The Application of Genetic Programming For Feature Construction in Classification

A thesis

submitted to the School of Computing Sciences

at the University of East Anglia

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

By

Mohammed Ahmed Yahya Muharram

July 2005

Abstract

This Thesis addresses the task of feature construction for classification. The quality of the data is one of the most important factors influencing the performance of any classification algorithm. The attributes defining the feature space of a given data set can often be inadequate, making it difficult to discover interesting knowledge. However, even when the original attributes are individually inadequate, it is often possible to combine such attributes in order to construct new ones with greater predictive power.

The goal of this Thesis is to restructure the feature space in order to improve the performance of decision tree classification techniques on complex, real world data. The proposed framework involves the use of genetic programming to evolve (construct) new attributes, which are non–linear combinations of the original attributes. This approach incorporates a number of decision tree splitting mechanisms in the fitness measures of the genetic program.

The empirical results obtained are encouraging and show that classification techniques can definitely benefit from the inclusion of an evolved attribute in terms of the accuracy and model size (for decision tree classifiers). When compared to existing approaches, the use of a decision tree splitting criteria as the fitness of the genetic program prove to be competitive and robust in terms predictive accuracy. Additionally, some of the evolved attributes manage to uncover physical properties in the data.

Acknowledgements

Firstly, I thank God for giving me the ability to complete this research and to learn a little more about one small aspect of His universe.

This thesis could not have been done without the help of a few individuals who deserve thanks and credit. First of all, I owe a great deal of thanks to my supervisor, Dr. George Smith, for his sharp insight, expertise and enthusiasm. Because of his various responsibilities, his time was often in high demand, but he provided me with all the time that I needed, and always returned my submitted work quickly and with valuable comments. His continuing encouragement and support has always inspired me and boosted my confidence in myself before my work. I would also like to extend my most sincere thanks to my second supervisor, Dr. Beatriz de la Iglesia, for her continuous support, with her kind and helpful attitude to my questions. Many thanks go to members of the Mathematical Algorithms Group, particularly those in the MAG-Lab for their friendship and advice. A great deal of thanks and appreciation also go to friends like Omar Almansoori, Samhan Arab, Hamdan Dammaj, Nasir Galal, Adel Mutlak, and many others for the long–years of friendship.

Of course, I am grateful to my parents without whom none of my university education would have come to existence. They have always been there for me and

supported me in every aspect of life. Their love, care and confidence have always motivated me and gave me hope through the good times and in those more difficult moments. Not to forget their financial support, even when things were tight, from the moment I started my higher education learning English through to the completion of my PhD. To my sisters and brothers who have also been very supportive and caring, "Thank you too!".

Most importantly, I owe my deepest gratitude to my wife, Shireen, who enriched my heart and enlightened the way when things seemed dark. Her contribution to this research has been her love, patience and willing sacrifice of time and effort in order to support me through difficult times, and this may never be fully assessed. Her moral support has made every single day a suitable environment for research. Last but not least, I owe a special thanks to my newborn son, Yazan, for the joy he has brought to my life.

Norwich, England Mohammed Muharram

November, 2004

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Overview

This Thesis concerns the application of evolutionary algorithms for constructive induction, and in particular the use of genetic programming to automatically construct features from data for supervised classification. This introductory chapter serves as a guide for the whole thesis, giving the reader a flavour of the background and motivation for the work, the Thesis contributions and outline, as well as a brief description of the experiments carried out.

1.1 Background to the Research

As data manipulation technologies rapidly advance, people rely increasingly on computers to accumulate data, process data, and make use of data. Knowledge discovery in databases consists of intelligent tools that handle massive data sets and find useful patterns that help people make use of the data [28]. Data Mining, a major process of knowledge discovery, is concerned with uncovering patterns hidden in the data. Data classification is a common data mining task that deals with methods for assigning a set of input objects to a set of decision classes. The input objects are usually described by a set of features (such as numbers or a string of symbols). Each feature

represents a characteristic of the given object. The classification task is accomplished by assigning the input objects to a class commonly described by a set of features [58]. Accurate classification is important for making effective decisions in many domains such as those in commercial, medical and academic organisations. Accuracy of the results produced by a classification system greatly depends on how well a problem is represented using a set of features. In most applications of classification systems, considerable effort is placed on finding an appropriate feature set that produces acceptable results.

A classification problem is not properly represented if it is defined by using redundant or irrelevant features, or if important interactions and relationships between the predicting features are not taken into consideration [38]. To overcome these obstacles, a tool must be developed to modify the initial feature set so that the problem is presented adequately to the classifier. This is a difficult task, because the space of all potential representations for a given problem is very large. Many attempts have been made to find appropriate representations for different problem domains. The amount of effort required to properly represent a problem is variable and depends on the problem's formulation. If the appropriate set of features is available and known, then the representation of a given problem becomes more understandable. For cases where there are small number of candidate features, an adequate representation or feature set can commonly be found through extensive trial and error. For many real–world problems however, we are faced with a situation in which there are many candidate features, there is little insight or knowledge as to which feature is more suitable, and it is not even clear that there is an appropriate set of features available. This Thesis will focus on the development of a system for modifying the initial feature set in order to provide a suitable set of features for a given problem.

Feature construction modifies the initial feature set by adding to it, one or more new features which are constructed using the original features [62]. The aim of feature construction is to find new features which can improve the performance of the inductive classifier. There are essentially two approaches to feature construction. The first is to rely on the classifier itself to construct features while performing the classification, a hybrid approach. The second is to use some technique that constructs features separately from the inductive classifier, referred to as constructive induction. Constructive induction is a method for improving the representation of an inductive problem by constructing new features (by combining existing features in various ways). It has proven effective in improving the classification accuracy in hard domains (domains for which standard inductive classifiers perform poorly) [67]. When performing constructive induction, the system faces a combinatorial explosion of potential features that it could construct, but only a few of these is useful.

The work carried out in this Thesis uses evolutionary algorithms, namely genetic programming, for performing constructive induction for classification, particularly decision tree induction. The aim is to evolve new features so that when they are added to the existing set of features, decision tree inductive techniques produce more accurate, concise and understandable models. Other techniques, such as an MLP neural network are also used in the experimental work to provide a comparison.

1.2 Key Contributions of the Thesis

This Thesis deals with the pre–processing task of feature construction for improving the representation of the feature space for classification. The main contributions of this Thesis include:

- A model has been developed using an evolutionary algorithm, namely genetic programming (GP), for restructuring the feature space by evolving new more predictive features for a number of classification techniques, particularly, decision tree classification algorithms.

- As univariate decision tree algorithms, such as C5, CART and CHAID, induce trees by splitting data using tests on a single attribute at a time, important relationships and interactions between the features are not taken into account. The proposed GP system constructs new features which are non–linear combinations of all or some of the original features in the hope of finding interesting relationships between these features. The GP system constructs a single feature which is then added to the original feature set to form an augmented set (original features + an evolved feature).

- **The GP incorporates the splitting mechanism of a decision tree classifier as its fitness for constructing new features**.

- The application of GP for feature construction has improved the predictive capabilities of decision tree classification techniques through its successful application to problems varying from artificial data through to complex, noisy real–world data.

- Analysis of the results showed that the accuracy of the decision tree models improved using the augmented attribute set, sometimes dramatically.

- It also improved the performance of a non–decision tree classifier, an MLP artificial neural network, on the same data.

- Bias analysis was performed to ascertain whether decision tree classifiers have more advantage from enclosing an attribute, constructed using a GP whose

fitness measure incorporates the splitting criterion of the associated tree.

