EVOLUTIONARY ALGORITHMS FOR EFFECTIVE CLASSIFICATION

BY

Kotecha Radhika N.

(10MICT06)

Nirma University, Ahmedabad

May 2012

EVOLUTIONARY ALGORITHMS FOR EFFECTIVE CLASSIFICATION

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Information & Communication Technology

By:

Kotecha Radhika N.

(10MICT06)

Guided By:

Dr. Sanjay Garg

Prof. Vijay Ukani

Nirma University, Ahmedabad

May 2012

Declaration

This is to certify that

i) The thesis comprises my original work towards the degree of Master of Technology in Information and Communication Technology at Nirma University and has not been submitted elsewhere for a degree.

ii) Due acknowledgement has been made in the text to all other material used.

Kotecha Radhika N.

Certificate

This is to certify that the Major Project Part - II Report entitled **"Evolutionary Algorithms for Effective Classification"** submitted by **Kotecha Radhika N. (Roll No : 10MICT06)**, towards the partial fulfillment of the requirements for degree of Master of Technology in Information and Communication Technology from Institute of Technology, Nirma University, Ahmedabad is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree.

Dr. Sanjay Garg
Guide & Professor,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Prof. Vijay Ukani
Co-Guide & Associate Professor,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Prof. Gaurang Raval
PG - ICT Coordinator,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Prof. D. J. Patel
Professor & Head,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Ketan Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad.

Acknowledgement

"Step after step, the ladder is ascended".

I owe a huge debt of gratitude to my guide, **Dr. Sanjay Garg**, Department of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad for the valuable inputs at every step of this work. The appreciation and continual support he has imparted has been a great motivation to me in reaching a higher goal. His in-depth knowledge about the subject has provided me a strong base and direction towards this work. His guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

I will always be grateful to my co-guide, **Prof. Vijay Ukani**, Department of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad for the valuable time and suggestions right from the very early stage of this work. Above all and the most needed, he provided me unflinching encouragement and support in various ways.

My deepest thank you is extended to **Prof. Gaurang Raval**, PG ICT - Coordinator, Department of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad for an exceptional support and continual encouragement throughout the Major Project.

It gives me an immense pleasure to thank **Prof. D. J. Patel**, Hon'ble Head of Computer Science and Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr. Ketan Kotecha**, Hon'ble Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation he has extended throughout course of this work.

I would also thank the Institution, all the faculty members in Department of Computer Science and my colleagues without whom this project would have been a distant reality.

Words fail to express appreciation for the constant support given by my parents throughout the work. I feel grateful and blessed.

Finally, Thank You God for always being with me.

- Kotecha Radhika N.

Abstract

The amount of raw data being accumulated in the databases is increasing at an inconceivable rate. However, these data-rich databases are poor in providing substantial information. This is where data mining comes into picture. Specifically, data mining is "the process of extracting or mining information from large amount of data". Data classification has been an active area of research in data mining. It consists of assigning a data instance to one of the predefined classes/groups based upon the knowledge gained from previously seen (classified) data.

Real world problems demand classifiers that are accurate as well as easy to interpret. Traditional comprehensible classifiers like decision trees are very accurate at classifying new instances, but with increase in the size of datasets and/or the number of classes, and/or with increase in number of attributes, the trees induced are very large in size and hence difficult to interpret. On the other hand, evolutionary algorithms like Genetic Programming(GP) when applied to classification problems give trees that have smaller size but are not very accurate.

Thus the work proposes an algorithm GPeCT, that employs GP as an optimization technique to yield efficient and effective classification. The goal is to evolve a classifier that performs a trade-off between accuracy and comprehensibility in order to produce an optimal decision tree classifier (for n-class where n>=2) using GP.

When evaluated on some benchmark datasets, the proposed algorithm obtained by merging Genetic Programming and decision tree outperforms the traditional classification techniques in terms of a combination of accuracy and comprehensibility.

Keywords: Accuracy, Classification Trees, Comprehensibility, Genetic Programming, Multi-class classification.

Contents

List of Figures

List of Tables

Abbreviations

ACO ..... Ant Colony Optimization
BDT ..... Binary Decision Tree
BFS ..... Breadth First Search
CART ..... Classification and Regression Tree
CTG ..... Cardiotocography
DFS ..... Depth First Search
DT ..... Decision Tree
EA ..... Evolutionary Algorithm
EP ..... Evolutionary Programming
ES ..... Evolutionary Strategies
FM ..... Fuzzy Methods
GA ..... Genetic Algorithm
GkNN ..... Genetic k Nearest Neighbor
GP ..... Genetic Programming
GPeCT ..... Genetic Programming based Evolution of Classification Tree
GUI ..... Graphical User Interface
kNN ..... K Nearest Neighbor
MAP ..... Maximum A Posterior
MATLAB ..... Matrix Laboratory
MOGP ..... Multi Objective Genetic Programming
NN ..... Neural Networks
RHS ..... Right Hand Side
UCI ..... University of California at Irvine
WEKA ..... Waikato Environment for Knowledge Analysis

Chapter 1

Introduction

The digital revolution has brought an explosion of stored data to our world, and it's increasing at an ever faster rate. This collection of data is quickly outpacing the ability for humans to understand or make use of it. Scientists across the world have joined in the hunt to discover valuable knowledge from this digital information[1].

Data mining is a field that was born to deal with these difficulties. Defined as "the process of extracting or mining knowledge from large amounts of data"[2], data mining techniques go through large volumes of data quickly and attempt to transform that data in a way that will enhance the discovery of new knowledge. Data mining uses a combination of many disciplines: from statistics and pattern recognition to artificial intelligence and machine learning.

Though data mining has been successful in many of its applications, it is still a relatively new field. New techniques and innovations are found frequently, and yet there is much potential for growth.

This introduction chapter presents the motivation source of the work carried out, the scope of the work, followed by some basics of data mining and machine learning.

1.1 Motivation

Classification is a form of data analysis that can be used to extract models describing important data classes or to predict future data trends. Such analysis can help to provide us a better understanding of the data at large. A lot of research has been done to develop an efficient classification technique for a problem with binary classes. However, in real world, the problems are scattered into multiple classes which is a field rarely touched by researchers.

Further, classification algorithms generally yield results in either symbolic or non-symbolic form. Algorithms that provide non-symbolic results only assign class labels to the data instances and do not convey the reason as to why the algorithm led to that classification. Examples include Bayesian classifiers, k Nearest Neighbor (kNN) classifier, Neural Networks, etc. However in real world applications, it is also required to know the reasons behind classifying any data into a particular class. Algorithms that provide such reasons fall in the category of symbolic classifiers.

Decision trees are symbolic classifiers, i.e. they have high comprehensibility and are accurate too. But, with increase in number of classes or when training data becomes very large, size of the tree increases considerably. This leads to loss of "comprehensibility", which was considered as one of the best properties of decision tree.

Evolutionary algorithms like Genetic Algorithms (GA) and Genetic Programming (GP) have been found successful in solving numerous classification problems. Also, tree structures are an encoding scheme for GP individuals. The classification trees generated using GP have smaller number of nodes and hence are more comprehensible.

Hence the need to develop an optimal multiclass classifier that performs a trade-off between accuracy and comprehensibility has been the motivation of the work.

1.2 Scope of the work

To achieve the objective of developing an evolutionary algorithm for effective classification, the work seeks to develop an optimal decision tree classifier using Genetic Programming: a subset of evolutionary algorithms. The parameter for optimization is the accuracy and additionally, the size of the decision tree so as to address the important issue of ease in model-interpretation and comprehension. The approach considers designing a n-class (n>=2) classifier. Thus the scope is limited to proposing a new classification algorithm that is a hybrid of DT and GP.

1.3 Fundamentals of Data Mining

In data mining or machine learning, the general goal is to find some implicit knowledge from a set of data. Various learning strategies are used, depending on the data used for training the algorithm.

1.3.1 Learning Strategies

Learning strategies include: supervised, reinforcement, unsupervised, and hybrid.

- **Supervised Learning Strategy**

In supervised learning, the system is provided with the correct answer for each training example. The task of the system is to learn the relationship between the input examples, and the answers. For example, a system could be shown a number of images of faces, each with a name. The system could then be shown a different image of one of the faces, and would output the name of the face[3].

- **Reinforcement Learning Strategy**

In reinforcement learning, the system is provided with hints toward the correct answers, but not the exact answers. The aim of the system is to use the hints over time,

which point toward the correct answers or actions. For example, an elevator could be given a reward each time it correctly predicts which floor to go to[4][3].

- **Unsupervised Learning Strategy**

In unsupervised learning, the system is not provided with any answers, or correct outputs. The learning process usually aims to find patterns and correlations in the data. For example, a shop could record the items that people buy; a learning system could then find correlations between different items that are bought together[3].

- **Hybrid Learning Strategy**

Hybrid learning involves a mixture of the previous strategies[3].

This work is related to supervised learning, that is, the class labels of the training data are already available and are used to built a classifier that will predict the class labels of the test data.

1.3.2 Testing Methods

Cross-Validation[5] is a common method to test the performance of a classifier. For N-fold cross validation the data set is partitioned into N separate, equal-sized groups. Training occurs N times, with each using a different group as the test set, and the other N - 1 groups as the training set. The performance of the system is then the average performance of the N trainings.

Another method used is the Holdout method[2]. Here, the given data are randomly partitioned into two independent sets, a training set and a test set. Typically, two-thirds of the data are allocated to the training set, and the remaining one-third is allocated to the test set. The training set is used to derive the model, whose accuracy is estimated with the test set. Throughout the work, holdout method is used for testing the performance of classifiers.

1.4 Organization of Thesis

Following this introduction, chapter 2 presents an overview of the traditional classification techniques, followed by introduction of evolutionary algorithms and its application in classification. The chapter also shows the implementation results of these traditional techniques.

Chapter 3 presents a literature survey of previous work to date in the domains of genetic programming and decision tree that is relevant to the work presented in the remainder of the thesis.

Chapter 4 proposes an algorithm Genetic Programming based Evolution of Classification Tree(GPeCT) accompanied with the detailed explanation of the algorithm.

Chapter 5 describes the methodology used in implementing each step of the algorithm whereas chapter 6 shows the experimental results and proves how the proposed algorithm outperforms GP and Decision Tree on benchmark datasets.

The thesis ends with discussing the conclusions derived from the work and explores some future enhancements that can be made to the algorithm.

Chapter 2

Classification and Evolutionary Algorithms

Classification has been an active area of research in data mining. It is a form of data analysis that can be used to extract models describing important data classes and can help provide us with a better understanding of the data at large. Classification predicts categorical (discrete, unordered) classes, given any unseen data.

Evolutionary algorithms (EAs) are stochastic optimization techniques that imitate some aspects of natural evolution. They try to obtain good solutions by creating a population of data structures that represent them, and evolve that population by changing those solutions, combining them to create new ones, and, on average, making the better-suited survive and the discarding worst-suited individuals. EAs are a set of modern met heuristics used successfully in many applications with very high complexity.

This chapter presents some of the fundamental concepts of classification and evolutionary algorithms that form the basis of the work. The chapter is ordered into five major sections. Section 2.1 presents an introduction to classification followed by a

brief review of the traditional techniques in section 2.2. Section 2.3 and 2.4 respectively cover the basics and types of evolutionary algorithms. Section 2.5 describes the implementation of the traditional methods used for classification.

2.1 Introduction to Classification

Classification is a task of assigning a data instance to one of the predefined classes/groups based upon the knowledge gained from previously seen (classified) data.

Basically, classification is a two step process. First, a training set consisting of records whose class labels are known must be provided. The training set is used to induce a classification model, which is subsequently applied to the test set consisting of records with unknown class labels[1]. Figure 2.1 shows this two-step classification process. Evaluation of the performance of a classification model is based on the counts of test records correctly and incorrectly predicted by the model.

Classification plays a major role in numerous practical situations such as classifying e-mails as spam or not spam, for a bank in classifying loan applications as either safe or risky, diagnosing medical conditions, categorizing news stories as finance, weather, business, entertainment, sports, etc. There are numerous methods used to build classifiers. A selected few are briefly described in the following section.

2.2 Brief Review of Classification Techniques

This section presents a brief review of traditional classification techniques. More emphasis is laid on Decision Trees, which is described in subsection 2.2.1, whereas the rest of the techniques are described in brief in subsection 2.2.2

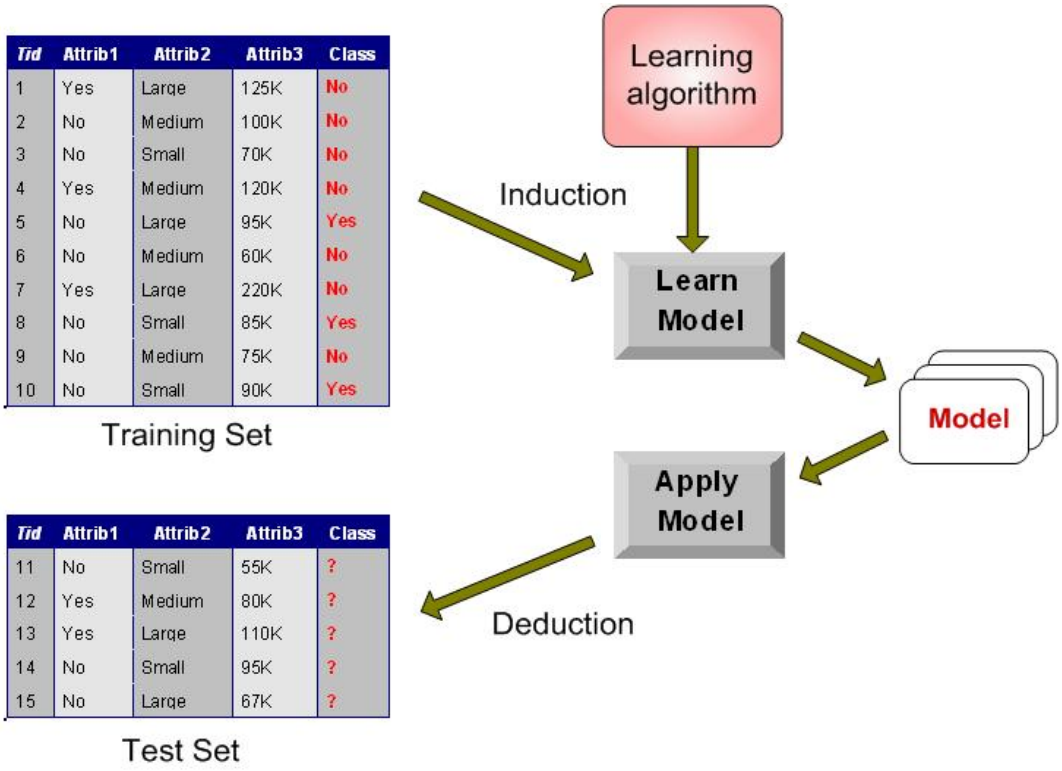


Figure 2.1: Illustrating Classification Task[1]

2.2.1 Decision Tree

A decision tree (or classification tree) is a flowchart-like tree structure where the non-terminal nodes (root and other internal nodes) contain attribute test conditions to separate records that have different characteristics. Each branch represents an outcome of the test and each terminal node (leaf) holds a class label.

