{(x_3+1){x_4}^2-x_2x_4-9.86082} \geq 0 \Rightarrow \mathbf{x} \in$ *class 2*

if ${x_4}^2 + x_1x_3 + 2.36042x_3 - 5.15982x_1 - 12.17934 \geq 0 \Rightarrow \mathbf{x} \in$ *class 3*

Table 3.5: Performance Evaluation for IRIS, WBC and BUPA Data

Methods	IRIS			W.B.C.			BUPA		
	$P_{ts}(\%)$	m_{err}	sd	$P_{ts}(\%)$	m_{err}	sd	$P_{ts}(\%)$	m_{err}	sd
R1	96.7	3.3	0.6	95.2	4.8	4.1	60.9	39.1	3.6
R2	98.0	2.0	0.5	96.7	3.3	0.2	69.0	31.0	2.3
R3	97.3	2.7	0.5	97.2	2.8	0.3	70.0	30.0	2.1
R4	98.7	1.3	0.4	96.4	3.6	0.5	69.9	30.1	1.9

In case of WBC data, 95.2% of the test data are classified correctly without using any post-processing method. After OR-ing operation, we achieved 96.7%. As there was no heuristic rule available, we used only the weight based scheme. Table 3.5 shows that with both OR-ing and (then) weighting schemes, we achieved 97.2% test accuracy (2.8% error). Without OR-ing operation, the weighting scheme could provided the result to 96.4%. This data set has been extensively studied in [80] where 33 *non*-GP classifiers are compared. Table 3.7 summarizes the errors reported in [80]. For consistency, we have rounded off the errors to one digit after decimal. For this data set the error for the 22 tree/rule based classifiers and 9 statistical classifiers varied between 3.1 - 8.5 and 3.4 - 4.8 respectively [80]. For the two neural networks, the error rates are 2.8 and 3.4. For the 5 GP methods given in [81], the errors for this data set varied between 3.2 to 3.6. For the proposed classifier the error rate is 2.8. Thus, performance of our method is good. The comparisons are summarized in Table 3.7.

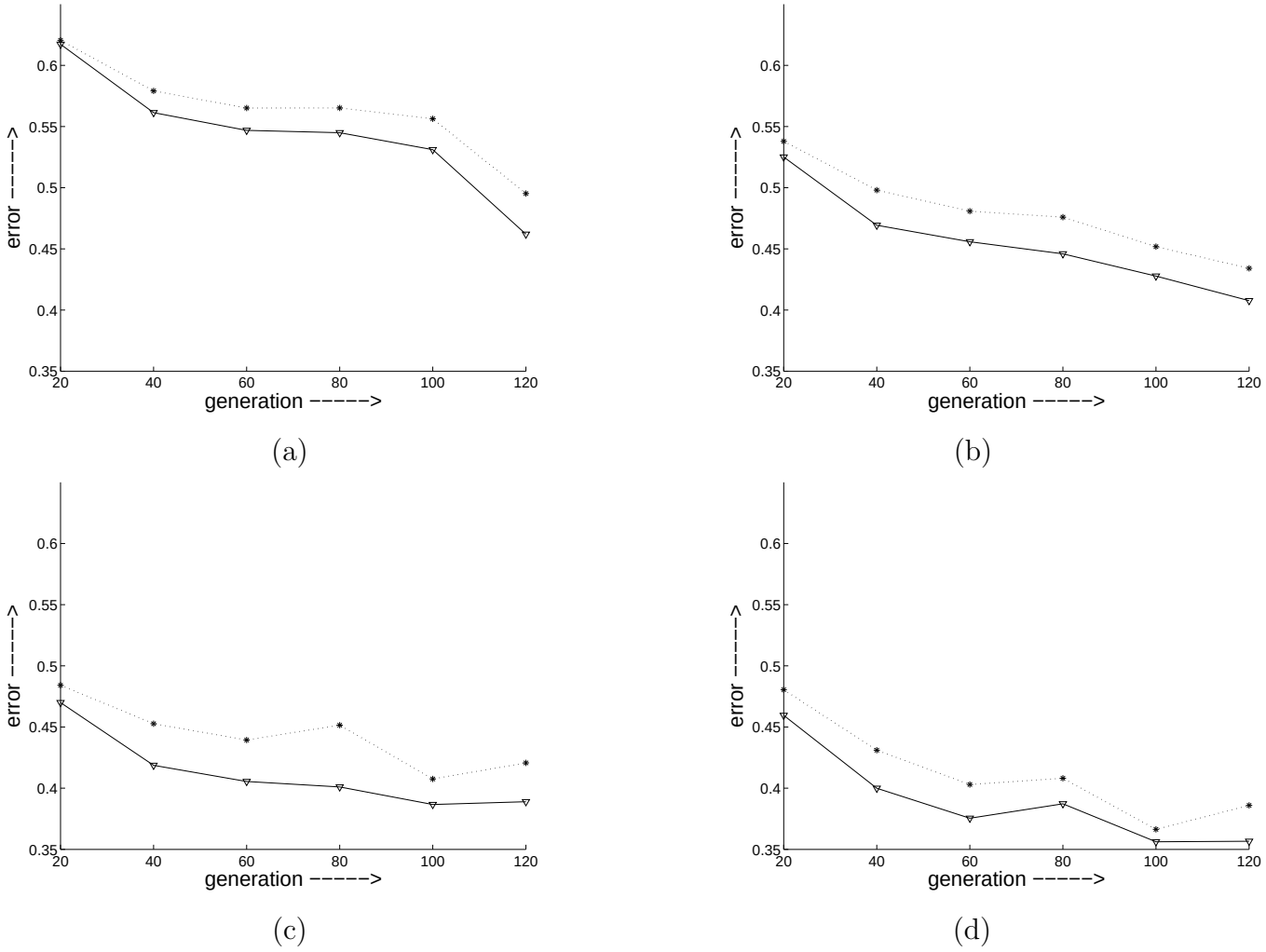
For the BUPA data, 60.9% test points are classified correctly without using any post-processing method. After OR-ing operation, we achieved 69% test accuracy. Since we do not get any heuristic rule, we used only the weighting scheme. The weighting scheme along with the OR-ing operation, provided a test accuracy of 69.9%. In this case the weighting scheme alone could provided 69.9% recognition accuracy. The BUPA data set is also tested against 33 classifiers in [80] and against 5 GP classifiers in [81].

Table 3.6: Performance Evaluation for Vehicle and RS Data

Methods	Vehicle			RS data		
	$P_{ts}(\%)$	m_{err}	sd	$P_{ts}(\%)$	m_{err}	sd
Rw1	48.6	51.4	3.8	70.9	28.6	5.3
Rw2	58.2	41.8	2.8	73.9	25.7	5.2
Rw3	58.9	41.1	2.5	75.5	24.0	5.5
Rw4	61.8	38.3	2.3	80.0	19.5	2.9
Rw5	58.6	41.4	2.2	76.1	23.4	5.4
Rw6	56.9	43.1	2.0	76.0	23.4	5.1

In case of the vehicle data set, we achieved only 48.6% classification accuracy without using any post-processing scheme (Table 3.6). Using OR-ing operation we obtained 58.2% test accuracy. The average number of heuristic rules generated without using OR-ing (n_h) and after using OR-ing (n_{hor}) for the vehicle and RS data sets are shown in Table 3.9. Using the heuristic rules after OR-ing operation, we got 58.9% test accuracy. Combination of OR-ing, heuristic and weighting schemes could provided 61.8% test accuracy. These results demonstrate the usefulness of all post-processing schemes. With only heuristic scheme and weighting method we get 58.6% and 56.9% test accuracies respectively. The errors produced by our method are also compared in Table 3.7 with other 38 approaches. Our method is quite comparable with the five GP based methods and the neural network based classifiers. As shown in Table 3.8, the mean GP run time for the vehicle data is 9 minutes 12 seconds. Without any post-processing, the mean GP run time is 9 minutes 6 seconds. So the computational overhead for post-processing operation is negligible (less than 2%) compared to the mean GP run time.

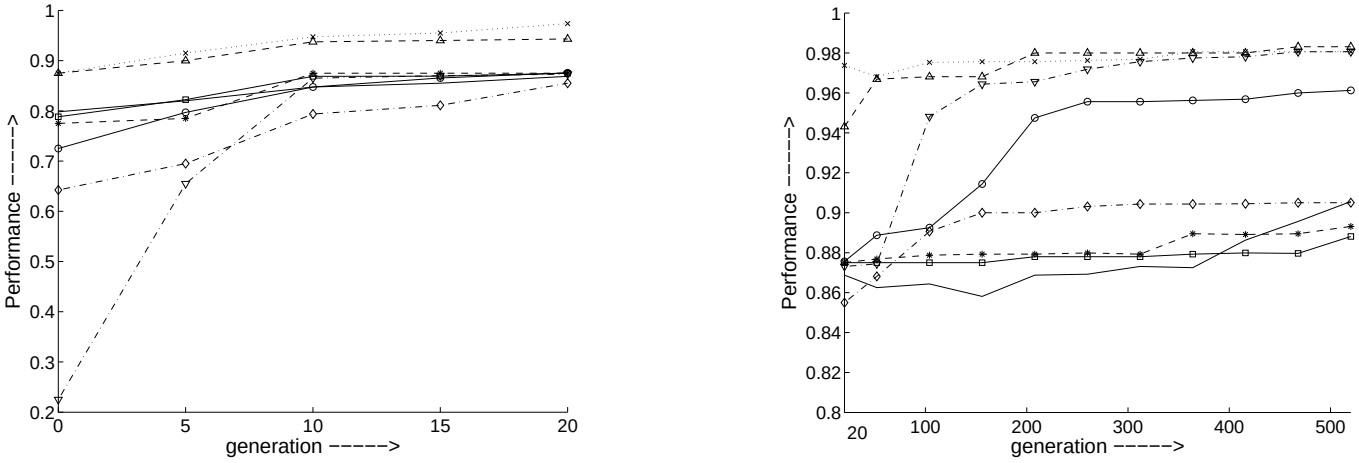
Table 3.6 also includes the results for the RS data set. For this data set 70.9% test data are classified correctly without using any post-processing method. Table 3.6 reveals that OR-ing operation increases the result by 3%. For the RS data set we obtained on average 3.7 heuristic rules, if we do not use ORing and 3.0 heuristic rules after OR-ing (Table 3.9). OR-ing and heuristic rules can increase the recognition accuracy by 4.6%. Combination of all three operations (OR-ing, heuristic rules and then weighting method) could increase the accuracy by 9.1%. Thus we achieved 80% recognition accuracy with all three operations. In Table 3.6 we also included the results with only



(a) (b)

(c) (d)

Figure 3.2: Variation of training and test error rates for the vehicle data : (a) No post-processing is used, (b) Only OR-ing is used, (c) OR-ing and heuristic rules are used, and (d) OR-ing, heuristic rules and weighting scheme are used



(a) (b)

Figure 3.3: (a) performance of the eight trees for RS data up to 20 generations and (b) performance of the same eight trees from generations 20 to 520.

weighting scheme and only heuristic scheme just for understanding of the effectiveness of each post processing scheme. Using the maximum likelihood (ML) method and artificial neural networks, 79.0% and 79.6% test accuracies are attained for this data set as reported in [75].

In any learning scheme, when the training error attains the minimum value, the generalization (test error) may not be the best. In order to investigate this, the misclassification (error rate) for the training and test sets in a typical run for the Vehicle data are noted after every 20 generations and displayed in Fig. 3.2. Figure 3.2 has four panels: (a) shows the error rates without using any post-processing scheme, (b) shows the error rates after OR-ing, (c) depicts the error rates after using OR-ing and heuristic rules and (d) shows the error rates after using OR-ing, heuristic rules and weighting scheme. The solid lines with triangular marks show the error rates for the training data. The dotted lines display the error rates for the test data. For panels (a) and (b) we find that both training and test error rates decrease with generations, but for panels (c) and (d) we observe that the best training and test error rates occur around generation number 100. We have analyzed such plots for other data sets also. In most cases we found that training error and test error follow a consistent pattern and the training and test errors decrease with generations. However, panels (c) and (d) of Fig. 3.2 suggest that use of a validation set may further improve the classifier

performance.

Table 3.7: Comparison of mean error rate m_{err} with other approaches

Data sets	WBC	BUPA	Vehicle
Our method	2.8	30.1	38.3
Five other GP approaches [81]	3.2 - 3.6	30.8 - 34.3	37.6 - 46.6
Twenty-two Tree/rule based methods [80]	3.1 - 8.5	27.9 - 43.2	20.6 - 48.7
Nine Statistical approaches [80]	3.4 - 4.8	28.0 - 40.1	14.5 - 22.4
Two NN classifiers [80]	2.8,3.4	32.9,33.0	37.2,37.4

Table 3.8: Mean GP Run Time

Data Sets	IRIS	WBC	BUPA	Vehicle	RS-DATA
Time (hour:min:sec)	0:0:8	0:3:28	0:3:05	0:9:12	2:50:00

So far we have not investigated how different trees evolve with generations. As an illustration, in Fig. 3.3 we show the performance of the 8 trees of a typical GP run for RS data. Fig. 3.3(a) shows the performance of the trees in the first 20 generations and Fig. 3.3(b) shows the performance of trees from 20 generations to 520 generations. The performance has been measured as the ratio of number of training samples correctly classified by a tree and the total number of training samples. Figure 3.3 reveals that some trees start exhibiting very poor performance initially but improve faster than the trees which start with relatively better performance. This demonstrates the effectiveness of the concept of *unfitness* of the tree. It also shows that the performance of trees improve with generations.

Table 3.9: Average number of heuristic rules obtained for each Data set

Data Set	IRIS	WBC	BUPA	Vehicle	RS Data
n_h	0	0	0	3.2	3.7
n_{hor}	0	0	0	2.6	3.0

3.4 Conclusions

We have proposed a GP approach to design classifiers. It needs only a *single* GP run to evolve an optimal classifier for a multi-class problem. For a c-class problem, a multi-tree classifier consists of c trees where each tree represents a classifier for a particular class. Unlike other approaches, we take an integrated view where GP evolves considering performance over all classes at a time.

Each tree recognizes patterns of a particular class and rejects patterns of other classes. Trees are independent of each other and each has an equal responsibility for classification, but all trees are tied together through fitness evaluation of chromosomes which governs the evolution of GP. An individual is selected according to its *fitness* value for genetic operation. However, its trees are selected according to their degree of *unfitness*. In this way, we give more opportunities to unfit trees to rectify themselves by genetic operations (crossover and mutation). At the same time, we reduce the chance of unwanted disruption of already fit trees by the genetic operations. In the case of crossover operation, we not only allow exchange of *subtrees* between trees meant for same class, but also complete exchange of some *trees* designated for the same class. Our mutation operation is designed to reduce the destructive nature of conventional mutation.

To obtain a better classifier we have proposed a new scheme for OR-ing two individuals. We have used a heuristic rule based scheme followed by a weight based scheme to resolve conflicting situations. The heuristic rules model typical situations where the classifier indicates ambiguous decisions and try to resolve them. The weight based scheme assigns a higher weight to a tree which is less responsible for making mistakes.

Our contributions in this paper are summarized as follows: (i) We Proposed a comprehensive scheme for classifier design for multi-category problems using a multi-tree concept of GP. It is an integrated evolutionary approach where classifier trees for all classes are evolved simultaneously.
(ii) For genetic operations, trees are selected on the basis of their *unfitness*.
(iii) We proposed a modified crossover operation.
(iv) We used a new mutation operation called non-destructive directed point mutation.
(v) An OR-ing operation is introduced to optimize the classifier - this can be applied with any GP based classifier.

(*vi*) We proposed a weight based scheme for conflict resolution. This is a modified version of the scheme suggested in [67].
(*vii*) The heuristic rule in [67] is also modified.

We tested our classifier with several data sets and obtained quite satisfactory results. Our future work will focus on reducing the size of the trees and feature analysis simultaneously with designing the classifier.

Chapter 4

Simultaneous feature selection and Classifier Design using Genetic Programming [A2]

4.1 Introduction

In the last chapter, we presented our Genetic Programming based classifier design scheme. There we did not consider feature selection which is also an important pattern recognition task. In this chapter, we propose a GP system that simultaneously selects useful features and designs the classifier.

Feature selection is a process to select useful features to obtain an efficient and improved solution to a given problem. Ideally, the feature selection process should select an optimum subset of features from the set of available features which is necessary and sufficient for solving the problem. Feature selection is important because all available features may not be useful. Some of the features may be redundant while some others may cause confusion during the learning phase. These features unnecessarily increase the complexity of the feature space which in turn demands more computation time for learning or finding a solution to the given problem.

Feature selection(FS) algorithms can be grouped [26] based on characteristics of searching: *exhaustive, heuristic and random*. Alternatively, they can also be categorized [26] into five groups based on the evaluation function that is used for evaluating the utility of the features: *distance* [66, 90], *information* [78, 76, 109], *dependence* [86], *consistency* [27, 77] and *classifier error rate*. The FS techniques which use classifier error rate as the criterion are called *wrapper* type algorithms [69]. In wrapper approach, the classifier for which features are being selected, itself is used as the evaluation function. Since

the suitability of features depends upon the concerned learning algorithm or classifier, the wrapper type algorithms usually perform better(in terms of accuracy) compared to other type called *filter* type techniques. However, wrapper approaches require more computation time than filter approaches. In a filter type method the selection is done independent of the learning algorithms. In this case, the relevant features are filtered out from the irrelevant ones prior to the learning. Good surveys on feature selection include [26, 15]. Analysis and discussion of various feature selection techniques are available in [74, 94, 54, 64, 30, 42, 69]. The Sequential Forward Selection (SFS) [31] methods start from an empty set and gradually add features selected by some evaluation function while the Sequential Backward Selection (SBS) [31] schemes start with the complete feature set and discard features one by one till an optimum feature subset is retained. However, in SFS once a feature is selected, it cannot be rejected later and reverse is true for SBS. Sequential forward floating selection(SFFS) [101] avoids this drawback. In [64, 74], it has been shown that SFFS performs better over many conventional FS techniques. In [90], branch and bound method is used to find an optimal feature subset with an assumption of a monotonic evaluation function. Relief [66] is a feature weight-based statistical approach whereas in [121], tabu search is used to find useful features. Some neural network based feature selection schemes are discussed in [95, 107], and [118]. Support vector machines(SVMs) have also been used for feature selection. In [83] a feature selection technique based on pruning analysis for SVMs is proposed. It enjoys characteristics of both filter and wrapper approaches. Filter approach reduces computational time whereas wrapper approach improves classification accuracy. Authors in [13] discuss feature selection and feature ranking using SVMs.

Evolutionary algorithms have also been used for feature selection [110, 111, 21, 97, 2, 108, 62]. Evolutionary algorithms are random search techniques. Typically, in a genetic algorithm (GA) based feature selection approach [110], [111], each individual (chromosome) of the population represents a feature subset. For an n-dimensional feature space, each chromosome is encoded by an n-bit binary string $\{b_1, b_2, ..., b_n\}$, $b_i = 1$ if the i_{th} feature is present in the feature subset represented by the chromosome and $b_i = 0$, otherwise. A classifier is used to evaluate each chromosome(or feature subset). Typically each chromosome is evaluated based on the classification accuracy and the dimension of the feature subset(number of 1's). In [74], it has been shown that GA based feature selection performs better than many conventional FS techniques

for high-dimensional data. In an evolutionary paradigm, a population of candidate solutions to the problem is allowed to evolve to achieve the desired optimal or sub-optimal solution using a set of genetically motivated operations.

Foroutan and Sklansky [110] used branch and bound technique for feature selection using GA. Casillas et al. [21] devised a genetic feature selection scheme for fuzzy rule based classification systems. In [103], ADHOC is a genetic algorithm based feature selection scheme with C4.5 induction learning. Pal et al. [97] proposed a new genetic operator called self-crossover for feature selection. Jack and Nandi [62] used GA for feature selection in machine condition monitoring contexts.

Unlike GA, there have been only a few attempts to use Genetic Programming(GP) [72, 7] for feature selection. Gray et al. [50] analyzed GP classifiers designed for two-class problems and decided on the features to be selected. They have not considered any special step for feature selection during the GP evolution. Ahluwalia and Bull [2] proposed a coevolutionary approach for feature extraction and selection using *automatic defined function* (ADFs) of GP. Each ADF is assigned its own independent sub-population which co-evolves with other ADF sub-populations and a population of main program trees. They used the k-nearest neighborhood(K-NN) algorithm for classification. Sherrah et al. [108] used GP for feature extraction/selection. They used a generalized linear machine as the classifier. In [52], a GP-based feature extraction scheme is proposed for the classification of bearing faults. The craeted features are then used as the inputs to a neural classifier for the identification of six bearing conditions. In [53], GP is used to generate features for breast cancer diagnosis.

In [95] authors proposed a novel scheme of selecting the relevant features on-line while training a neural network. In the on-line approach, selection of features and construction of the classifier are done simultaneously producing a good system for a given problem. Chakraborty and Pal also introduced neuro-fuzzy schemes for on-line feature selection [22], [23].

This chapter presents an *on-line* feature selection algorithm using Genetic Programming (GP), named, GP_{mtfs} (multi-tree genetic programming based feature selection). For a c-class problem, a population of classifiers, each having c trees is constructed using a randomly chosen feature subset. The size of the feature subset is determined probabilistically by assigning higher probabilities to smaller sizes. The classifiers which

are more accurate using a small number of features are given higher probability to pass through the GP evolutionary operations. As a result, we can achieve a good classifier using a small feature subset. We introduce two new crossover operations to suit the feature selection process.

Use of GP for on-line feature selection has some advantages. GP provides a mathematical representation of the classifier. As the classifier uses only a few selected features, the mathematical representation of the classifier can be easily analyzed to know more about the underlying system.

In this chapter, we conduct simultaneously feature selection and classifier design using GP. We have combined some additional steps in classifier design method to do feature selection. The classifier design method GP_{mtc} given in Chapter 3 can be upgraded for feature selection. Before combining additional components to GP_{mtc} for feature selection, we have attempted to simplify the classifier design approach. This reduces the computational time and complexity GP_{mtc}. Moreover, we have considered a modified fitness function in this classifier design approach GP_{mt}. In the following section, we present the changes made in GP_{mtc} approach to obtain the simplified algorithm, GP_{mt}. After designing the GP_{mt} approach, we include the additional steps given in section 4.3 for simultaneously feature selection. This algorithm with feature selection ability is called GP_{mtfs}.