- A comparative study with existing approaches was performed to address the strengths and weaknesses of the proposed model.
- Further analysis on the tree size showed that all classifiers produce smaller trees using the augmented feature set.
- A classification GP system was developed to compare the performance of a GP for classification with the GP developed for constructive induction.
- Further analysis of the constructed features revealed that, in some cases, the features uncovered important physical characteristics about the data.
- The experimental work was extended to include the application of GP for feature construction on a confidential commercial data set. [Note: this is provided as a separate attachment due to its confidential nature.]

1.3 Outline of the Thesis

This Thesis is divided into nine chapters, including this chapter. It includes three literature review chapters and four chapters containing details and analysis of the experimental work. The last chapter concludes the Thesis.

A review of the current literature relevant to this Thesis starts in Chapter 2. It begins by presenting an overview of knowledge discovery in databases, introducing the main themes of the process. We outline the various phases involved and we present two methodologies. Some related areas are also highlighted. The pre-processing task is also addressed and some related work is mentioned. The last part of the chapter

outlines some of the well–known techniques for data classification.

Chapter 3 introduces decision tree induction for classification. The classification methods used in the experimental work of this Thesis, namely, C5, CART and CHAID, are described in greater detail.

Chapter 4 serves as an introduction to evolutionary computation. First, the topics of search and optimisation are introduced. Next, each of the four paradigms of evolutionary computation are outlined, with more focus on genetic programming, since this approach forms the basis for the constructive induction model. We also address previous work in the applications of evolutionary algorithms, particularly in data pre–processing and data classification.

Chapter 5 outlines the experimental setup and methodology. We present the data sets used in the experiments and the sampling method. The GP system used for constructing the features is described along with parameter settings and the fitness functions used. Then, we present the experimental methodology and classification techniques used in the experiments. Finally, we show some examples of the evolved features and their goodness in terms of the splitting criteria.

The error rates of classification are presented in Chapter 6. We compare the error rates of classification using the original and the augmented (original + evolved) attribute sets. The results are validated using the statistical measure (T-Test). We then analyse the results to check for bias, i.e. check whether the performance of any of the decision tree models is biased towards the attribute set which has an attribute constructed using a GP whose fitness function incorporates the splitting criteria of the associated tree. A comparative study is performed to examine the effectiveness of

the current work with other linear and evolutionary approaches to feature construction.

Further analysis is performed in Chapter 7. We first analyse the size of the resultant trees of the decision tree classification models using the original and augmented attribute sets. Then we look, in particular, at the evolved attributes of two data sets, where some classifiers achieve 100% accuracy, to see if they could describe physical properties of the data. Also in this chapter, we show the results of an experimental study on the use of GP for classification and compare it to the results shown in Chapter 6.

In Chapter 8, we extend previous work on a confidential, commercial, real-world data by performing constructive induction to improve the performance of classification. Other pre–processing approaches are also described in this chapter.

Chapter 9 concludes the Thesis outlining the overall summary of research and performed work, limitations of the current system and suggestions for future work.

Chapter 2

Data Mining and Knowledge Discovery in Databases

2.1 Introduction

Over the past few decades, rapid advancements in computer technologies have allowed organisations to gather vast quantities of data for various purposes. Database technology provides efficient and sophisticated access to a myriad of information and allows the creation and maintenance of massive databases [29]. However, it is far easier to collect data than to analyse it [2] and extract information from it. Traditionally, data was analysed manually but as it grows in size, many hidden patterns and potentially useful relationships may not be recognised by the analyst. This resulted in the growing interest in the field of *Knowledge Discovery in Databases* (KDD), and a particular process of it, *data mining*. Data mining applies data analysis and discovery algorithms to identify patterns in data.

This chapter provides an overview of the multi–disciplinary field of KDD. In Section 2.2, we show KDD as a sequence of various stages, describing two methodologies, namely the CRISP-DM and the KDD-Roadmap. The areas of most relevance to the

Thesis, data pre–processing and data classification techniques, are outlined in Sections 2.3 and 2.4, respectively.

2.2 Knowledge Discovery in Databases (KDD)

In recent years, the advances in computer technologies and the accompanying decrease in their cost has expanded the means available to collect and store data. As an immediate consequence, the amount of information stored has been increasing at a very fast pace.

Conventional data analysis techniques are useful to create informative reports from data (statistics), or to confirm predefined hypotheses about the data (on–line analytical processing). However, the huge volumes of data being collected create new challenges for such techniques as organisations look for ways to make use of the stored data . In business, people would like to gain an edge over competitors and, therefore, it is reasonable to believe that data collected over an extended period contains hidden knowledge about the business or patterns characterising customer behaviour. For example, the manager of a large supermarket would like to have answers to questions such as who will buy, what items are bought together and in what quantities. To answer such questions the data analyst needs new tools and techniques to explore the data in the search for answers to questions which were not considered when the data was collected. The answers to such questions are, in general, not explicitly available in the data.

Knowledge discovery is a research field that focuses on the development of tools which search for hidden patterns in large collections of data. In [28], KDD is defined as the "*non-trivial process of identifying valid, novel, potentially useful, and ultimately*

understandable patterns in data".

KDD has emerged from the intersection of research in such fields as databases, machine learning, statistics and artificial intelligence.

The process of KDD is usually defined as a sequence of specific steps. Its definition can vary slightly from one author to another [28, 15, 96], however, there is a general consensus on the main elements of this process. We define the process of KDD as a sequence consisting of the following three phases; *pre-processing*, *mining*, and *post-processing*.

2.2.1 The Pre-processing Phase

The pre-processing phase is concerned with defining the problem at hand and is usually performed before hardly any learning is carried out. The general purpose of pre-processing data is to reshape and transform the data in order to make it ready for the mining phase. This phase is considered the most time consuming as it can take up to 80% of the whole KDD process time [42]. Major processes within this phase include:

- **Domain Understanding**

  This involves the formation of the picture of what exists in the domain and gathering relevant facts about the domain. Here the problem space and the solution space are explored using reports and visualisation tools to identify what the outcome of the KDD process should look like.

- **Data Preparation, Transformation and Cleansing**

  This involves converting data into a shape suitable for mining. It includes integrating data from different resources and transforming it into a form suitable for mining. Data cleansing addresses problems arising from noise and inconsistent

data. It is also deals with outliers and missing values in the data.

- **Feature Space Transformation**

  This involves modifying the feature space of the data to enhance quality of the data and hence the performance of the mining algorithms. At this point, some learning about the features in the data may be performed. The aim is to focus on relevant features by reducing the feature space and eliminating irrelevant features and/or by introducing new features. This is usually achieved using three common feature space transformation techniques, namely, *feature selection*, *feature extraction* and *feature construction* [52, 67]. Feature selection reduces the set of features by choosing the ones that most contribute to the decision at hand. Feature extraction replaces the original set of features with a new, potentially better, feature set (usually smaller than the original feature set), constructed using the original feature set. Feature selection and feature extraction are referred to as data reduction techniques, as they reduce the dimensionality of the feature space. Feature construction, on the other hand, creates one or more new, potentially better, features from the existing set of features, which are added to the original ones. Feature selection, extraction and construction are explained in greater detail in Section 2.3.

2.2.2 The Mining Phase

The data mining phase is a process concerned with the application of algorithms to uncover patterns, associations, anomalies and statistically significant structures in data [29]. It typically refers to the case where the data is too large and/or too complex to allow either a manual analysis or analysis by means of simple queries. The data mining task is categorised by many researchers in the KDD community into *descriptive* and *predictive* data mining, depending on the desired outcome of this task

[96].