While applying decision trees for classification, instances are classified by applying test conditions, starting at the root node and following the appropriate branch based on the outcome of the test. The branch may lead to another internal node for which a new test condition is applied or to a leaf node. In the latter case, the class label associated with the leaf node is applied to the instance[1].

A decision tree is constructed by recursively dividing the training data into successively purer partitions. A partition is pure if all the tuples in it belong to the same class[2]. The basic strategy is as follow[2][1]:

a. The tree starts with a single node representing all the training instances.

b. If all these instances belong to the same class, then the node becomes a leaf and is labeled with that class. If the instances belong to more than one class, an attribute that best classifies the examples is selected and child nodes are created corresponding to each outcomes of test conditions on that attribute. For categorical attributes, the outcomes of the test at node correspond directly to the known values of the attribute whereas continuous-valued attributes may be discretized to form the test conditions. The instances are distributed to the children based on these tests.

c. The algorithm is applied recursively until either all of the tuples in a partition belong to the same class or all the attributes in a path starting from the root have been tested.

A typical decision tree is shown in Figure 2.2 which represents a case where a bank loan manager wants to predict whether giving loan to a person is safe or not.
Figure 2.2 is merely an example of decision tree induced from the dataset using the algorithm Classification And Regression Trees(CART). Practically, there are many different decision trees possible depending on the attributes selected for classification.

Also, there are several measures[1][6] for selecting the attribute that best divides the training data such as Information Gain, Gini Index, etc. Throughout the paper, we have used Gini Index (used by the algorithm CART) as an impurity measure to induce decision trees and it is described in the following subsection.

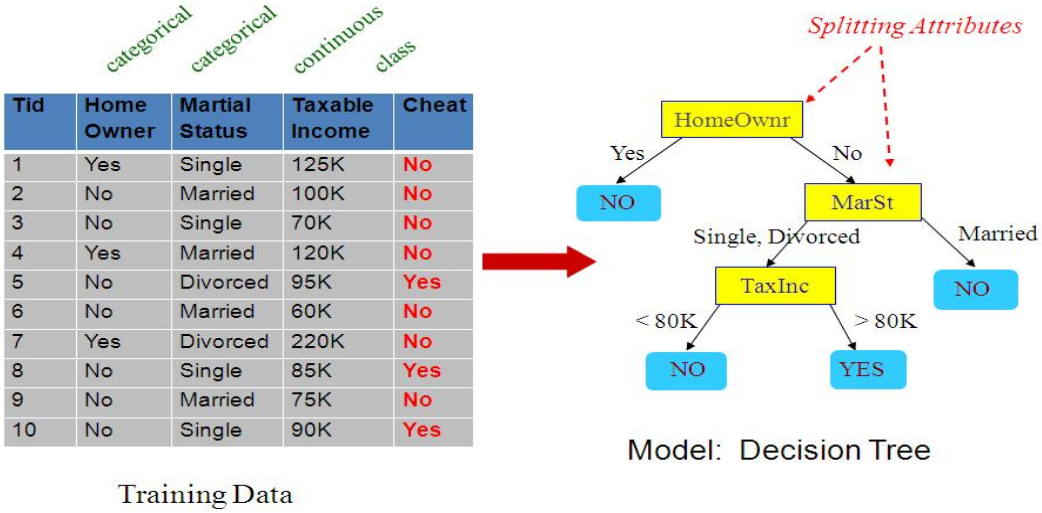


Figure 2.2: An example Decision Tree[1]

Gini Index measure for selecting attributes

The Gini index[2] which is used by CART, measures the impurity of D, a data partition or set of training tuples, as shown in equation 2.1:[2]

$$\text{Gini(D)} = 1 - \sum_{\text{i=1}}^{\text{m}} \text{p}_\text{i}^2 \tag{2.1}$$

where p_i is the probability that a tuple in D belongs to class C_i and is estimated by $|C_i,D|/|D|$. The sum is computed over m classes. The Gini index considers a binary split for each attribute.

The following subsections describe the method of employing Gini index for selecting attributes with discrete-value and continuous valued attributes.

Discrete-Valued Attributes

Assume A as a discrete-valued attribute having v distinct values, $a_1, a_2, ..a_v$, occurring in D. To determine the best binary split on A, all of the possible subsets that can be formed using known values of A are examined. Each subset, S_A, can be considered

as a binary test for attribute A of the form "A ϵ S_A?". Given a tuple, this test is satisfied if the value of A for the tuple is among the values listed in S_A. When considering a binary split, a weighted sum of the impurity of each resulting partition is computed[2]. For example, if a binary split on A partitions D into D_1 and D_2, the Gini index of D given that partitioning is as shown in equation 2.2[2]:

$$\text{Gini}_\text{A}(\text{D}) = \frac{|\text{D}_1|}{|\text{D}|}\text{Gini}(\text{D}_1) + \frac{|\text{D}_2|}{|\text{D}|}\text{Gini}(\text{D}_2) \tag{2.2}$$

For each attribute, all the possible binary splits is considered. For a discrete-valued attribute, the subset that gives the minimum Gini index for that attribute is selected as its splitting subset.

Continuous-Valued Attributes

For continuous-valued attributes, each possible split-point must be considered. The midpoint between each pair of (sorted) adjacent values is taken as a possible split-point. The point giving the minimum Gini index for a given (continuous-valued) attribute is taken as the split-point of that attribute. For a possible split-point of A, D_1 is the set of tuples in D satisfying A <=split point, and D_2 is the set of tuples in D satisfying A > split point[1][2].

The reduction in impurity that would be incurred by a binary split on a discrete or continuous-valued attribute A is shown in equation 2.3:[2]

$$\Delta\text{Gini(A)} = \text{Gini(D)} - \text{Gini}_\text{A}(\text{D}) \tag{2.3}$$

The attribute that maximizes the reduction in impurity (or, equivalently, has the minimum Gini index) is selected as the splitting attribute. This attribute and either its splitting subset (for a discrete-valued splitting attribute) or split-point (for a continuous valued splitting attribute) together form the splitting criterion.

Decision trees are highly accurate and require less time for construction. Moreover, the major advantage of decision tree is its comprehensibility, that is, they are easy to interpret. However, decision trees cannot handle incremental data and are required to be constructed all again. Also, when the number of outcome classes is more or the training data is very large (which is apparent in data mining), decision trees induced are very large. This leads to loss of its comprehensibility[7] and hence this area needs attention.

2.2.2 Other Classification Techniques

The following section describes few other commonly used classification techniques with their advantages and open issues.

Naive Bayes Classifier

Naive Bayes[8][9] is a simple but effective classifier based upon the principle of Maximum A Posteriori (MAP).

Given a problem with K classes $\{C_1, \ldots, C_K\}$ with so-called prior probabilities $P(C_1), \ldots, P(C_K)$, class label c can be assigned to an unknown example with features $x = (x_1, \ldots, x_N)$ such that $c = \text{argmax}_c\, P(C = c \mid x_1, \ldots, x_N)$, that is choose the class with the maximum a posterior probability given the observed data. This a posterior probability can be formulated, using Bayes theorem, as in equation 2.4 [9]:

$$P(C = c|x_1, ..., x_N) = \frac{P(C = c)P(x_1, ..., x_N|C = c)}{P(x_1, ..., x_N)} \tag{2.4}$$

In equation 2.4, as the denominator is the same for all classes, it can be dropped from the comparison. Now, it is required to compute the so-called class conditional probabilities of the features given the available classes. This can be quite difficult taking into account the dependencies between features. The Naive Bayes approach

is to assume class conditional independence i.e. the attributes $x_1, \ldots, x_N$ are independent given the class.

This assumption simplifies the numerator of the above equation to be $P(C = c)$ $P(x_1 \mid C = c) \ldots P(x_N \mid C = c)$, and then choosing the class c that maximizes this value over all the classes $c = 1, \ldots, K$. Clearly this approach is naturally extensible to the case of having more than two classes, and was shown to perform well despite of the underlying simplifying assumption of conditional independence. The major advantage of the Naive Bayes classifier is that it requires short time for training the classifier. Also, each training example has an effect on the prediction and each training example in turn would increase/decrease the probability that a prediction is correct. However, the assumption of independence among attributes is not true always and hence the accuracy of Naive Bayes classifier is unstable[7]. An exception occurs when there is an equal probability for each class label value in the Naive Bayesian algorithm. In[10], a novel algorithm named NB+ which is an extended version of the traditional Naive Bayesian algorithm has been presented which solves this problem.

k Nearest Neighbors Classifier

Nearest neighbor classifiers[2] are based on learning by analogy. The training samples are described by n-dimensional numeric attributes. Each sample represents a point in an n-dimensional space. In this way, all of the training samples are stored in an n-dimensional pattern space. When given an unknown sample, a k-nearest neighbor classifier searches the pattern space for the k training samples that are closest to the unknown sample[11]. An unknown sample is assigned the most common class among its k nearest neighbors. Closeness is defined in terms of a distance metric, such as Euclidean distance. Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes (e.g. height, weight, etc.)[2]. The value of k is generally determined using a validation set or using cross validation.

Nearest neighbor classifiers are instance-based or lazy learners, in that they store all of the training samples and do not build a classifier until a new (unlabeled) sample needs to be classified. This contrasts with eager learning methods, such as decision tree, which construct a generalization model before receiving new samples to classify[11]. kNN classifiers require less training time and are also accurate. However, they are found sensitive to the choice of the similarity function selected for comparing instances. Also, unlike decision tree induction, nearest neighbor classifiers assign equal weight to each attribute[11]. This may cause confusion when there are many irrelevant attributes in the data.

In [12], a variant called Genetic k Nearest Neighbor (GkNN) is proposed wherein genetic algorithm is combined with kNN to improve its classification performance. Instead of considering all the training samples and taking k-neighbors, the GA is employed to take k-neighbors straightaway and then calculate the distance to classify the test samples.

2.3 Introduction to Evolutionary Algorithms

Real world applications are most likely to encounter problems that are hard to solve. Some examples for these kind of problems are the traveling salesman, knapsack problem, etc. To approach such hard problems, a couple of concepts were introduced in the past decades, which were inspired by nature. Some of the more popular and successful examples are Neural Networks (NN), Fuzzy Methods (FM) and Evolutionary Algorithms (EA or also known as Evolutionary Computation), Ant Colony Optimization (ACO), etc.

EAs follow the principle of "Survival of the Fittest" laid down by Charles Darwin. One of the major advantages of EA methods compared to other methods is, that they

only need little problem specific knowledge and that they can be applied on a broad range of problems. EA methods only need the target (fitness) function for a given problem, which is to be optimized. EAs differ from more traditional optimization techniques in that they involve a search from a "population" of solutions, not from a single point[13].

2.4 Brief Review of Evolutionary Algorithms

Several different techniques are grouped under the generic denomination of EA. Section 2.4.1 describes Genetic Programming and its applications in classification whereas the other EAs are discussed in section 2.4.2.

2.4.1 Genetic Programming

Genetic Programming(GP), pioneered by John R. Koza[14], is an evolutionary algorithm that uses variable sized tree structures to codify individuals. This innovative and flexible technique has been applied to solve numerous real world problems. The work uses GP to optimize classification techniques described in section 2.2.

The vital features of Genetic Programming are as follow[13]:

a. The use of a population (a group) of individuals that may be regarded as candidate or partial solutions, unlike the other techniques that operate on only one solution.

b. A generational inheritance method. Genetic Operators are applied to the individuals of a population to give birth to a new population of individuals (hence developing the next generation). The main genetic operators are reproduction, crossover and mutation. Crossover swaps some genetic material from two individuals to form two new offspring for the new population, whereas mutation

randomly changes a small portion of the genetic material of only one individual to produce one new offspring.

c. A fitness function is used in order to measure the quality (how well it performs) of an individual so as to efficiently search for the desired solution. Fitter individuals are more apt to be selected to take part in the procreation of the next generation of individuals, thus, increasing the probability that its genetic material will survive throughout the evolutionary process.

The basic GP algorithm is explained graphically in Figure 2.3:

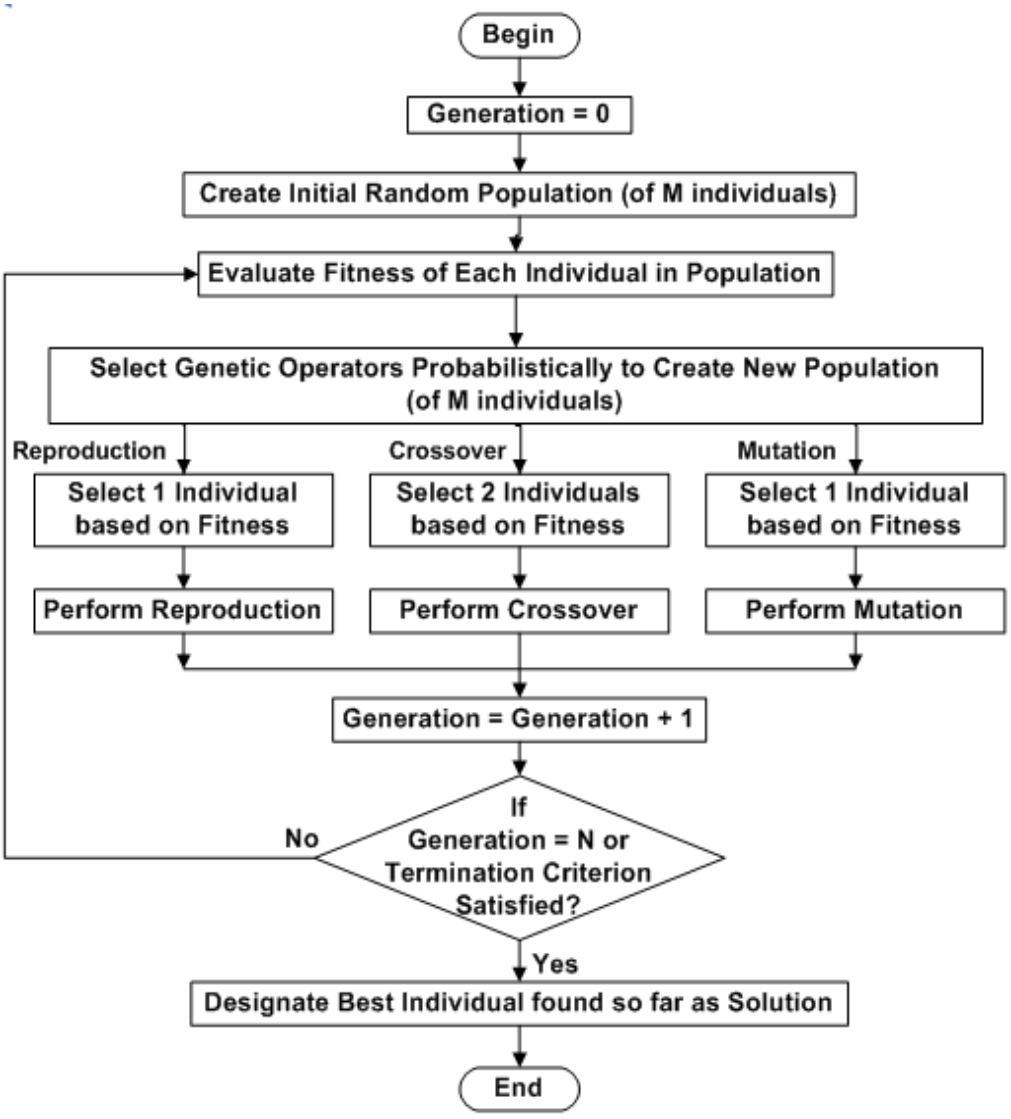


Figure 2.3: Flowchart of Genetic Programming

The generation of initial population of a GP run is performed by the random generation of individuals. The internal nodes of the tree are called functions, while the leaf nodes of the tree, which take no arguments, are called terminals. Starting from the root node of a tree, one function from the function set is selected at random to be the root node. Further, new functions or terminals are selected at random from the function and terminal sets to form the arguments of the root function[15].