4.2 Designing Classifiers without Feature Selection

We consider the following changes in GP_{mtc} to get GP_{mt}.

4.2.1 Modified Fitness Function

A tree T_k of the GP classifier act as a classifier for a two-class problem. It considers class k as the desired class and remaining classes together as the other class. This two-class data is typically an unbalanced data set. Let us illustrate it with an example. Consider a 4-class problem in which the number of training samples belonging to each class is 50. Thus, for the k_{th} class, $N_k = 50$ and $N - N_k = 150$. That means, tree ${T^l}_k$ is trained to discriminate between 50 training samples of k_{th} class and 150 training

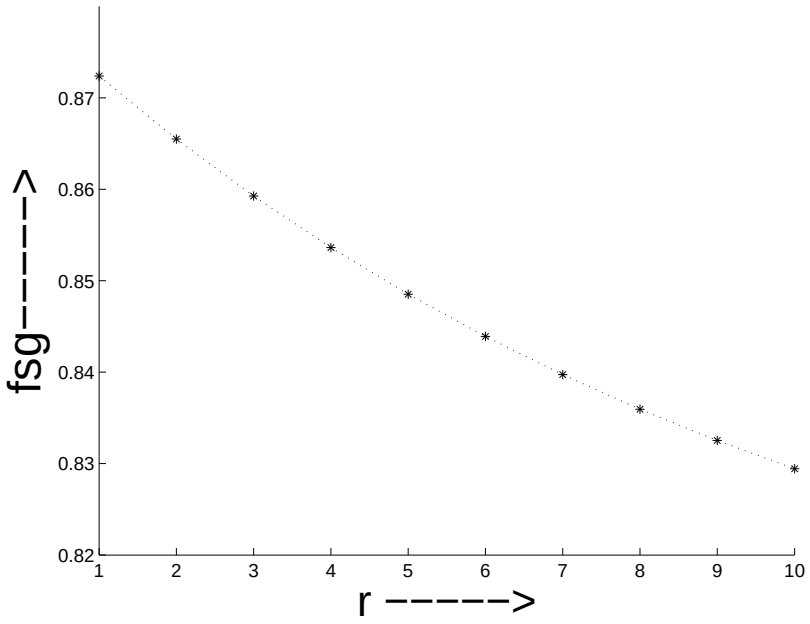


Figure 4.1: Decrease of fitness(f_{sg}) value with increase in r for a_f =0.1 and n = 10 in Eq. 6

samples of the remaining classes. For this two class problem, we have more samples from the second class. To reduce the effect of the unbalanced sample size, we assign more weight to the correct classification of training samples $\mathbf{x_i} \in$ *class* k by the tree T_k.

The population of classifiers is allowed to learn using a set of training samples, X = $X_1 \cup X_2 \cup \cdots X_k \cdots \cup X_c$, $X_i \cap X_{j \neq i} = \phi$, $\mid X_k \mid = N_k$ and $\mid X \mid$ = N. Each classifier classifies all training samples, $\mathbf{x_i} \in X_k$, $\forall i = 1, 2, \cdots, N_k$ and $k = 1, 2 \cdots, c$. The fitness function measures *how well* it performs the classification task.

We consider the following fitness function to compute the *fitness* of the l_{th} classifier of the population-

$$f^l = \sum_{k=1}^{c} \sum_{i=1}^{N_k} \sum_{j=1}^{c} {h^l}_{k,i,j}. \tag{4.1}$$

The two outer summations in (4.1) are used to consider all data points in X to compute the fitness of a classifier. ${h^l}_{k,i,j}$ is the contribution of j_{th} tree of l_{th} classifier (${T^l}_j$) for

the i_{th} data point($\mathbf{x_i}$) of X_k. So for a training point $\mathbf{x_i} \in X_k$ we expect ${T^l}_k(\mathbf{x_i}) \geq 0$ and ${T^l}_{j\neq k}(\mathbf{x_i}) < 0$. Consequently both ${T^l}_k(\mathbf{x_i}) \geq 0$ and ${T^l}_{j\neq k}(\mathbf{x_i}) < 0$ should increase the fitness function. To achieve this we define,

$$
\begin{aligned}
{h^l}_{k,i,k} &= \tfrac{N-N_k}{N_k} \text{ if } {T^l}_k(\mathbf{x_i}) \geq 0 \\
&= 0 \text{ if } {T^l}_k(\mathbf{x_i}) < 0 \\
{h^l}_{k,i,j\neq k} &= 0 \text{ if } {T^l}_{j\neq k}(\mathbf{x_i}) \geq 0 \\
&= 1 \text{ if } {T^l}_{j\neq k}(\mathbf{x_i}) < 0.
\end{aligned}
$$

Usually, the number of training samples in the k_{th} class, N_k, is smaller than the total number of training samples of classes other than the k_{th} class, $N - N_k$. The tree ${T^l}_k$ learns to discriminate between two classes, *class k* and all other classes taken together. Since the sizes of these two classes are highly unbalanced and the GP attempts to minimize the number of misclassifications, we give more importance to correct classification to class k. Let us illustrate it with an example. Consider a 4-class problem in which the number of training samples belonging to each class is 50. Thus, for the k_{th} class, $N_k = 50$ and $N - N_k = 150$. That means, tree ${T^l}_k$ is trained to discriminate between 50 training samples of k_{th} class and 150 training samples of the remaining classes. For this two class problem, we have more samples from the second class. To reduce the effect of the unbalanced sample size, we assign more weight to the correct classification of training samples $\mathbf{x_i} \in$ *class k* by the tree T_k. This is equivalent to copying each point in the training set of class k by $\frac{N-N_k}{N_k}$ times [67]. Thus in the stated 4-class problem, if a data point from class k is correctly classified by tree ${T^l}_k$, then the fitness is increased by 3 ($= \frac{150}{50}$).

If the identity of the classifier is not important, then for clarity we will ignore the superscript. So, a sample $\mathbf{x} \in X_k$ is correctly classified by the *tree* T_k, if $T_k(\mathbf{x}) \geq 0$ and it is correctly classified by tree $T_{j(\neq k)}$, if $T_j(\mathbf{x}) < 0$. A *classifier* correctly classifies a sample $\mathbf{x}$ if and only if all of its constituent trees correctly classify $\mathbf{x}$.

In GP_{mt}, we are not considering post-processing methods involving *heuristic rules* and *OR-ing operation* discussed in Chapter 3. This simplifies the scheme. We are also not utilizing step-wise learning as it will affect the feature selection. We use weight based scheme for conflict resolution.

4.3 Designing classifiers with on-line feature selection

For simultaneous feature selection and classifier design, we use the following additional steps in the proposed GP_{mtfs} scheme.

4.3.1 Selection of a feature subset for each chromosome

Let P be the population size. Before initializing each individual(classifier) C^l, $l \in \{1, 2, ..., P\}$, a feature subset G_l is randomly chosen from the set of all n available features. The size of the feature subset G_l for each classifier C^l is determined probabilistically. The probability to select a feature subset of size $r(\leq n)$ is taken as,

$$p^s{}_r = \frac{n-r}{\sum_{j=1}^{c}(n-j)} = \frac{2}{n-1} - \frac{2r}{n(n-1)}. \tag{4.2}$$

$p^s{}_r$ decreases linearly with increase in r. Note that $\sum_{r=1}^{n} p^s{}_r = 1$. We use Roulette wheel selection to determine the size of the feature subset,r, with probability $p^s{}_r$.

After, deciding the number of features r, we randomly select r features to construct the feature subset G_l. The classifier is now initialized as discussed earlier with the chosen feature subset G_l instead of all n features. The chosen feature subset G_l may be represented by a vector $\mathbf{v_l}$ whose j_{th} element $v_{lj} = 1$, if the j_{th} feature is present in the selected feature subset, otherwise $v_{lj} = 0$. For example, if the total number of features is 5 and out of these 5 features $\{x_1, x_2, x_3, x_4, x_5\}$, if $r = 2$ features, x_1 ,x_3, are selected to construct a classifier, then $\mathbf{v_l}$ will be (1,0,1,0,0).

4.3.2 Fitness Function

The fitness function is required to assign higher fitness value to the classifier which can classify correctly *more* samples using *fewer* features. Thus the fitness function is a multi-objective one, which needs to consider both correct classification and number of features used. We use the following fitness function:

$$f_s = f \times (1 + ae^{-\frac{r}{n}}). \tag{4.3}$$

In (4.3), f is the fitness value obtained by Eq(4.1), r is the cardinality of the feature subset used, n is the total number of features and a is a parameter which determines the relative importance that we want to assign for correct classification and the size of the feature subset.

The factor $e^{-\frac{r}{n}}$ decreases exponentially with increase in r and so is the fitness function. Thus, if two classifiers make correct decision on the same number of training points, then the one using fewer features is assigned a higher fitness. We decrease the penalty for using larger feature subset with generations to improve the classification accuracy. So initially we use fewer features, but as learning progresses we give more importance to better classification performance. To achieve this, we decrease a as:

$$a = 2a_f \left(1 - \frac{gen}{M}\right). \tag{4.4}$$

where a_f is a constant. M is the number of generations GP is evolved, *gen* is the current generation number. We have taken $a_f = 0.1$.

After each generation, to select the best chromosome, we use the fitness function f_{sg}:

$$f_{sg} = f \times (1 + a_f e^{-\frac{r}{n}}). \tag{4.5}$$

It is similar to f_s in (4.3), except that here a_f does not change with generation. It is because the best chromosome of each generation should be compared with the best chromosome of the previous generation to get the best chromosome of the run.

Figure 4.1 displays the fitness function(4.5) with $f = 0.8$, n = 10 and $a_f = 0.1$. It shows the effect of r, the size of the feature subset, on the fitness value.

After initialization, the population evolves using genetic operations iteratively.

4.3.3 Crossover Operation

We use two crossover operations to suit feature selection. These are:

(1) Homogeneous crossover called *Crossover_hg*

(2) Heterogeneous crossover called *Crossover_ht*. The heterogeneous crossover depends on the degree of similarity between two parents.

The probability of using homogeneous crossover is computed as

$$P_{hg} = \frac{gen}{M}. \tag{4.6}$$

Thus, the probability of using homogeneous crossover increases linearly with generations (*gen*) from 0 to 1. The *Crossover_ht* is used with probability $P_{ht} = 1 - P_{hg}$. So, P_{ht} decreases linearly from 1 to 0 with increase in *gen*.

Crossover_hg restricts the crossover operation between classifiers which use the same feature subset. This crossover operation completely avoids gradual use of more and more features by the classifiers. *Crossover_ht* allows crossover between classifiers using different feature subsets. However, we shall see later it is biased towards crossover between classifiers which use more common features. Consequently, this will check the gradual use of more and more features with generations. As a result of this, with generations the classifiers are not expected to use all features.

Homogeneous Crossover, *Crossover_hg*

This crossover operation restricts the crossover between classifiers which use the same feature subset. After selecting a chromosome as the first parent C_1, we select the second parent C_2 from the group of chromosome that uses the same feature subset as used by C_1. If there is no such chromosome, then we use heterogeneous crossover. The algorithm, *Crossover_hg*, is given in this section.

Heterogeneous crossover, *Crossover_ht*

This is a biased crossover between two classifiers which use more common features. At first, we randomly select a set of τ classifiers. The classifier which has the highest fitness value f_s among this set of classifiers is taken as the first parent C_1 for crossover. After that we randomly select another set of τ classifiers to select the second parent C_2. The degree of similarity s_j of the $j_{th}, j = 1, 2, ..., \tau$, classifier of this second set with C_1 is calculated as

$$s_j = \frac{\sum_{k=1}^{n} v_{0k} v_{jk}}{Max(\sum_{k=1}^{n} v_{0k}, \sum_{k=1}^{n} v_{jk})} = \frac{\mathbf{v_0}^T \mathbf{v_j}}{Max\{||\mathbf{v_0}||, ||\mathbf{v_j}||\}}. \tag{4.7}$$

Algorithm 1 *Crossover_hg (Population)*

$m_\tau = 0, m_1 = 1, Max_1 = 0, Max_2 = 0$
for $j = 1$ to τ **do**
  Randomly take a classifier $C^l \in Population$
  if $f_s(C^l) \geq Max_1$ **then**
    $Max_1 = f_s(C^l),\ k = l$
  end if
end for
$P_1 = C_k$ {first parent for crossover}
$C_i = C_{k+1}$ { now for second parent P_2}
while $((m_1 < P)\ and\ (m_\tau < \tau))$ **do**
  if $\mathbf{v}(C_i) = \mathbf{v}(P_1)$ **then**
    if $f_s(C_i) \geq Max_2$ **then**
      $Max_2 = f_s(C_i),\ P_2 = C_i$ {if both use same feature subset}
    end if
    $m_\tau = m_\tau + 1$
  else
    $C_i = C_{i+1}$ {if i = P, then $i + 1 = 1$}
  end if
  $m_1 = m_1 + 1$
end while
if $m_\tau > 0$ **then**
  Compute ${p_i}^1, i = 1, 2, \cdots, c$ of P_1
  Select a tree ${T_j}^1$ of P_1, by the Roulette-wheel selection using ${p_i}^1$
  Select a node type *ntype* {with probability q_{fc} to select *function* node type and q_{tc} to select *terminal* node type}
  Randomly select a node of the *ntype* type from ${T_j}^1$ and ${T_j}^2$ (of P_2) independently
  Swap the subtrees rooted at the selected nodes of ${T_j}^1$ and ${T_j}^2$
  Swap ${T_g}^1$ with ${T_g}^2$, $\forall\ g = j + 1, ..., c$
else
  Do heterogeneous crossover
end if

In Eq(4.7), $v_{0k} = 1$, if C_1 uses the k_{th} feature

$= 0$, otherwise.

Similarly, $v_{jk} = 1$, if j_{th} chromosome of the second set uses the k_{th} feature

$= 0$, otherwise.

Clearly, s_j lies in [0,1]. $s_j = 0$ means j_{th} classifier uses a completely different feature subset than that by C_1 and $s_j = 1$ implies that the j_{th} classifier uses the same subset of features as that by C_1. The j_{th} classifier of the second set is selected as C_2 with probability proportional to m_j:

$$m_j = f_{sj} + \beta s_j. \tag{4.8}$$

Here, f_{sj} = fitness of the j_{th} classifier, β = a constant that controls the effect of similarity. We have taken $\beta = 0.2$. In (4.8), the factor βs_j is responsible for crossover operation between classifiers using more common features.

To start with, since we randomly generate classifiers using different(randomly taken) feature subsets, the population uses a number of different feature subsets. If we start with homogeneous crossover, then for large dimensional data, the computation cost to search homogeneous group of classifiers will be more. Also, as discussed earlier, there is a chance that after selecting a classifier(C_1) for homogeneous crossover, we may not get another classifier (C_2) which uses the same feature subset as used by C_1. Moreover, since there are $(2^n - 1)$ possible feature subsets, for large n it is not possible to take all feature subsets to construct classifiers. So during the initial period of learning, we use *Crossover_ht* which may change some feature subsets and there by allowing GP to examine some new feature subsets. As the learning progresses, we decrease the probability of using *Crossover_ht* (and hence increase the probability of using *Crossover_hg*). As a result, with generations the good feature subsets should dominate and the population is expected to use a small number of feature subsets. In our scheme, usually feature selection is accomplished in the first few generations, so we do not use step-wise learning as done in [87]. Because step-wise learning uses a small subset of training samples at the beginning of the evolution and with a small subset of training samples the feature selection process may not be very fruitful.

Mutation, Termination criteria, conflict resolution, validation of classifier CF remain

the same as discussed earlier. The complete algorithm, *Classifier*, is given below for a better understanding.

Algorithm 2 *Classifier*

gen = 0, $fit_g = 0$, $fit_r = 0$ { fit_g = highest fitness value(f_{sg}) of that generation and fit_r =highest fitness(f_{sg}) till that generation}
Initialize population of classifiers $\{C_l\}, \forall l = 1, 2, \cdots, P$ {max. possible fitness $f_{max} = 1 + a_f \times exp(-1/n)$}
while gen < M and $fit_r < f_{max}$ **do**
  for $l = 1$ to P **do**
    Evaluate fitness f_s and f_{sg} of each classifier C_l
    $fit_g = max\ f_{sg}(C_l)$
    k = arg $max_l\ f_{sg}(C_l)$
  end for
  if $fit_g > fit_r$ **then**
    $fit_r = fit_g$
    CF = C_k
  end if
  Perform *Breeding* {all genetic operations}
  gen = gen + 1 {go to the next generation}
end while
Compute weights $\{w_i\}$ {of the best classifier CF}

4.4 Experimental Results

We have used seven data sets for validating our methodology. These data sets, named, Iris [4], WBC [14], Wine [14], Vehicle [14], WDBC [82], Sonar [49, 14] and GENE [25, 47] cover examples of small, medium and large dimensional data. Table 4.1 summarizes these data sets. The five data sets used in chapter 3 are not sufficient to validate the feature selection method. Hence we have introduced four new higher dimensional data sets and dropped two low dimensional data sets.

4.4.1 Data Sets

Description of Iris, WBC and Vehicle data sets have been given in Chapter 3. The other four data sets have been briefly described below.

Wine

Wine data set [14] consists of 178 points in 13 dimension distributed in 3 classes. These data are the results of chemical analysis of wines grown in a particular region of Italy but derived from three different cultivators. The analysis determined the quantities of 13 constituents found in each of the three types of wine.

Table 4.1: Data sets

Name of Data Set	No of classes	No of Features	Size of Data set
Iris	3	4	150 (50+50+50)
WBC	2	9	683 (444+239)
Wine	3	13	178 (59+71+48)
Vehicle	4	18	846 (212 + 217 + 218 + 199)
WDBC	2	30	569 (357+212)
Sonar	2	60	208 (97+111)
GENE	2	7129	72 (47+25)

WDBC

This Wisconsin Diagnostic Breast Cancer (WDBC) [82] data set contains observations on 569 patients with either Malignant or Benign breast tumor. Each data point consists of 30 features, which are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. These features describe characteristics of the cell nuclei present in the digitized image. Out of 569 samples, 357 belong to Malignant and remaining 212 samples belong to Benign classes.

Sonar

This data set [14, 49] contains 208 patterns obtained by bouncing sonar signals off a *metal cylinder* and *rocks* at various angles and under various conditions. Each pattern is represented by 60 attributes. Each attribute represents the energy within a particular frequency band.

GENE

GENE [25, 47] is a set of *DNA microarray gene expression* levels of 72 persons having either acute lymphoblastic leukemia (ALL) or acute myeloid leukemia (AML). Each sample is represented by 7129 gene expressions. 38 samples are used for the training and the remaining 34 samples are used for the testing. Out of 38 training samples, 27 belong to ALL (class 1) and remaining 11 samples belong to AML (class 2). The test set consists of 20 ALL and 14 AML samples.

4.4.2 GP Parameters

The GP parameters which are common for all data sets are given in Table 4.2 and the GP population size(P) which differs with data sets are listed in Table 4.3.

Note that, the parameters in Tables 4.2 and 4.3 are not specific to our algorithm. These parameters are required for any GP based classifier design.

We consider larger populations for higher dimensional data because the number of possible feature subsets increases with the number of features(dimension). Use of a large population helps GP to explore more possibilities and hence one can expect GP to evolve to a good solution without using many generations. If we allow large number of generations and large size of the classifiers then the classifiers may overfit(memorize) on training samples. Consequently, the classifiers may give better performance on the training data but poor performance on the test data. Choosing the optimal population size for a given problem is a difficult task and we do not study this problem here. The optimal number of generations can be determined using a validation set. To achieve a better generalization one should choose small values of M, m_h and m_n. Here we allow GP to evolve only up to 30 generations for all data sets.

Table 4.2: Common Parameters for all Data sets

Parameters	Values
Probability of crossover operation, p_c	0.80
Probability of reproduction operation, p_r	0.05
Probability of mutation operation, p_m	0.15
Probability of selecting a function node during crossover operation, q_{fc}	0.8
Probability of selecting a terminal node during crossover operation, q_{tc}	0.2
Probability of selecting a function node during mutation operation, q_{fm}	0.7
Probability of selecting a terminal node during mutation operation, q_{tm}	0.3
Tournament size, τ	10
Total number of generations the GP is evolved, M	30
Initial height of a tree	2-6
Maximum allowed nodes for a tree, m_n	350
Maximum allowed height of a tree, m_h	12

Table 4.3: Population Size

Data set	IRIS	WBC	Wine	Vehicle	WDBC	Sonar	GENE
P	1000	1000	1500	1500	2000	2000	5000

4.4.3 Results

We performed our experiments using lilgp [124] on Alpha server DS10. During classification by the classifier CF, we have used weight scheme (described in chapter 3) for conflict resolution. We run GP 10 times for each data set. Except for the GENE data set, we use the following computational protocol:

Each GP run involves a 10-fold cross-validation (Thus, each GP run consists 10 runs, each with one of the 10 folds, so total 100 runs). We compare the average performance of the classifiers obtained by the proposed method (GP_{mtfs})using a set of selected features with the classifiers designed using *all* features (using GP_{mt}) as described in Section 4.2. To do this, we also run GP 10 times using GP_{mt} method on each data set. Here also we do a 10-fold cross-validation.

In case of GENE data, we do not use 10 fold validation to keep the results consistent with other results reported in the literature on this data set. For this data set, as used by other authors, we use a specific training and test partition as mentioned in Section 4.4.1. We use the following notations in the subsequent sections.

A_s is the average percentage of correctly classified test samples by the best classifiers (CF of each run) using the selected feature subset over 10 GP runs, A_{all} is the average percentage of correctly classified test samples by the best classifiers using all features (without any feature selection) over 10 GP runs, n is the total number of features, n_s is the average number of selected features.

The average classification (test) accuracy A_{all} with all features and A_s with the selected features, and the average number of selected features n_s over 10 GP runs for six data sets are given in Table 4.4. The mean run time for each data set is shown in Table 4.5. The mean run time includes time taken for partitioning the data set, evolving GP for simultaneous classifier design and feature selection, input/output file handling, post processing, and validation. Table 4.5 shows that *run time* does not necessarily increase with increase in dimension of data sets. Because, the run time also depends on factors like size of population P, number of generations M, size of classifiers, number of training samples N and the distribution of the training samples in the feature space. If all training samples are correctly classified by the best classifier during the GP run, then the GP run terminates before completion of M generations and hence it reduces the run time.

Since different feature subsets are selected in different runs, to give an idea about the importance of the features, we count the frequency of the selected features. The normalized values of those frequencies are given in Table 4.9 for Iris, WBC and Wine data sets and in Table 4.10 for Vehicle data. These values may be (roughly) used for ranking the features, although, that is not our objective.