- **Descriptive Data Mining**

  The main purpose of descriptive data mining is to describe and present general interesting properties of the data. It characterises the data based on regularities found in its examples. The most popular descriptive data mining techniques include *cluster analysis*, *summarisation* and *visualisation*, all of which are described in the following subsections:

  Cluster Analysis Cluster analysis is the task of dividing objects within a data set into a number of groups or clusters of objects, where the cluster is a collection of "similar" data objects [27, 48]. Similarity is generally expressed by some metric such as distance measures. The term clustering is used to describe the methods for grouping unlabelled data. Clustering is considered as a form of unsupervised learning since the data objects have no predefined class labels. For instance, given a data set of customers, clustering be used to identify subgroups of customers with a similar buying behaviour so they can be targeted for a new sales campaign, i.e. customer segmentation.

  Summarisation and Visualisation Summarisation involves finding a compact description for a subset of the data. This includes summarising the statistical attributes of the data, such as means and standard deviations. Data visualisation aids the data miners to look for potentially meaningful links among features. Visualisation techniques can range from simple scatter and histogram plots to 3D movies.

- **Predictive Data Mining**

  Predictive data mining is a search for patterns in data that can generalise to accurate future decisions [96]. It allows the user to submit records with some unknown field values, and the system will predict the unknown output values based on previous patterns discovered from the data. Major tasks of predictive data mining include *classification* and *regression*, presented below:

  Classification Data classification is one of the major data mining tasks, involving the prediction of one attribute, a discretised class, based on a set of predicting attributes [96]. For example, given a data set of patients, to predict whether a patient could develop a certain disease or could better respond to a certain drug, the predictor attributes should contain relevant medical information.

  Regression Regression is a statistical technique used for prediction. It is analogous to classification with the difference being that regression predicts a numerical attribute from numerical data [44]. For instance, predicting the relative performance of computer processing power given a set of computer configurations [99].

There are many other tasks that can be used for both description and prediction. They include tasks for exploring data to find certain regularities and interesting characteristics, including a search for similar sequences or subsequences, and mining sequential patterns, periodicities, trends and deviations. The two most common approaches include *association rules discovery* [2, 3] and *time–series analysis* [30]. Association rules discovery finds rules about items that appear together in an event such as a purchase transaction. Market-basket analysis is a well–known example of association discovery. Time–series analysis is very similar, in that a sequence is an

association related over time. For example, one may predict the trend of the stock values for a company based on its stock history, business situation, competitors' performance, and current market.

2.2.3 The Post-processing Phase

The post-processing phase requires deeper involvement from domain experts and is concerned with the steps taken after the data is pre-processed and mined. Processes within this phase include:

- **Results Evaluation**

  This involves evaluating the discovered patterns where "interestingness" measures are used to identify patterns that represent knowledge. Due to the nature of data mining, i.e. the search for patterns in data, it is likely that patterns will be found. However, these patterns may simply be due to some variation in the data rather than representing some real world phenomenon [28]. A common method used to validate results is to use separate training and testing sets of data. The training data is used to discover patterns then the patterns are tested on the unseen test data. The performance of the patterns on the test data gives an indication of the strength of the patterns on other unseen data. Another validation method that is often used when limited volumes of data are available, is *cross validation* [97]. Other aspects of evaluation include checking that patterns are understandable and that they are novel and potentially useful, in accordance with the definition of the KDD process [30]. The complexity of patterns and their correspondence with known facts contribute to how easy these are to understand. Domain experts will often be able to determine if patterns are novel and potentially useful.

- **Knowledge Interpretation**

  Interpretation of discovered knowledge would normally be carried out by domain experts. As in the evaluation stage, it would be expected that new knowledge would tend to fit with the existing knowledge and would be explainable by the domain experts. Obviously, some difference between known and discovered knowledge is required for patterns to be novel. However, if the differences are very great it may be because errors have been made in the KDD process.

- **Knowledge Deployment**

  The final stage of the process is the exploitation of the discovered knowledge. It may be possible to exploit the results without further refinement or it may be that the results can be taken to represent a hypothesis that requires further validation. For example, in the medical domain, any results that might effect clinical practice should be subject to rigorous testing in order to avoid compromising a doctor's defence in a medical negligence case [102, 84]. Exploitation of results may include modification of procedures or the production or modification of software to embed the discovered knowledge. For example, the software used to assess risk in insurance companies may be updated as a result of a KDD exercise.

The KDD process is interactive and iterative which means that at any stage, if the results are not satisfactory then this may suggest the necessity for better data cleansing or for different settings of the data mining algorithm parameters.

There are many methodologies that provide a mainstream for the KDD lifecycle [1]. Most of the available tools have their own models. In most cases, such models are tightly bound to the respective tools and platforms. Efforts towards standardisation of the KDD process models are scattered in different industries and research institutions.

In the following Subsections, we briefly address two business–orientated approaches to KDD, namely the CRISP-DM [21] model and the KDD-Roadmap [24, 23].

2.2.4 The CRISP-DM Model

CRoss-Industry Standard Process for Data Mining (CRISP-DM), first conceived in 1996, is a project developing a domain–independent KDD model [98, 21]. The CRISP-DM model is a consortium formed by leading DM companies. It provides a methodology that is supposed to guide data analysts throughout a DM lifecycle, see Figure 1 (adapted from [98]).

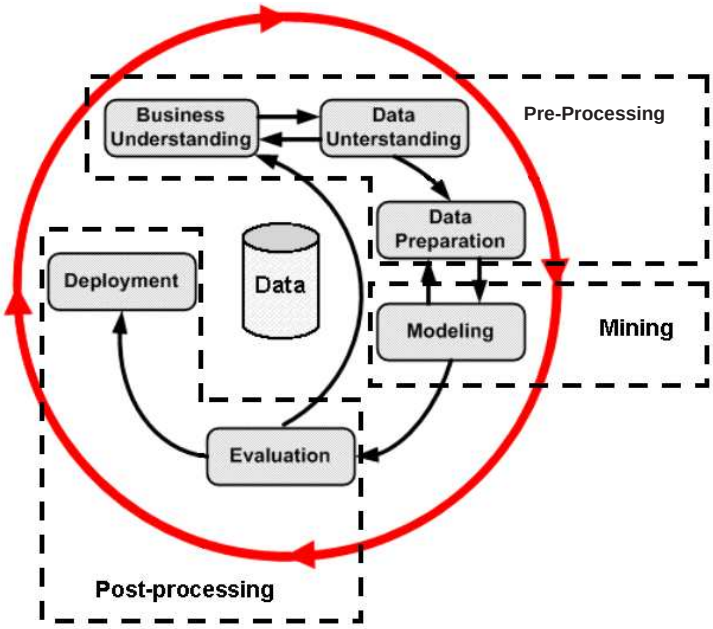


Figure 1: The CRISP-DM Model.

In the following, we briefly outline the phases within the CRISP-DM model:

1. **Business (or Problem) Understanding** This phase focuses on understanding

the requirements and objectives of the project from a business perspective. It is also concerned with developing an initial technical problem definition and a project plan.

2. **Data Understanding** This phase starts with an initial analysis of the data by identifying data quality problems and discovering first insights into the data.

3. **Data Preparation** This phase is concerned with making the data ready for the modelling (mining) phase. Subtasks include sampling of the data and and other pre–processing tasks including, transformation and cleansing of data.

4. **Modeling** In this phase the mining techniques are applied on the data. Revisiting previous stages is often needed.

5. **Evaluation** At this phase the data mining model is thoroughly evaluated, and the steps executed to construct the model are reviewed, to ensure no important business issues are missed and that the desired business objectives are met. At the end of this phase, a decision is made as to whether or not the outcome of the whole process can be deployed.