This process continues until every leaf node of the tree consists of a terminal. The individuals (trees) developed through this manner undergo genetic operations as described above and form the new generation. The new population in turn undergoes genetic operations until the termination criterion is reached or maximum number of generations have been developed.

Figure 2.4 shows an example GP tree which is the representation of the program max(x+x,x+3*y). A set of key parameters are used for deciding the specific charac-

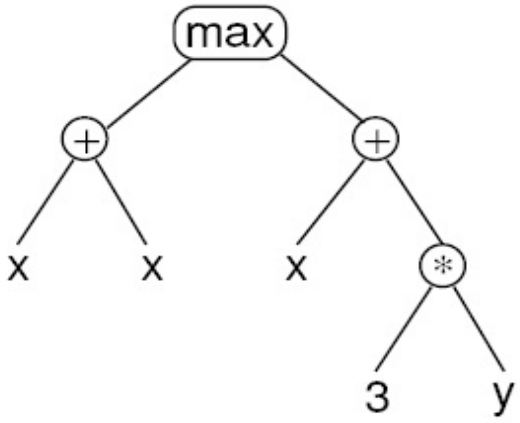


Figure 2.4: An example GP tree[16]

teristics of how the overall GP process operates. These key parameters are described as follows:

1. Function and Terminal Set

The terminal and function set used in GP is typically driven by the nature of the problem domain. Terminals are usually some randomly assigned numeric constants whereas the function set consists of all the fundamental operations which are performed upon the terminal values to produce desired output. To prevent GP trees from growing to disproportionate sizes, a maximum size of trees is often set[15].

Table 2.1 and Table 2.2 respectively show a sample of some of the functions and terminals found in GP literature[16].

Table 2.1: Example of primitives in GP function set

Kind of Primitive	**Examples**
Arithmetic	+, *, /
Mathematical	sin, cos, exp
Boolean	AND, OR, NOT
Conditional	IF-THEN-ELSE
Looping	FOR, REPEAT

Table 2.2: Example of primitives in GP terminal set

Kind of Primitive	**Examples**
Variables	x, y
Constant values	3, 0.45
0-arity functions	rand, randbetween

Two conditions must be satisfied to ensure that GP can be successfully applied to a specific problem: sufficiency and closure. Sufficiency states that the terminals and nonterminals (in combination) must be capable of representing a solution to the problem. Closure requires that each function of the nonterminal set should be able to handle all values it might receive as input. In practice, we often need to evolve programs that handle values of different types, and this makes it difficult to meet the closure requirement[13].

2. Population Size

The number of individuals within the evolving population is one of the most important parameters of a GP run. Simple problems are often solved with small population

sizes of 50 individuals, whereas difficult problems are approached with population sizes larger that 10,000[17].

3. Maximum Number of Generations.

The parameter indicates the number of generations to run before stopping. Generations from less than 20 to many thousands have been used.

4. Initial Population creation method.

There are a number of different approaches for generating this random initial population. Here, two of the simplest (and earliest) methods (the full and grow methods), and a widely used combination of the two known as Ramped half-and-half is described[15][18].

In both the full and grow methods, the initial individuals are generated so that they do not exceed a user specified maximum depth. The depth of a node is the number of edges that need to be traversed to reach the node starting from the tree's root node (which is assumed to be at depth 0). The depth of a tree is the depth of its deepest leaf.

- **Full Method**

In the full method (so named because it generates full trees, i.e. all leaves are at the same depth) nodes are taken at random from the function set until the maximum tree depth is reached. Beyond that depth, only terminals can be chosen.

- **Grow Method**

The grow method, on the contrary, allows for the creation of trees of more varied sizes and shapes. Nodes are selected from the whole primitive set (i.e., functions and terminals) until the depth limit is reached. Once the depth limit is reached only terminals may be chosen (just as in the full method).

- **Ramped half and half Method**

Because neither the grow or full method provide a very wide array of sizes or shapes on their own, Koza (1992) proposed a combination called ramped half-and-half. Half of the initial population is constructed using full method and half is constructed using the grow method. This is done using a range of depth limits (hence the term "ramped") to help ensure that we generate trees having a variety of sizes and shapes[16].

5. Fitness Function.

The fitness measure specifies what needs to be done. The fitness measure is the primary mechanism for communicating the high-level statement of the problem's requirements to the genetic programming system. The fitness measure implicitly specifies the search's desired goal. An effective fitness mechanism for classification is the accuracy of the individual on the training set[15].

6. Selection Method.

The fitter individuals should contribute more than poorer individuals to later generations. This is achieved by the selection mechanism, which is used whenever selecting an individual for use in a new population. Many selection methods could be used, including proportional, rank, and tournament selection[15].

- **Proportional Selection**

Proportional selection can be visualized as spinning a roulette wheel. The size of the segment on the wheel that applies to an individual is proportional to its fitness. As such, the probability of an individual being selected is proportional to its fitness.

- **Rank Selection**

When using rank selection, the rank of each individual is found, from first to last in the population. The probability of an individual being selected is based on a function of the rank of the individual.

- **Tournament Selection**

When selecting an individual using tournament selection, instead of competing for the selection over the entire scope of the population, few individuals are randomly selected from the population. The individuals within the tournament then compete for a chance to be selected to pass genetic material into the next generation.

7. Genetic Operators to be used and their probability.

The genetic operators of GP are described as follow[3]:

- **Reproduction**

To ensure that the fitness of individual in a population is never less than that of previous generations, the reproduction, or elitism, operator is used. This consists of simply copying the best few individuals of a generation's population directly to the next.

- **Mutation**

In mutation, a single individual is selected from the population and copied to a mating pool. A mutation point is chosen randomly, somewhere in the tree, and the subtree below the mutation point is replaced with a new, randomly generated subtree. The new individual is then copied into the new population. This is pictured in figure 2.5. Mutation is used to ensure diversity of individuals in the population and for introducing new genetic material.

- **Crossover**

In crossover (or recombination), two individuals are selected from the population and copied to a mating pool. A crossover point is randomly chosen in each tree, and the subtrees below the crossover points are swapped. The two new trees, are then copied to the new population. This is pictured in figure 2.6.

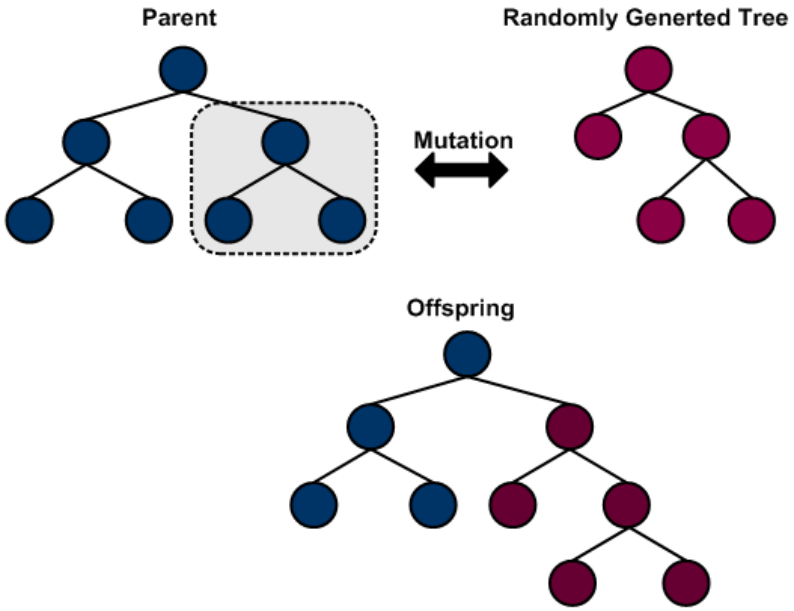


Figure 2.5: Subtree Mutation

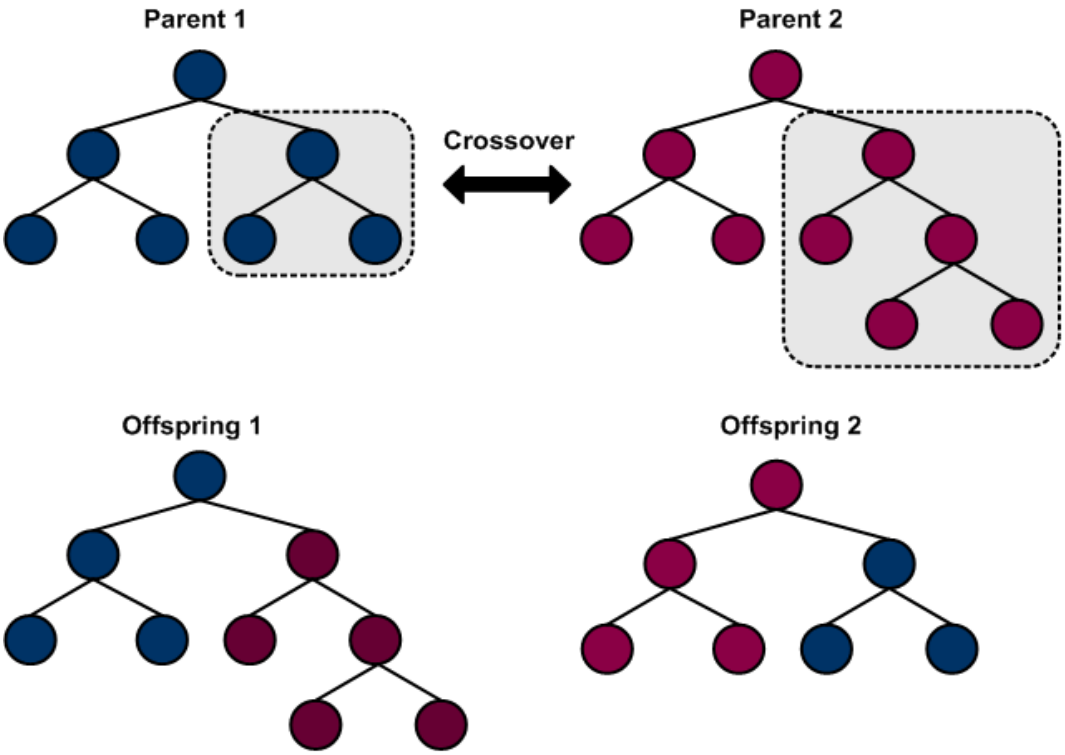


Figure 2.6: Subtree Crossover

Above are the basic crossover and mutation operations. Other variants have also been proposed in the literature. The probabilities of selecting the operators vary with the nature of the problem. However, in general, GP systems use a high level of crossover, and lower levels of mutation and reproduction.

8. Termination Criterion.

Another important key parameter, the termination criterion, may include a maximum number of generations to be run, as well as a problem-specific success predicate. The single best-so-far individual is then designated as the result of the run[15].

2.4.2 Other EA methods

This section describes three different methods derived from EA along with their area of application[3][19].

- Genetic Algorithms

Genetic Algorithms(GA) are the most popular type of EA. In GAs, the representation for solutions is typically a fixed length bit-strings. GAs are well suited for optimizing combinatorial problems.

- Evolutionary Strategies and Evolutionary Programming

Evolutionary Strategies (ES) and Evolutionary Programming (EP) were both use vectors of real values to represent individuals. In ES the individuals are used directly as solutions, in EP the individuals are interpreted as finite-state machines. In both ES and EP, mutation is often the most important evolutionary operator. ES and EPs are well suited for optimizing continuous functions.

2.5 Implementation of Basic Techniques

The above stated methods have been suggested and tested to solve the problem in the binary classification case. But the multiclass classification has been attempted by only few researchers. Real world problems generally consists of classifying data into multiple (more than two) classes. Hence the section shows experiments conducted on 5 multiclass datasets from various real domains. The aim behind applying the algorithms to datasets of various domains is to show the consistency of results in all domains. These datasets are available from machine learning repository of the University of California at Irvine(UCI). Table 6.1 lists the five datasets used in experimentation along with their details.

All the algorithms, i.e. Decision Tree (CART), Naive Bayes, kNN have been applied to test five data sets using Weka, Open Source, Portable, GUI-based workbench consisting collection of state-of-the-art machine learning algorithms and data pre processing tools. Whereas, GP implementation of HeuristicLab 3.5.33, an open source under GNU General Public License is used. All the experiments are performed on Windows Platform.

The results of predictive accuracy and comprehensibility on test instances of all the five data sets are tabulated in figures 2.4 and 2.5 respectively. The experiments use hold-out method for evaluating the performance of the classifier, where 66% of dataset was used for training the classifier and the rest was used for testing it.

Table 2.3: Composition of Data Sets

	Attributes	Instances	Classes
Cardiotocography	21	2126	3
Robot Navigation	24	5456	4
Nursery	8	12960	5
Forest Cover Type	54	581012	7
Poker Hand	10	1025010	10

Table 2.4: Predictive Accuracy (in %) of Five Datasets

	Decision Trees	Naive Bayes	kNN	GP
Cardiotocography	92.52	82.15	92.25	63.27
Robot Navigation	98.92	52.35	85.50	71.55
Nursery	72.71	90.67	96.95	59.89
Forest Cover Type	63.49	61.34	63.14	53.07
Poker Hand	66.16	50.06	64.74	50.18

Cardiotocography(CTG) and Robot Navigation are numeric datasets whereas Nursery and Poker hand are text datasets. Forest cover type is a mixed dataset with numeric and text attributes together.

The results show that decision trees are on an average best in terms of accuracy. But, the size of decision tree becomes very large as the number of instances and classes increase.

Table 2.5: Comprehensibility (Tree Size)of Five Datasets

	Decision Tree	Naive Bayes	kNN	GP
Cardiotocography	101	N/A	N/A	66
Robot Navigation	53	N/A	N/A	16
Nursery	197	N/A	N/A	83
Forest Cover Type	22983	N/A	N/A	3984
Poker Hand	137745	N/A	N/A	30202

The classification accuracy of Naive Bayes classifiers is not stable due to the assumption of attribute independence.

With the increase in the number of attributes, the accuracy of kNN decreases as they give equal importance to each attribute.

When GP is used for classification, the size of the tree obtained is quite less compared to decision tree and is the major advantage of using GP as a classifier.