In [34], results of 3 feature selection algorithms, Forward Sequential Selection (FSS), Backward Sequential Selection (BSS) and relevance in context (RC), are available. RC uses a clustering like approach to select sets of locally relevant features in the feature space. In [78], results of another 3 feature selection algorithms called Information theoretic algorithm (IFN), Relief and ABB are available. Relief is a feature weight based statistical approach. ABB is a breadth first search, backward selection algorithm, with some simple pruning abilities. For comparison, we included results of these 6 methods on Iris, WBC and Wine data. We use results of ADHOC [103] for comparison of performance on Vehicle data. In addition, we use results of two neural network based feature selection schemes NNFS1 and NNFS2 given in [107] and [118] respectively and a Signal-to-Noise ratio based feature selection algorithm, SNRFS [9] to compare our results on WBC data. NNFS1 uses cross-validation classification errors to select features. NNFS2 uses a network pruning algorithm to remove redundant and irrelevant attributes one by one. We compare the performance of our scheme on Sonar data with results of 4 mutual information (MIF) based schemes [76], NNFS2 [118] and ADHOC [103].

Many methods have been used to analyze the data set *GENE*. Golub et al. [47] proposed a neighborhood analysis method to analyze it. Furey et al. [45] applied signal-to-noise ratio (SNR) criteria for feature selection and used support vector machine (SVM) for classification. Ben-Dor et al. [10] used a nearest neighbor method, support vector machine with quadratic kernel, and AdaBoost for classification. Principal component analysis was used by Nguyen et al. [92] to extract features. Then they used linear and quadratic discriminant analysis for classification. Cho and Ryu [25] considered various criteria for feature selection and applied multi-layer perceptron(MLP) network, k-NN, SVM, self-organizing map(SOM) and decision tree to classify the samples. Cho and Ryu [25] also used ensembles of these techniques for classification.

Rowland [104] used Genetic Programming to classify this gene expression data. Rowland partitioned the training data into *train* and *validation* sets. GP was run 15 times

on the *train* set and the best classifier of each run was used to classify the *validation* data. The three best classifiers which produced the minimum absolute difference between the errors on the *train* and *validation* sets, were combined by a voting scheme to classify the test samples.

Table 4.4: Average Performance

Methods	With all features		With selected features	
	n	A_{all} (%)	n_s	A_s (%)
Iris	4	98.65	1.56	98.69
WBC	9	97.42	2.23	96.84
Wine	13	95.49	4.08	94.82
Vehicle	18	78.37	5.37	78.45
WDBC	30	97.26	6.72	96.31
Sonar	60	84.74	9.45	86.26

Table 4.5: Mean Run Time

Data Sets	Iris	WBC	Wine	Vehicle	WDBC	Sonar	GENE
Time(min:sec)	0:30	2:20	1:10	7:30	4:10	2:30	15:40

Iris

From Table 4.4, we find that for Iris data, on average over 10 GP runs, the best classifiers could correctly classify 98.69% test data using on average only 1.56 features. This clearly indicates that the selected features have very good discriminating power. Table 4.6 shows that our method performs better compared to several other methods.

In most cases (GP runs), we observed that our methodology selected feature subsets {3}, {4} and {3,4}. Note that, {3,4} means 3^{rd} and 4^{th} features. From these observations, it can be concluded that the 3^{rd} feature (i.e., Petal length) and 4^{th} feature (i.e., Petal width) have good discriminating power for classification. This finding is consistent with feature analysis results represented by other researchers [22, 95]. The weights of the features are given in Table 4.9.

As an illustration, we show the expressions corresponding to the three trees of the best classifier of a typical run:

TREE1: x_3 - 4.8

Table 4.6: Comparison with other methods for IRIS data

Methods	FSS	BSS	RC	IFN	Relief	ABB	Fuzzy	GP_{mtfs}
$A_s(\%)$	92.6	92.6	94.4	94.0	96.0	90.0	96.58	98.69
n_s	2.2	2.6	4	1	2	1	3	1.56

TREE2: $(x_3$ - 2)(4.9 - $x_3)$

TREE3: 2.2 - x_3

Where x_3 is the 3^{rd} feature. After analyzing the above expressions, for a pattern $\mathbf{x}$, we find the following decision rules for the three classes:
If $x_3 \geq 4.8$ then $\mathbf{x} \in$ *Class 1*
If $x_3 \in [2, 4.9]$ then $\mathbf{x} \in$ *Class 2*
If $x_3 < 2.2$ then $\mathbf{x} \in$ *Class 3*

These rules suggest that we need techniques like weight based method to make decisions in the overlapping regions [2.0,2.2] and [4.8,4.9]. Note that, the best GP classifier may not always produce such simple rules.

Fig.4.2 shows the distribution of Iris data in feature space of 3^{rd} (petal length) and 4^{th} (petal width) features.

WBC

For this data set, on average our method constructed classifiers, selecting only 2.23 features. Our method selected features 6, 2 and 3 in most of the runs. With these selected features we could achieve 96.84% test accuracy. Table 4.4 shows the average performance.

Table 4.7 compares our method with six other methods. In this case too, the proposed method outperforms other methods. The weight (the normalized frequency with which each feature is selected) computed for each feature is given in Table 4.9. Table 4.9 reveals that features 2,3 and 6 are very important.

To demonstrate the decision rules for *WBC* we show one of the best outputs (best in terms of simplicity of expressions) corresponding to a GP run:

TREE1: $3.84 - x_2$

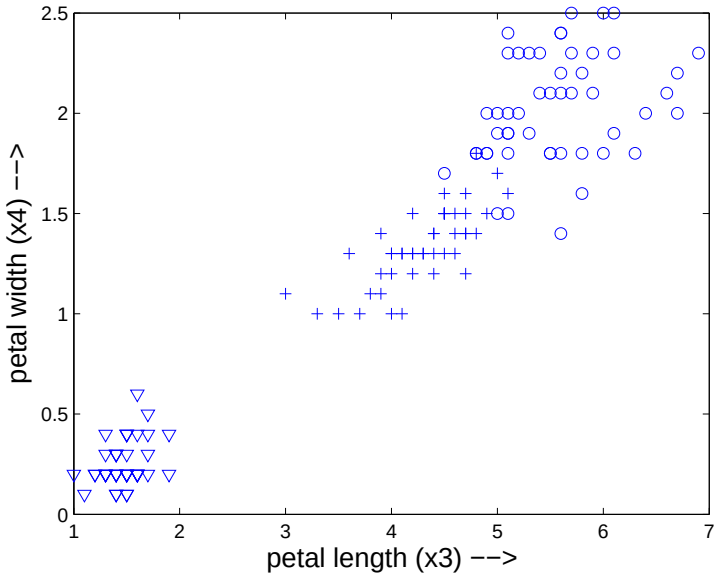


Figure 4.2: Iris Data using 3rd and 4th feature

Table 4.7: Comparison with other methods for WBC data

Methods	IFN	Relief	ABB	SNR	NNFS1	NNFS2	GP_{mtfs}
$A_s(\%)$	94.0	93.6	93.6	92.53	94.15	95.77	96.84
n_s	3	2	3	1	2.7	2	2.23

TREE2: $x_2 - 3.47$

where x_2 is the second feature. We know that for the WBC data set x_2 is an integer valued feature. So, after analyzing the above expressions, we can write the following simple decision rule:

If $x_2 \leq 3$ **then**
$\mathbf{x} \in Class\ 1$
else
$\mathbf{x} \in Class\ 2.$

This is probably one of the simplest rule that can do the classification task with a reasonably good accuracy of 92.39%.

Wine

For wine data set also the average number of selected features and the classifier performance over all runs are given in Table 4.4. It shows that on average with only 4.08 selected features, GP is able to construct classifiers with 94.82% accuracy. The computed importance (weights) of the features are included in Table 4.10. Table 4.8 compares our method with three other methods [78]. In this case GP_{mtfs} outperforms IFN and ABB but Relief is marginally better than GP_{mtfs}.

Table 4.8: Comparison for Wine and Vehicle data

Data	Wine				Vehicle	
Methods	IFN	Relief	ABB	GP_{mtfs}	ADHOC	GP_{mtfs}
A_s	91.7	95.0	79.0	94.82	69.6	78.45
n_s	3	3	2	4.08	7	5.37

Vehicle

For Vehicle data, on average, 5.37 features are selected (used) by the classifier. With about 5 features 78.45% test points are classified correctly as given in Table 4.4. The ADHOC [103] method reported a test accuracy of 69.6% using 7 features while our method could achieve 78.45% test accuracy using a lesser number of features as shown in Table 4.8. The classifier designed with all features classifies 78.37% of test data.

The weights of some of the features are listed in the Table 4.10, the rest are not included as they have negligible weights.

Table 4.9: Weights of features for Iris, WBC and Wine data

Features	1	2	3	4	5	6	7	8	9	10	11	12	13
Iris	0.15	0.07	0.42	0.36									
WBC	0.031	0.232	0.182	0.050	0.052	0.260	0.061	0.110	0.022				
Wine	0.108	0.044	0.063	0.056	0.054	0.092	0.103	0.043	0.066	0.118	0.043	0.091	0.119

Table 4.10: Weights of features for Vehicle data

Features	7	13	10	11	8	12	16	9	14
Weights	0.142	0.112	0.108	0.104	0.101	0.098	0.094	0.077	0.073

WDBC

The average performance of the best classifiers over all 10 runs shows that with only 6-7 features, 96.31% test data can be classified correctly (Table 4.4). We do not include the feature weights for WDBC in Table 4.9 because the number of features is too large to accommodate in Table 4.9. However, the most important 7 features are 28, 8, 21, 10, 13, 15, and 23.

Sonar

On average, we achieved 86.26% accuracy with about 9-10 features (Table 4.4). While using all 60 features, the average performance is 84.74%. This again emphasizes that more features are not necessarily good. Table 4.11 shows that GP_{mtfs} performs much better than ADHOC and the 4 mutual information based selection schemes [76]. For this data set NNFS1 outperforms our method both in terms of accuracy and number of features used.

Table 4.11: Comparison with other methods for Sonar data

Methods	ADHOC	4 MIF schemes	NNFS1	GP_{mtfs}
$A_s(\%)$	76	76.45-81.70	93.81	86.26
n_s	16	9	3.87	9.45

GENE

For this data set, the average accuracy on the test data achieved by our method is 92.55% using on average 10.45 features out of 7129 features. In one of the GP runs, the best classifier could classify correctly 33 of the 34 test samples (97.1% accuracy) using 13 features. Note that, during the generation of the initial population, the algorithm may select a large number of features for some classifiers, but since the depth of initial trees are restricted to 2-6, most of the selected features will not be used. Consequently, GP will evolve using only a small number of features, which is further moderated by our crossover operations. In [47], Golub et al. reported correct classification of 29 test samples with high confidence. Nguyen et al. [92], [25] achieved 97.1% (best test) accuracy using the logistic discriminant classifier. On the other hand, the test accuracy of the classifiers studied by Cho and Ryu [25] varied from 58.8% to 100%. Rowland [104] obtained 91.1% test accuracy using Genetic Programming.

Our experiments on Sonar and GENE suggest that the proposed method can easily do a good job for moderately, large and very large dimensional data sets also.

4.4.4 Effect of noise features

Here we study the sensitivity of our method on bad (noise) features that are synthetically injected to a few real data sets. To produce noisy data, we add 4 randomly generated values, x_{n1}, x_{n2}, x_{n3} and x_{n4}, to each data point of WBC, Wine and Vehicle data. We generate the values of x_{n1} and x_{n2} in [0,1] and [-1,1] respectively. To decide on the domains of x_{n3} and x_{n4}, we randomly select two features of the data set and use their domains. Once the domains are decided we randomly generate values of x_{n3} and x_{n4} from their respective domains. For example, if the domain of a selected feature is [15.5,30.0], then for each data point we use a random number generator to obtain a value in [15.5,30.0] and add it as a noisy feature value. To see the effect of the

noise features we run our scheme with feature selection (GP_{mtfs}) and without feature selection (GP_{mt}) on the augmented noisy data sets. We run GP 5 times with 2-fold cross-validation using both the schemes. The average accuracies for WBC, Wine and Vehicle data are given in Table 4.12. The performance of our scheme on data sets with noise features almost remains the same as that on data sets without noise.

We have observed that the proposed feature selection scheme GP_{mtfs} only occasionally chooses any noise feature, but on average there is a marginal increase in the number of features used. Addition of noise features causes confusion in feature space. This may lead the classifier to select more non-noise features to counter balance the complexity in feature space caused by noise features. Moreover, if a noise feature gets selected, the system may need a few good features to balance its influence. Hence, there is a marginal increase in the number of selected features. An interesting observation is that even without feature selection, our scheme can find reasonably good classifiers from the noisy data. This is a very good attribute of the proposed GP based classifier design scheme.

In addition to this comparison of average performances, we also compare the performance of these two methods over noisy data using 5×2 cross validation paired t test [33]. Let μ_1 and μ_2 be the means of the test errors using GP_{mt} and GP_{mtfs} respectively. Let the null hypothesis be $H_0 : \mu_1 = \mu_2$ and the alternative hypothesis be $H_1 : \mu_1 > \mu_2$. The computed t values, $\tilde{t}$ for WBC, Wine and Vehicle data are given in Table 4.13. The cut-off value for rejecting the null hypothesis at 95% confidence level for t with 5 degrees of freedom [33] is 2.015. Table 4.13 shows that the $\tilde{t}$ value for each of these 3 data sets is greater than this critical value of t. So the null hypothesis is rejected at 95% confidence level.

Table 4.12: Average Performance for noisy data

Methods	With all features		With selected features	
	n	A_{all} (%)	n_s	A_s (%)
WBC	13	94.54	3.46	95.92
Wine	17	93.86	5.27	94.33
Vehicle	22	76.32	6.12	78.16

Table 4.13: $\tilde{t}$ Statistic for the noisy data sets

Data set	WBC	Wine	Vehicle
$\tilde{t}$	2.022	2.038	2.104

4.5 Conclusions

We proposed a methodology for on-line feature selection and classifier design using GP. In a *single* run of GP our method automatically selects the required features while designing classifiers for a multicategory classification problem.

We generated the initial population in such a manner that the classifiers use different feature subsets. The initialization process generates classifiers using smaller feature subsets with higher probability. The fitness function assigns higher fitness values to classifiers which classify more samples using fewer features. The multi-objective fitness function helps to accomplish both feature selection and classifier design simultaneously. In this regard, we proposed two modified crossover operations suitable for feature selection. As a byproduct, we obtained a feature ranking method.

The effectiveness of our scheme is demonstrated on seven data sets having dimensionality varying between 4 and 7129. Our experimental results established that the proposed scheme is very effective in selecting a small set of features and finding useful classifiers. The obtained results (Table 4.4) reveal that the proposed method can achieve almost the same performance using a small set of features as that with all features. We have also demonstrated the effectiveness of our scheme on data sets with known redundant or bad features added synthetically. We have compared the performance of our methodology with results available in the literature with both filter and wrapper type approaches. Wrapper (and on-line) FS algorithms perform better than filter approaches, at the cost of computational time. Our proposed algorithm performed better for both two class and multi-class problems.

Some of the important characteristics of the proposed algorithm are:

- It provides a mathematical description of the classifier which is easy to analyze and interpret.

- The feature selection process is integrated into classifier design in such a manner that the algorithm can pick up the required features and obtain the classifier using them.
- The fitness function prevents the use of more features and hence helps to achieve more readability of the trees extracted by the system.
- Since the number of features to be used is not predefined, the algorithm has more flexibility.
- Using the output of the algorithm, we can obtain a ranking of the features.
- It can deal with a c-class ($c \geq 2$) problem in a single run of GP.

We have used arithmetic functions to design classifiers. So, our methodology is applicable to numerical attributes only. For data with nominal attributes, the logical functions like AND, OR , NOT may be considered instead of arithmetic functions.

Chapter 5

Evolution of Fuzzy classifiers Using Genetic Programming [A3]

5.1 Introduction

In the previous chapters, we have discussed GP based methods for classifier design and feature selection. In this chapter, we propose a GP based classifier design scheme that incorporates fuzzy logic. The GP system provides fuzzy rule based classifiers for a multi-class problem.

While designing models, we may incorporate fuzzy logic concepts to address vagueness and uncertainty in data. In the literature, several works are available, where Artificial Neural Networks(ANNs) are combined with fuzzy logic to design models (e.g. classifiers) [91, 22, 93]. Many authors have already used genetic algorithms to design fuzzy rules [19, 105, 79, 20, 65, 58] including fuzzy classifier rules [60, 61]. A few attempts have been made to evolve fuzzy classifier rules using GP. In this case, a classifier constitutes a set of fuzzy rules (or a rule base). Evolution (or design) of fuzzy rules using a technique like GP does not need the help of domain experts.

Tsakonas has employed GP in [116] to develop four different types of classifiers. The author has used four context-free grammars to describe decision trees, fuzzy rule-based systems, feed forward neural networks and fuzzy Petri-nets with GP. To evolve fuzzy rule-based classifiers using GP, the author has initialized a population of rule bases where each rule base represents a possible classifier. In [99], GP has been used to find suitable structure of fuzzy neural networks for classification. After developing the structure, authors have optimized parameters using gradient-based technique. GP

operators have been incorporated with Simulated annealing to obtain fuzzy rule based classifiers in [106]. In [84], a co-evolutionary approach has been used to discover fuzzy classification rules. A GP population involving fuzzy rules and a simple evolutionary algorithm involving membership function definitions are co-evolved to seek a well adapted fuzzy rule set and a set of membership function definitions. GP has been used in [35] for the generation of fuzzy rule bases in classification and diagnosis of medical data.

A solution (individual) of the GP population can represent an individual fuzzy rule or a rule base (the classifier). In the former case, the individuals (rules) are evolved in a co-operative manner and the rules of the population act together as a classifier. In the later case, each individual is a rule base representing a classifier. These rule bases or classifiers are evolved in a competitive manner and the best classifier(s) is considered as the final desired classifier. We have adopted this later approach.

The performance of a fuzzy classifier depends on both its structure and parameters. In most approaches, the structure is assumed a-priori. GP has the ability not only to find the structure of the model (fuzzy classifier) but also the parameters it involves.

In this study, we consider fuzzy classifier rules [96] of the form:

R_i: **If** x_1 is A_{i1} **AND** x_2 is A_{i2} **AND**....**AND** x_p is A_{ip} **then class is** j. Here A_{ij} is a fuzzy set used in the i-th rule and defined on the domain of attribute x_j.

For classification of a pattern $\mathbf{x} \in \mathcal{R}^p$, the antecedent (if-clause) of the fuzzy rules are evaluated using some T-norm. The degree to which the antecedent of a rule is satisfied is called the firing strength of the rule. These firing strengths are used to determine the class membership of $\mathbf{x}$. Usually the class corresponding to the rule with the largest firing strength is assigned to $\mathbf{x}$.

Generally, all features are used to design a fuzzy rule. However, a rule may not need all features. In fact, some features may be redundant/irrelevant and some may even be derogatory. So, a rule may be created using a subset ($q \leq p$) of features.

We have used Genetic Programming (GP) to evolve fuzzy classifier rules for multi-class problems. Our GP methodology generates the required fuzzy classifier by determining: (i) number of rules, (ii) features(clauses) used in a rule, and (iii) values of the parameters involved. For a c class problem, an individual C_l consists of c trees

$\{{T^l}_1, {T^l}_2, ..., {T^l}_c\}$. Each tree ${T_k}^l$ contains a set of ${b_k}^l$ rules representing class k.

5.2 Procedure

The fuzzy sets A_{ik} of a fuzzy rule can be determined based on prototypes [96]. A prototype represents a cluster of patterns. If $\mathbf{m}_i \in \mathcal{R}^p$ represents a prototype and if $\mathbf{x} \in \mathcal{R}^p$ represents an arbitrary pattern then a fuzzy rule can be written as:

R_i: If $\mathbf{x}$ is CLOSE TO $\mathbf{m}_i$ then class is k.
This rule can further be written in a more understandable form as below:

R_i: If x_1 is CLOSE TO m_{i1} AND ... AND x_p is CLOSE TO m_{ip} then class is k.

Note that, the two forms of the rules are not necessarily the same. We can model fuzzy set 'CLOSE TO m_{ij}' using different functions such as triangular, trapezoidal and Gaussian membership functions. Here, we have used Gaussian membership function to represent a fuzzy set. The Gausssian membership function is given by:

$$\mu_{CLOSETO}(x_j, m_{ij}) = exp^{-\frac{[x_j - m_{ij}]^2}{{\sigma_{ij}}^2}} \tag{5.1}$$

In (1), m_{ij} is the j_{th} component of prototype $\mathbf{m}_i$ and σ_{ij} is the spread of the membership function. As membership value depends on values of m_{ij} and σ_{ij}, so it is necessary to find appropriate values of these parameters for better performance. At first, we take some initial values for these parameters to initialize the population of classifiers. The evolutionary process then evolves the classifiers including the parameters and provides suitable classifiers (and hence suitable values of the parameters).

To compute firing strength of the rule, in this investigation, the MIN operator is used as the T-norm. So, the minimum of all membership values in a rule gives the firing strength of the rule.

As mentioned earlier, in our GP methodology, for a c-class problem, a classifier consists of c trees. Fig 5.1 represents a classifier. Each tree T_k has an "OR" node as the root and a set of b_k rules, $\{{R_i}^k\}, i = 1, 2, ..., b_k$ as children to the root. The set of rules $\{{R_i}^k\}$ of tree T_k represents the class k. So, we do not include consequent part (then-clause) in the rules.

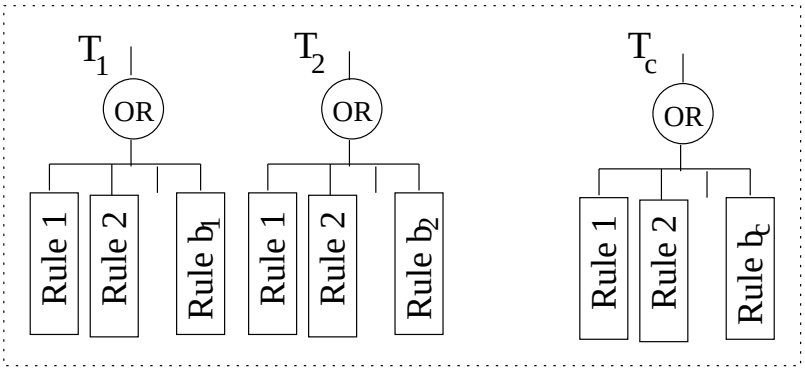


Figure 5.1: Representation of a fuzzy classifier

A typical rule $R_i{}^k$ (using features x_1, x_3, x_4 and x_5) may be written as:

$R_i{}^k$: x_1 is CLOSE TO m_{i1} AND x_3 is CLOSE TO m_{i3} AND x_4 is CLOSE TO m_{i4} AND x_5 is CLOSE TO m_{i5}.

where CLOSE TO m_{ij} is defined by a Gaussian function with two parameters m_{ij} and σ_{ij}.