6. **Deployment** The deployment phase can be as simple as generating a report or as complex as implementing a repeatable data mining process. In many cases it will be the customer, not the data analyst, who will carry out the deployment steps.

2.2.5 The KDD Roadmap

Here the KDD process is viewed as a roadmap [24, 23]. In this roadmap, shown in Figure 2 (adapted from [24]), the KDD process contains eight sub-phases, each of which consists of a number of processes. Inspired by the software engineering life-cycle, the

map contains one and two way roads, and locations representing the phases. After a phase is completed, a valid route is taken to the next one. The processes within the KDD Roadmap include:

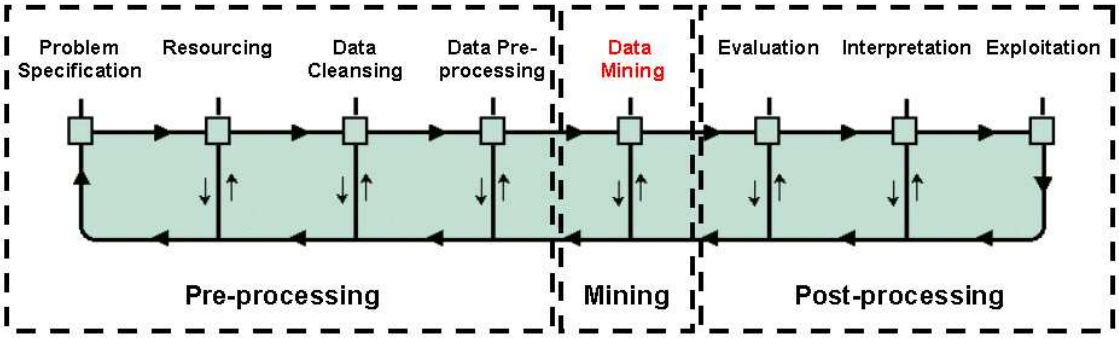


Figure 2: The KDD Road Map.

1. **Problem Specification** The purpose of this phase is to develop a tightly defined problem definition from a loosely defined data mining project idea. It is necessary to undertake some preliminary database examination in order to determine factors such as the size of the database and the proportion of missing data. At this stage the data itself may not be available so a description of the data might be used. The resources necessary to carry out the project should be determined and the feasibility of the project confirmed.

2. **Resourcing** This phase comprises the collection of the resources required to carry out the project. This will usually include hardware and data mining software. Often the most time consuming resource to collect will be the data itself. The data may not be readily available. For example, it may come from different sources and require combining and formatting.

3. **Data Cleansing** Data cleansing is concerned with ensuring that the data is correct, i.e. that errors in the data are removed. The problem of handling missing values and outliers might also be addressed at this stage although they could also be dealt with in the pre-processing or data mining stages. The distinction between data cleansing and pre–processing is that no learning or knowledge discovery takes place when data cleansing. Data cleansing is usually carried out only once for each database, whereas pre–processing may be repeated as the process continues.

4. **Data Pre–processing** Pre–processing is undertaken to prepare the data so that it is ready for input into the data mining phase. This includes preparing and transforming the data to improve its quality and hence the performance of the data mining algorithms, such as predictive accuracy and reducing the learning time. This can be achieved by reducing the amount of data, focussing only on the relevant data. Major processes involved within this phase include feature selection, construction and extraction, which are discussed in greater detail in Section 2.3.

5. **The Mining Process** This phase includes searching for patterns in data, typically with the aid of powerful algorithms to automate part of the search. These methods come from disciplines such as statistics, machine learning and pattern recognition.

6. **Results Evaluation** This phase is used to assess the success of the overall process. There are several approaches to evaluate the validity of the discovered patterns; the performance on the test set, the simplicity, suitability, and so on.

7. **Interpretation** This phase is concerned with the understandability of the discovered knowledge. It involves employing visualisation and knowledge presentation

techniques to present the knowledge to the user.

8. **Exploitation** Putting the results into use is the final step of the post–processing phase. Based on the outcome of this stage, the user may incorporate the knowledge into the performance system, taking actions based on knowledge or review the specification and implementation of the problem set in the pre–processing phase.

2.3 Data Pre-processing

As mentioned in Subsection 2.2.1, pre–processing data for a data mining task usually consumes the bulk of the effort invested in the entire KDD process [42], see Figure 3. The figure shows that about 80% of the KDD time is spent on data acquisition and pre–processing. This is due to the fact that real world data is often large, noisy, messy, and contain missing or irrelevant values which make it of a disappointingly low quality [44]. Therefore, the gap between the format of source or raw data and that required by data mining algorithms must be bridged before data is presented to the data miner.

Since data is rarely collected for the purposes of mining, many databases contain features that have little or no relation with the class attribute. Hence, data pre–processing is mainly concerned with the cleansing, integration, transformation, and reduction of data. For the purpose of this research, we address the problem of finding appropriate data representation for the data mining task of classification. Restructuring the feature space of the problem is very significant and has resulted in vigourous research by the machine learning and KDD communities. Researchers have developed several techniques and methods to deal with this problem. The most common methods are feature selection (FS), feature extraction (FE), and feature construction

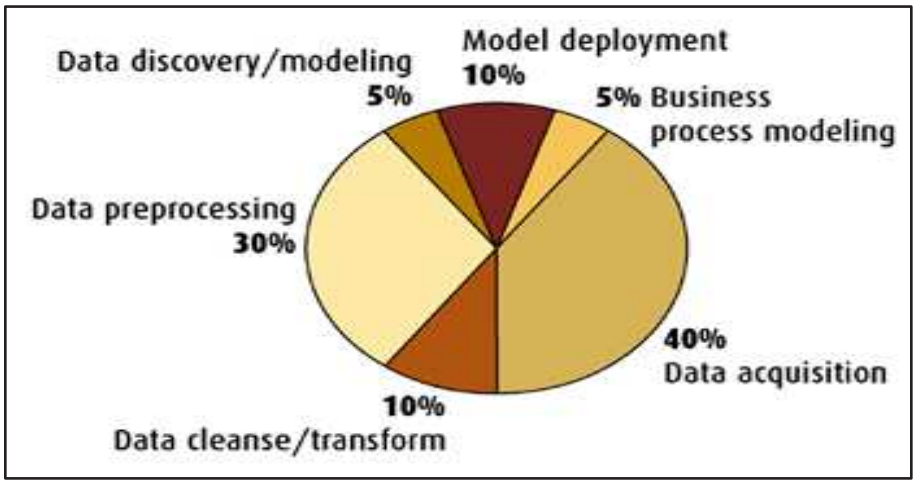


Figure 3: KDD Effort Distribution [42].

(FC) [63].

In the following Subsection, we address the problem of focusing on relevant features and how they affect data quality. Then, we elaborate on the pre–processing methods of feature selection, extraction and construction, reporting on some previous work done in each of these areas.

2.3.1 Relevant Features and Data Quality

In a typical supervised machine learning task, data is represented as a table of examples or instances. Each instance is described by a fixed number of measurements, or *features*, along with a label that denotes its class. Features, also known as *attributes*, are typically one of two types: *categorical* or *numerical* [80].

The features within a data set provide the information that make classification feasible. One can assume that expanding the feature space can always improve class

description. In reality, however, adding more features can actually lead to a degradation in the classifier performance [11]. In many real-world problems, there are often far too many features for the classifier, most of which are irrelevant or redundant. Choosing relevant features, and eliminating irrelevant ones, is a crucial problem in machine learning. Before a classifier can move beyond the training data to make predictions about novel test cases, it must decide which features to use in these predictions and which to ignore. Intuitively, one would like the classifier to find subsets of the feature population that are *relevant* to the target class and worthy of focused analysis [12].