Real world problems demand a classifier that is not only accurate on classifying unseen instances, but the classifier should also be easy to interpret. These results conclude that the decision trees are good at accuracy whereas GP when used for classification produces trees with noticeably smaller size. This result directs the work towards combining Decision Tree and GP in order to produce an optimal classifier in terms of accuracy and comprehensibility.

Few researchers have made proposals of merging genetic programming and decision tree. The major Reasons for merging GP and Decision Tree [20][21][22]:

- In literature, Decision Trees have been proved best as far as accuracy and training time are concerned. However, when training data are given incrementally, decision trees cannot be used. Collaborating Decision Tree with GP can facilitate this
- Moreover, when training data is large or number of classes are more, decision tree becomes very large. This can be solved using tree size as a fitness measure in GP and hence the issue of comprehensibility can be addressed
- Tree structures are used as encoding scheme for GP individuals. Hence combining decision tree classifier with GP is an preferable approach

Thus as the experimental and theoretical conclusions suggest that merging GP with Decision Trees is a preferable approach, the work hereafter focuses on only GP and Decision Tree. The next chapter shows the work done in literature in concern with merging GP and DT.

Chapter 3

Related Work and Issues

The use of evolutionary algorithms for training classifiers has been studied in the past few decades. Decision trees are one of the most frequently used representations for classifiers. A vast amount of literature has been devoted to this form of classification. Moreover, Genetic programming (GP) is a flexible and powerful evolutionary technique with some features that can be very valuable and suitable for the evolution of classifiers. Also, the experimental results and theoretical conclusions derived from the previous chapter suggest that the application of GP to evolve decision trees seems to be an obvious approach. Hence this chapter presents a survey on existing literature about the application of genetic programming to classification (and specifically decision trees), to show the different ways in which this evolutionary algorithm can help in the construction of accurate and reliable classifiers.

3.1 Survey

In [23], evolutionary approach based classification model is tested and compared with non evolutionary approach. The paper concludes that decision trees are better for classification when a choice is made among non-evolutionary approach based classification algorithms. EAs have highest predictive accuracy as compared to other non-evolutionary approach based classification algorithms. In terms of training time,

EAs are slow but the comprehensibility of EA based classifiers is better compared to the non-evolutionary approach based classification algorithms. Hence, EA is the best choice when comprehensibility is a selection criterion and decision tree is the first choice when training time is a selection criterion.

In [20], design of decision trees through integration of C4.5 and GP is presented where the population is initialized by C4.5. Specifically, each tree in the initial population is designed by C4.5, using a part of the training examples. By doing so, relatively good DTs from the very beginning can be obtained which can be used while waiting for better DTs to emerge. The design method in an evolutionary process containing two phases:

a. Select part of the training examples at random from the whole training set, and design a BDT using C4.5. Repeat this for all trees in the initial population.

b. Evolve the trees using GP.

The paper shows that trees obtained through integration of C4.5 and GP (C4.5 + GP) are even worse than C4.5. The paper concludes that the approach requires modification.

In [22], different ways of establishing the classification technique using GP has been stated and in each method the issues which need to be analyzed for GP are mentioned.

The methods which can be used to create a decision tree are as follows:

a. Each individual in a generation represents a class. It can be carried out in two ways. One way is to repeat the GP algorithm as many times as the number of classes for which rules have to be made; we consider a specific class each time and carry out the GP algorithm for that class only. The second way is to use a Genetic Program in which the number of individuals in each generation

is many times bigger than the number of classes, and then in each generation some individuals are considered for one class.

b. All classed are expressed through one individual. In this case each individual in a generation covers all the classes.

The paper states that some additional conditions have to be checked which are:

- In the first method where each individual is planned for one class, care should be taken that the individuals participating in the Crossover belong to the same class
- When one sub-tree from an individual is exchanged with a sub-tree from another individual, care should be taken that the established paths from the root toward the leaf nodes not have contradictory conditions, nor should there be cases of repetition

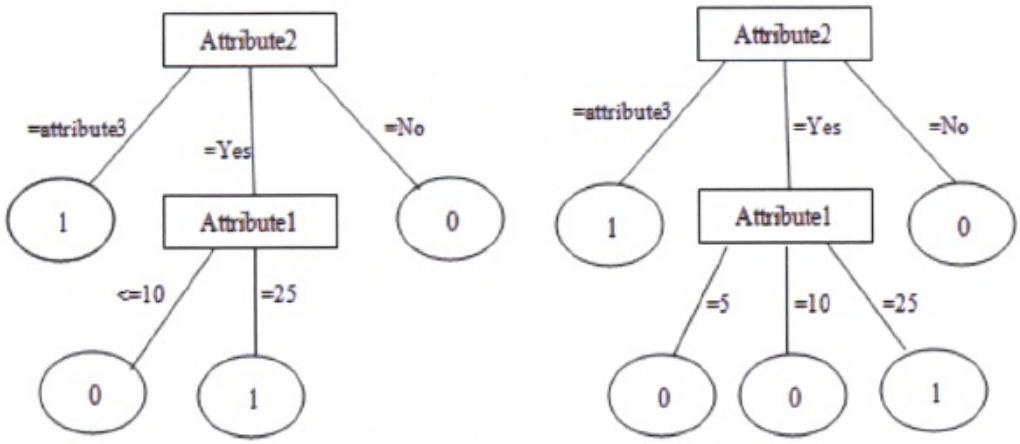


Figure 3.1: Example of decision trees created using GP

The paper shows that using while using GP with decision trees, it is possible to compare same-type attributes in data sets in order to create rules of classification in a way as to reduce the number of rules significantly as compared to the traditional methods where such a comparison was not possible. This is picturised in figure 3.1

The paper [24] presents an approach for designing classifiers for a multiclass problem using Genetic Programming (GP). For a c-class problem, an individual of population will contain a tree for every class. Hence a multi-tree classifier consisting of c-trees is evolved. Each tree will represent a classifier for a particular class. Further, the paper says that 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. Also the selection of trees for the mutation operation is according to their unfitness, so, the chance of unwanted disruption of already fit trees is reduced and the chance of evolution of weak trees is increased. Further, a kind of directed mutation is used, which always accepts mutations that improve the solution. After mutation, a decision is to be made as to whether the mutated tree will be retained or ignored.

In [25], an MOGP system designed to optimize two goals is described, but these goals are not fixed. The authors make a general proposal, allowing for any pair of goals to be considered. In this paper, a cost-sensitive point of view is adopted, and related pairs of goals are suggested, such as false negative rate/false positive rate, sensitivity/specificity, or recall/precision, but any other pair of goals could be employed.

In [21], decision trees are generated through evolution of training data. Usually, DTs are designed from fixed data. Decision trees (DTs) are often considered comprehensible because there is a reasoning process for each conclusion. When the data set is large, however, DTs obtained may become very large, and they are no longer comprehensible. To increase the comprehensibility, it is necessary to reduce the tree sizes. The papers states that to find the minimum DT is an NP-complete problem. Hence, the basic idea is to evolve a small data set that can cover the domain knowl-

edge as good as possible. From this dataset, a small but good DT can be designed. Here, N training sets are extracted at random from the entire training set D and a DT from each training set is constructed using C4.5. Fitness of each tree is then measured using the validation set V (=D here). Genetic operators are further applied on the training sets(subsets of D) that generate fit decision trees. The rationale is, if the best training set, after evolution, can cover the domain knowledge well, the DT designed from it must be good as well.

In [26], the authors investigate the feasibility of using Genetic Programming in dynamically changing environments to evolve decision trees for classification problems and proposes an new version of Genetic Programming called Adaptive Genetic Programming. To cope up with the changing environment, a new control parameter known as culling is introduced.

Culling is a mechanism to jump start the exploration capability of an evolutionary algorithm. It was applied by removing a portion of the worst solutions from a population and replacing these with randomly generated solutions. Replacement solutions were constructed by generating a pool of trees of size 1, only consisting of terminal nodes as the root nodes, and then applying the mutational operators to these solutions, afterwards inserting them into the population.
Also, various mutation operators are explained which are: Growing, Truncation, Shrinking, Node Altering, Physical swap and Logical Swap.

In [27], GP is employed to achieve a selection of features. Each individual is a tree encoding a classifier represented as a discriminant function, and classification accuracy is used as the fitness function. The authors capitalize on the implicit feature selection ability of GP classifiers by applying GP for classification in a two-stage scheme. First, a certain number of GP runs are carried out, each one resulting in a best-of-run classifier. Then, in the second stage, the GP is run again using only the

features most frequently used in the best classifiers obtained in the previous stage. However, the goal of the first stage is just to discover the most interesting features, and the classifiers obtained are discarded. The final classifier is induced in the second stage using the features selected in the first stage.

Further, [28] presents an analytical study on the influence of the fitness function and mutation operator on the quality of the decision trees induced, taking into account both classification accuracy and tree size. One of the main points of interest when constructing decision trees with GP is how to obtain accurate and parsimonious classifiers, since simple classifiers are more comprehensible to humans. This concern is addressed by many researchers, not only when evolving decision trees, but also when other representation formalisms are employed.

In [29], a new mutation technique called Mutation on RHS has been proposed. Given a condition, the method consists of changing the right hand side of the condition into a randomly chosen (allowed) new one. This can either be a value of the attribute or another attribute. Also, the paper uses pruning as a Genetic Operator.

The potential of GP to include any kind of operation into individuals is used in [30] in order to evolve oblique decision trees. The paper shows that in GP, individuals tend to become larger over time. This phenomenon is known as bloat. Disadvantages include overtraining, longer execution time for evaluating individuals and lower understandability of the trees for humans. Hence in this paper, the fitness function takes into account not only the classification accuracy but also applies two size penalty factors, one for the number of nodes and the other for the depth of the tree. Also, it is examined that the exact amount of mutation and crossover was of no high importance, as long the mutation rate is larger than 0 percent. Hence the mutation and crossover rate was set to 0.5 as there was a peak in validation accuracy at 50 percent in the experimentation shown in the paper.

In [31], a survey of evolutionary algorithms that are designed for decision-tree induction is presented. In this context, most of the paper focuses on approaches that evolve decision trees as an alternate heuristics to the traditional top-down divide and conquer approach. Additionally, some alternative methods that make use of evolutionary algorithms to improve particular components of decision-tree classifiers are shown.

The paper provides an up-to-date overview that is fully focused on evolutionary algorithms and decision trees and does not concentrate on any specific evolutionary approach.

Also, it provides a taxonomy, which addresses works that evolve decision trees and works that design decision-tree components by the use of evolutionary algorithms. Further, the paper addresses some more important issues and open questions that can be the subject of future research in concern with combining decision tree and evolutionary algorithms.

In [32], the authors have proposed an algorithm based on genetic programming to search for an appropriate classification tree according to some criteria. The classification tree obtained is transferred into a rule set, which can then be fed into a knowledge base to support decision making and facilitate daily operations. Two new genetic operators, elimination and merge, are designed in the proposed approach to remove redundancy and subsumption, thus producing more accurate and concise decision rules than that without using them. The paper also presents experimental results from the credit card data to show the feasibility of the proposed algorithm.

3.2 Summary

This extensive literature survey shows various approaches used to merge GP and Decision Tree.

In the proposed algorithm, the strategy of using decision trees for producing initial population instead of initializing the population randomly is adopted. Further, emphasis is laid on the sayings from [24] that the nature of mutation is destructive and a solution to this is suggested in the proposed algorithm.

Taking motivation and lessons from this survey, the next chapter proposes a new algorithm that merges GP to Decision Trees and attempts to produce an optimal classifier in terms of accuracy and comprehensibility.

Chapter 4

Proposed Algorithm

This chapter proposes a new algorithm named **G**enetic **P**rogramming based **E**volution of **C**lassification **T**ree **(GPeCT)** that performs a trade-off between accuracy and comprehensibility. As the name suggests, the proposed algorithm is a hybrid version of GP and Classification Tree(Decision Tree).

The algorithm is especially designed for large datasets and for problems involving multi-class classification because the increase in dataset size or number of classes directly affects the size of the trees, resulting into trees that are not comprehensible enough. As a solution to this issue, the proposed approach gives equal importance to size of the tree and its accuracy by designing the fitness function accordingly. Undoubtedly, the approach works perfectly well for binary classification problems and small datasets.

The proposed algorithm uses some lessons learnt from the literature survey and combines them to produce a new classification technique. The basic approach is to form a population of decision trees and evolve them using GP with accuracy and number of nodes as fitness parameters.

4.1 Algorithm GPeCT

The proposed approach is shown in algorithm 1.

Algorithm 1 GPeCT. Evolves a classification tree.

Input: Data Partition, D_t, which is a set of training instances and their associated class labels

Output: *BEST_SO_FAR*, the optimized classification tree

1. **for** $i = 1$ to *POP_SIZE* **do** // Initial population
2. Create a decision tree from a random subset of D_t
3. **end for**
4. Evaluate fitness of each individual in the initial population using fitness measure of Equation 4.1 and D_t
5. **while** *GENERATION* $<$ *MAX_GENERATIONS* **or** Termination criterion not satisfied **do**
6. **repeat**
7. Perform genetic operations on parent trees to generate offspring trees for the new population
8. **if** (*Size of Individual* $<$ *c1* $*$ *Size of BEST_SO_FAR*) **then**
9. Evaluate its fitness and increment *NEW_POP_SIZE*
10. **else if** (*Accuracy of Individual* $>$ *c2* $*$ *Accuracy of BEST_SO_FAR*) **then**
11. Evaluate its fitness and increment *NEW_POP_SIZE*
12. **else**
13. Discard the individual
14. **end if**
15. **until** *POP_SIZE* individuals are produced
16. *GENERATION* $\leftarrow$ *GENERATION* $+ 1$
17. *POP_SIZE* $\leftarrow$ *NEW_POP_SIZE*
18. **end while**
19. Save the population of last generation for *incremental learning*
20. Designate best individual found so far as *BEST_SO_FAR*

4.2 Detailed Analysis of GPeCT

The input to the algorithm is the set of training instances along with their associated class labels. As an output, the algorithm gives us a classification tree *BEST_SO_FAR*, which is an optimal classifier in terms of accuracy and comprehensibility. Initially, *POP_SIZE* is set equal to the desired size of the initial population and as the name suggests, the parameter *MAX_GENERATIONS* is set equal to the number of maximum generations the algorithm is desired to run.

The detailed explanation of the algorithm 1 is as follow:

4.2.1 Initializing Population

The algorithm begins by creating an initial population made up of decision trees. Forming initial population using decision trees has a plus point over initializing the population randomly with terminals and functions. The advantage is that initializing population using Decision Trees would give us good (in terms of accuracy) trees from the beginning as compared to the random trees[20][33].

Further, to obtain the initial population, the training data is divided into *POP_SIZE* random parts. Moreover, it is made sure that each record in the training data is covered at least once in any of the part. Furthermore, from each part, a decision tree is formed and added to the population. This is how initial population of size *POP_SIZE* is formed.