In our GP approach, the above rule is represented as a tree as shown in Fig 5.2.

The root node of a rule $R_i{}^k$ is an "AND" node. The right children of "AND" node is a "CLOSETO" node and the left children is either an "AND" node or a "CLOSETO" node. CLOSETO node has three children: a feature x_j, m_{ij} and σ_{ij}.

Let us call this methodology as GP_{fc}. The procedure, GP_{fc}, for evolution of the GP-fuzzy classifier is discussed in the following sections.

5.2.1 Initialization

To start the evolution, at first a population of P classifiers is randomly generated. For a c class classification problem, a classifier C^l consists of c trees. To create a tree $T_k{}^l$ an "OR" node with b_k arity (children) is taken. Each children to the "OR" node is a rule $R_i{}^k, i \in \{1, 2, ..., b_k\}$. To determine the depth of the rule $R_i{}^k$, a randomly selected value d_i in the range of initial height of tree is taken. Then the tree structure of the rule $R_i{}^k$ is generated with height $d_i - 1$ (excluding the OR node) using a randomly chosen feature subset, S_i, of size $d_i - 2$ and a prototype $(\mathbf{m_i}, \sigma_\mathbf{i})$.

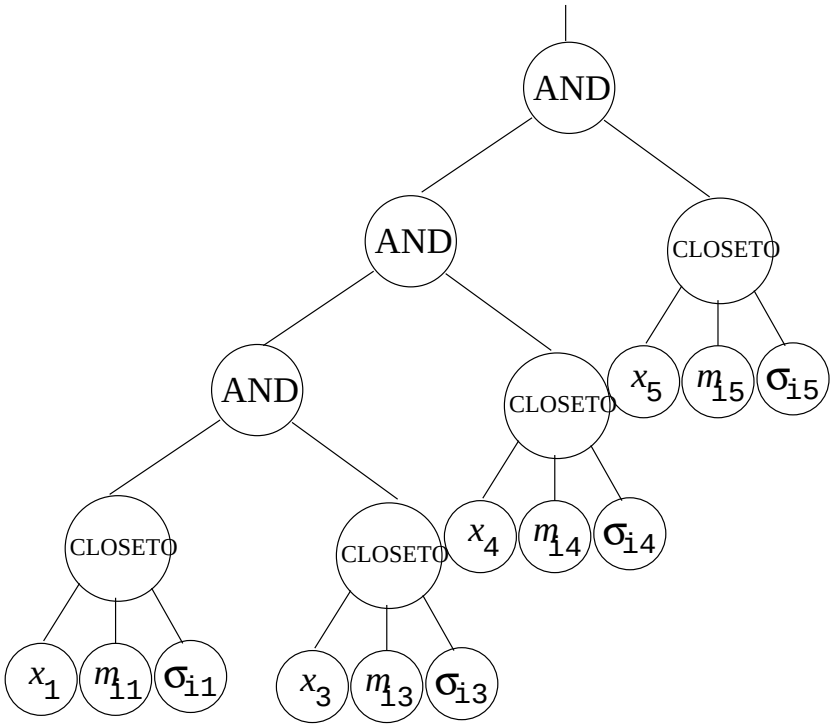


Figure 5.2: A typical tree representation of fuzzy rule

We produce the prototypes $(\mathbf{m_i}, \sigma_\mathbf{i})$ either randomly or by using a clustering algorithm; here we use the popular fuzzy K-means (FKM) clustering algorithm [12]. Both procedures are given below:

Random

We randomly take a value in the j^{th} feature domain as the value of μ_{ij}, i.e., $\mu_{ij} \in [{f^j}_{min}, {f^j}_{max}]$. ${f^j}_{min}$ and ${f^j}_{max}$ are the minimum and maximum values of the j^{th} feature in the training set. To obtain a value for σ_{ij}, we randomly take a value $r \in [\beta_1, \beta_2]$. Then we multiply r by the range of the j^{th} feature, W_j. Instead of taking $\beta_1 = 0$, we have taken a small value $\beta_1 = 10^{-4}$ for it. This prevents generation of almost zero value of σ_{ij}. Similarly, we have taken $\beta_2 = 0.25$ that prevents to produce large σ_{ij}. Note that, our goal is to represent a cluster of points by a rule so that membership value of patterns near to the prototype is high and away to prototype is low. But if we take very large value of σ_{ij}, then the Gaussian function will cover a large area and the difference between membership value of a near point and far point (w.r.t. prototype) may not be strong enough to discriminate.

Fuzzy K-means

We run fuzzy K-means algorithm to find K prototypes from the training data for each class. Then each training pattern is assigned to the prototype for which it has highest membership value. It may happen that no pattern is assigned to a prototype. In this case, the prototype is discarded. This provides $g_h \leq K$ prototypes for a particular class h. The mean vector $(\mathbf{m})$ and the vector of standard deviations (σ) of patterns to each cluster (associate with a prototype) are computed to obtain the set of initial values $\{\mathbf{m}_i, \sigma_i\}$. Note that, the j_{th} component of σ_i is computed as the standard deviation of the the j_{th} component of all training patterns falling in the i_{th} class.

The type of the prototype to use in a rule ${R_i}^k$ is decided with probability. Probability to select a random prototype is p_{rp} and the probability to select an FKM prototype is $1-p_{rp}$. The structure as given in Fig. 5.2 is maintained while generating the rule. The CLOSE TO nodes, each having a feature x_j of the chosen feature subset S_i and the corresponding parameters (m_{ij}, σ_{ij}), are used to generate the rule in the descending

order of index j from right top most to left bottom (as shown in Fig. 5.2). This makes it easy to maintain the validity of rules during the genetic operations.

In the example shown in Fig 5.2, the chosen feature subset is $\{x_1, x_3, x_4, x_5\}$. The right top most children (to the root node AND) is a CLOSETO node having x_5 and prototype (m_{i5}, σ_{i5}) as children. The left children to the root is an AND node. Again the right children to this AND node is the CLOSETO node having the next feature (in decreasing order) x_4 of the feature subset and the corresponding prototype (m_{i4}, σ_{i4}). The left children to this second AND node is also an AND node. This process is repeated till all but one feature is left in the feature subset. At this step, instead of taking an AND node, a CLOSETO node is created using the last feature (having the lowest index) of the feature subset and the corresponding prototype. Here the last feature is x_1.

5.2.2 Fitness Measure

In the evolutionary process, good solutions are evolved toward better solutions. To access how good a solution, each solution is evaluated and a fitness value is assigned to it. We evaluate a classifier (solution),C^l, of the GP population by allowing it to classify all training samples. For a training sample $\mathbf{x}$, the firing strength of each rule ${R_i}^k$ of tree ${T_k}^l$ is computed. The maximum firing strength is considered as the output $({\mu_k}^l)$ of the tree ${T_k}^l$. If output ${\mu_g}^l$ is the highest among outputs $\{{\mu_k}^l\}$ of all trees and $\mathbf{x} \in \ class\ g$, then we say that $\mathbf{x}$ is correctly classified by the classifier C^l. The number of correctly classified training samples ${n^l}_{rc}$ by the classifier C^l is computed. If N_{tr} is the total number of training samples, then the fitness value of the classifier is defined as

$$f^l = \frac{{n^l}_{rc}}{N_{tr}}. \tag{5.2}$$

5.2.3 Crossover

The good solutions are recombined to produce new solutions. It is expected that the new solutions will drive toward better and desired solutions. This recombination or crossover operation plays a vital role in the evolutionary process. For the crossover

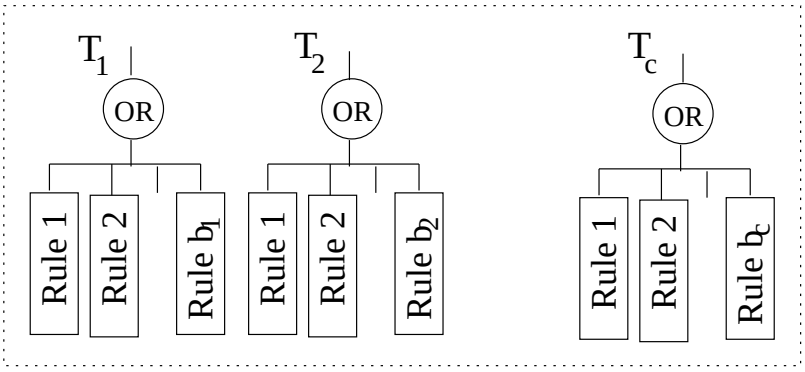


Figure 5.3: Parent 1 for crossover

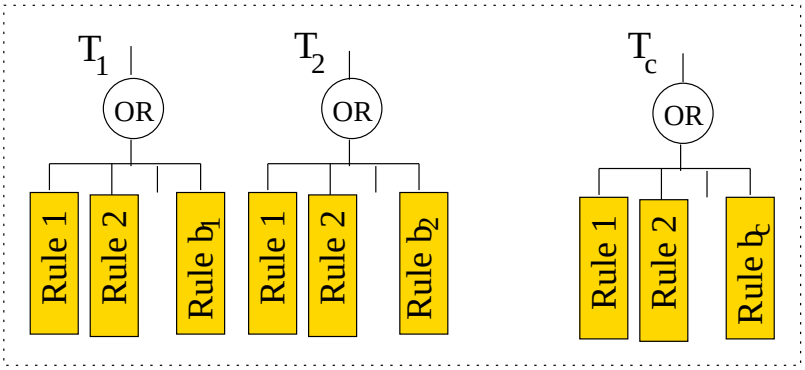


Figure 5.4: Parent 2 for crossover

operation, we select two classifiers C^{p1} and C^{p2} of the GP population based on the fitness value. A tree ${T_g}^{p1}$ of the first parent is randomly selected. Then an AND node of this tree ${T_g}^{p1}$ is randomly chosen. After that, an AND node of ${T_g}^{p2}$ of the second parent is randomly chosen in such a way that after swapping the subtrees with these AND nodes as root nodes will produce valid rules. Now the subtrees under these two AND nodes are swapped. In addition to this, we swap trees ${T_k}^{p1}$ of C^{p1} with ${T_k}^{p2}$ of C^{p2} for all $k > g$. This above mentioned crossover operation is repeated with probability p_c. Note that, such a crossover produces one new rule in each of the two classifiers involved and also swaps a set of rules between two classifiers. Figs. 5.3 - 5.6 illustrate the crossover operation. In the example, a subtree of rule 2 in tree T_1 of first parent is selected for swapping with a subtree of rule 1 in tree T_1 of second parent. After swapping subtrees, the trees following tree T_1 of both parents are swapped.

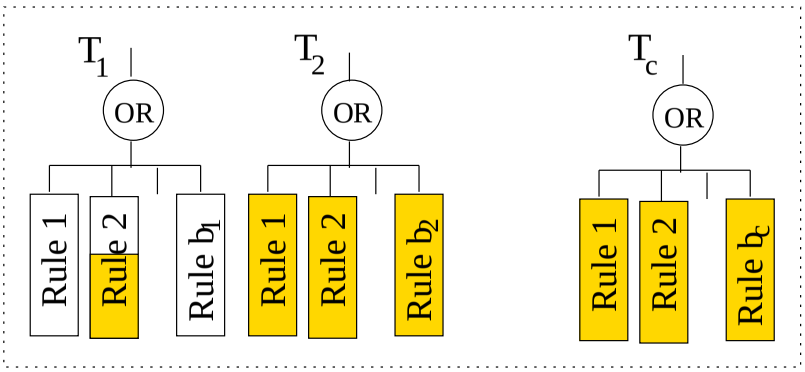


Figure 5.5: Offspring 1 after crossover

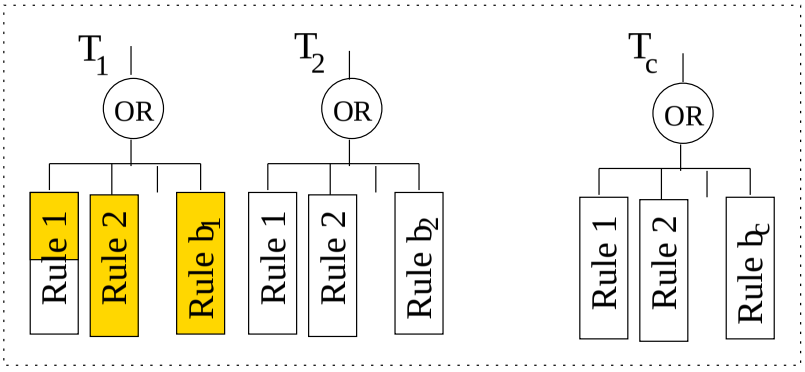


Figure 5.6: Offspring 2 after crossover

5.2.4 Mutation

Mutation is a process to alter a single solution. To obtain the desired model (fuzzy classifier), we need to alter both the structure and parameters. To facilitate this, we use two mutation operations: *Sub-tree* mutation and *parameter* mutation. At first a classifier of the GP population is randomly chosen. Then a tree of the classifier is randomly selected for the mutation operation. After that, the type of mutation operation is decided with probability. Both mutation operations are given below:

1. *Sub-tree Mutation*: This alters the structure including the parameters of a rule. This subtree mutation is the standard GP mutation operation. First, we have tried the standard GP mutation operation by randomly selecting a subtree for mutation. However, we have observed that it is no more effective than mutation restricted to choosing only subtrees with AND node as the root. So, instead of randomly selecting a subtree, we select a subtree with an AND node as root for mutation. After selecting a subtree with an AND node as the root, we find the lowest index, L_I and the highest index H_I of the features used by the subtree. Then a valid rule is randomly generated using a randomly selected subset of the features $\{x_{L_I}, x_{L_i+1}, ..., x_{H_I}\}$. The chosen subtree of the tree is replaced with the generated rule. For example, in Fig. 2, if the left child of the root node is selected for sub-tree mutation, then $L_I = 1$ and $H_I = 4$. So a randomly generated subtree using a subset of $\{x_1, x_2, x_3, x_4\}$ is used to replace the subtree rooted at the left child of the root node.

2. *Parameter Mutation*: We have introduced this mutation operation to alter values of parameters (m_{ij}, σ_{ij}). We randomly select a rule (of the selected tree) for the mutation. Then value of each (m_{ij} and σ_{ij}) of this rule is altered as follows:

Variation of m_{ij}: Let a_j and b_j be the smallest and highest values of the j_{th} feature, respectively. Then, define $W_j = b_j - a_j$. Now generate a random value $\delta \in [0, W_j/100]$ and another random value $r \in [0, 1]$. If $r < 0.5$, then we add δ to m_{ij}. Otherwise, we subtract δ from it. We do not accept the variation, if the altered value falls outside the domain of the feature.

Variation of σ_{ij}: A random value $\rho \in [0, 0.1 \times \sigma_{ij}]$ is generated. Now another random value $r \in [0, 1]$ is generated. If $r < 0.5$, then we add ρ to σ_{ij}. Otherwise, we subtract ρ from it. However, if the altered value is very small ($< W_j * 10^{-3}$), then we do not accept the variation.

We do sub-tree mutation with probability 0.8 and parameter mutation with probability 0.2. Before choosing these values, we have experimented on three sets of values: (0.9,0.1),(0.8,0.2) and (0.7,0.3). Out of these three cases, the performance was the best for (0.8,0.2) case.

After mutating the classifier, we evaluate it. If the fitness of the mutated classifier is higher or equal to that of original classifier then we accept the mutation. Otherwise, we accept the mutated classifier with probability 0.5.

After every generation, we expect the new best classifier to have a higher fitness than that of previous best classifier. But sometimes the increase in fitness is negligible or zero. If the variation is not sufficient, then we need to give sufficient perturbation to the GP population. This is accomplished by using mutation operation with a higher probability. Mutation may improve, degrade or may not even affect the performance. Since we use a directed mutation operation which accepts poor mutated solution with a probability, the chance of obtaining poor solutions is comparatively low. Thus, if the improvement in fitness over a set of successive generations is not satisfactory, we increase the probability of mutation. To do this, we maintain the so far best classifier and proceed as follows. Let B_t be the best classifier evolved in generation t with fitness f_{B_t} and SB_{t-1} is the so far best classifier with fitness $f_{SB_{t-1}}$ up to generation $t-1$. Now, if $f_{B_t} > f_{SB_{t-1}}$, then $SB_t \leftarrow B_t$; if $f_{B_t} = f_{SB_{t-1}}$ and the total number of nodes in B_t is smaller than that in SB_{t-1}, then $SB_t \leftarrow B_t$. Otherwise, $SB_t \leftarrow SB_{t-1}$. In this way, we maintain the so-far best classifier. Now, we sum the absolute difference in fitness of the so far best classifiers in 10 consecutive generations. If this sum is less than a predefined value, d_{th}, then we increase the probability of mutation operation p_m by γ_m. Now the question arises, what would be the values of these parameters?

Crossover operation plays a vital role to transfer good characteristics of the current individuals to the next generation by (sexually) combining two or more good solutions. Hence, a very high probability (say, 0.8) is given to crossover operation and low probability to mutation and reproduction operations. Let p_c, p_m and p_r be the probabilities of crossover, mutation and reproduction operations respectively. We have taken $p_c = 0.75$, $p_m = 0.15$ and $p_r = 0.1$ at the beginning. However, as mentioned above, we increase p_m during the evolutionary process when required. In our experiments, we have observed that the initial fitness value ($f^b{}_{ini}$) of the best classifier for different data sets varied from about 0.5 to 0.98. Thus, on average, we assume $f^b{}_{ini} = 0.75$.

We need to continue the evolution till a classifier could classify all training samples (${f^b}_{fin} = 1$) or a pre-defined number of generations, generation = 50 (= M), is reached. This means that if ${f^b}_{ini} = 0.75$ and ${f^b}_{fin} = 1$, then we have to increase the fitness (of the best classifier) by 0.25 in at most 50 generations. This indicates that in every 10 generations, the increment in fitness value of the best classifier is expected to be $\delta f^b = 0.05$. So, we use $d_{th} = 0.05$. The increment in p_m should be small such that at the end of the evolution, p_m is not very high. Let us assume the maximum allowed value of p_m is 0.25. In other words, we may increase p_m at most by 0.1 in 50 generations. If we divide this increment over five steps (10 generations each), then the increment for each step is 0.02. Hence we use $\gamma_m = 0.02$. These choices seem plausible, but not necessarily the best choices. The most desirable values for these parameters may be chosen using a validation data.

5.2.5 Cleaning of poor Rules

All rules of a tree may not be useful. Some of them may be poor/inaccurate and some may be inactive (not used in decision making). Inaccurate rules are primarily responsible for misclassification. So, we need to remove them. Also we should drop inactive rules. By keeping only few good rules, we can give more chance to these good rules to participate in genetic operations. This increases the chance of obtaining better rules in lesser time. Removing poor rules not only can improve the performance but also can reduce computational time to a great extent. Hence, during evolution, we remove the inaccurate and inactive fuzzy rules. In this experiment, we evolve GP up to 50 generations. Note that, an attempt to remove poor rules at an early (premature) stage of evolution may have derogatory effect on the final rule base. Hence, the first round of cleaning is done if and when the average fitness of the population reaches 0.7 (i.e. 70 % accuracy) during the first 25 generations. Then at gen=25, we again apply our rule cleaning scheme. After gen=25, we allow the population to evolve without any further pruning.

To decide on the rules to be removed, we evaluate each rule ${R_i}^k$ and assign a fitness value h_{ik} to it as described below:

If rule ${R_i}^k$ has the highest firing strength among all rules of tree T_k for a training sample $\mathbf{x}$, then increase h_{ik} by 1 if $\mathbf{x} \in$ class k; otherwise, decrease h_{ik} by 1 if $\mathbf{x} \notin$

class k. If $h_{ik} > 0$ then we keep the rule ${R_i}^k$; otherwise we remove it. In this pruning scheme, if a rule makes more misclassification than correct classification, we remove it. However, we make sure that pruning process will retain at least two rules for every class.

5.2.6 Termination of GP and Validation

The GP is terminated when all N training samples are classified correctly by a classifier of the GP population or a predefined number, M, of generations are completed. The best classifier of the population is the required classifier, CF. The classifier CF is validated by allowing it to classify test samples. The test accuracies are discussed in the following section.

5.3 Experimental Results

We have used the same set of data sets that have been used in chapter 3. These data sets are IRIS, BUPA, WBC, Vehicle and RS-data. In addition to these five bench mark data sets, we have considered Diabetes data [14] for comparison with the result of an existing GP based method. Table 5.1 summarizes all six data sets. A brief summary of Diabetes data has been given below:

Table 5.1: Data sets

Name of Data Set	No of classes	No of Features	Size of Data set
Iris	3	4	150 (50+50+50)
BUPA	2	6	345 (145+200)
Diabetes	2	8	768 (500+268)
WBC	2	9	683 (444+239)
RS	8	7	262144 (see chapter 3)
Vehicle	4	18	846 (212 + 217 + 218 + 199)

Diabetes: This two-class Pima Indians Diabetes data set has 768 instances in 8-

dimension. Out of 768 cases 500 belong to class 1 and remaining 268 belong to class 2.

5.3.1 GP parameters

The GP parameters that are common for all data sets are given in Table 5.2 and those that differ are given in Table 5.3.