There are many learning methods that can select or extract features. However, in general, they do not cope well with high-dimensional domains. *Data reduction* is one of the methods proposed to tackle the problem of high-dimensional data. There are three types of data reduction: *feature reduction*, *case reduction* and *feature smoothing* (reducing the number of values in a feature) [96]. Reducing high-dimensional feature spaces is addressed in the next two Subsections. The main objective of feature reduction is to improve the quality of the data by using fewer features and focusing on the relevant data [63].

2.3.2 Feature Selection

Since each additional feature used as part of the classification procedure can increase the cost and running time of a classification system as well as reduce the accuracy of the result, there is a strong demand for developing methods to select smaller feature subsets. At the same time there is a potentially opposing need to include a sufficient set of features to achieve high accuracy rates and provide better understanding of

results. This has led to the development of a variety of techniques within the machine learning community for finding an optimal subset of features from the superset of original features [99].

Kira et al. [52] define feature selection as "*the problem of choosing a small subset of features that ideally is necessary and sufficient to describe the target concept*". It is concerned with identifying and removing as many of the irrelevant and redundant features as possible. Given a data set with original features A, after applying feature selection, we retain a smaller subset B of the original features. The new subset should provide more or similar learning power to the classifier.

There are two main approaches to feature selection: the *filter* approach and the *wrapper* approach [62]. The sole difference between these two approaches is that the filter approach selects features independent from their effect on the performance of the induction algorithms. It aims at selecting a subset of the features that preserves as much as possible the relevant information found in the entire set of features [53, 39].

There are few methods that implement the filter approach to feature selection. The *FOCUS* algorithm [4] was designed for noise-free Boolean domains and it follows the *MIN-FEATURES* bias. It examines all feature subsets and selects the minimal subset of features that is sufficient to predict the class targets for all records in the training set. *Relief* is an instance–based feature selection method introduced by Kira et al. [52] and later enhanced by Kononenko [54]. Relief was originally developed for two-class problems and was later extended (*Relief–F*) to handle noise and multi-class data sets. Relief works by randomly sampling an instance from the data and then locating its nearest neighbour from the same and opposite class. The values of the features of the nearest neighbours are compared to the sampled instance and used

to update relevance scores for each feature. This process is repeated for a number of instances. The idea is that a useful attribute should differentiate between instances from different classes and have the same value for instances from the same class. One major disadvantage of the filter approach is the non–involvement of the classification algorithm in selecting the feature subset [39].

In contrast, in the wrapper approach, the selection is performed using the classification algorithm as part of the evaluation function [53]. *Information gain* and *gain ratio* [80] are good examples of measuring the relevance of features for decision tree induction. They use the *entropy* measure to rank the features based on the information gained; the higher the gain the better the feature. Moore et al. [70] proposed another model using an instance-based algorithm, called *RACE*, as the induction engine, and *leave-one-out cross–validation* (LOOCV) as the subset evaluation function. Searching for feature subsets is done using backward and forward *hill-climbing* techniques. John et al. [49] proposed a similar method and applied it to ID3 and C4.5 on real world domains. Langley et al. [60] also used LOOCV in a *nearest-neighbour* algorithm. Caruana et al. [20] test the forward and backward stepwise methods on the *Calendar Apprentice* domain, using the wrapper model and a variant of ID3 as the induction engine. Wrapper models are usually slower than filter models in the sense that inductive learning is carried out more than once. Overfitting is another drawback of the wrapper approach.

2.3.3 Feature Extraction

Feature extraction is another approach for minimising the dimensionality of the feature space. The assumption here is that there exists a transform function T that

converts the original feature set $F = \{f_1, f_2, .., f_n\}$ of n features into another set $AF = \{af_1, af_2, .., af_m\}$ of m features which has lower dimensionality ($m < n$) and is more conducive to classification, where $af_i = T(f_1, f_2, .., f_n)$. Feature extraction replaces the set of original features by searching for a minimum set of new features that satisfy some performance criteria [63].

One of the most fundamental feature extraction techniques is *principal component analysis* (PCA) [50]. First introduced by Pearson [77] and independently by Hotelling [47], the PCA calculates the covariance matrix of the data set and finds its corresponding eigenvalues and eigenvectors. Each of the eigenvectors (α_i) is a transform that acts on the original feature set to create a new feature; they are called *principal components*. The eigenvalue (λ_i) corresponding to each eigenvector denotes the variance of the data over each principal component. The number of significant eigenvalues is taken as a measure of intrinsic dimensionality. The underlying assumption here is that principal components with the largest variances make the best features. Thus, the PCA utilises the eigenvalues as the basis for selecting the best m features. The chosen eigenvectors are used to transform the original data set to a new one.

A more sophisticated feature extraction technique known as *Decision Boundary Feature Analysis* (DBFA), was developed by Lee and Landgrebe [61]. This approach winnows the data set using a k–Nearest Neighbor (k–NN) algorithm to remove data points that obscure the true class boundaries. With the classes more clearly separated, the algorithm proceeds to identify all potential decision boundaries. A vector normal to each decision boundary is then computed. Each of these vectors represent the direction of optimal class separation for its respective decision boundary. The covariance matrix describing this set of vectors is termed the *Effective Decision Boundary Feature Matrix* (EDBFM). The EDBFM is the covariance matrix of unit

vectors normal to the class decision boundaries. EDBFM is used in a manner analogous to the covariance matrix in the PCA to derive a new feature set. The key difference is that the eigenvalues of the EDBFM are far more accurate indicators of features that increase class separation.

Another approach to extracting features is a *feed–forward* Neural Network, presented in [85]. A *multi–layer perceptron* with one single hidden layer is adopted for this task where, the basic idea is to use the hidden units as newly extracted features. The extracted features are evaluated using a performance measure based on their predictive accuracy. The extracted features that result in the best predictive accuracy are chosen. These features are the non–linear transformation from input units to hidden units. The algorithm is designed to construct a network with the minimum number of hidden units (i.e. minimum number of new features) and the minimum number of connections between the input and hidden layers: the network construction algorithm parsimoniously adds one more hidden unit to improve predictive accuracy; and the network pruning algorithm generously removes redundant connections between the input and hidden layers if predictive accuracy does not deteriorate.

2.3.4 Feature Construction

Much research has focused on restructuring the original feature space representation by reducing its dimensionality. Most feature selection techniques focus on the relationship between the individual predicting attributes and the class attribute. Any *combination* of predicting attributes that presents a much stronger prediction may therefore be missed, if the operators available to the induction process are insufficient to identify that combination.

One approach to overcome this problem is to allow the induction process the flexibility to identify and construct these powerfully predictive combinations. For instance, in the work of [94], where the authors use a feed–forward neural network to extract knowledge from corporate accounting reports, it is interesting to note that the first hidden layer of nodes were inclined to construct ratios from some of the raw accounting data. It is widely recognised that accounting ratios, rather than the basic accounting data, are more useful in terms of what can be deduced about a company's financial status. Turning to decision trees, *OC1* is an oblique tree induction technique designed for the use with continuous real-valued attributes. During the induction stage, OC1 considers linear combinations of attributes, and partitions the data set into both oblique and axis–parallel hyperplanes [74].