The algorithm used to create initial decision trees is the CART. The reason for selecting CART as the decision tree algorithm out of the many is as follow:

CART, that uses Gini index as an attribute selection measure forms binary trees. From the literature survey it is observed that binary trees as preferable for GP as

most of the researchers have applied evolutionary algorithms on binary decision trees only. Further, CART has a provision to deal with noisy data. Hence it is preferable over the highly used C4.5 that uses Gain Ratio as an attribute selection measure, as C4.5 does not implicitly deal with noise.

The above method of initializing population can be presented in a nutshell in figure 4.1:

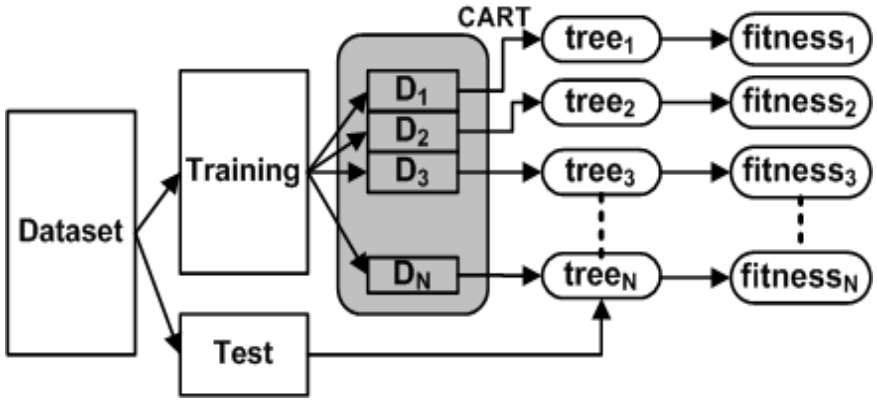


Figure 4.1: Method of Initializing Population in GPeCT

4.2.2 Fitness Measure

The fitness of each individual (tree) is evaluated using the fitness measure shown in equation 4.1:

$$\text{Fitness} = \frac{\text{Accuracy of the Decision Tree}}{[\text{Number of nodes in the tree}]^{\gamma}} \tag{4.1}$$

where γ is a control parameter representing a tradeoff between accuracy and size. Its value can range from 0.05 to 0.1.

The fitness function is proposed such that a trade off between accuracy and size of the tree(comprehensibility) can be provided. As the value of γ increases, the value of the denominator increases, i.e. trees with big size are assigned less fitness. The impact of trade-off factor γ is shown in Table 4.1.

Table 4.1: Impact of γ on Fitness Measure

Accuracy	*No. of Nodes*	γ	$[No.\ of\ Nodes]^{\gamma}$	*Fitness Measure*
80	100	0.02	1.26	63.49
90	100	0.02	1.26	71.43
90	200	0.02	1.3	69.23
80	100	0.1	1.58	50.63
90	100	0.1	1.58	56.96
90	200	0.1	1.70	52.94

Table 4.1 has been mathematically as well as experimentally proved. The table shows that, with the same value of γ, when the size of the tree is same but accuracy of one is more over the other, the fitness of the tree with high accuracy is obtained higher. On the other hand, if accuracy of two trees are same but size of one is more than the other, the fitness of tree with smaller size is more. And that does exactly what the algorithm wants to perform, that is, performs a trade off between accuracy and comprehensibility.

With less value of γ, the size parameter has less effect. With more value of γ, the size parameter has higher effect and the fitness obtained is less.

4.2.3 Genetic Operations

The algorithm performs three genetic operations on the parent trees of existing population to generate the new population. They are:

1. Reproduction
2. Crossover
3. Mutation

The method of selecting individuals for genetic operation, operation probability, etc. are presented in Table 4.2:

Table 4.2: Parameters concerning genetic operations in GPeCT

	Reproduction	**Crossover**	**Mutation**
Probability	0.2	0.6	0.2
Number of individuals used and produced per operator	1	2	1
Method used to select individuals for genetic operation	Rank Selection	Tournament Selection	Reverse Rank Selection

The probability of reproduction operator is set 0.2. Further, for reproduction, the best individuals are preferred using the Rank Selection Method. This is to make sure that the fittest members are more likely to be passed on to the next generation, and past optimal solutions are not lost[34].

The probabilities of crossover and mutation are set 0.6 and 0.2 respectively. For crossover, tournament selection is performed and for mutation, Reverse Rank Selection is used. That is, it prefers unfit individuals over the fit individuals. This is to save good trees from undergoing the random changes of mutation, because as per[24][34][35], the nature of mutation is destructive at times. Hence, the probability of selecting unfit individuals for mutation is set more using Reverse Rank Selection.

As we are using Reverse Rank Selection, assigning the probability of selection of mutation operator equivalent to (i.e. as high as) reproduction operator is acceptable.

4.2.4 Validation of Individuals

The paper introduces a new validation process for the individuals produced through the genetic operations. The individuals with tree size less than c_1 times the best so far are validated. If any individual has size more than it, its accuracy is checked. If its greater than c_2 times the best so far, it is validated. Otherwise it is rejected and its fitness is not calculated. This allows saving the time in calculating the fitness of already known bad individuals. The fitness of only validated individuals are calculated.

The individuals who pass the above test and get validated are added to the new population and the counter of *NEW_POP_SIZE* is increased by one per individual. This parameter at the end of the generation gives the new population size which is to be used for next generation. This way, the algorithm uses variable population size.

However, a generation is incremented only if required amount of individuals are produced. It does not matter if those individuals are accepted or rejected. Hence, to check if the generation should be incremented, with every genetic operation, the count of number of individuals produced is increases accordingly. Once the required amount of individuals, i.e. *POP_SIZE* individuals are produced, the next generation is started. For this new generation, *NEW_POP_SIZE* becomes *POP_SIZE*.

4.2.5 Termination Criterion

The algorithm stops when the termination criterion is met or maximum number of generations have been developed. The termination criterion here is that for p consecutive generations, the best q fitness values achieved are the same. For example, considering $p = 2$ and $q = 3$, when the best classifier obtained, second best and the third best out of the population are same for 2 consecutive generations, the algorithm terminates.

Once the termination criterion is met or the maximum number generations have evolved, the *BEST_SO_FAR* reported by now is made the output classifier and this is how the algorithm works.

4.2.6 Incremental Learning

Furthermore, in real world applications, data might be produced incrementally. Hence any classifier must be capable of incorporating incremental data. However, this is not possible using only CART or any other decision tree algorithm[36]. The proposed algorithm has this advantage over the traditional decision tree algorithm that, data can be given incrementally here. The population of the last generation produced while evolving the classifier is saved. Later, when new training instances come at hand, trees induced from this data are added into the population and one more GP run is conducted. The *BEST_SO_FAR* obtained after this run becomes the final classifier. This classifier would incorporate training instances available earlier as well as the newly arrived instances. Thus the proposed algorithm allows incremental learning.

Chapter 5

Implementation

This chapter covers the details of the tool used for implementation followed by a block diagram picturizing the modular flow of the project and other relevant details. The chapter also presents the snapshots of the running algorithm and explains the significance of some steps on the final output.

5.1 Tools Used

CART and the proposed algorithm GPeCT have been implemented in MATLAB 7.8.0 (R2009a) whereas GP for classification is implemented in HeuristicLab 3.5.33, an open source under GNU General Public License. All the experiments are performed on Windows Platform.

The reason behind choosing MATLAB for Implementation of the proposed algorithm is as follow:

- Interactive interface
- Debugging facilities
- High quality graphics and visualization facility

- MATLAB's add on feature in the form of toolboxes: making it possible to extend the existing capabilities of the language with ease
- Can manipulate large amount of data: which is the basic requirement in Data Mining
- Researchers in [37] suggest that a way to speed up the GP algorithm is to make a faster implementation using C/C++/MATLAB rather than JAVA: Hence the well known tool WEKA is not used.
- MATLAB has an innumerable in-built functions ready in it which makes the implementation work very easy.

Initially, a plug in toolbox GPlab[38] was to be used for the implementation of the algorithm. However, the decision trees formed were ordered in BFS manner and the crossover, mutation, etc. functions used by the toolbox were applicable on trees ordered in DFS manner of traversal. As combining both of them was not possible, the toolbox was not used and the sole implementation is done in MATLAB and all the major required functions were designed explicitly.

5.2 Modular Flow

The block diagram of figure 5.1 shows the modular flow of the project.

In the PreRequisiteCheck stage, the dataset is imported and checked for noisy data. The algorithm also facilitates selecting few attributes out of many.

In the second stage, initial population is generated and evaluated for their efficiency using the training data itself. The algorithm practices to keep the best individuals at the top of array so that its comparison with newly developed individuals can be done easily. Hence a sort function is used that sorts individuals based on fitness.

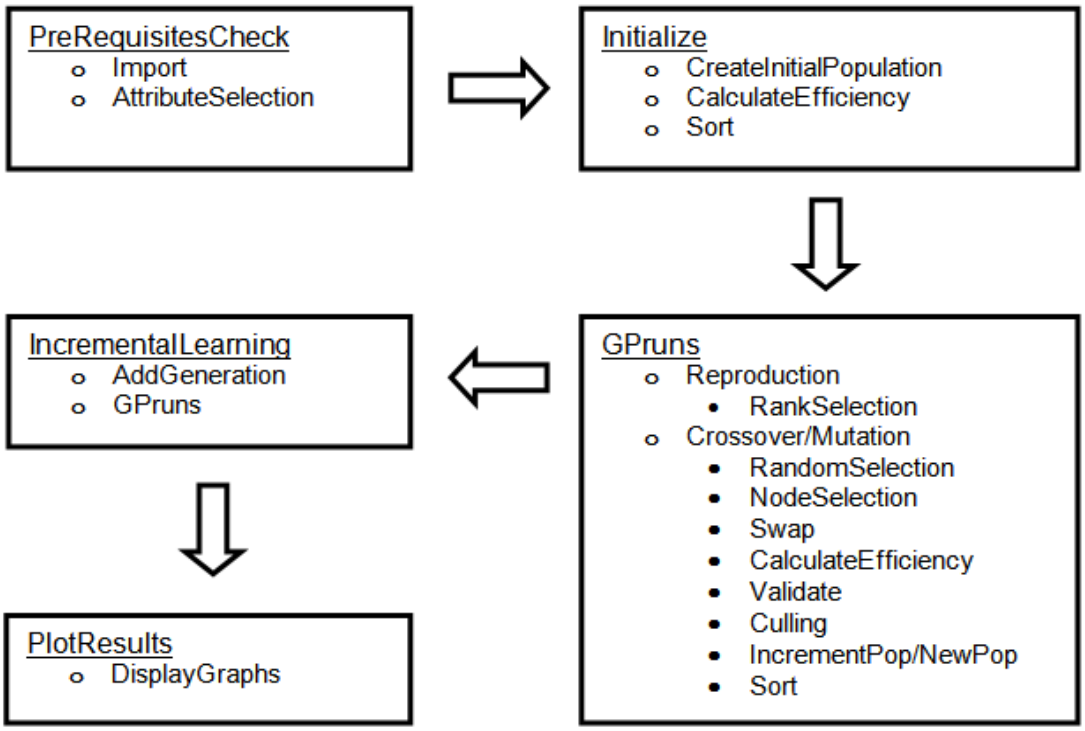


Figure 5.1: Block diagram of modules implemented

The third stage performs the GP runs until the termination criterion is reached and presents the best solutions at the top of the array. The module GPruns include performing Reproduction with Rank Selection method implemented in it. Further, it includes Crossover and Mutation functions with the stated sub-functions like swap, etc. The detailed working of crossover and mutation is explained in section 5.3

The results of the last generation are saved for incremental learning, which occurs after the algorithm has produced its best classifier and some new training data arrives at hand. In order to perform incremental learning, one more GP run is conducted on the saved population. This is why GPruns module is called again from Incremental-Learning module.

Finally, the results are shown in form of graphs and the GPeCT classifier is produced as output in form of a tree structure that is easy to interpret.

5.3 Methodology

The method of implementing each of the above stated modules and their significant impact on the overall performance of the algorithm is explained in this section.

5.3.1 Pre Requisite Checking

Figure 5.2 shows a GUI that enables selection of the attributes the user wants to allow for participating in classification. This feature is provided as the data might be noisy or some attributes may be irrelevant for classification. Such attributes can be eliminated here.

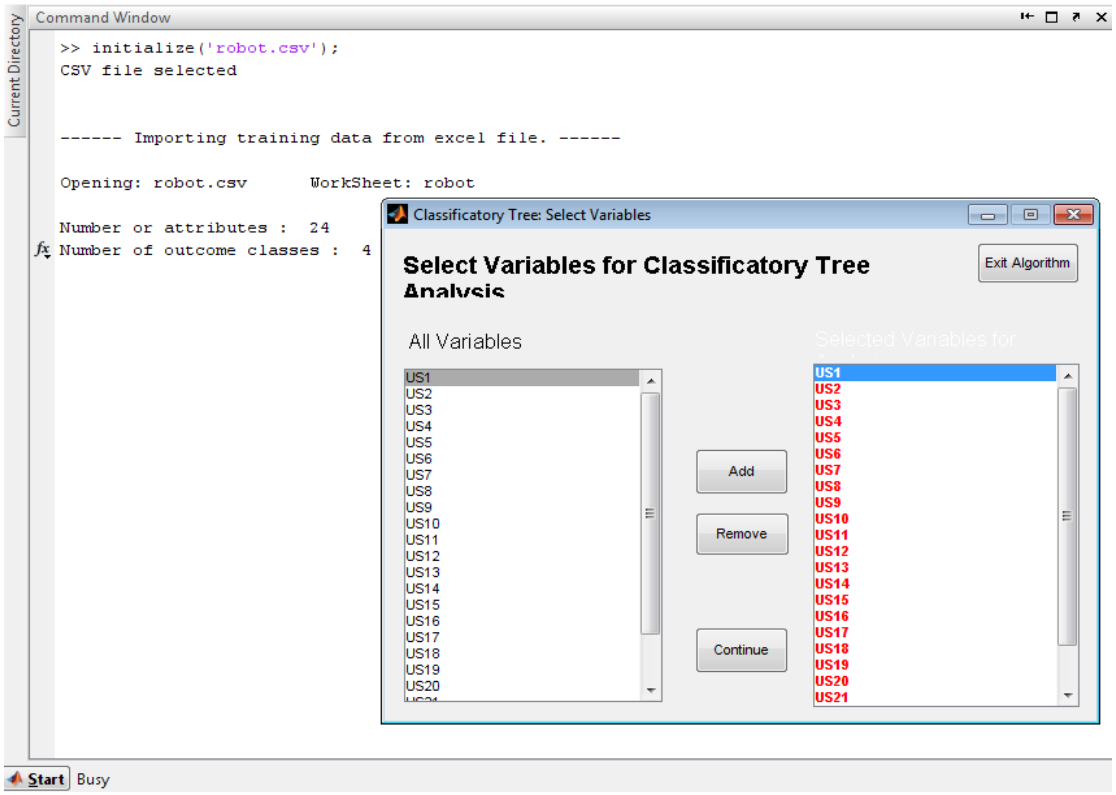


Figure 5.2: GUI enabling selection of attributes for classification

The implementation imports data directly from an excel or csv file, using the first row as attribute names (necessary). The first column is the outcome group which must be numeric. Although the initial population trees are developed using parts of dataset, fitness of the classifier is determined using the entire training set.