Table 5.2: Common GP Parameters for all Data sets

Parameters	Values
Probability of crossover operation, p_c	0.75
Probability of reproduction operation, p_r	0.10
Probability of mutation operation, p_μ	0.15
Tournament size, τ	5
Total number of generations the GP is evolved, M	50

Table 5.3: Other GP Parameters

Data set	IRIS	BUPA	Diabetes	WBC	RS	Vehicle
Population Size	200	200	200	200	300	300
Initial height of a tree	4-6	4-8	4-8	4-8	4-8	6-12
Maximum height of a tree	6	8	10	11	9	17

The vehicle data is in a larger dimension than the other studied data sets and also it is a difficult data set. Also, RS data has many classes and it is a complex data. Hence we have taken a larger population for the vehicle and RS data. The height of a tree depends on the number of features. If there are F features then the height of the tree will be $F+2$. So, in case of Iris data, if a rule consists of all 4 features then the height of the tree will be 6. Similarly, for Diabetes and WBC, the maximum possible heights of the trees are 10 (8+2) and 11 (9+2) respectively. As the dimension (number of features) increases, the chances of redundant features also increase, hence, we restrict rules not to use all features. Thus, for Vehicle data a rule is allowed to use up to

15 features. Note that, these numbers are heuristically chosen - our objective in this paper is *not* to do feature analysis. For Iris and BUPA data sets, the initial height of a tree is 4-6. This means, we generate rules containing 2 $(= 4 - 2)$ to 4 $(= 6 - 2)$ features. We have kept the same initial height for Diabetes, WBC and RS data sets as they have almost the same dimension.

5.3.2 Results

We have used the (modified) lilgp [124] to conduct the experiments. We conducted three experiments. In the first and second experiments, except for RS and Diabetes data, we used 10-fold cross-validation to estimate the accuracy. For RS data, as mentioned in chapter 3, we run GP 10 times using 200 points from each class for training and remain points for test. Diabetes data has been used in third experiment to compare with an existing GP based method. For other data sets, in both experiments, GP was run 10 times each time involving a 10-fold cross-validation (total $10 \times 10 = 100$ runs). Here, performance of the fuzzy rules using 5 different proportions of FKM prototypes and Random prototypes is studied. In the first case(1:0), i.e., rules use only the FKM prototypes and in the last case (0:1) rules use only random prototypes. In the intermediate cases $(1 - p_{rp} : p_{rp})$, rules use FKM prototypes with probability $1 - p_{rp}$ and random prototypes with probability p_{rp}.

Table 5.4: Test Accuracy with initial 10 Rules per class

FKM:Rand	Iris	BUPA	WBC	RS	Vehicle
1:0	93.45 ± 0.79	67.38 ± 1.79	97.34± 0.38	75.23 ± 0.89	68.48 ± 0.76
0.75:0.25	94.10 ± 0.69	65.96 ± 1.43	97.40 ± 0.52	79.41 ± 0.72	70.02 ± 0.89
0.50:0.50	94.43 ± 0.53	65.14 ± 1.68	97.47 ± 0.31	81.33 ± 0.58	70.34 ± 0.85
0.25:0.75	96.53±0.49	65.09 ± 1.46	97.59 ± 0.41	80.09 ± 0.64	71.56 ± 0.68
0:1	96.33 ± 1.08	68.83 ± 1.98	96.88 ± 0.55	76.57 ± 0.78	69.08 ± 0.79

In the first and second experiments, we have initialized classifiers with $b_k = 10$ and $b_k = 5$ rules per tree T_k respectively. For the first case (1:0), when we use only FKM prototypes, we try to find b_k FKM prototypes. But for some data sets, the FKM

Table 5.5: Average number of Rules with initial 10 rules per class

FKM:Rand	Iris	BUPA	WBC	RS	vehicle
1:0	7.2	7.4	6.8	8.7	8.9
0.75:0.25	6.2	6.8	5.4	7.3	6.6
0.50:0.50	4.3	4.9	5.1	6.4	5.0
0.25:0.75	3.3	3.7	4.1	4.8	3.4
0:1	2.3	2.8	2.7	2.9	2.5

resulted in fewer prototypes. If the total number of FKM prototypes, g_k, is less than b_k for class k, then some FKM prototypes will be used more then once in tree T_k. In the intermediate cases (when both FKM and random prototypes are used), the FKM algorithm is used to obtain $(1 - p_{rp}) * b_k$ prototypes. For example, in case 2 of the first experiment, 0.75 * 10 $\approx$ 8 FKM prototypes are attempted to obtain.

Tables 5.4 and 5.6 show the average (test) accuracies along with standard deviation for Iris, BUPA, WBC, RS and Vehicle data sets. The performance of the classifiers that use both types of prototypes is found to be better than the performance using either type. Also, it is observed that the performance of classifiers using only random prototypes is quite good. It is interesting to note that for Iris data in Table 5.4 the accuracy is 96.33% using only random prototypes while the accuracy is 93.45% using only the FKM prototypes. And in Table 5.6, it is 95.80% using only random prototypes as against 93.30% using only FKM prototypes. Similarrly, for BUPA data, random generated rules outperform FKM prototype based rules. In Table 5.6, the average test accuracy is 69.20% using only random prototypes.

Tables 5.5 and 5.7 show the average number of rules of a tree over all trees in the final GP population. When we use only random prototypes we obtain trees with few rules. The reason is that as the prototypes are generated randomly, the proportion of bad prototypes is high. Our pruning scheme removes the unwanted rules arising from bad prototypes. This results in trees with fewer rules. On the other hand, when only FKM prototypes are used, almost all rules become important because the associated prototypes are placed in dense areas of the data. Since, we do not try to optimize the number of rules, we may end up with more rules to model some areas of the input

space. As shown in the Table 5.5, we obtained only 2-3 rules although we start with 10 rules using only random prototypes.

Table 5.6: Test Accuracy with initial 5 Rules per class

FKM:Rand	IRIS	BUPA	WBC	RS	Vehicle
1:0	93.30 ± 0.71	64.68 ± 1.36	97.22 ± 0.40	78.14 ± 0.74	70.38 ±1.14
0.75:0.25	95.65 ± 0.68	66.13 ± 1.17	97.18 ± 0.24	77.37 ± 0.82	69.83±1.23
0.50:0.50	95.93 ± 0.70	69.32 ± 1.24	97.26 ± 0.36	80.47 ± 0.68	70.61± 0.97
0.25:0.75	97.13 ± 0.65	68.22 ± 1.65	97.58± 0.45	78.93 ± 0.59	69.08± 1.31
0:1	95.80 ± 0.63	69.20 ± 1.36	97.50 ± 0.29	76.82 ± 0.92	65.22± 1.53

Table 5.7: Average number of Rules with initial 5 rules per class

FKM:Rand	IRIS	BUPA	WBC	RS	Veh
1:0	3.3	3.9	2.3	4.4	4.2
0.75:0.25	3.1	3.0	2.7	3.4	3.8
0.50:0.50	2.8	2.9	2.3	3.6	3.3
0.25:0.75	2.3	2.2	2.3	2.7	2.8
0:1	2.1	2.2	2.4	2.1	2.3

It can be observed that the overall performance of classifiers with 5 initial rules per class is slightly better than that using initial 10 rules per class. Two possible reasons for these are: More rules per class increases the chance of having more poor rules and hence more chance for poor rules to get involved in genetic operation. As a result, useful rules may get less chance to evolve. Also more rules may lead to memorization of the training data resulting in poor test performance.

In the second experiment, for Iris data we have obtained 97.13% accuracy when the composition of FKM and random prototypes is 0.25:0.75. Again, for WBC data the average test accuracy is 97.58% with 0.25:0.75 composition of the two types of prototypes. In [84], the GP was used to evolve fuzzy classification rules. The reported accuracy for Iris data using 10-fold validation is 95.3%.

In first experiment, with RS data, when the proportion is 0.50:0.50, we obtain the average accuracy of 81.33%. In case of Vehicle data, the average accuracy is 71.56% when the composition is 0.25:0.75.

To visualize how the learning (evolution) converges toward better solutions, we have plotted the best fitness and the average fitness of the solutions with generation for two data sets: Iris and WBC. For each data set we consider two extreme cases, using only FKM prototypes and using only random prototypes. Figure 5.7 and Figure 5.8 depict variation of the best and average fitness of a typical run for IRIS data when only FKM and only random prototypes are used, respectively. Although the variation of average fitness is similar in the two cases, the change in the best fitness is more smooth in case of random prototype. Similarly, Figure 5.9 and Figure 5.10 display the variation of the best fitness and average fitness of a typical run for WBC when only FKM and only random prototypes are used, respectively. The characteristic of Figs. 5.7 and 5.8 are also present here. This demonstrates the consistency of our algorithm. As the FKM prototypes quantize the data much better than the random prototypes, we obtain better classifiers with higher fitness value at the beginning in the former case. Figures 5.7 - 5.10 correspond to the cases with initially 5 rules per class.

As an illustration, we show the expressions corresponding to the three trees of the best classifier for Iris in a typical run:

TREE1: (OR (AND (Ct x2 0.18770 1.42674) (Ct x3 0.05320 0.20993)) (AND (Ct x1 0.27560 3.69521) (Ct x3 0.12370 0.27334)))

TREE2: (OR (AND (Ct x1 0.26417 2.59364) (Ct x3 0.14816 1.19223)) (AND (AND (AND (Ct x0 0.35443 6.33675) (Ct x1 0.28237 2.93938)) (Ct x2 0.22653 4.59761)) (Ct x3 0.13607 1.44758)) (AND (AND (Ct x0 0.87542 4.58146) (Ct x1 0.27787 3.46339)) (Ct x2 0.47887 4.53061)))

TREE3: (OR (AND (Ct x1 0.26858 3.13959) (Ct x3 0.14736 2.03583)) (AND (Ct x2 0.51499 5.96831) (Ct x3 0.54374 1.20857)))

The above expressions can be simplified as follows:

Class 1:

R1: If x3 CLOSE TO$_{\sigma_{i3}=0.19}$ 1.43 AND x4 CLOSE TO$_{\sigma_{i4}=0.05}$ 0.21 **then** *Class 1*

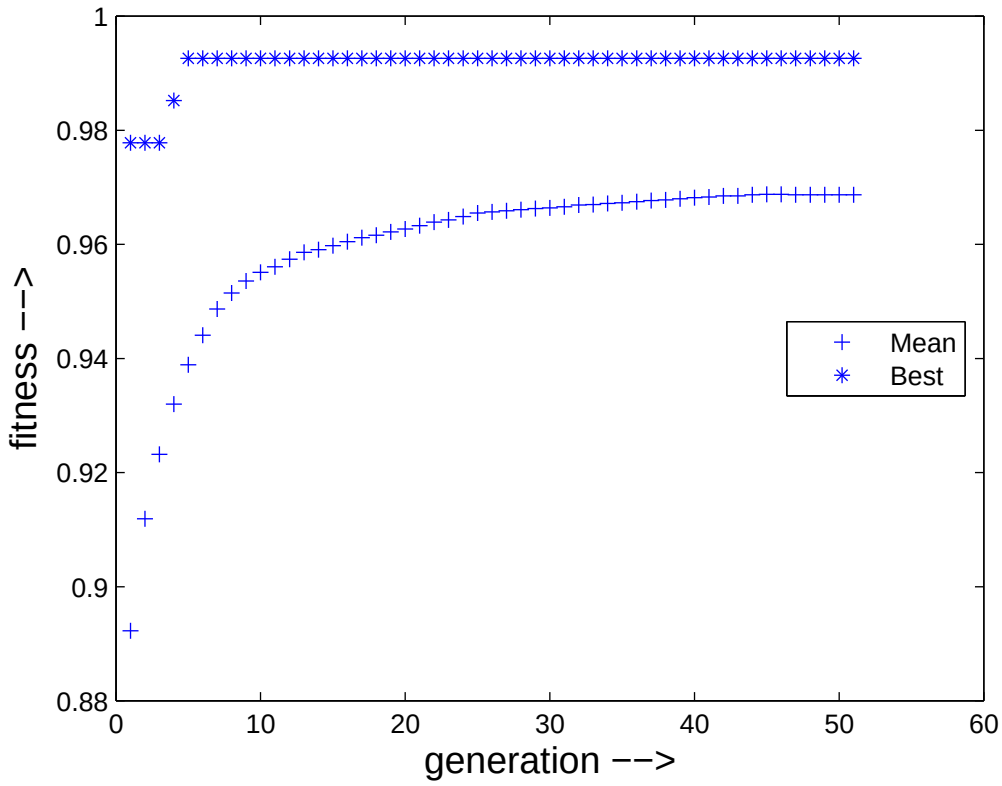


Figure 5.7: Best fitness and Mean fitness over generation for IRIS: all FKM prototypes

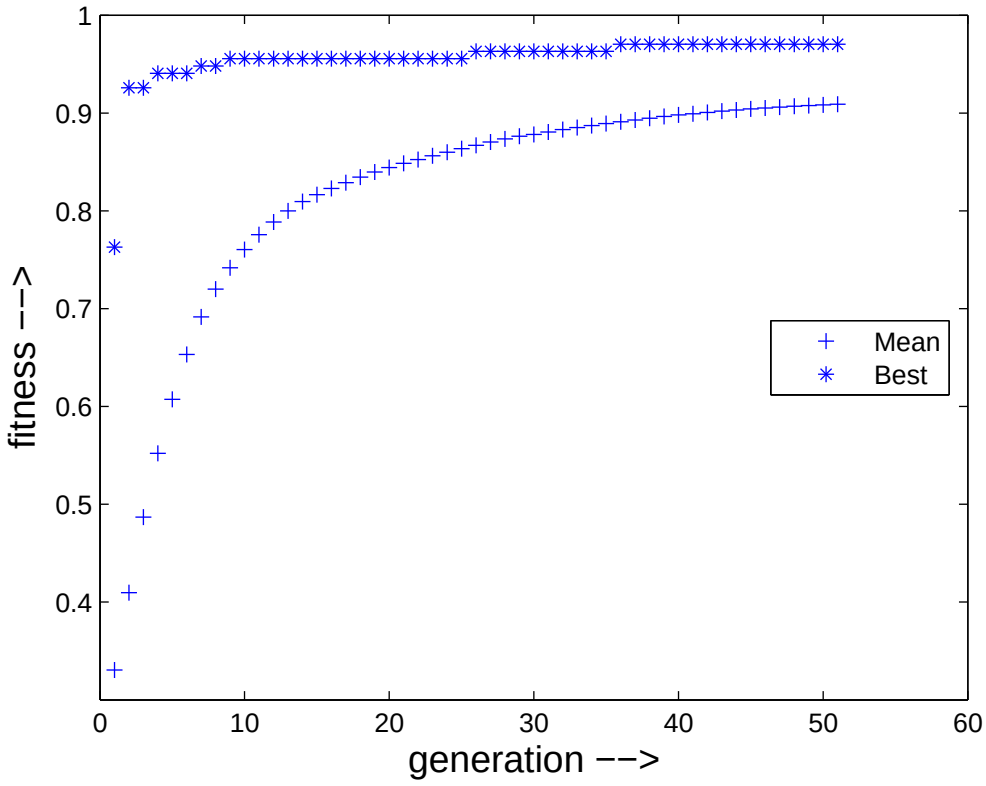


Figure 5.8: Best fitness and Mean fitness over generation for IRIS: all Random prototypes

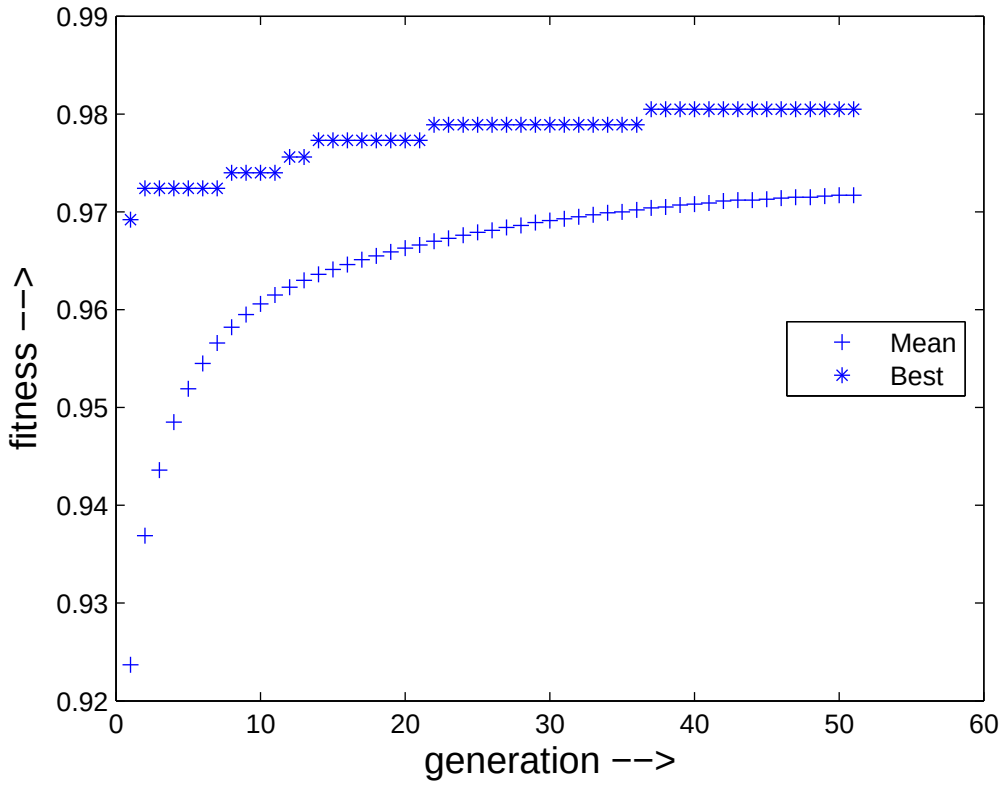


Figure 5.9: Best fitness and Mean fitness over generation for WBC: all FKM prototypes

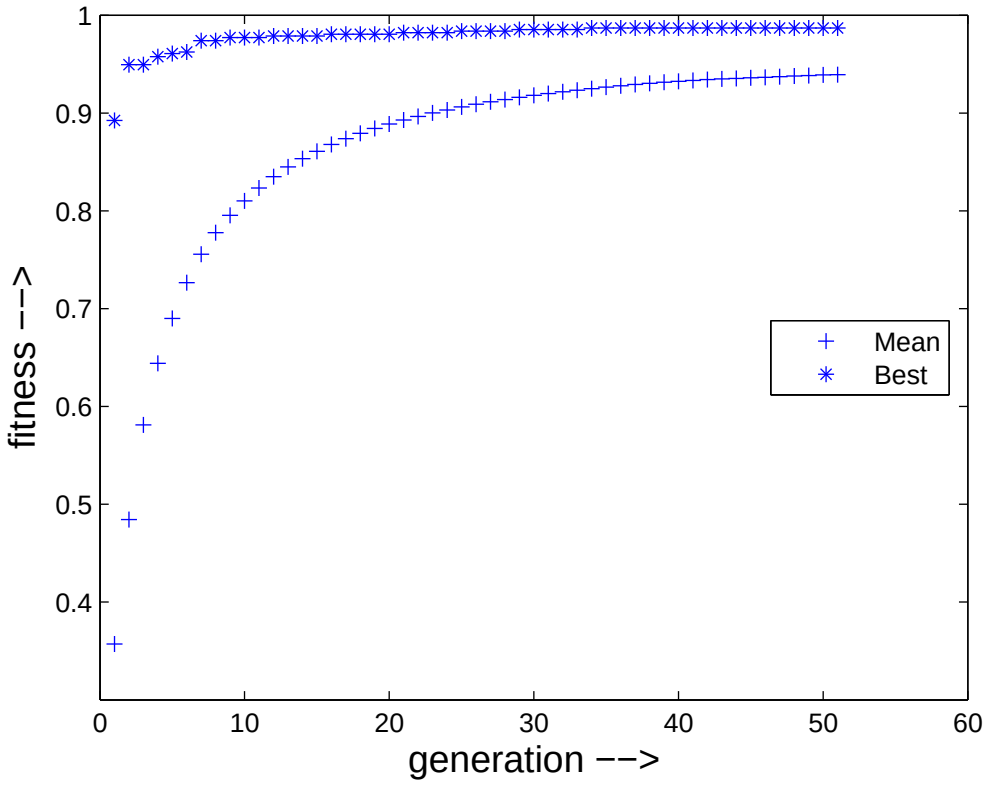


Figure 5.10: Best fitness and Mean fitness over generation for WBC: all Random prototypes

R2: **If** x2 CLOSE $\text{TO}_{\sigma_{i2}=0.28}$ 3.70 AND x4 CLOSE $\text{TO}_{\sigma_{i4}=0.12}$ 0.27 **then** *Class 1*

Class 2:

R1: **If** x2 CLOSE $\text{TO}_{\sigma_{i2}=0.26}$ 2.59 AND x4 CLOSE $\text{TO}_{\sigma_{i4}=0.15}$ 1.19 **then** *Class 2*

R2: **If** x1 CLOSE $\text{TO}_{\sigma_{i1}=0.35}$ 6.34 AND x2 CLOSE $\text{TO}_{\sigma_{i2}=0.28}$ 2.94 AND x3 CLOSE $\text{TO}_{\sigma_{i3}=0.23}$ 4.60 AND x4 CLOSE $\text{TO}_{\sigma_{i4}=0.14}$ 1.45 **then** *Class 2*

R3: **If** x1 CLOSE $\text{TO}_{\sigma_{i1}=0.87}$ 4.58 AND x2 CLOSE $\text{TO}_{\sigma_{i2}=0.28}$ 3.46 AND x3 CLOSE $\text{TO}_{\sigma_{i3}=0.48}$ 4.53 **then** *Class 2*

Class 3:

R1: **If** x2 CLOSE $\text{TO}_{\sigma_{i2}=0.27}$ 3.14 AND x4 CLOSE $\text{TO}_{\sigma_{i4}=0.15}$ 2.04 **then** *Class 3*

R2: **If** x3 CLOSE $\text{TO}_{\sigma_{i3}=0.52}$ 5.97 AND x4 CLOSE $\text{TO}_{\sigma_{i4}=0.54}$ 1.21 **then** *Class 3*

In the above rules, the parameters are rounded to 2 digits. When we apply the above classifier on all points of Iris data set, it could correctly classify all but one point.

In [116], GP was used to develop fuzzy classifier rules. For comparison with this available GP method, named G^3P, we have conducted the third experiment. For G^3P, authors have partitioned the data sets including Diabetes and WBC into 3 sets: training set with 50% samples, validation set with 25% samples and test set with 25% samples. Since we have not used validation set, we have considered 75% samples for training and 25% samples for test. We selected the samples for test in the same maner as described in [116]. Then we run GP with different number of rules using different types of prototypes. For each case, GP was run 20 times and the test errors were calculated. Tables 5.8 and 5.9 show the best test error, average and standard deviation of errors over 20 GP runs for WBC and Diabetes data sets respectively. In the first case, trees were generated with 2 rules per class using only FKM prototypes. In the second case, a tree is initialized with 5 rules using FKM prototypes with probability 0.25 and random prototypes with probability 0.75. In the third case, GP evolution was started with 10 rules per tree with only random prototypes. For the WBC data, with different combinations of FKM and random prototypes, we have obtained the best error rates in the range 1.17 %-1.75% and average error rates in 2.3% - 3.66%.