Another approach is to construct new attributes (feature construction) which are combinations of the original attributes, the objective of the construction technique being to identify highly predictive combinations of the original attribute set and hence, by including such combinations as new features (attributes), to improve the predictive power of the attribute vector. Feature construction is an approach to augmenting the feature space by creating additional features. The aim is to discover the relationships between the original features in the hope of increasing the expressive (prediction) power of the original feature set [63]. Given a set $F = \{f_1, f_2, .., f_n\}$ of n features, after constructing ($m \geq 1$) new features, the augmented feature set is $AF = \{f_1, f_2, .., f_n, f_{n+1}, f_{n+2}, ..f_{n+m}\}$. For example, given the two features *height* and *weight* it might be advantageous to construct a feature *body mass index* (BMI), which is expressed by $weight \div height^2$.

Since manual feature construction is difficult, the question of how to automate this task is addressed by many practitioners. This has resulted in the concept of

constructive induction. Constructive induction is concerned with the automated construction of new features to facilitate concept learning [67]. There are essentially two approaches to constructive induction in relation to data classification; one method is as a separate pre–processing stage, in which the new attributes are constructed before any induction process, i.e. before the classification algorithm is applied to build the model. The second approach is an integration of construction and induction, in which new attributes are constructed within the induction process. The latter is therefore a *hybrid* induction algorithm.

Matheus et al. [67] proposed the CITRE system, a framework for constructive induction. CITRE uses background knowledge in two different ways: domain–knowledge constraints, to eliminate less desirable new attributes beforehand and domain–dependant transformations to generalise newly constructed attributes in ways meaningful to the current problem. The work by Wnek et al. [100] categorised constructive induction into four approaches: *data–driven*, *hypothesis–driven*, *knowledge–based* and *multi–strategy* constructive induction (MCI). The data–driven approach constructs new features based on analysis of the available data by applying various operators. The hypothesis–driven approach constructs new features based on the hypotheses generated previously. Useful concepts in the induced hypotheses (e.g. rules) can be extracted and used to define new features. The knowledge–based approach constructs new features applying existing knowledge and domain knowledge which is particularly helpful in determining the type of new compound features and choosing suitable operators. The MCI approach integrates the data–driven and the hypothesis–driven approaches with empirical induction and deduction.

Constructive induction systems usually consist of two components: one for constructing new attributes, and the other for generating concepts. After the construction, new attributes are treated exactly in the same way as the primitive (original) ones. For a non-trivial learning problem the number of possible constructive operators, such as logical operators and mathematical operators, and the number of possible operands, are usually very large, so it is not feasible to search through all the possible combinations.

Usually, systems select as a bias, a goal set of possible operators and reduce the search space of new attributes. *Conjunction*, *disjunction*, and *negation* are commonly used constructive operators, whereby they produce binary attributes. Another type of constructive operator is $M\text{-}of\text{-}N$ including the variants *at-least* $M\text{-}of\text{-}N$, *at-most* $M\text{-}of\text{-}N$, and *exactly* $M\text{-}of\text{-}N$ [105]. A $M\text{-}of\text{-}N$ operator generates boolean attributes. It consists of a value M and a set of N conditions based on existing attributes. An *at-least* $M\text{–}of\text{–}N$ attribute is true if at least M of the N conditions are true. Zheng [105] proposes the $X\text{-}of\text{-}N$ constructor that returns the number of true conditions. It generates ordered discrete values. One of the well–known constructive induction systems is the *FRINGE* family of algorithms [76]. All algorithms from this family iteratively build a decision tree based on the existing attributes (initially only primitive attributes), and then construct new attributes by using conjunctions and/or disjunctions of two conditions from the tree paths. The new attributes are added to the set of existing attributes, and the process is repeated. The conditions used for generating new attributes are chosen from fixed positions in the paths, either near the root and/or near the fringe of a tree.

2.4 Mining techniques for the Classification Task

There are many techniques used for data classification. We describe only three, namely *k–nearest neighbour*, *artificial neural networks* and *decision trees*.

2.4.1 k-Nearest Neighbour (k–NN)

k-nearest neighbour is a predictive technique suitable for classification models. k represents a number of similar cases or the number of items in a group [69]. With the k-NN technique, the model is created using the training data. When a new case or instance is presented to the model, the algorithm looks at all the data to find a subset of cases that are most similar to it and uses them to predict the outcome. k–NN is based on a concept of distance, and uses some metric to measure this similarity. For continuous attributes, Euclidean distance can be used, while for categorical variables, one has to find a suitable way for calculating the distance between attributes in the data. Choosing a suitable metric is a very delicate task, since different metrics used on the same training data can result in completely different predictions. This means that a domain expert is needed to help determine a good metric.

2.4.2 Artificial Neural Networks (ANNs)

Artificial Neural Networks are computational algorithms based on the virtues of the biological brain rather than on the true physical details [11]. Interconnections of synthetic neurons are used to learn from observed data in a manner that is amenable to parallelism, robust to noise and fault tolerant. ANNs are among the most complicated of the classification algorithms. They are often considered as a black box. A neural network requires a lot of data for training, thus consuming time, but once trained, it can make predictions for new cases very quickly, even in real time. Moreover, neural

networks can provide multiple outputs representing multiple simultaneous predictions.

Neural networks are defined by their architecture. A typical ANN consists of three groups of layers: an input layer, one or more hidden layers, and an output layer. A *feed–forward* neural network is perhaps the most commonly used technique for classification. It consists of several layers of neurons with the output of each neuron in layer k connected to the input of each neuron in layer $k + 1$. In other words, data is presented at the input layer and the signals propagate through the layers of the network to produce a result at the output layer, see Figure 4. A typical feed–forward neural network is the *multi–layer perceptron* (MLP). In an MLP network, data is fed into the input layer and transformed by weights and transformation functions at each neuron as it flows through the network. The network output is a resultant transformation that forms the relationship between inputs (predicting fetures) and the output (class feature) [17]. MLPs are trained to find relationships by presenting the network with historical values of inputs and outputs. Training is the search for a set of weights which best matches the inputs onto the output for the samples in the data set. Training the network is thus an optimisation problem where the optimal solution lies somewhere in weight space.

2.4.3 Decision Trees

Decision tree learning is one of the most widely used and practical methods for inductive learning. It is a method for predicting discrete-valued classes, using a set of predicting attributes. Decision trees classify data by successively splitting the training set into partitions based on splitting measures. Decision trees perform many tests on the predicting attributes, starting at the root node, and then try to arrive at the

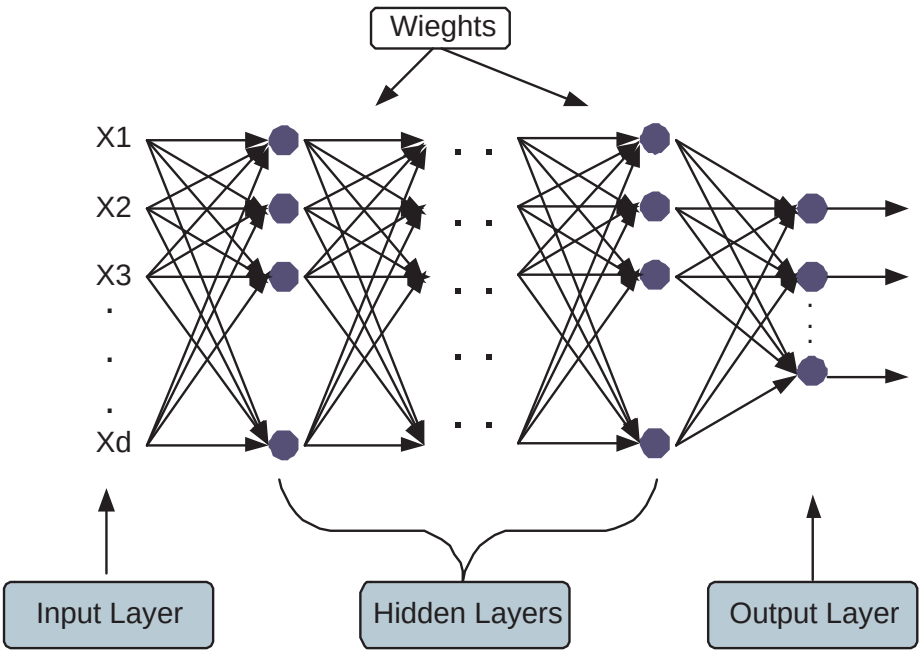


Figure 4: A multi-layer perceptron artificial neural network.

best sequence for predicting the class. Each test creates branches that lead to more tests, until testing terminates in a leaf node. The path from the root to the leaf nodes is the rule that predicts the class of the data samples belonging these leaf nodes. The rules are expressed as {*if antecedents then consequent*} form.