Hence, the accuracy reflects the effect of applying the classifier on the entire dataset and not the only part on which it was trained.

5.3.2 Initial Population

Figure 5.3 shows a snapshot of some of the trees of the initial generation.

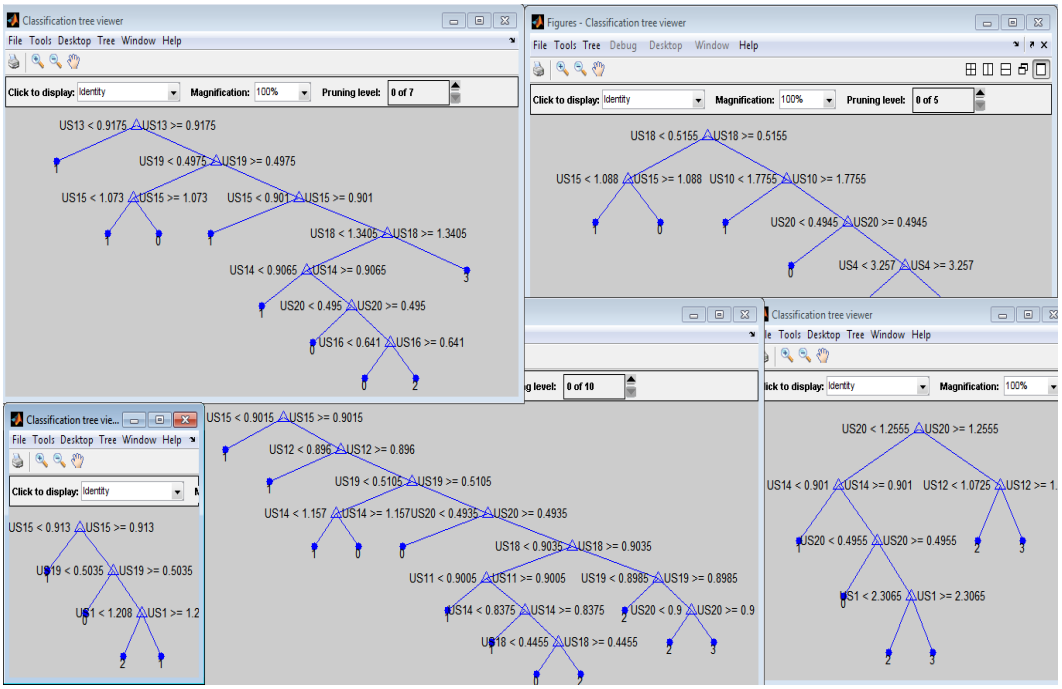


Figure 5.3: Sample of initial population generated(Decision Trees)

Few best individuals formed as the initial population on the Robot Navigation dataset described earlier are listed as in table 5.1

Table 5.1: A list of few Best Individuals

Individual Number	Fitness	Accuracy	No. of Nodes
16	65.710079	74.701472	13
3	63.069364	75.118023	33
71	61.863498	73.451819	31
42	59.618168	70.785893	31
65	58.222897	71.008053	53
84	57.769857	70.952513	61

The results show that by using Decision Trees for creating initial population, gives good trees from the very beginning which matches the theoretical assumption[33].

5.3.3 Genetic Operations

Once the initial trees have been developed, the next step is to perform crossover, mutation and reproduction upon them based on the results. Figures 5.5 to 5.8 show how crossover and mutation is implemented.

Figure 5.8 shows an offspring obtain by performing genetic operation on parents. In case of crossover, two parents undergo the above stated process whereas for mutation, one parent is mutated with a randomly created new tree.

Whenever a leaf node is to be swapped with a leaf node, the task becomes very simple and simply one node swap is required. However, if a node that has branches becomes the point of crossover or mutation, it is to be pruned first and then swapped with the crossover or mutation node of the other tree.

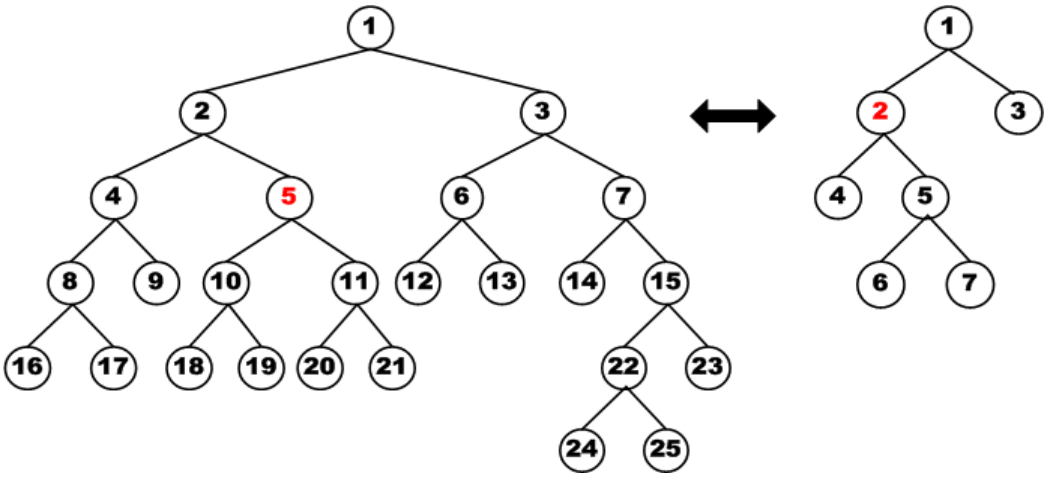


Figure 5.4: Step - 1: Randomly selecting crossover/mutation point

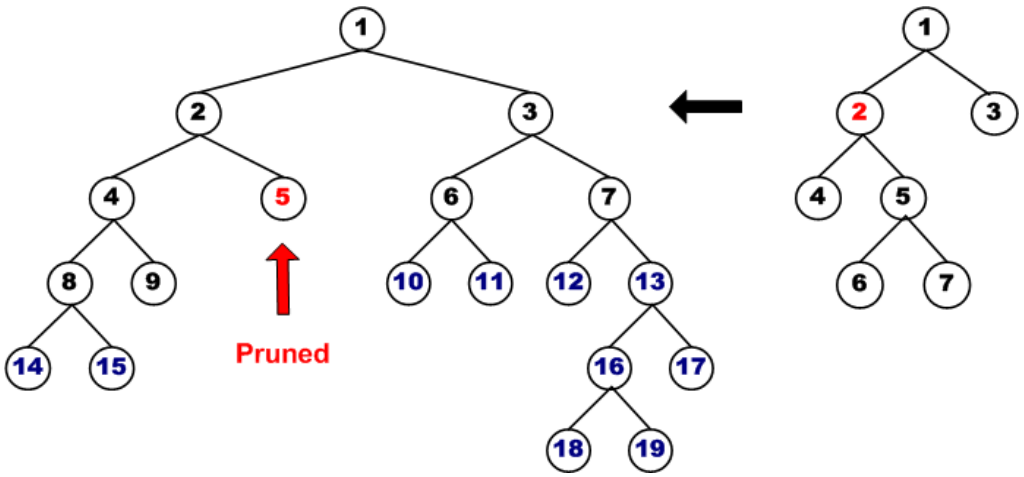


Figure 5.5: Step - 2: Pruning the node undergoing crossover/mutation

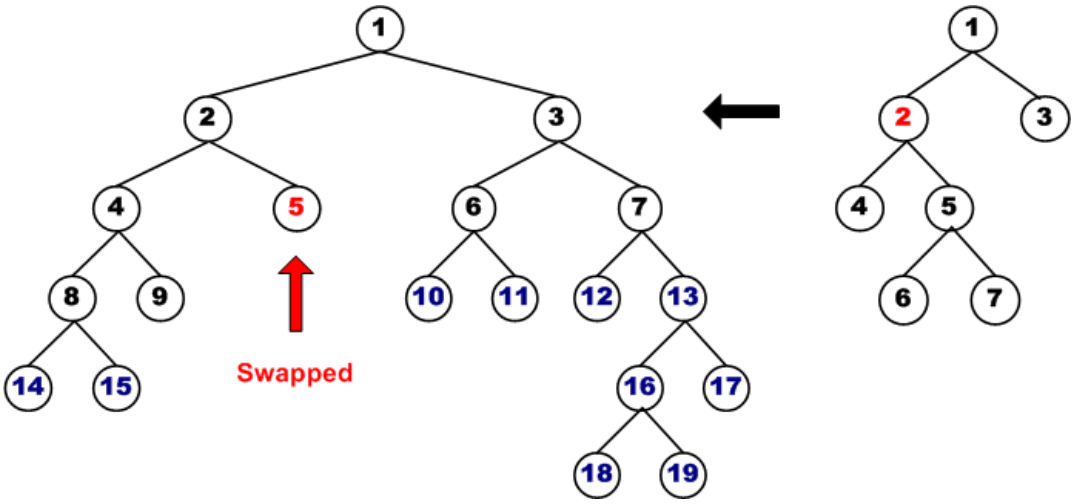


Figure 5.6: Step - 3: Substituting node 2 of Tree 2 at node 5 of Tree 1

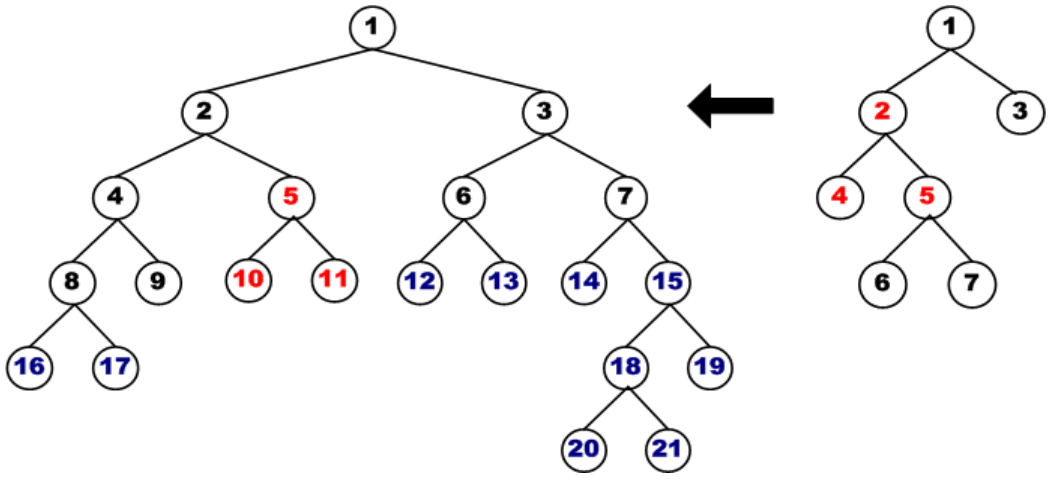


Figure 5.7: Step - 4: Addition of new nodes and changing properties of all following nodes

If the node that is added has branches, crossover or mutation takes place by finding the next node with branch and giving the number of its children to children of the swapped node. All the properties of the nodes that follow are to be changed.

Building crossover and mutation functions have been the heart of the implementation as it not only involves simple swapping of nodes, but all the properties of the nodes such as its parent, its children, etc. have to be swapped. Also, crossover and mutation are responsible for consuming the major time the algorithm runs. Further, the proposed algorithm works well with both numeric and text data.

5.3.4 Output Classification Tree

An example of the final classification tree developed on the Robot Navigation dataset is shown in figure 5.9. The view function of MATLAB is used to display the tree. The final tree is comprehensible enough to analyze on the given dataset.

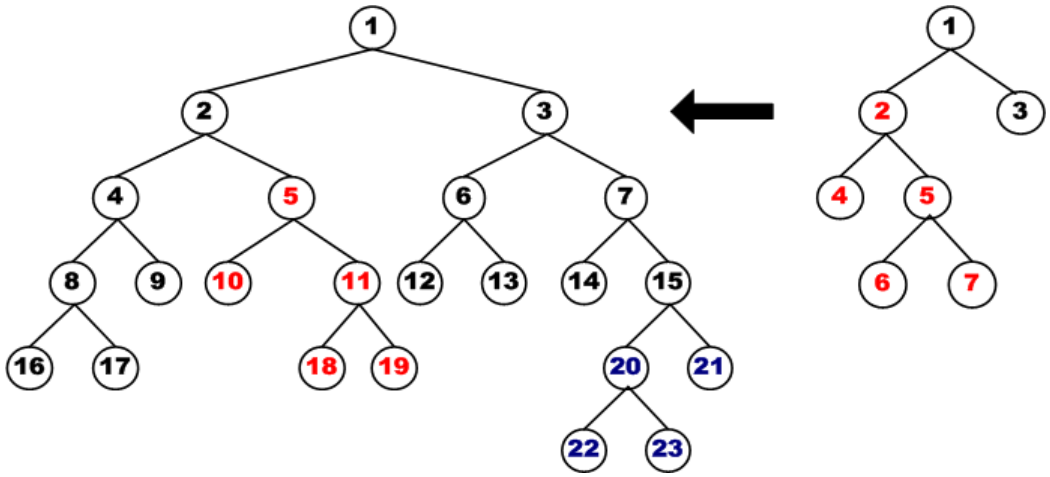


Figure 5.8: Step - 5: Addition of new nodes and changing properties of all following nodes

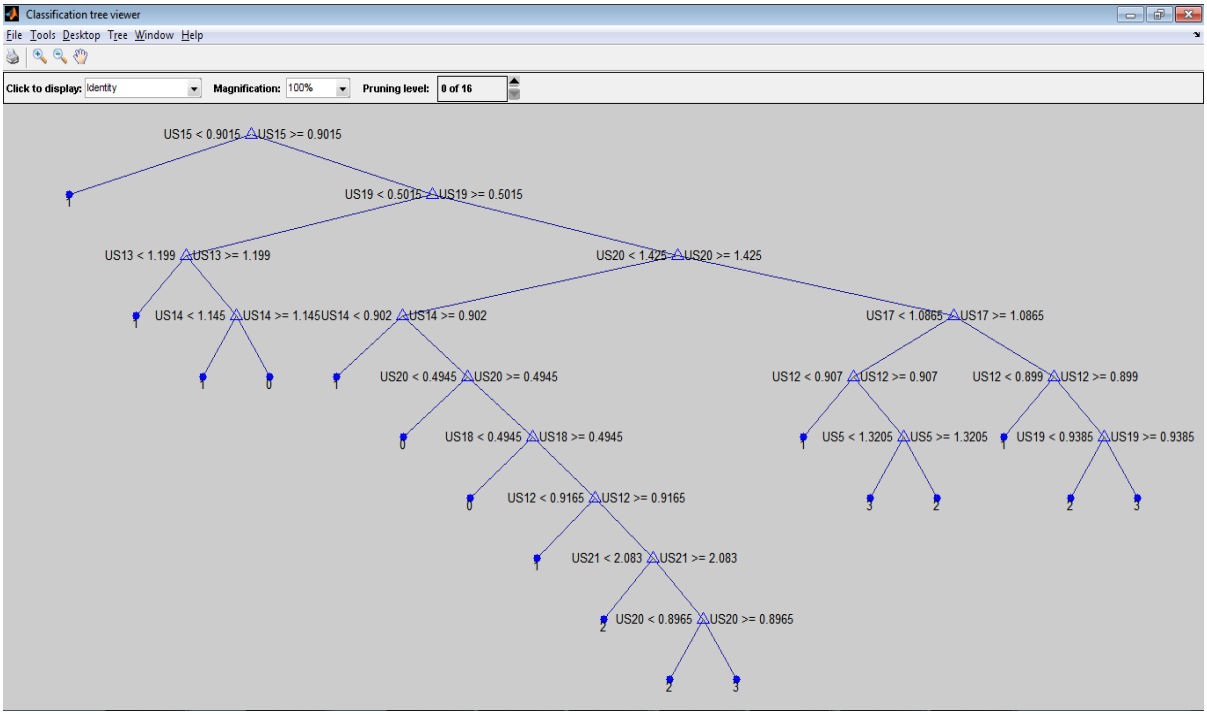


Figure 5.9: Output Classification Tree

Chapter 6

Results

Chapter 4 proposes an algorithm that merges Decision Tree and GP and the results of that proposed algorithm are discussed in this algorithm. Here a comparison is made between the well known Decision Tree algorithm CART, Genetic Programming as classifier and the proposed algorithm GPeCT.