The reported best and average error rates obtained using G^3P were 2.29% and 4.39% respectively. For the diabetes data, the best error varied between 19.27% to 20.83% and average error was in 23.16% - 23.30% as shown in Table 5.9. In [116], the best error reported using G^3P is 21.98% and average error is 26.47%. For both data sets, our GP method performed better than the G^3P method.

Table 5.8: Test Error for WBC for comparison

FKM:Rand	Max Rules	Best	Avr	SD
1:0	2	1.75	2.81	0.438
0.25:0.75	5	1.17	2.3	0.45
0:1	10	1.75	3.66	0.89

Table 5.9: Test Error for Diabetes for comparison

FKM:Rand	max Rules	Best	Avr	SD
1:0	2	19.27	23.16	1.71
0.25:0.75	5	20.83	23.30	1.71
0:1	10	19.79	23.23	1.72

Table 5.10: Reported test error in other GP work [116]

Data Sets	Best	Avr	SD
WBC	2.29	4.39	1.42
Diabetes	21.98	26.47	3.40

5.4 Conclusion

We have used Genetic Programming to develop fuzzy classifier rules. For a c-class problems, a classifier was represented by c trees. Each tree T_k constituted a set of rules for class k. Our GP methodology, GP_{fc}, provided the required fuzzy classifier by

determining: 1) number of rules, 2) features(antecedent clauses) used in a rule, and 3) values of the parameters involved.

The initial rule base was generated using randomly generated prototypes, FKM prototypes and a mixture of the two. In this context, we have proposed a new mutation operation to alter the parameters. We have also introduced a scheme to remove inaccurate and inactive rules. The method was validated on five benchmark data sets. It is observed that the performance of the evolved classifiers is better when we use a mixture of FKM and random prototypes to initialize the rule base, over the two extreme cases with only FKM and only random prototypes. It is interesting to note that of the two extreme cases, use of only random prototypes is found to work better than the use of only FKM prototypes in most cases. Performance of the fuzzy rule based classifier is as good as the performance of our proposed classifier scheme proposed in chapter 3 in addition to the interpretability of fuzzy rules.

There are many avenues where further investigation is needed. For example, such a method can be used for system identification task. For a few parameters required by the method either we have used fixed values or chosen some values based on a few trials. All these parameters can be chosen using a validation set.

Chapter 6

Texture Generation and Classification using Genetic Programming [A4, A5, A6]

6.1 Introduction

In the previous chapters, we studied the role of Genetic Programming for classifier design and feature selection. There we have developed general purpose methodologies. In this classifier we focus on a specific application area, texture generation and classification. In particular, in this chapter, we have used GP to generate textures and we have applied our GP based classifier schemes (developed in the previous chapters) to classify textures.

Texture [113] is a property of the surface or structure of an object. Despite wide use of texture, there is no well accepted definition of texture. Texture describes the appearance and feel of a surface which consists of somewhat mutually related elements(primitives). It may be described by two components: primitives (local properties), out of which the texture is composed of and the mutual relationship among the primitives [113]. Textures may be natural or synthetic. Images of grass, dog fur, pebbles are some examples of natural texture. Computer or human generated textures are called synthetic textures. Computer generated textures are in great demand in fields like art, fashion and textile designing, animation, and so on.

A texture produced by an algorithm or a procedure is called a procedural texture [37]. For each point of the procedural texture, the procedure gives the corresponding gray level or color value. Although the procedural representation is extremely compact compared to the image representation, it is difficult to find/write procedures that will

generate textures for some target application. Evolutionary algorithm [46, 72] is a possible solution to this major problem. We can easily generate interesting textures using evolutionary algorithms. Typically, each solution of the EA population is a procedure to produce a texture. A fitness value is assigned to each solution indicating how the solution is fit for the problem. In texture generation problem, a higher fitness value is assigned to a procedure that produce better texture. However, As there is no precise definition of texture, so an automatic evaluation of textures is not an easy task. In comparison, a human being can easily identify and assess a texture. If we allow computer to generate textures and user to determine which textures are good according to his/her choice in the process of generation, then interesting textures can be created. Generation of textures based on this principle is called interactive texture generation.

Sims [112] has used evolutionary algorithms to evolve lisp expressions of procedures to produce interesting images, textures and animation. An interactive (procedural) texture generation scheme called Genshade is available in [59]. However, interactive texture generation could be a tedious process if a user needs to assign a fitness value to every generated texture. Consequently some methods generate textures similar to a(or a set of) given reference texture(s). This type of GP based schemes are presented in [120] which evolve procedures to produce textures similar to a given reference texture. This is an interesting approach but requires reference texture(s) and also produces only similar textures with respect to the given reference texture(s). But, the most optimal texture will be identical to the given reference texture. A review of evolutionary design by computer is available in [11].

From the above discussion, we may realize that interactive approach can generate very interesting textures according to user's choice as compared to the second approach. But it will be a tedious job if the user has to assign fitness value to each procedure. Hence, we need to device a hybrid approach that can combine the advantages of both approaches. This motivated us to propose a new approach to generate textures. Our proposed approach integrates features of both interactive and automatic approaches. This produces a variety of interesting textures according to user's choice in an interactive approach but with much reduced burden on user.

Our scheme *occasionally* seeks user's intervention to evaluate the generated textures during the evolutionary process. To reduce burden over the user, the generated textures

are passed through a filtering process and then through a clustering scheme. We use contrast of each generated image/texture to filter out very poor textures such as images with not much variation in intensities. After this, a generated texture is placed in a cluster of existing generated textures which is more *similar* to the generated texture and a fitness value related to the cluster is assigned to the new texture. If the new generated texture is quite *dissimilar* from the existing textures, then our scheme displays the new texture to the user to assign a fitness value by visually examining the goodness of the texture. We use statistical features to compute the *similarity* between two textures.

In addition to texture generation, we have also used our GP based classifier systems of the previous chapters to classify textures. In texture classification, a given image/texture is assigned to its predefined texture category. Texture classification is used in fields like remote sensing image analysis, medical image analysis, and document processing. As we have already proposed methods for classifier design using GP, so we have not introduced any new classifier design approach for texture classification. However, before applying our GP classifiers, it is required to represent each texture by a vector of features. The simplest approach to represent a (grey) texture by its two dimensional grey value matrix. But it will merely represent the texture as an image and a large dimensional feature vector containing the gray value of pixels will represent the texture. Thus, we need to extract such features from a texture that can represent the property of the texture. For texture classification/segmentation, using such features we can classify or group *similar* textures into one class or group and discriminant *dissimilar* textures into different classes/groups. Laws [**?**] observed the following properties that play crucial roles in describing texture: uniformity, density, coarseness, roughness, regularity, linearity, directionality, frequency and phase. Some of these qualities are interrelated. There are different approaches to represent a texture [117]. We have considered statistical approach to extract features that can capture important characteristic of texture and it is comparatively easy to use. We have used our classifier system GP_{mt} and fuzzy rule based classifier scheme GP_{fc} to classify natural textures.

We have presented our texture generation and classification schemes in section 5.2 and 5.3 respectively.

6.2 Proposed Texture Generation using GP

For each point of the procedural texture, the procedure gives the corresponding grey value or color value. In this work, we use GP to generate tree representation of procedures to produce greyscale (procedural) textures. The steps of our modus operandi are described in subsequent sections.

6.2.1 Initialization

A population of P trees is generated randomly using a set of functions $F = \{+, -, *, /, Log, Sin, Cos\}$ and a set of terminals $S = \{i, j, R\}$, where i, j are co-ordinate variables in 2-dimensional space and R contains randomly generated constants in [0.0, 100.0]. The sinusoidal functions *Sin* and *Cos* are useful to generate repeated patterns in a texture. Variations in images are desired to create interesting textures. However, the high frequency components of an image cause *aliasing* [37]. Aliasing creates flaws and unpleasant artifacts in the image. One such example is the staircase-like patterns in the image. High frequency components may be reduced to a certain extent using smoother functions like logarithm. However, we should be discreet in using logarithmic function as the function itself creates high frequency components. We define the functions as follow:

Sin(x)
{
  return $255 \times$ *sin(x)*
}

Cos(x)
{
  return $255 \times$ *cos(x)*
}

Log(x)
{

if x **greater than** 0 return $log_e\ (x)$.
else if x **less than** 0 return $log_e\ (-x)$.
else return -15.
}

Each tree $T_l(i,j), l = 1, 2, ..., P$ of the population represents a procedure to generate a texture. Let $m_r \times m_c$ be the size of the textures we are intending to generate. For each point $z(i,j)$ of an $m_r \times m_c$ matrix A_l, the tree T_l returns a real value ${a^l}_{ij}$. We associate an $m_r \times m_c$ matrix B_l to each A_l, such that each coefficient ${b^l}_{ij}$ is a grey-label, that is, ${b^l}_{ij} \in \{0, 1, 2, ..., G\}$ and is defined as,

$$ {b^l}_{ij} = \frac{{a^l}_{ij} - min_l}{max_l - min_l} \times G \tag{6.1} $$

Where G is the maximum grey-label (say G = 255) and max_l and min_l are the maximum and minimum co-efficients of the matrix A_l respectively.

Note that, a grey image is nothing but a 2-dimensional (grey-label) matrix and a color image is a set of three 2-dimensional matrices. Thus, this matrix B_l is simply a greyscale image produced by the tree T_l. We expect the generated image B_l to be a texture. In terms of genetic systems, a tree T_l is a genotype (or chromosome) and the image/texture B_l is its corresponding phenotype.

If we will consider the entire texture generation system as a pattern recognition system (PRS), then this phase resembles data acquisition/preprocessing phase of the PRS.

6.2.2 Filtering of Textures

Some of the generated images/textures may be very poor due to very limited intensity variation. For example, images with constant grey label. Of course, we do not desire to have these images. So if we assign negligible fitness values to procedures that have generated images with very low intensity variation then these procedures will not survive the process of evolution. This motivated us to calculate (normalized) contrast c_l of the image B_l and assign this value to procedure(tree) T_l as its fitness, if c_l is less than a minimum contrast value c_{min}. If $c_l \geq c_{min}$, then we assign a user defined fitness value to T_l as described in the following sections. The normalized value of contrast

[48], c_l is computed as follow:

$$Contrast = \frac{1}{G^2} \sum_i \sum_j (i-j)^2 P_{\phi,d}(i,j) \tag{6.2}$$

Where G is the maximum grey label and $P_{\phi,d}$ is the co-occurrence matrix with displacement d and direction ϕ [48]. The matrix describes how frequently two pixels with gray-levels i and j appear in the image separated by a distance d in direction ϕ.

6.2.3 Computation of feature vector

We compute a vector of features $\mathbf{v_l}$ for each generated texture B_l whose contrast is higher or equal to the minimum desired contrast value c_{min}. There are many methods to extract features from a texture [117]. Statistical methods compute different properties and are suitable if texture primitive sizes are comparable with the pixel sizes [113]. Our GP system uses mathematical functions that usually generate texture of small primitives. Moreover, it is comparatively easy to extract a feature vector using statistical approach. So, we use statistical approach to extract features. Two important categories of statistical features are co-occurrence based features and auto-correlation coefficients. We compute four co-occurrence based features [113] and auto-correlation coefficients [113] with different displacements and directions. The four co-occurrence based features are contrast, entropy, inverse difference moment and correlation with displacements d = 2,4,6 and 8 along directions ϕ = 0 and 90 degrees. The autocorrelation coefficients are computed with displacements (0,2),(0,4),(0,6),(0,8), (2,0),(4,0),(6,0) and (8,0). The above computed 32 co-occurrence based features and 8 autocorrelation coefficients(features) of the image B_l together produce the feature vector $\mathbf{v_k}$ with $n = 40$ components. Let $P_{\phi,d}$ be the (normalized) co-occurrence matrix. Then the Co-occurrence matrix based features are computed as follows [113]:

$$Entropy = \sum_i \sum_j P_{\phi,d}(i,j) log_2 P_{\phi,d}(i,j) \tag{6.3}$$

$$Contrast(normalized) = \frac{1}{G^2} \sum_i \sum_j (i-j)^2 P_{\phi,d}(i,j) \tag{6.4}$$

$$Inverse\ difference\ moment\ = \sum_i \sum_{j(\neq i)} \frac{P_{\phi,d}(i,j)}{(i-j)^2} \tag{6.5}$$

$$Correlation = \frac{\sum_i \sum_j [ij\ P_{\phi,d}(i,j)] - \mu_x \mu_y}{\sigma_x \sigma_y} \tag{6.6}$$

where μ_x, μ_y are means and σ_x, σ_y are standard deviations:

$$\mu_x = \sum_i i \sum_j P_{\phi,d}(i,j)$$

$$\mu_y = \sum_j j \sum_i P_{\phi,d}(i,j)$$

$$\sigma_x = \sum_i (i-\mu_x)^2 \sum_j P_{\phi,d}(i,j)$$

$$\sigma_y = \sum_j (j-\mu_y)^2 \sum_i P_{\phi,d}(i,j)$$

The autocorrelation coefficient $a_c(p,q)$ is computed as follow [113]:

$$a_c(p,q) = \frac{m_r m_c}{(m_r-p)(m_c-q)} \frac{\sum_{i=1}^{m_r-p} \sum_{j=1}^{m_c-q} f(i,j) f(i+p, j+q)}{\sum_{i=1}^{m_r} \sum_{j=1}^{m_c} f^2(i,j)} \tag{6.7}$$

where p,q is the position difference in the i, j direction, m_r, m_c are the image dimensions, and $f(i,j)$ is the function value (gray value) at (pixel) position (i,j). This can be considered as the feature extraction/analysis phase of the PRS.

6.2.4 Fitness

If B_l is the first generated texture whose contrast value $c_l \geq c_{min}$, then we display B_l for the user to assign a fitness value f_l(a non-negative real value) according to his/her assessment. Now we assume B_l to constitute a single element cluster. The feature vector $\mathbf{v_l}$ will represent the center,O_1, of this first cluster U_1. And, the user defined fitness value f^l is considered the fitness ${f_1}^u$ (say) of the first cluster U_1.

However, for subsequent generated textures we use the following steps. If contrast c_l of a generated texture B_l is higher or equal to c_{min}, then we calculate the distances between the feature vector $\mathbf{v_l}$ (of the texture B_l) and center of each existing cluster.

Let c be the number of existing clusters. Let ${D_k}^l, k \in \{1, 2, \cdots, c\}$ be the distance between feature vector $\mathbf{v_l}$ and center O_k of the existing cluster $U_k, k = \{1, 2, \cdots, c\}$. We compute the distance as follow:

$$ {D_k}^l = \sqrt{\frac{\sum_{g=1}^{n} w_g \left(\frac{O_{kg} - v_{lg}}{O_{kg} + v_{lg}}\right)^2}{\sum_{g=1}^{n} w_g}}. \tag{6.8} $$

Where n is the total number of features and w_g is a weight assigned to the g_{th} feature. We have considered $w_g = 1$ for co-occurrence based features and $w_g = 4$ (32/8) for autocorrelation features. The weights are chosen to realize as if we are using equal number of features from both categories.

Let ${D_h}^l$ be the minimum distance among all c distances ${D_k}^l, k \in \{1, 2, \cdots, c\}$. If the minimum distance ${D_h}^l$ is less than a maximum allowed distance D_{max}, then we include B_l into the h_{th} cluster U_h and assign the fitness ${f_h}^u$ of the h_{th} cluster to the procedure T_l. After this, we update the cluster center of h_{th} cluster center as follow:

$$ {\mathbf{O_h}}^{new} = \frac{n_h {\mathbf{O_h}}^{old} + \mathbf{v_l}}{n_h + 1}. \tag{6.9} $$

Here n_h is the number of textures in the h_{th} cluster. However, if the minimum distance ${D_h}^l \geq D_{max}$, then the texture B_l is considered as a new texture which is different from the other existing textures. So we create a new cluster U_{c+1} with the texture B_l. This new texture B_l is displayed to the user to decide a fitness f_l (≥ 0.0) for B_l. Then f_l is assigned as the fitness to the procedure T_l and also as the fitness ${f^u}_{c+1}$ to the new cluster U_{c+1}. As the new cluster U_{c+1} now contains only one texture B_l, $\mathbf{v}_l$ will represent the cluster center O_{c+1}.

The process of fitness assignment and clustering is continued for each procedure $T_l(i, j)$ of the population. After this process of evaluation, we apply genetic operations (reproduction, crossover and mutation) on the population as described in chapter 2. As there is no target to stop the evolution, we continue this process up to a predefined number of iterations, M. This clustering process can be considered as the classification/clustering phase of the PRS. During this process of evolution, our scheme generates a large number of procedures and their corresponding (procedural) textures.

The block diagram of the GP system is given in Figure 6.1. The (phenotype) steps

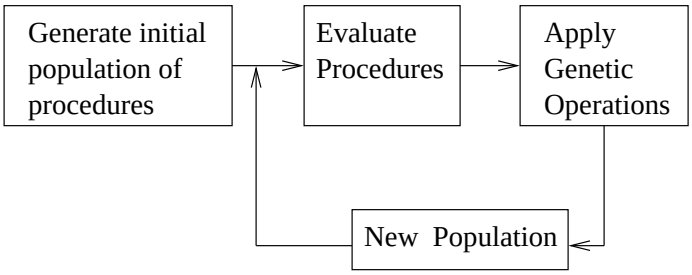


Figure 6.1: Block diagram of the GP system

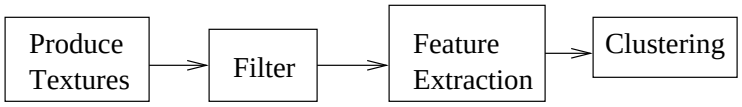


Figure 6.2: Evaluation of procedures on the basis of their produced image/texture

to evaluate procedures are given in Figure 6.2 that resembles a pattern recognition system.

6.2.5 Performance of Texture Generation Method

We have used *lilgp* [124] to perform our experiment. The GP parameters used are listed below:
Population size = P = 100
Maximum number of generations = M = 5
Probability of reproduction operation = $p_r = 0.1$
Probability of crossover operation = $p_c = 0.7$
Probability of mutation operation = $p_m = 0.2$
Maximum allowed height of a tree = 8
Maximum allowed number of nodes for a tree = 200

The Problem dependent parameters are:
Minimum Contrast value = $c_{min} = 0.03$
Maximum allowed distance = $D_{max} = 0.25$

Values of both problem dependent parameters are determined after conducting a couple of experiments. If value of D_{max} is large, then fewer clusters are created. This reduces burden on the user for repeatedly assigning fitness values to newly generated textures. However, it may allow dissimilar textures to be in the same cluster. On the other hand, if value of D_{max} is small then many clusters containing similar textures will be created. This will assign the same fitness value to only almost identical textures. However, the user will be asked repeatedly to assign fitness values to newly generated textures. So the choice of D_{max} is very important. We have made a few trial runs with different choices of D_{max} and based on the results we decided on $D_{max} = 0.25$.

Based on our experiments, we observed that, typically in the first generation of GP run, many generated procedures produced images with almost constant gray label. Our filtering scheme assigned very small fitness values to these procedures so that these did not survive the process of evolution. After that, many interesting textures were produced. The proposed clustering scheme was able to assign fitness values to many procedures. Occasionally user's input was sought to assign fitness values to some new procedures. Almost all textures in a cluster were similar. Only in a few cases, fewer textures in a cluster were not very similar. We noticed that instead of running GP for large number of iterations, if it is run for fewer iterations but for many times with different initial GP populations, then many more interesting textures could be produced.

We obtained many interesting textures (and corresponding procedures). We have included only 54 textures produced by our GP system in Figures 6.3 - 6.5. Each figure consists of 9 textures.

As an illustration, we show four evolved procedures and their corresponding textures in Figs. 6.6 and 6.7.