The process of creating the decision tree is called *induction*. Most decision tree inductions go through three phases: a tree growing (splitting), stopping and pruning, and these are different for every decision tree algorithm. The splitting criterion is the most important measure for differentiating between decision tree algorithms. Classification tree algorithms can be divided into those that yield binary splits, such as CART [16], Quest [64] and OC1 [74], and those that yield trees with multiway splits, such as C4.5/C5 [80, 83] and CHAID [51]. In keeping with the theme of this Thesis, classification decision trees are further discussed in Chapter 3.

Chapter 3

Data Classification: Decision Tree Induction

3.1 Introduction

The subject of this Thesis is to restructure the feature space for decision tree classification. Learning how to classify objects to one of a predefined set of categories or classes is a characteristic of intelligence that has been of keen interest to researchers in data mining as well as machine learning. Identifying the essential characteristics of a set of examples that represent their class is of enormous use in focusing the attention of a person or computer program. The ability to perform classification and to be able to learn to classify gives people and computer programs the power to make decisions. The efficacy of these decisions is affected by performance of the classification task. In machine learning, the classification task is commonly referred to as supervised learning. In supervised learning there is a specified set of classes, and example objects are labelled with the appropriate class. The goal is to generalise (form class descriptions) from the training objects that will enable novel objects to be identified as belonging to a particular class. In contrast to supervised learning is unsupervised learning where, the goal is to decide which objects should be grouped together, in other words,

the learner forms the classes itself. Of course, the success of classification learning is heavily dependent on the quality of the data provided for training, where a learner has only the input (training set) to learn from. If the data is inadequate or irrelevant then the concept descriptions will reflect this and misclassification will result when they are applied to new data (testing set).

In Section 3.2, we address some of the characteristics of data and representation issues for classification. We describe the decision tree induction techniques related to the work of this Thesis in Section 3.3.

3.2 Data Representation

In a typical classification task, data is represented as a table of examples or instances. Each instance is described by a fixed number of measurements, or features, along with a label that denotes its class. Features (sometimes called attributes) are typically one of two types: categorical (nominal or dicrete), or numerical (real or integer). Table 1 shows fourteen instances of suitable and unsuitable days for which to play a game of golf [80]. Each instance is a day described in terms of the (categorical) attributes Outlook and Wind, and the (numerical) attributes Temperature and Humidity, along with the class label which indicates whether the day is suitable for playing golf or not. A typical application of a classification algorithm requires two sets of data: a training set and a testing set. The training set is used to produce the learned concept descriptions and a separate testing set is needed to evaluate the accuracy of these descriptions. When testing, the class labels are not presented to the algorithm. The algorithm takes, as input a test example and produces, as output, a class label (the predicted class for that example).

Table 1: The Golf data set.

Instance #	Outlook	Temperature	Humidity	Wind	Class
1	sunny	85	85	false	Don't play
2	sunny	80	90	true	Don't Play
3	overcast	83	78	false	Play
4	rain	70	96	false	Play
5	rain	68	80	false	Play
6	rain	65	70	true	Don't Play
7	overcast	64	65	true	Play
8	sunny	72	95	false	Don't Play
9	sunny	69	70	false	Play
10	rain	75	80	false	Play
11	sunny	75	70	true	Play
12	overcast	72	90	true	Play
13	overcast	81	75	false	Play
14	rain	71	80	true	Don't Play

3.3 Decision Tree Induction

An induction algorithm, forms concept descriptions from example data. Concept descriptions are often referred to as the knowledge or model that the learning algorithm has induced from the data. Knowledge may be represented differently from one algorithm to another. As mentioned in the previous chapter, there are numerous approaches to classifying data. Perhaps one of the most common approaches is decision trees.

A decision tree is a tree–like data structure that can be employed in certain data classification tasks. A data set should include the following characteristics in order to be suitable for classification by a decision tree:

- Each instance in the data set must be represented by a fixed number of attributes and must belong to a class.

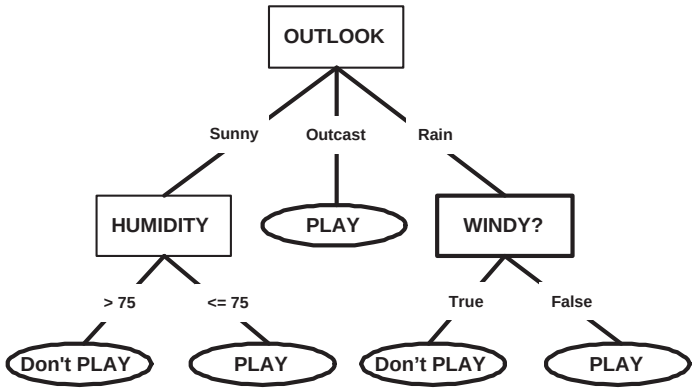


Figure 5: A decision tree for the "Golf" data set. Branches correspond to the partitioning of records; leaves indicate classifications.

- Attributes may be categorical or numerical.
- Classes must be discrete–valued, i.e. categorical.

For a given record, the classification process starts from the root node. The attribute in the node is tested, and the value determines which branch is to be taken. This process is repeated until a leaf node is reached. The record is then classified as the class of the leaf.

For example, Figure 5 shows a simple decision tree for classifying the "Golf" data set presented in Table 1.

Most decision tree inductions go through three phases: tree growing or *splitting*, *stopping* and *pruning*, and these are different for every decision tree algorithm.

- Splitting: The tree growing phase is an iterative process which involves splitting the training data into progressively smaller subsets. The first iteration considers the root node that contains all the data. Subsequent iterations work on derivative nodes that will contain subsets of the data. At each split, all features are analysed and the best split is chosen. One important characteristic of splitting

is that it is greedy, which means that the algorithm does not look at any other part of the tree to see if another decision would best produce a better overall result.

- Stopping: Decision tree algorithms usually have several stopping rules. These rules are usually based on several factors including maximum tree depth, minimum number of instances in a node considered for splitting, or its near equivalent, the minimum number of elements that must be in a new node. In most implementations the user can alter the parameters associated with these rules. Some algorithms, in fact, begin by building trees to their maximum depth. While such a tree can precisely predict all the instances in the training set, the problem with such a tree is that, more than likely, it has *overfitted* the data.

- Pruning: After a tree is grown, one can explore the model to find out nodes or subtrees that are undesirable because of overfitting, or rules that are judged inappropriate. Pruning removes leaves and is a common technique used to make a tree more general. Algorithms that build trees to maximum depth will automatically invoke pruning. In some products, users also have the ability to prune the tree interactively.

Decision trees that allow only a single test on a single attribute per branch node are known as *univariate* trees, whereas those permitting multiple tests on one or more attributes per node are commonly known as *multivariate* trees. For the purpose of this Thesis we will consider only univariate trees.