6.1 Data Sets

To verify the effectiveness of the algorithm proposed in this paper, we conducted experiments on different datasets from various real domains. The datasets have been taken from machine learning repository of the University of California at Irvine. These datasets are the same as described in chapter 2, but they are repeated for better comprehensibility.

The aim behind using different datasets for experimentation is to prove the consistency of the proposed algorithm in different domains. The details about these datasets are listed in Table 6.1. The number of instances in these datasets range from hundreds to more than a million, verifying the performance of proposed algorithm on datasets with different sizes.

Table 6.1: Composition of Data Sets

	Attributes	Instances	Classes
Cardiotocography	21	2126	3
Robot Navigation	24	5456	4
Nursery	8	12960	5
Forest Cover Type	54	581012	7
Poker Hand	10	1025010	10

In the experimentations, 66% of each of these datasets have been used for training the classifier and the rest for testing it. The next section describes in detail the results of experiments on five datasets, each representing a multi-class problem. The algorithm has also been tested on many other datasets representing multi-class as well as binary classification and the test results are shown in Table 6.4 at the end of the chapter. Although the algorithm has been designed especially for multi-class classification problems, it works equally well on problems with binary classes. Hence, experimental results have also been performed on binary classification problems in order to show the generalized behavior of the proposed algorithm on classification problems with any number of classes.

6.2 Experimental Results and Findings

Some initial runs were performed to compare settings for tournament size, trade off factor and validation factors. The results showing fitness, accuracy and number of nodes of individuals for different values of trade-off factor is shown in Table 6.2. The results show that 0.05 is the optimal value of trade-off factor γ on the above datasets. Hence the algorithm uses value of γ as 0.05 throughout the experiments.

Table 6.2: Optimal value of Trade-off factor γ

Dataset	Trade-off factor γ	Fitness	Accuracy	No. of Nodes
Cardiotocography	0.01	92.31	96.56	123
	0.05	**75.24**	**94.07**	**87**
	0.1	61.35	88.5	39
Robot Navigation	0.01	95.34	99.45	68
	0.05	**82.86**	**99.25**	**37**
	0.1	56.69	76.86	21
Nursery	0.01	78.86	83.24	223
	0.05	**62.89**	**79.65**	**113**
	0.1	50.48	72.44	37

Table 6.3 summarizes all the concluded optimal GPeCT parameters used for experiments.

Table 6.3: Optimal values of GPeCT Parameters

Parameter	Value
Initial Population Size	100 [17]
Maximum Generations	100 [17]
Tournament Size	7 [17]
Tradeoff Factor γ	0.05
Validation Factor *c1*	5
Validation Factor *c2*	0.5
Termination Criterion	Same 3 BEST_SO_FAR for 3 consecutive generations

The heart of the algorithm are the crossover and mutation operations which lead to increase in accuracy from a smaller value to significantly high values. The average results of how the crossover and mutation operations perform on all the stated datasets is shown in figure 6.1 and 6.2 respectively.

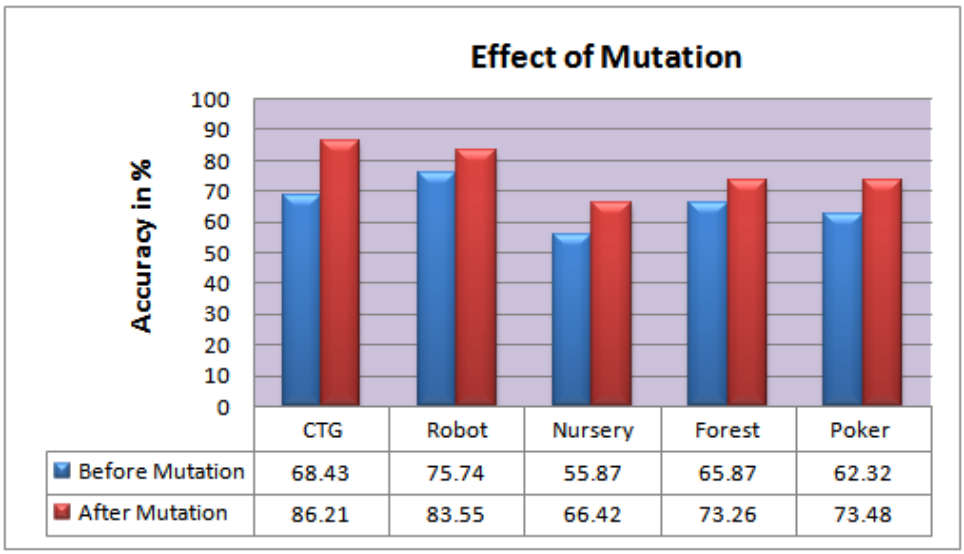


Figure 6.1: Effect of Mutation

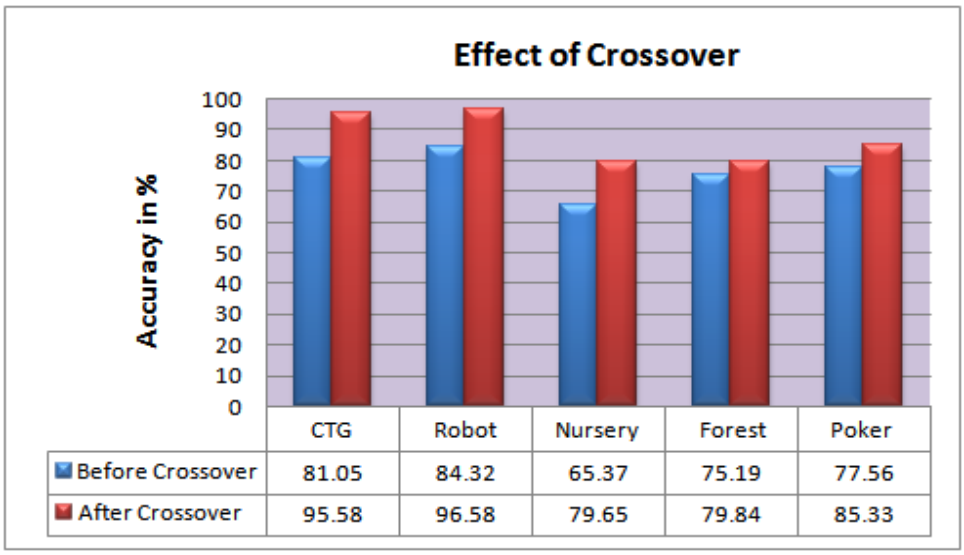


Figure 6.2: Effect of Crossover

The results show that on an average, mutation raises the accuracy of weak individuals to a higher value whereas crossover increases their accuracy to the highest values.

Figure 6.3 shows how accuracy of the individuals increase with generations. The results show that after a finitely small number of generations, the termination criterion is reached and the algorithm terminates. In such cases, the graph becomes a straight line showing that the accuracy is no more increasing.

For Nursery dataset, the fitness becomes stable after 17 generations whereas on Robot Navigation, it takes 13 generations for fitness to stabilize. On the other hand, GPeCT runs for 19 generations on Cardiotocography(CTG) dataset to terminate.

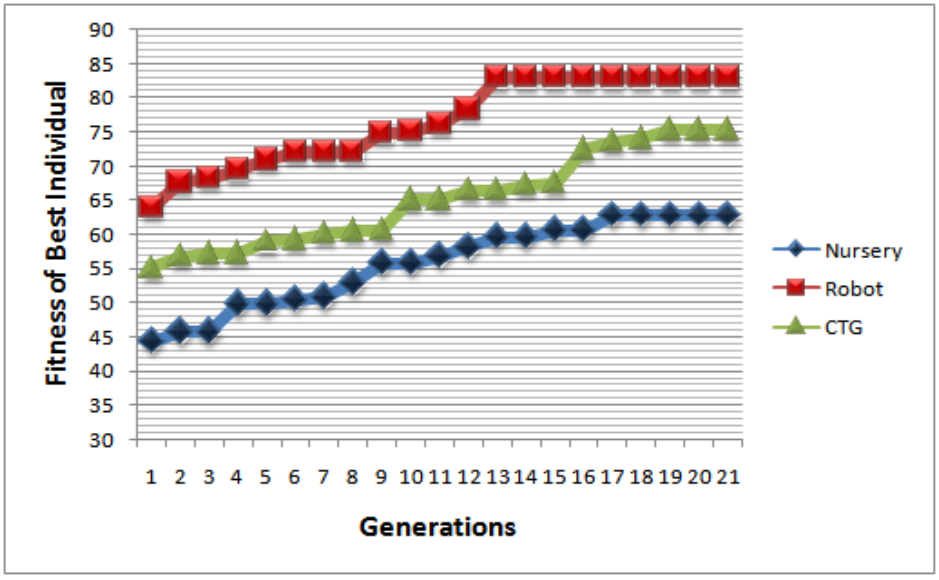
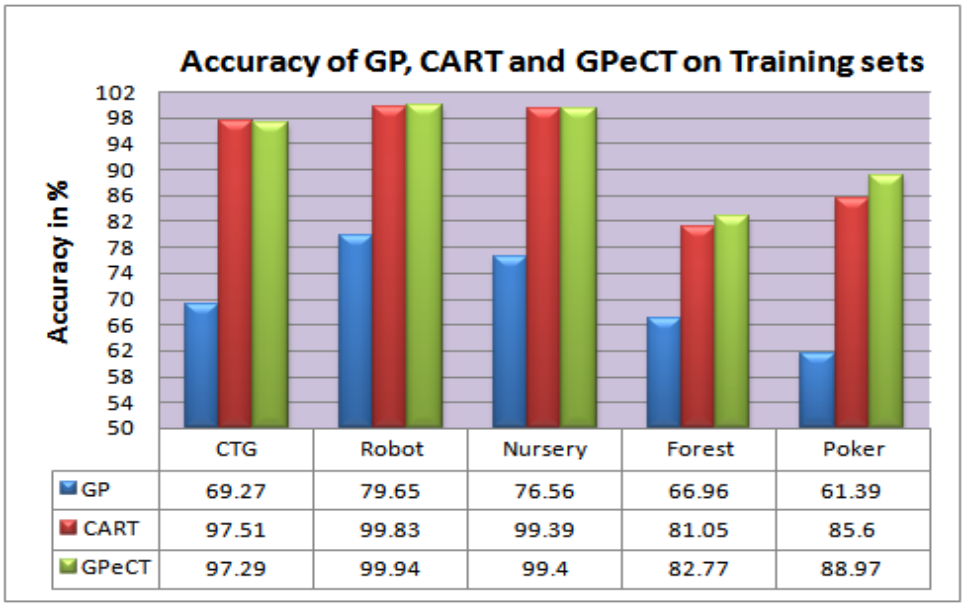


Figure 6.3: Increase in Fitness of Individuals with Generations

Figures 6.4 and 6.5 respectively show the results of accuracy (in %) on training and test instances whereas Figure 6.6 shows the results comprehensibility (number of nodes) of the classifier on the above stated datasets for the three algorithms CART, GP for classification and GPeCT. As explained in chapter 2, with increase in number of classes and length of dataset, CART (i.e. decision tree) performs good in terms



	CTG	Robot	Nursery	Forest	Poker
GP	69.27	79.65	76.56	66.96	61.39
CART	97.51	99.83	99.39	81.05	85.6
GPeCT	97.29	99.94	99.4	82.77	88.97

Figure 6.4: Accuracies of GP, CART and GPeCT on Training instances

of accuracy but comparison of classifier is less, whereas GP produces classification trees with less number of nodes and hence is highly comprehensible. This can also be concluded from the results above. Hence the goal of our proposed approach was to perform a trade-off between accuracy and number of nodes and the results shown in figure 6.5 and 6.6 verify that we have been successful in performing the trade-off. Not only a trade-off, in most of the cases, the accuracy has also increased in comparison to CART.

On training instances, the classifier performs as good as CART in terms of accuracy whereas the number of nodes have significantly decreased. On the test instances, GPeCT outperforms GP and CART in most of the cases.