Procedure 1 (Figure 6.6(a))

$$Cos(j) * Cos(i * j)$$

Procedure 2 (Figure 6.6(b))

$$Sin\left(\frac{i*j}{2.5}\right)$$

Procedure 3 (Figure 6.7(a))

$$Cos(14.765\ i * j)$$

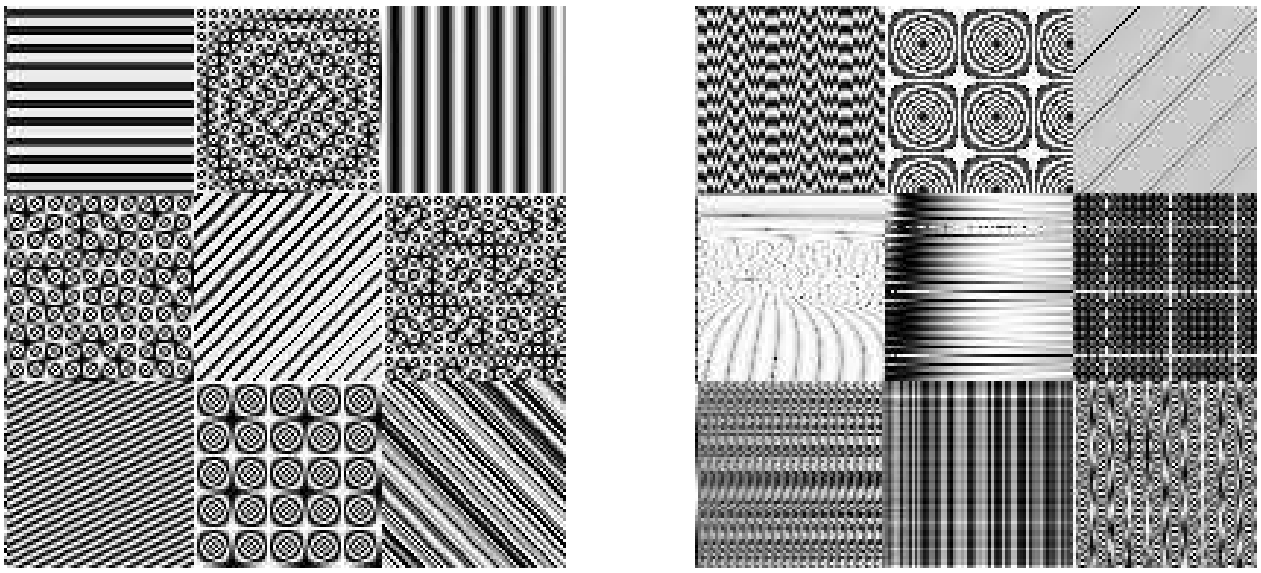


Figure 6.3: Generated Textures by our GP system

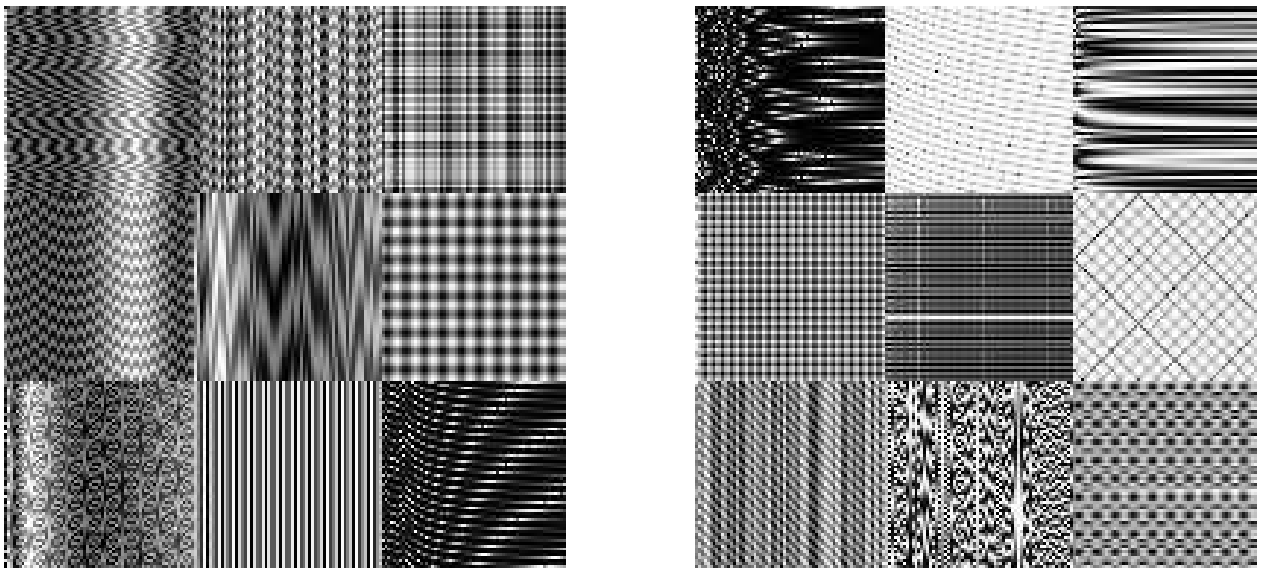


Figure 6.4: Generated Textures by our GP system

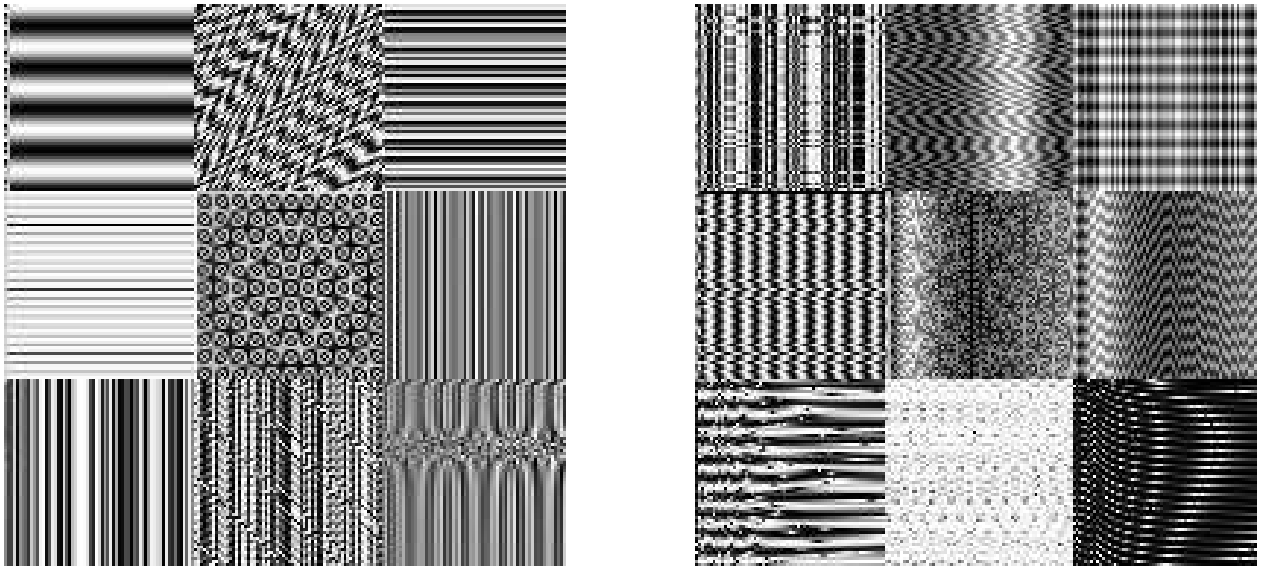


Figure 6.5: Generated Textures by our GP system

(1) (2)

Figure 6.6: Textures produced by the given procedures 1 and 2

Procedure 4 (Figure 6.7(b))

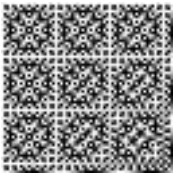
$Cos(93.5 + j) + Cos(Sin(i))$

These textures are example candidates that can be used for textile and fashion design.

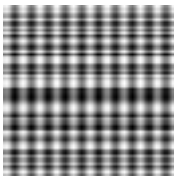
6.3 Texture Classification using Genetic Programming

In this section, we have used our proposed GP based classifier schemes GP_{mt} and GP_{fc} (fuzzy rule based classifier)to classify textures.

Texture classification is one of the problem domains in the field of texture analysis.



(3)



(4)

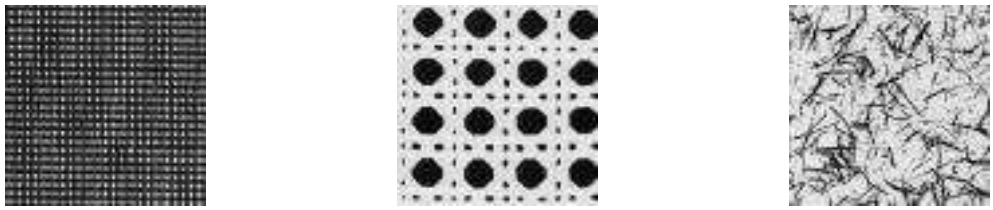
Figure 6.7: Textures produced by the given procedures 3 and 4

Texture has no precise (mathematical) definition. It also depends on orientation, scale and other visual appearances. So texture analysis is a challenging task. Tuceryan and jain [117] have categorized the various texture analysis methods into four categories: statistical, geometrical, model-based and signal processing. In this investigation, we have considered statistical approach to represent textures. Please note that a successful classification requires an efficient description of image texture. Among statistical approaches to represent texture, co-occurrence features [55] are widely used. So, we have considered co-occurrence matrix based features in addition to auto-correlation and edge frequency features to represent texture.

We have taken 3 sets of natural textures [17] to conduct 3 texture classification experiments. Each natural texture represents a class. Experiment 1 is a 3-class problem. Figure 6.8 shows three textures (types) representing 3 classes. Figure 6.9 shows textures considered for experiment 2. Four classes of textures are considered for the third experiment and the corresponding textures are shown in Figure 6.10.

6.3.1 Preparation of Data Sets

Each given natural texture is partitioned into small textures to generate a set of texture patterns for training set and test set. The natural textures for the experiments have dimension equal to or more then 640×640. Hundred textures of dimension 64×64 are created by partitioning each given natural texture. Then a vector of statistical features($\mathbf{x}$) is extracted from each 64×64 texture. Now each texture is represented by the corresponding statistical feature vector $\mathbf{x}$. In this manner, 100 patterns for each class (given natural texture type) are created to develop and validate the classifier. We extract following statistical features [113].



(Class 1) (Class 2) (Class 3)

Figure 6.8: Textures for Experiment 1

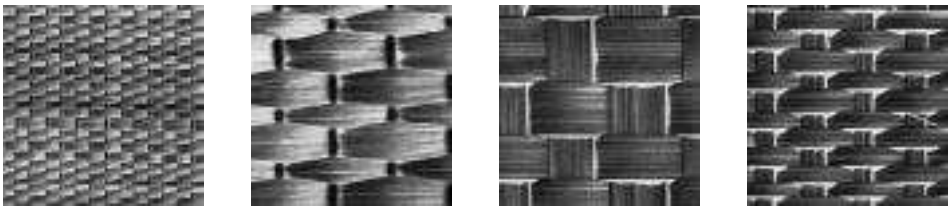
(Class 1) (Class 2) (Class 3)

Figure 6.9: Textures for Experiment 2

1. Co-occurrence based features: These are based on repeated occurrence of same gray-level configuration in the texture. We compute co-occurrence matrices in direction 0 and 90 with displacements 2 and 4. Each of these 4 matrices is used to calculate contrast, entropy and correlation features. This creates $4 \times 3 = 12$ co-occurrence based features.

2. Autocorrelation coefficients: These pertain to spatial frequencies in an image/texture. For each texture, autocorrelation coefficients with displacements (0,2), (0,4), (2,0) and (4,0) are computed. This provides 4 autocorrelation coefficients.

3. Edge Frequency: Comparison of edge frequencies in texture can be used to discriminate textures. We compute edge frequency of each texture to include as another feature. The above 17 features are used to represent each texture. A subset of 100 patterns from each class is kept to test the classifier and the remaining patterns (training set) are used to train the algorithm.



(Class 1) (Class 2) (Class 3) (Class 4)

Figure 6.10: Textures for Experiment 3

6.3.2 Classification Results

We run both GP based schemes GP_{mt} and GP_{fc} ten times for each experiment. Each GP run involves a 10-fold cross-validation (Thus, each GP run consists of 10 runs, each with one of the 10 folds, so total 100 runs). In each GP run, the test accuracy (the percentage of correctly classified test patterns) is computed. The average and standard deviation of test accuracies over all 10 GP runs are calculated in each experiment. For comparison, we applied the 1-nn (nearest neighbor) classifier on the texture data sets. Here also we used the 10-fold cross validation. In the first experiment, the average test accuracies are 99.6% (sd = 0.16) and 99.1% (sd = 0.13) using GP_{mt} and GP_{fc} respectively. That means the developed GP classifiers could correctly classified almost all test patterns. In average, 1-nn classifier classified 98.3% (sd = 0.08) test patterns correctly. In the second experiment, we obtained 92.2% (sd = 0.41) and 91.6% (sd = 0.20) average test accuracies respectively by GP_{mt} and GP_{fc}. This degradation of performance may be due to similarity of patterns in class 2 and class 3. For this experiment, we obtained only 81% (sd = 0.14) average test accuracy using the 1-nn classifier. In the third experiment, there is the similarity between classes is much stronger. In this cases, it is difficult to discriminate between classes. But, our non-fuzzy and fuzzy GP classifier systems are able to correctly classify 84% (sd = 1.22) and 85.4% (sd = 0.45) test samples (in average) respectively. For this difficult data set, the average test accuracy using 1-nn is 61% (sd = 0.34) only.

6.4 Conclusion

We have presented a Genetic Programming based scheme to generate procedural textures. The ability of genetic programming to evolve procedures to generate textures and the human ability to judge texturedness are blended in a judicious manner. To reduce the burden over the user drastically, the generated textures are passed through a filtering process and then through a clustering scheme. We use contrast of each generated texture to facilitate the filtering process. The textures that survive the filtering are placed in the clusters of existing generated textures based on their *similarity* with existing clusters. If a new generated texture is significantly *dissimilar* from the existing textures, then our scheme displays the new texture to the user to assign a fitness value by visual inspection. We use statistical features to find the *similarity* between two textures. Our GP system provides grey label textures. To generate color textures, we need to produce three matrices one for each of red, green and blue components of the texture. For this, each tree is required to return a vector (of 3 color values) for each texture point or combination of three trees may be used to produce a single color texture.

We have also used our GP based classifier schemes to evolve multi-tree classifiers for texture classification problems. We conducted experiments on 3 sets of well-known Brodatz natural textures. Statistical features of textures were extracted to represent them. Performance of GP classifier schemes were good in all 3 experiments. In experiment 1, the data set was comparatively easy as compared to experiment 3. In experiment 3, there was more similarity among classes and hence the accuracy was not very high. In this chapter, we found that the performance of fuzzy rule based classifier GP_{fc} is as good as that of GP_{mt}. In this investigation, we have considered statistical features. However, other features may be used to represent textures.

Chapter 7

Conclusion and Scope of further work

7.1 Conclusion

In this thesis, we proposed Genetic Programming based methodologies to solve certain pattern recognition tasks. These tasks were classifier design, on-line feature selection, texture generation and classification, and rule based fuzzy classifier design. The beauty of GP is that it provides the (mathematical) expression of the model (classifier) that can be analyzed. Except texture generation task, in all other works, a GP population of multi-tree chromosomes was evolved where each chromosome represented a classifier. For a c-class problem, a multi-tree classifier consists of c trees where each tree represents a classifier for a particular class. Our approach needs only a *single* GP run to evolve a classifier for a multi-class problem. Each tree of the multi-tree classifier recognizes patterns of a particular class and rejects patterns of other classes. Trees are independent of each other and each has an equal responsibility for classification, but all trees are tied together through fitness evaluation of chromosomes which governs the evolution of GP. In the case of crossover operation, we not only allow exchange of *subtrees* between trees meant for the same class, but also complete exchange of some *trees* designated for the same class. Our mutation operation is designed to reduce the destructive nature of conventional mutation.

In Chapter 1, we briefed about pattern recognition (PR), Evolutionary algorithms (EAs), motivation and scope of the thesis. In Chapter 2, we elaborated PR and EAs to provide necessary background to understand the thesis.

In Chapter 3, the GP based multi-tree classifier design method was proposed. The basic

concept of classifier design, presented in this chapter, has been used in other chapters except the texture generation scheme. An individual is selected according to its *fitness* value for genetic operation. However, its trees are selected according to their degree of *unfitness*. In this way, we give more opportunities to unfit trees to rectify themselves by genetic operations (crossover and mutation). At the same time, we reduce the chance of unwanted disruption of already fit trees by the genetic operations. To obtain a better classifier we have proposed a new scheme for OR-ing two classifiers. We have used a heuristic rule based scheme followed by a weight based scheme to resolve conflicting situations. The heuristic rules model typical situations where the classifier indicates ambiguous decisions and try to resolve them. The weight based scheme assigns a higher weight to a tree which is less responsible for making mistakes. We tested our classifier with several data sets and obtained quite satisfactory results.

In Chapter 4, the proposed classifier design method in the previous chapter was refined and modified so that we can do both feature selection and classifier design simultaneously. The initialization process generates classifiers using smaller feature subsets with higher probability. The fitness function assigns higher fitness values to classifiers which classify more samples using fewer features. The multi-objective fitness function helps to accomplish both feature selection and classifier design simultaneously. In this regard, we proposed two modified crossover operations suitable for feature selection. As a byproduct, we obtained a feature ranking method. The fitness function prevents the use of more features and hence helps to achieve more readability of the trees extracted by the system. Since the number of features to be used is not predefined, the algorithm has more flexibility.

The effectiveness of our scheme is demonstrated on seven data sets having dimensionality varying between 4 and 7129. Our experimental results established that the proposed scheme is very effective in selecting a small set of good features and finding useful classifiers. The obtained results reveal that the proposed method can achieve almost the same performance using a small set of features as that with all features. We have also demonstrated the effectiveness of our scheme on data sets with redundant or bad features added synthetically. We have compared the performance of our methodology with results available in the literature with both filter and wrapper type approaches. Wrapper (and on-line) FS algorithms perform better than filter approaches, at the cost of computational time. Our proposed algorithm performed better for both two class

and multi-class problems.

In Chapter 5, fuzzy rule based classifiers were evolved using GP. Each tree T_k of the multi-tree classifier constitutes a set of rules for class k. During evolutionary process, the inaccurate/inactive rules of the initial set of rules were removed by a cleaning scheme. This allowed only good rules to sustain and that eventually determined the number of rules. In stead of using all features in a rule, our GP scheme used only a subset of features and thereby determined the features used in a rule. The rules were constructed using prototypes. The prototypes were obtained by randomly generating the values and by using fuzzy K-means algorithm. We proposed a new mutation operation to alter the rule parameters. Hence, the GP scheme not only optimized the structure of rules but also optimized the parameters involved. This resulted in good fuzzy rule based classifiers. Moreover, the resultant fuzzy rules can be analyzed. The method was validated on six benchmark data sets. The performance of the GP Scheme was quite satisfactory. We observed that using both types of prototypes causes better performance as compared to using only one type of prototype. It is interesting to note that of the two extreme cases, use of only random prototypes works better than the use of only FKM prototypes in most cases. This emphasizes the novelty of the proposed scheme.

In Chapter 6, we presented Genetic Programming based schemes to generate and classify textures.The textures generated by procedures are called procedural textures. GP can evolve procedures to produce "good" textures. There is no well accepted definition of texture. So, it is not easy to assess a texture and assign a fitness value to it automatically. On the other hand, human can easily evaluate a texture. The ability of genetic programming to evolve procedures to generate textures and the human ability to judge textureness are blended in a judicious manner. If the user evaluate each produced texture during evolution, then it would be a very tedious/ boring job for the user. To reduce the burden over the user drastically, the generated textures are passed through a filtering process and then through a clustering scheme. We use contrast of each generated texture to facilitate the filtering process. The textures that survive the filtering are placed in the clusters of existing generated textures based on their *similarity* with existing clusters. If a new generated texture is significantly *dissimilar* from the existing textures, then our scheme displays the new texture to the user to assign a fitness value by visual inspection. We use statistical features to find

the *similarity* between two textures.

We used our both GP based classifier schemes GP_{mt} and GP_{fc} (fuzzy) to evolve multi-tree classifiers for texture classification problems. We conducted experiments on 3 sets of well-known Brodatz natural textures. Statistical features of textures were extracted to represent them. Performance of GP classifier scheme on textures was good in all 3 experiments.

The integrated approach to feature selection and classifier design can be easily adapted to regression/ function approximation/ forecasting problems. The concept of unfitness of trees can be used in case of multiple-input-multiple-output function approximate type systems.

7.2 Scope of the further work

In this thesis, we have addressed mainly two pattern recognition tasks: classification and feature selection. It would be interesting to use GP for clustering because GP can provide description of the clusters. In our methodologies, we have primarily focused on accuracy of the classifiers. Size of the tree too can be considered while evaluating solutions.

In Chapters 3 and 4, We have used arithmetic functions to design classifiers. So, our methodology is applicable to numerical attributes only. For data with nominal attributes, the logical functions like AND, OR , NOT may be considered instead of arithmetic functions.

In Chapter 5, we have proposed a GP scheme that evolves fuzzy rule based classifiers. Although our scheme does not use all features in each rule, we did not try to do feature selection along with rule generation. In this scheme, we may incorporate feature selection as we have done in Chapter 4. We may use validation set to find the values of the parameters.

Our GP system in chapter 6 generates provides grey label textures. This may be extended to color texture. To generate color textures, we need to produce three matrices one for each of red, green and blue components of the texture. For this, each tree is required to return a vector (of 3 color values) for each texture point or combination of

three trees may be used to produce a single color texture.

Different algorithms required selection of different parameters. Although, we have usually selected them by some intuitively plausible schemes, a better approach would be to use validation data for this.

Bibliography

[1] D.Agnelli, A.Bollini and L.Lombardi, " Image Classification: an evolutionary approach", *Pattern Recognition Letters*, vol. 23 pp. 303-309, 2002.

[2] M. Ahluwalia and L. Bull, "Coevolving functions in genetic programming", *journal of systems architecture*, vol. 47, no. 7, pp. 573-585, July 2001.

[3] R.Aler, D.Borrajo and P.Isasi," Using Genetic Programming to learn and improve control knowledge", *Artificial Intelligence*, vol. 141, pp. 29-56, October 2002.

[4] E. Anderson, "The Irises of the Gaspe peninsula", *Bulletin of the American IRIS Society*, vol. 59, pp. 2-5, 1935.

[5] T. Back, U. Hammel, and H.-P. Schwefel, " Evolutionary Computation: Comments on the History and Current State", *IEEE Transactions on Evolutionary Computation*, vol 1, no 1, pp 3 - 17, April 1997.

[6] T. Back and H.-P. Schwefel, " Evolutionary Computation: An overview", *Proceedings of IEEE International conference on Evolutionary Computation*, 1996.

[7] W.Banzhaf, P.Nordin, R.E.Keller and F.D.Francone,*Genetic Programming: an introduction*, Morgan Kaufmann Publishers,1998.

[8] A. Bastian, " Identifying fuzzy models utilizing Genetic Programming", *Fuzzy sets and systems*, vol 113, no 3, pp 333-350, 2000.

[9] K.W. Bauer, S.G. Alsing, and K.A. Greene, "Feature screening using signal-to-noise ratios",*Neurocomputing*, vol. 31, pp. 29-44, 2000.

[10] A. Ben-Dor, L. Bruhn, N. Friedman, I. Nachman, M. Schummer, and N. Yakhini, "Tissue classification with gene expression profiles",*J. Comput. Biol.*, vol. 7, pp. 559-584, 2000.

[11] P. Bentley, " Aspects of Evolutionary Design by Computers", *Advances in Soft Computing - Engineering Design and Manufacturing*, Springer-Verlag, London, pp 99-118,1999.

[12] J.C.Bezdek, J.Keller, R.Krisnapuram, and N.R. Pal, *Fuzzy Models and Algorithms for pattern recognition and Image Processing*, Kluwer Academic Publishers, 1999.

[13] J. Bi, K.P. Bennett, M. Embrechts, C.M. Breneman and M. Song, "Dimensionality reduction via sparse support vector machines", *Journal of Machine Learning Research*, vol. 1, pp. 1-48, 2002.

[14] C.L. Blake and C.J. Merz,*UCI Repository of machine learning databases*, University of California, Department of Information and Computer Science, 1998. http://www.ics.uci.edu/ mlearn/MLRepository.html

[15] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning", Special issue of *Artificial Intelligence* on 'Relevance', vol. 97, pp. 245-271, 1997.

[16] C.C.Bojarczuk, H.S.Lopes and A.A.Freitas, "Genetic Programming for knowledge Discovery in Chest Pain Diagnosis", *IEEE Engineering in Medicine and Magazine*, vol 19, no.4, pp 38- 44, 2000.

[17] P. Brodatz, *Textures: A Photographic Album for Artists and Designers*, Dover, New York, 1996.

[18] E. K. Burke and G. Kendall (Ed.), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Springer,2005.

[19] B. Carse, T. C. Fogarty, and A. Munro, "Evolving fuzzy rule based controllers using genetic algorithms", *Fuzzy Sets Syst.*, vol. 80, pp. 273 293, June 1996.

[20] J. Casillas, B. Carse, L. Bull, "Fuzzy-XCS: A Michigan Genetic Fuzzy System", *IEEE Trans. Fuzzy System*, vol. 15, Issue 4, pp 536 - 550, Aug 2007.

[21] J.Casillas, O.Cordon, M.J.Del Jesus and F.Herrera, " Genetic feature selection in a fuzzy rule-based classification system learning process for high-dimensional problems", *Information Sciences*, vol. 136, pp. 135-157, 2001.

[22] D. Chakraborty and N.R. Pal, " A Neuro-fuzzy scheme for simultaneous feature selection and fuzzy rule-based classification",*IEEE Transactions on Neural Networks*, vol. 15, No. 1, pp. 110-123, 2004.

[23] D. Chakraborty and N.R. Pal, "Integrated feature analysis and fuzzy rule-based system identification in a neuro-fuzzy paradigm", *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, vol. 31, No. 3, pp. 391-400, 2001.

[24] B.-C. Chien, J.Y. Lin and T.-P. Hong, " Learning discriminant functions with fuzzy attributes for classification using genetic programming", *Expert Systems with Applications* vol. 23, pp 31-37, 2002.

[25] S-B Cho, and j. Ryu, "Classifying gene expression data of Cancer using classifier ensemble with mutually exclusive features", *Proceedings of IEEE*, vol. 19, No. 11, pp. 1744-1753, 2002.

[26] M. Dash and H.Liu, "Feature selection for classification", *Intelligent Data Analysis*, vol. 1, no. 3, pp. 131-156, 1997.

[27] M. Dash, and H. Liu, "Consistency-based search in feature selection", *Artificial Intelligence*, vol. 151, pp. 155-176, 2003.

[28] P. Day and A.K. Nandi,"Binary String Fitness Characterization and Comparative Partner Selection in Genetic Programming", IEEE Trans. on Evolutionary Computation, vol. 12, no. 6, pp. 724-735, 2008.

[29] P. Day and A.K. Nandi, "Robust Text-Independent Speaker Verification Using Genetic Programming", *IEEE Trans. on Audio, Speech, and Language Processing*, vol. 15, no. 1, pp. 285-295, 2007.

[30] R. K. De, N. R. Pal and S. K. Pal, " Feature analysis: Neural network and fuzzy set theoretic approaches",*Pattern Recognition*, vol. 30, pp. 1579-1590, 1997.

[31] P.A. Devijver and J.Kittler, *Pattern Recognition: a statistical approach*, Prentice Hall, 1982.

[32] J.M.Diada, T.F.Bersano-Begey, S.J.Ross and J.F.Vesecky ," Computer-Assisted Design of Image Classification Algorithms: dynamic and Static Fitness Evaluations in a Scaffolded Genetic Programming Environment", *Genetic Programming 1996:Proceedings of the First Annual Conference*, The MIT press, pp 279-284.

[33] T.G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms", *Neural Computation*, vol. 10, No. 7, pp. 1895-1923, 1998.

[34] P. Domingos, "Context-sensitive feature selection for lazy learners", *Artificial Intelligence Review*, vol. 11, pp. 227-253, 1997.

[35] G. Dounias, A. Tsakonas, J. Jantzen, H. Axer, B. Bjerregaard, and D. Keyserlingk, "Genetic programming for the generation of crisp and fuzzy rule bases in classification and diagnosis of medical data", in *Proc. 1st Int. NAISO Congr. Neuro Fuzzy Technologies*, Canada, 2002, Academic Press, [CD-ROM].

[36] R. Duda and P. Hart. *Patern Classification and Scene Analysis*, Wiley Interscience, NY, New York, 1973.

[37] D. Ebert, F.K. Musgrave, D. Peachey, K. Perlin, and S. Worley, *Texturing and Modelling: a procedural approach*, Toronto: Academic Press, 1994.

[38] A.I. Esparcia-Alcazar and Ken Sharman," Evolving Recurrent Neural Network Architectures by Genetic Programming", *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp 89-94

[39] I.De Falco, A.Della Cioppa, E. Tarantino,"Discovering interesting Classification rules with Genetic Programming", *Applied Soft Computing* vol. 23, pp. 1-13, 2002.

[40] G. Folino, C. Pizzuti, G. Spezzano, "GP ensembles for large-scale data classification", *IEEE Trans. on Evolutionary Computation*, vol. 10, Issue 5, pp 604 - 616, Oct 2006.

[41] C.Fonlupt," Solving the Ocean Color Problem using a Genetic Programming Approach", *Applied Soft Computing*, Vol 1, pp 63-72,June 2001.

[42] G. Forman, "An extensive empirical study of feature selection metrics for text classification", *Journal of Machine Research*, vol. 3, pp. 1289-1305, 2003.

[43] H.Forrest, J.R.Koza, Y.Jessen and M.William," Automatic Synthesis, placement and routing of an amplifier circuit by means of genetic programming", *Evolvable systems: From Biology to Hardware, Proc. of the third international conference, ICES 2000*, LNCS, vol 1801, pp 1-10,

[44] K. Fukunaga, *Introduction to statistical pattern recognition*, Academic Press, 1972.

[45] T.S. Furey, N. Cristianini, N. Duffy, D. W.Bednarski, M.Schummer, and D. Haussler, "Support vector machine classification and validation of cancer tissue samples using microarray expression data",*Bioinformatics*, vol. 16, no. 10, pp. 906-914, 2000.

[46] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Pearson Education, 1989.

[47] T.R. Golub, D.K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J.P. Mesirov, H. Coller, M. L. Loh, J.R. Downing, M.A. Caligiuri, C.D. Bloomfield, E.S. Lander,"Molecular classification of Cancer: Class discovery and class prediction by gene expression monitoring", *Science*, vol. 286, pp. 531-537, 1999.

[48] R. C. Gonzalez and R.E. Woods, *Digital Image Processing*, 2^{nd} edition, pearson Education.

[49] R. P. Gorman, and T. J. Sejnowski, "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets", *Neural Networks*, vol. 1, pp. 75-89, 1988.

[50] H.F.Gray, R.J.Maxwell, I.Martinez-Perez, C.Arus and S.Cerdan, " Genetic programming for classification and feature selection: analysis of ^{1}H nuclear magnetic resonance spectra from human brain tumor biopsies", *NMR IN BIOMEDICINE*, vol. 11, pp. 217-224, 1998.

[51] L.Gritz and J.K.Hahn," Genetic Programming for Articulated Figure Motion", *Journal of Visualization and computer Animation*,vol 6, no 3,pp 129-142,1995.

[52] H.Guo, L.B.Jack and A.K. Nandi, "Feature generation using genetic programming with application to fault classification", *IEEE Trans. on Systems, Man and Cybernetics - Part B: Cybernetics*, vol. 35, No. 1, pp. 89-99, 2005.

[53] H. Guo and A.K. Nandi, "Breast cancer diagnosis using genetic programming generated features", *Pattern Recognition*, vol. 39, no. 5, pp. 980-987, 2006.

[54] I. Guyon and A. Elisseeff, "An introduction to variables and feature Selection", *Journal of Machine Learning Research*, vol. 3, pp. 1157-1182, 2003.

[55] R.M. Harlick, "Statistical and Structural approaches to Texture", *Proceedings of the IEEE*, vol. 67, pp 786 - 804, 1979.

[56] N.R.Harvey, S.P.Brumby, S.Perkins, J.Theiler, J.J.Szymanski, J.J.Bloch, R.B.Porter, M.Galassi and A.C.Young,"Image Feature Extraction: GENIE Vs Conventional Supervised Classification techniques", *IEEE Transactions on Geoscience and Remote Sensing*,vol. 4, no 2, pp 393-404, 2002.

[57] S. Haykin, *Neural networks: a comprehensive foundation*, Prentice Hall, 1994.

[58] F. Hoffmann, D. Schauten, S. Holemann, "Incremental Evolutionary Design of TSK Fuzzy Controllers", *IEEE Trans. Fuzzy Systems*, vol. 15, issue 4, Aug 2007, pp 563 - 577.

[59] A E. Ibrahim, "Genshade: an evolutionary approach to automatic and interactive procedural texture generation", Doctoral thesis, College of Architecture, A&M University, Texas, 1998.

[60] H. Ishibuchi, T. Nakashima, and T. Murata, "Performance Evaluation of Fuzzy Classifier Systems for multidimensional pattern classification problems", *IEEE Trans. on SMC-B*, vol. 29, pp 601-618, Oct. 1999.

[61] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, " Selecting fuzzy if-then rules for classification problems using genetic algorithms", *IEEE Trans. Fuzzy Syst.*, vol. 3, no. 3, pp 260 - 270, 1995.

[62] L.B. Jack and A.K. Nandi, "Genetic algorithms for feature selection in machine condition monitoring with vibration signals",IEE Proceedings- Part VIS, vol. 147, no. 3, pp. 205-212, 2000.

[63] C. Jacob, *Illustrating Evolutionary Computation with Mathematica*, Morgan Kaufmann Publishers, 2001.

[64] A. Jain and D. Zongker, "Feature selection: evaluation, application, and Small sample performance", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 19, no. 2, pp. 153-158, Feb. 1997.

[65] R. W. Johnson, M.E. Melich, Z. Michalewicz, M. Schmidt, "Coevolutionary optimization of fuzzy logic intelligence for strategic decision support", *IEEE Trans. on Evol. Compt.*, vol. 9, Issue 6, pp 682 - 694, Dec 2005.

[66] K.Kira and L.A.Rendell, "The feature selection problem: Traditional methods and a new algorithms", *Proceedings of ninth national conference on artificial intelligence*, pp. 129-134, 1992.

[67] J.K. Kishore, L.M.Patnaik, V.Mani and V.K.Agrawal," Application of Genetic Programming for Multicategory Pattern Classification", *IEEE Transactions on Evolutionary Computation,* vol.4, no.3, pp 242-258, 2000.

[68] G. J. Klir and Y. Bo, *Fuzzy sets and fuzzy logic: theory and applications*, Prentice Hall, 1995.

[69] R. Kohavi and G.H. John, "Wrappers for feature subset selection", *Artificial Intelligence*, vol. 97, pp. 273-324, 1997.

[70] A. Konar, *Computational intelligence: principles, techniques and applications*, Springer-Verlag, 2005.

[71] M.Koppen and B.Nickolay," Genetic Programming based Texture Filtering Framework",*Pattern recognition in soft computing Paradigm*, chapter 12, (Edited by Nikhil R. Pal), World Scientific.

[72] J.R.Koza,*Genetic Programming: On the programming of computers by means of natural selection.* Cambridge, MA, M.I.T.press 1992

[73] J.R.Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge, MA, 1994.

[74] M. Kudo and J. Sklansky, "Comparison of algorithms that select features for pattern classifiers", *Pattern Recognition*, vol. 33, pp. 25-41, 2000.

[75] A.S. Kumar, S. Chowdhury and K.L. Mazumder, "Combination of neural and statistical approaches for classifying space-borne multispectral data", *Proc. of ICAPRDT99, Calcutta, India*, pp. 87-91, 1999.

[76] N. Kwak and C.-Ho Choi, "Input feature selection for classification problems", *IEEE Transactions on Neural Networks*, vol. 13, no. 1, pp. 143-159, Jan. 2002.

[77] G.V. Lashkia, and L. Anthony, "Relevant, irredundant feature selection and noisy example elimination", *IEEE Transactions on Systems, Man, and Cybernetics-Part B:Cybernetics*, vol. 34, Issue 2, pp. 888-897, April 2004.

[78] M. Last, A. Kandel, and O. Maimon, "Information-theoretic algorithm for feature selection", *Pattern Recognition Letters*, vol. 22, pp. 799-811, 2001.

[79] Gwo-Ching Liao, Ta-Peng Tsao, "Application of a fuzzy neural network combined with a chaos genetic algorithm and simulated annealing to short-term load forecasting", *IEEE Trans. on Evol. Compt.*, vol 10, Issue 3, pp 330 - 340, June 2006.

[80] T.-S. Lim, W.-Y. Loh and Y.-S. Shih, " A Comparison of Prediction Accuracy, Complexity and Training Time of Thirty-three Old and New Classification Algorithms", *Machine Learning Journal*, Vol 40, pp 203-228, 2000.

[81] T.Loveard and V.Ciesielski, "Representing Classification Problems in Genetic Programming", *Proceedings of the Congress on Evolutionary Computation*, pp 1070-1077, IEEE Press, May 2001.

[82] O.L. Mangasarian, W.N. Street and W.H. Wolberg, "Breast Cancer diagnosis and prognosis via linear programming", *Operation Research*, vol. 43, No. 4, pp. 570-577, 1995.

[83] K.Z. Mao, "Feature subset selection for support vector machines through discriminative function pruning analysis", *IEEE Transactions on Systems, Man, and Cybernetics-Part B:Cybernetics*, vol. 34, no. 1, Feb. 2004.

[84] R.R.F.Mendes, F.B.Voznika, A.A.Freitas and J.C.Nievola, " Discovering Fuzzy Classification Rules with Genetic Programming and Co-Evolution", *Proceedings of fifth European Conference PKDD 2001*, Lecture Notes in Artificial Intelligence, 2168, pp 314-325.

[85] Z. Michalewicz and M. Michalewicz, " Evolutionary Computation Techniques and Their Applications", *1997 IEEE Conference on Intelligent Processing Systems.*

[86] A.N. Mucciardi and E.E. Gose, "A Comparison of seven techniques for choosing subsets of pattern recognition", *IEEE Transactions on Computers*, C-20, pp. 1023-1031, Sep. 1971.

[87] D. P. Muni, N. R. Pal, and J. Das, "A novel approach for designing classifiers using genetic programming", *IEEE Trans. Evolut. Comput.*, vol. 8, no. 2, pp. 183 196, April 2004.

[88] D. P. Muni, N. R. Pal, and J. Das, "Genetic Programming for Simultaneous Feature Selection and Classifier Design", *IEEE Trans. SMC - B*, vol. 36, no. 1, pp. 106 - 117, Feb 2006.

[89] R.J. Nandi, A.K. Nandi, R.M. Rangayyan and D. Scutt, "Classification of breast masses in mammograms using genetic programming",*Medical and Biological Engineering and Computing*, vol. 44, no. 8, pp. 683-694, 2006.

[90] P.M. Narendra and K.Fukunaga, "A branch and bound algorithm for feature selection", *IEEE Transactions on Computers*, C-26(9), pp. 917-922, Sept. 1977.

[91] D. Nauck and R. Kruse, "A neuro-fuzzy method to learn fuzzy classification rules from data", *Fuzzy Sets Syst.*, vol. 89, pp. 277 288, 1997.

[92] D.V. Nguyen and D.M. Rocke, "Tumor classification by partial least squares using microarray gene expression data", *Bioinformatics*, vol. 18, No. 1, pp. 39-50, 2002.

[93] K. Pal, N. R. Pal and J. M. Keller, "Some neural net realizations of fuzzy reasoning", *Int. Journal of Intell. Systems*, vol. 13, pp 859-886, 1998.

[94] N.R. Pal," Soft computing for feature analysis", *Fuzzy Sets and Systems*, vol. 103, pp. 201-221, 1999.

[95] N.R. Pal and K. Chintalapudi, "A connectionist system for feature selection", *Neural, parallel and Scientific Computations*, vol. 5, pp. 359-381, 1997.

[96] N.R. Pal, V. Kumar, and G.K. Mandal, " Fuzzy Logic Approaches to structure preserving dimensionality reduction", *IEEE Trans. on Fuzzy Systems*, vol. 10, pp 277 - 286, June 2002.

[97] N.R. Pal, S. Nandi and M.K. Kundu, "Self-crossover: a new genetic operator and its application to feature selection", *International Journal of systems and science*, vol. 29, No. 2, pp. 207-212, 1998.

[98] W. Pedrycz (Ed.), *Fuzzy Evolutionary Computation*, Kluwer Academic Publishers, 1997.

[99] W. Pedrycz and M. Reformat, " Evolutionary Fuzzy Modeling", *IEEE Trans. on Fuzzy Systems*, vol. 11, pp. 652 - 665, Oct. 2003.

[100] R.Poli,"Genetic Programming for image analysis",*proceedings of the first international conference on Genetic Programming*, Stanford, pp 363 - 368, July 1996.

[101] P. Pudil, J. Novovicova and J. Kittler, "Floating search methods in feature selection", *Pattern Recognition Letter*, vol. 15, pp. 1119 - 1125, 1994.

[102] P.J. Rauss, J.M.Daida and S.Chaudhary, "Classification of Spectral Imagery Using Genetic Programming", *Proceeding of the Genetic and Evolutionary computation conference*, GECCO-2000.

[103] M.Richeldi and P.Lanzi, "Performing effective feature selection by investigating the deep structure of the data",*Proceedings of second international conference on knowledge discovery and data mining*, AAAI Press, CA, pp 379-383, 1996.

[104] J.J. Rowland, "Model selection methodology in supervised learning with evolutionary computation", *Biosystems*, vol. 72, pp. 187-196, 2003.

[105] M. Russo, "Genetic Fuzzy learning", *IEEE Trans. on Evolutionary Computation*, vol. 4, Iss 3, pp 259 - 273, Sept 2000.

[106] L. Sanchez, I. Couso, J.A. Corrales, "Combining GP operators with SA search to evolve fuzzy rule based classifiers", *Information Sciences*, vol. 136, pp. 175 - 191, 2001.

[107] R. Setiono and H. Liu, "Neural-network feature selector",*IEEE Transactions on Neural Networks*, vol. 8, no. 3, May 1997.

[108] J. Sherrah, R.E. Bogner and A. Bouzerdoum, "Automatic selection of features for classification using genetic programming", *Proceedings of the 1996 Australian New Zealand Conference on Intelligent information systems*, pp. 284-287,IEEE.

[109] S.Y.M. Shi, and P.N. Suganthan, "Feature Analysis and classification of Protein Secondary Structure Data", *Lecture Notes in Computer Science*, vol. 2714, pp. 1151-1158, Springer, 2003.

[110] W. Siedlecki and J. Sklansky, "A Note on genetic algorithms for large-scale feature selection", *Pattern Recognition Letters*, vol. 10, pp. 335-347, 1989.

[111] W. Siedlecki and J. Sklansky, " On automatic feature selection", *International Journal of Pattern recognition and Artificial Intelligence*, vol. 2, No. 2, pp. 197-220, 1988.

[112] K. Sims, "Artificial Evolution for Computer Graphics", *Computer Graphics*, vol. 25,no. 4, pp 319-328, July 1991.

[113] M. Sonka, V. Hlavac and R. Boyle, *Image Processing, Analysis, and Machine Vision*, 2^{nd} edition, PWS publication.

[114] S.A.Stanhope, J.M. Daida , "Genetic Programming for Automatic Target Classification and Recognition in Synthetic Aperture Radar Imagery", *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pp 735-744.

[115] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*, Addison-Wesley Publishing Company, 1974.

[116] A. Tsakonas, "A Comparison of classification accuracy of four genetic programming-evolved intelligent structures", *Information Sciences* vol. 176, pp. 691 - 724, 2006.

[117] M. Tuceryan and A.K. Jain, "Texture Analysis", *The Handbook of Pattern Recognition and Computer Vision (2nd Edition)*, pp 207-248, World Scientific Publishing Co., 1998.

[118] A. Verikas, M. Bacauskiene, "Feature selection with neural networks", *Pattern Recognition Letters*, vol. 23, pp. 1323-1335, 2002.

[119] P.A. Whigham and P.F. Crapper," Modelling Rainfall-Runoff using Genetic Programming",*Mathematical and Computer Modeling*, vol. 33, pp. 707-721, 2001.

[120] A. L. Wiens and B.J. Ross, "Gentropy: evolving 2D textures", *Computers and Graphics*, vol. 26, pp. 75- 88, 2002.

[121] H.Zhang and G.Sun. "Feature selection using tabu search method", *Pattern Recognition*, vol. 35, pp. 701-711, 2002.

[122] L. Zhang, L.B. Jack and A.K. Nandi,"Fault detection using Genetic Programming",Mechanical Systems and Signal Processing, vol. 19, no. 2, pp. 271-289, 2005.

[123] L. Zhang and A.K. Nandi, "Fault Classification using Genetic Programming", *Mechanical Systems and Signal Processing*, vol. 21, no. 3, pp. 1273-1284, 2007.

[124] D. Zongker, W.F. Punch,*lilgp 1.0 User's Manual*, Technical Report, MSU Genetic Algorithms and Research Application Group (GARAGe),http://garage.cps.msu.edu

Publications of the Author Related to the Thesis

A1. Durga Prasad Muni, Nikhil R. Pal and J. Das, " A Novel Approach for Designing Classifiers Using Genetic Programming", *IEEE Transaction on Evolutionary Computation*, vol 8, No. 2, April 2004, pp 183-196.

A2. Durga Prasad Muni, Nikhil R. Pal and J. Das, "Genetic Programming for Simultaneous Feature Selection and Classifier design", *IEEE Transaction on Systems, Man and Cybernetics-B*, vol 36, No. 1, Feb 2006, pp 106-117.

A3. Durga Prasad Muni, Nikhil R. Pal and J. Das, "Evolution of Fuzzy Classifiers using Genetic Programming", *IEEE Transaction on Evolutionary Computation*, (Communicated).

A4. Durga Prasad Muni, Nikhil R. Pal and J. Das, "Texture Generation for Fashion Design using Genetic Programming", *Proceedings of 9th International Conference on Control, Automation, Robotics and Vision, ICARCV 2006, Singapore, IEEE Xplore.*

A5. D.P. Muni and J. Das, " Interactive Texture Generation Using Genetic Programming", *Proceedings of National Conference on Recent Advances in Power, Signal Processing and Control*, 2004, Rourkela, pp 151-154.

A6. D.P. Muni and J. Das, "Texture Classification using Genetic Programming", *Proceedings of International Conference on Information Technology, Haldia*,2007, pp. 525-527.

A7. D. P. Muni, N. R. Pal and J. Das, "Multicategory Classifier design using Genetic Programming", *Proceedings of international conference on Communications, Devices and Intelligent Systems, CODIS-2004*, pp 597-599.