For example, on CTG and Nursery datasets, the accuracy obtained by applying GPeCT is remarkably more than CART. Moreover, as GPeCT is a result of merging GP to CART, the number of nodes have also been reduced as compared to CART. On the other hand, the accuracy of GPeCT on Robot Navigation dataset is almost

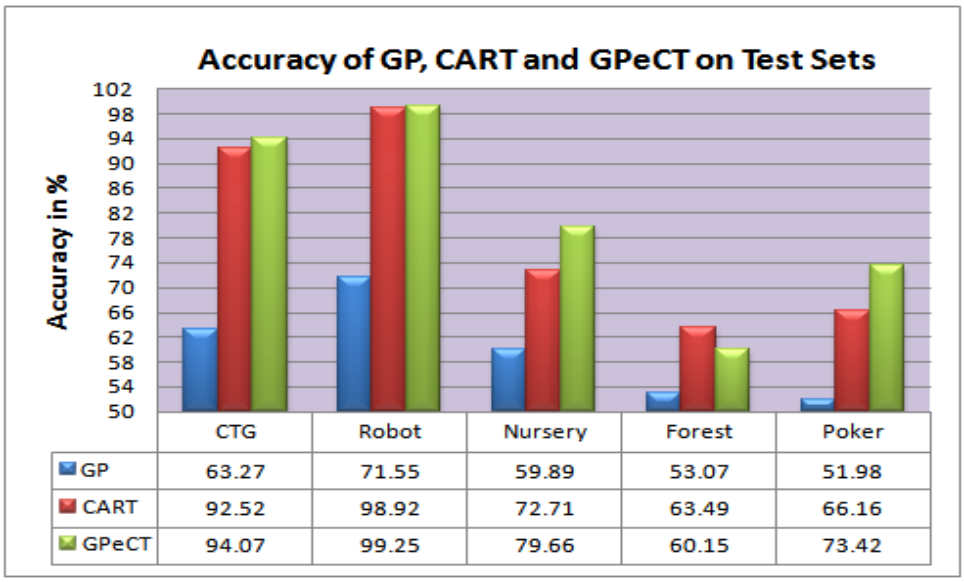


Figure 6.5: Accuracies of GP, CART and GPeCT on Test instances

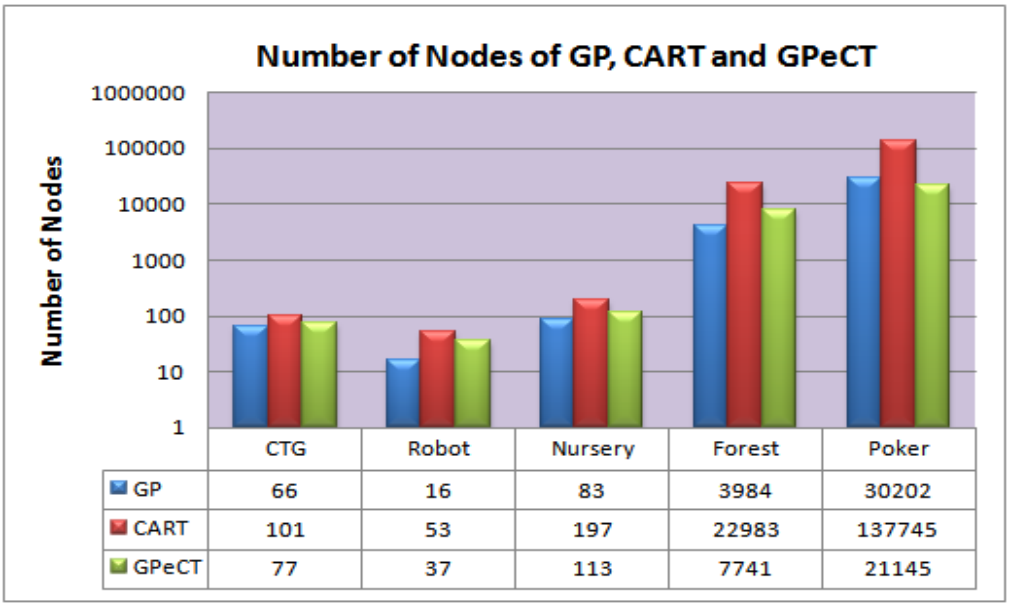


Figure 6.6: Number of nodes in trees generated by GP, CART and GPeCT

equivalent to CART, and the number of nodes in the classification tree are convincingly less than CART resulting into a triumphant trade-off.

The only database on which GPeCT has a notably less accuracy is on the Forest cover-type database. But, the number of nodes have decreased by approximately three times and hence the reduction in accuracy is quite acceptable.

Further, the largest dataset used, that is the Poker-hand dataset, is classified the best with GPeCT as compared to CART and GP. Not only is the accuracy of GPeCT fairly high, but the number of nodes have also been very significantly decreased (almost 6 times). This shows that GPeCT performs at its best when mining interpretable results from very large datasets.

6.3 Comments on Classification Time

As far as the time complexity is concerned, it is quite obvious that Genetic Programming takes more time than CART as it continues for a number of generations. GPeCT being a combination of CART and GP, clearly takes more time in construction of the classifier. However, the construction of classifier is a one-time cost and once generated, what is of importance is the amount of time taken to classify new instances using that classifier.

As the results show, the tree size of CART is considerably large and hence it takes considerably high time in classifying the data. GPeCT on the other hand, may take time in building the classifier, but once trained, the new instances can be classified quickly as the size of the tree is small (at least smaller than tree developed by CART or any other decision tree algorithm).

Undoubtedly, pruning[39] the trees developed by decision tree algorithm reduces the

size, but it happens only by compromising in accuracy to a greater extent. GPeCT, while performing trade-off, sees to the fact that accuracy remains convincingly high.

GPeCT has been verified on several other datasets representing binary as well as multiclass problems. The results on some of them are shown in Table 6.4. As the results of CART and GPeCT are more significant, only they are compared. Further, the accuracy tabulated is for the test instances as the actual performance of the classifier can be measured as how efficiently it classifies the unknown instances.

Table 6.4: Experimental results on different datasets

Dataset	No. of Attributes	No. of Instances	No. of Classes	Optimization Parameter	CART	GPeCT
Iris	5	150	3	% Accuracy	96.07	98
				No. of nodes	9	7
Ecoli	8	336	8	% Accuracy	74.34	81.42
				#Nodes	31	23
Diabetes	9	768	2	% Accuracy	74.23	81.54
				#Nodes	103	27
Vehicle	18	946	4	% Accuracy	66.43	69.58
				#Nodes	121	99
CMC	9	1473	3	% Accuracy	48.8	49.6
				#Nodes	249	189
Segment	20	1500	7	% Accuracy	96.17	96.66
				#Nodes	77	53
Car Evaluation	7	1728	4	% Accuracy	95.22	94.2
				#Nodes	65	39
Spambase	58	4601	2	% Accuracy	80.36	91.59
				#Nodes	239	133

The previous results and the results from this table show that in almost all the cases, GPeCT outperforms CART in terms of accuracy and comprehensibility.

Further, the results presented are an average of experiments conducted three times using the same algorithm parameters. The reason for conducting the same experiment thrice is because GP and hence the algorithm GPeCT has its roots in randomization. The parts of training data selected for generation of initial population, the selection of trees for crossover, mutation, etc. is all randomly done. Hence, to assure the consistency of output, the average of three experimental results is presented.

Chapter 7

Conclusions and Future Scope

The chapter presents the conclusion derived from the work done and further scope of the work.

7.1 Conclusions

The work proposes a new algorithm GPeCT developed with an aim to perform a trade-off between accuracy and comprehensibility of the classifier. The work focuses on large datasets and multi-class classification. The implementation results show that the proposed algorithm outperforms the traditional decision tree classifier CART, as well as GP. Not only has a trade-off been performed, but the accuracy of the GPeCT classifier is obtained more than decision tree on most of the datasets. The number of nodes have also been decreased noticeably giving us a comprehensible classifier.

Using CART to produce the initial population has performed a major role in increasing the accuracy, as we have been getting good trees from the very beginning. Secondly, the fitness measure proposed succeeds in imparting more emphasis on trees with high accuracy, as well as less nodes, which in turn increases the efficiency of the algorithm.

Further, the algorithm uses variable size population by performing an extra step that simple GP, in which it validates some of the individuals and reject the few. This extra step saves time required in performing operations on the individuals that can be rejected at an early stage. Also, it saves a lot of computation involved in calculating fitness of few so-called "useless" trees

Clearly, with the decrease in population size, the overall running time of algorithm also decreases. Lastly, using Reverse Rank Selection to select trees that undergo mutation results into a significant increase in accuracy of the weak individuals.

Summarizing it, the proposed algorithm has been successful in attempting to produce an optimal classifier, that is, a classifier with high accuracy and smaller number of nodes.

7.2 Future Scope of Work

A number of future directions require investigation. An attempt can be made to parallelize the initial population generation so as to decrease the running time of the algorithm.

Few more genetic operators can be added to the algorithm. For example, adding merge and elimination operators proposed in [32] can increase the performance of the algorithm.

A variation to the algorithm GPeCT can be adaptive GPeCT, where the probabilities of crossover and mutation can be made adaptable to the performance of the operators.

Some work in the literature has been done to utilize the feature selection capability of GP, which can be incorporated in the algorithm.

Appendix A

List of Websites

http://en.wikipedia.org/wiki/Evolutionary-algorithm

http://www.cs.sandia.gov/opt/survey/ea.html

http://www.geneticprogramming.com

http://archive.ics.uci.edu/ml

http://www.geneticprogramming.us

Appendix B

List of Publications

1. R. Kotecha, V. Ukani, and S. Garg, "An empirical analysis of multiclass classification techniques in data mining," in Proceedings of 2nd Nirma University International Conference on Engineering(NUiCONE), December 2011.

2. R. Kotecha, S. Garg, and V. Ukani, "Genetic Programming based Evolution of Classification Tree", in communication with IEEE Transactions on Knowledge and Data Engineering (TKDE).

References

[1] P. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison-Wesley, 2005.

[2] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. San Mateo: The Morgan Kaufmann Series in Data Management Systems, 2 ed., 2006.

[3] W. R. Smart, *Genetic Programming for Multiclass Object Classification*. PhD thesis, School of Mathematical, Statistical and Computing Sciences, Victoria University of Wellington, 2005.

[4] T. Mitchell, *Machine Learning*. New York: McGraw-Hill, 2 ed., 1997.

[5] M. Vijaya, K. Jamuna, and S. Karpagavalli, "Password strength prediction using supervised machine learning techniques," in *International Conference on Advances in Computing, Control, and Telecommunication Technologies*, pp. 401–405, 2009.

[6] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers-a survey," *IEEE transactions on Systems, Man, and Cybernetics, Part C*, November 2005.

[7] R. Kotecha, V. Ukani, and S. Garg, "An empirical analysis of multiclass classification techniques in data mining," in *Nirma University International Conference on Engineering(NUiCONE)*, (Ahmedabad), December 2011.

[8] M. Aly, "Survey on multiclass classification methods," Citeseer, Nov 2005.

[9] I. Rish, "An empirical study of the naive bayes classifier," in *IJCAI Workshop on Empirical Methods in Artificial Intelligence*, (Italy), pp. 41–46, 2011.

[10] A. alias Balamurugan, R. Rajaram, S. Pramala, S. Rajalakshmi, C. Jeyendran, and J. D. S. Prakash, "Nb+: An improved naive bayesian algorithm," *Knowledge-Based Systems*, vol. 24, pp. 563–569, October 2011.

[11] T. Phyu, "Survey of classification techniques in data mining," *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 2009.

[12] N. Suguna and K. Thanushkodi, "An improved k-nearest neighbor classification using genetic algorithm," *International Journal of Computer Science Issues*, vol. 7, July 2010.

[13] P. G. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *IEEE transactions on systems, man, and cybernetics-part C: applications and Reviews*, vol. 40, pp. 121–144, March 2010.

[14] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[15] T. Loveard, *Genetic Programming for Classification Learning Problems*. PhD thesis, Royal Melbourne Institute of Technology, 2003.

[16] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. PhD thesis.

[17] R. Feldt and P. Nordin, "Using factorial experiments to evaluate the effect of genetic programming parameters," in *Proceedings of European Conference on Genetic Programming*, vol. 1802 of LNCS, pp. 271–282, 2000.

[18] H. Jabeen and A. R. Baig, "Review of classification using genetic programming," in *International Journal of Engineering Science and Technology*, vol. 2, pp. 94–103, 2010.

[19] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. San Mateo, CA/Heidelberg, (Germany): Morgan Kaufmann, 1998.

[20] S. Oka and Q. Zhao, "Design of decision trees through integration of c4.5 and gp," in *Proceedings of 4th Japan-Australia Joint Workshop Intelligent and Evolutionary Systems*, pp. 128–135, 2000.

[21] T. Enodu and Q. Zhao, "Generation of comprehensible decision trees through evolution of training data," *Proceedings of Congress on Evolutionary Computation*, vol. 2, pp. 1221–1225, 2002.

[22] M. H. Saraee and R. S. Sadjady, "Optimizing classification techniques using genetic programming approach," in *Proceedings of the 12th IEEE International Multitopic Conference*, (Advanced Databases, Data Mining and Bioinformatics Research Laboratory), pp. 345–348, December 2008.

[23] P. Kumar and Saroj, "Classification models: Non evolutionary vs evolutionary approach," in *International Conference on Advances in Computing, Control, and Telecommunication Technologies*, pp. 341–343, 2009.

[24] N. S. Chaudhari, A. Purohit, and A. Tiwari, "A multiclass classifier using genetic programming," in *10th International Conference on Control, Automation, Robotics and Vision*, (Hanoi, Vietnam), pp. 1884–1887, December 2008.

[25] H. Zhao, "A multi-objective genetic programming approach to developing pareto optimal decision trees," *Decision Support Systems*, vol. 43, pp. 809–826, April 2007.

[26] M. Riekert, K. M. Malan, and A. Engelbrect, "Adaptive genetic programming for dynamic classification problems," *Congress on Evolutionary Computation*, pp. 674–681, 2009.

[27] R. A. Davis, A. J. Charlton, S. Oehlschlager, and J. C. Wilson, "Novel feature selection method for genetic programming using metabolomic h nmr data," *Chemometrics Intelligent Laboratory Systems*, vol. 81, pp. 50–59, March 2006.

[28] V. Slavov and N. I. Nokolaev, "Fitness landscapes and inductive genetic programming," in *Proceedings of 3rd International Conference Artificial Neural Networks and Genetic Algorithms*, (Berlin, Germany), pp. 414–418, Springer-Verlag, 1997.

[29] S. Rouwhorst and A. Engelbrecht, "Searching the forest: Using decision trees as building blocks for evolutionary search in classification databases," *In Proceedings of the Congress on Evolutionary Computation*, pp. 633–638, 2000.

[30] M. C. J. Bot and W. B. Langdon, "Application of genetic programming to induction of linear classification trees," in *Proceedings of the Third European Conference on Genetic Programming*, vol. 1802, (Berlin, Germany), pp. 247–258, Springer-Verlag, April 2000. Lecture Notes in Computer Science Series.

[31] R. C. Barros, M. P. Basgalupp, D. C. C. P. L. F. Andre, and A. A. Freitas, "A survey of evolutionary algorithms for decision-tree induction," *IEEE Transactions on Systmes, Man, and CyberneticsPart C: Applications and Reviews*, 2011.

[32] C. S. Kuo, T. P. Hong, and C. L. Chen, "Applying genetic programming technique in classification trees," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 11, pp. 1165–1172, October 2007.

[33] J. Eggermont, "Data mining using genetic programming: Classification and symbolic regression," Master's thesis, Institute for Programming research and Algorithmics, 2002.

[34] K. Macropol, "Genetic programming and decision trees applied to medical data mining," Master's thesis, California State University Channel Islands, 2007.

[35] D. P. Muni, N. R. Pal, S. Member, and J. Das, "A novel approach to design classifiers using genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 183–196, April 2004.

[36] W. C. Lee, "Genetic programming decision tree for bankruptcy prediction," in *Proceedings of the Joint Conference on Information Sciences, JCIS 2006*, Atlantis Press, October 2006.

[37] A. Shali, M. R. Kangavari, and B. Bina, "Using genetic programming for the induction of oblique decision trees," in *6th International Conference on Machine Learning and Applications*, pp. 38–43, 2007.

[38] S. Silva and J. Almeida, "Gplab-a genetic programming toolbox for matlab," in *In Proceedings of the Nordic MATLAB Conference*, pp. 273–278, 2005.

[39] F. Esposito, D. Malerba, G. Semeraro, and J. Kay, "A comparative analysis of methods for pruning decision trees," *IEEE transactions on Pattern Analysis Analysis and Machine Intelligence*, vol. 19, pp. 476–491, May 1997.