The construction of decision trees is referred to as *decision tree induction*. Generally, decision tree induction involves the following steps:

1. Start at a single node representing all data (root node).

2. If records are all in the same class then node becomes a leaf labelled with that class.

3. Otherwise, split using the attribute that "best" separates records into individual classes.

4. Repeat until:

   - All records in a node belong to the same class, or;
   - majority of records in same class; or
   - node contains too few records.

The fundamental difference between decision tree induction techniques lies in the splitting criteria. There are many decision tree induction algorithms that deploy the univariate approach including, C5 [83] and its predecessors ID3 [79] and C4.5 [80], CART [16] and CHAID [51], which are discussed in the Subsections that follow. These inductive techniques are exploratory data analysis methods used to study the relationship between the class and the predicting attributes. They are chosen in our research because of their wide application in data mining.

3.3.1 ID3

The *Interactive Decotimizer 3* (ID3) [79] is a simple decision tree algorithm used for data classification. It performs a heuristic top–down greedy search. Initially all training samples are placed in the root node. Then ID3 uses *information gain* as the splitting criterion for selecting the branching attribute of a node. Let the node contain a set S of cases, with $|C_j|$ of the cases belonging to one of the predefined classes C_j. The information needed for classification in the current node, referred to

as *entropy*, is

$$Entropy(S) = -\sum_{j=1}^{c} \frac{|C_j|}{|S|} \times \log_2 \left(\frac{|C_j|}{|S|} \right)$$

This value measures the average amount of information required to identify the class of a case. Assume that we use an attribute A as the branching attribute which, divides the cases into n subsets. Let S_i denote the set of cases in subset i. The information needed for the subset i is $Entropy(S_i)$. Thus the expected information required after choosing attribute A as the branching attribute is the weighted average of the subtree information:

$$Entropy_A(S) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} \times Entropy(S_i)$$

Thus, the information gain of splitting using attribute A, $Gain(A)$, is the expected reduction in entropy caused by partitioning the cases in accordance with attribute A:

$$Gain(A) = Entropy(S) - Entropy_A(S)$$

As a smaller value in the entropy corresponds to better classification, the attribute A with the maximum information gain is selected for the branching of the node.

After the branching attribute is selected, the training cases are divided by the different values of the branching attribute. If all cases belong to the same class, then this branch becomes a leaf node labelled with that class. If all cases are labelled with a class, the algorithm terminates. Otherwise, the process is recursively applied on each branch.

3.3.2 C4.5 and C5

C4.5 is a successor of ID3, also developed by Ross Quinlan. The use of information

gain measure has a serious deficiency that favors tests with many outcomes [80]. C4.5 improves this by using *gain ratio* as the criterion for selecting the branching attribute. The gain ratio is computed in two steps. Firstly, the $split\_info_A(S)$, which represents the potential information generated by dividing S into n subsets, is measured:

$$split\_info_A(S) = -\sum_{i=1}^{n} \frac{|S_i|}{|S|} \times log_2\left(\frac{|S_i|}{|S|}\right)$$

Secondly, $splitinfo_A(S)$ is used to normalise the gain criterion computed above to yield the gain ratio:

$$Gain\_ratio(A) = Gain(A)/split\_info_A(S)$$

The attribute with the maximum value of gain ratio is selected as the branching attribute.

C5 is the successor to C4.5 which also uses gain ratio to select the branching attribute. C5 offers more enhancements to C4.5 in terms of reduced memory usage, increased speed and improved accuracy [83].

3.3.3 Classification And Regression Tree (CART)

CART is another well-known decision tree classification algorithm, popularised by Breiman et al. [16]. Like the C5 family, for each attribute, CART tries to find the best split, then it selects the attribute with the best split. The criteria for finding the best split is to maximise the decrease in the *impurity* of the parent node. A node is pure if all its cases belong to one class. Thus, it prefers splits that put the largest class in one pure node. The impurity measure used in CART is referred to as the *gini index*. For instance, given a data set S with c classes, then

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2$$

where p_i is the relative frequency of class i in S.

If a test attribute A splits S into two subsets S_1 and S_2 with sizes N_1 and N_2 respectively, then

$$Gini_A(S) = \frac{N_1}{N} \times Gini(S_1) + \frac{N_2}{N} \times Gini(S_2)$$

where N is the total number of samples in the node.

Therefore, the decrease in impurity caused by splitting on attribute A, GI_A, is measured as

$$GI_A(S) = Gini(S) - Gini_A(S)$$

The attribute that provides the highest $Gini_A(S)$ implies the least interesting information, whereas the attribute that gives the smallest $Gini_A(S)$ implies the most interesting information and is therefore chosen to split the node.

3.3.4 Chi-squared Automated Interaction Detector (CHAID)

CHAID is a decision tree induction algorithm developed by G. V. Kass [51]. CHAID, like C5, is a greedy algorithm. However, the manner in which these two algorithms construct their tree differs greatly.

Suppose we have a class attribute with $c \geq 2$ classes and a predicting attribute with $n \geq 2$ different values. We reduce the given $c \times n$ contingency table to the most significant $m \times c$ table by merging categories of the predicting attribute, where $m = 2, 3, .., n$. CHAID uses the Chi^2 test of significance for determining which categories to merge. The $c \times n$ contingency table consists of n rows and c columns, where o_{ij} is the observed frequency of the i^{th} row in the j^{th} column, and e_{ij} is the expected frequency of the i^{th} row in the j^{th} column, such that

$$e_{ij} = \frac{RowTotal \times ColumnTotal}{GrandTotal}.$$

then the Chi^2 test is

$$Chi^2 = \sum_{j=1}^{n} \sum_{i=1}^{c} \frac{o_{ij} - e_{ij}}{e_{ij}}.$$

The full algorithm, found in [51], is outlined below:

1. For each predictor in turn, cross-tabulate the categories of the predictor with the categories of the dependent variable and do steps 2 and 3.

2. Find the pair of categories of the predictor (only considering allowable pairs as determined by the type of predictor) whose $2 \times d$ sub-table is least significantly different. If this significance does not reach a critical value, merge the two categories, consider this merger a new compound category and repeat step.

3. For each compound category consisting of three or more of the original categories, find the most significant binary split (constrained by the type of predictor). If this significance is beyond some boundary, implement the split to form two new categories and repeat step 2.

4. Calculate the significance of each optimally merged predictor and isolate the most significant one. If the significance is greater than a criterion value, subdivide the data according to the (merged) categories of the chosen predictor.

5. For each partition of the data that has not yet been analysed, return to step 1. This step may be modified by excluding from further analysis partitions with a small number of observations.

Each of these decision tree algorithms, namely C5, CART and CHAID, is used in the experimental work presented in later chapters.

Chapter 4

Background on Evolutionary Computation

4.1 Introduction

The previous two chapters presented a background to KDD, data mining and classification using decision tree induction, addressing issues related to the work of this Thesis. In this chapter, we introduce the topic of evolutionary computation. Firstly, evolutionary computation is presented in Section 4.2 as a set of algorithms inspired by the principles of biological evolution. In Section 4.3, we describe in more detail the paradigm of particular relevance to this Thesis, namely genetic programming. A survey of relevant applications of evolutionary algorithms is presented in Section 4.4. Section 4.5 summarises the chapter.

4.2 Evolutionary Computation

A *search algorithm* searches through the space of possible solutions. The term *search* is strongly related to optimisation because optimisation is actually a search for the optimal solution(s) [35]. In order to perform optimisation, it is crucial to have a way

of measuring the quality of any possible solution in the search space. This is achieved by employing an *objective (fitness) function*, which evaluates a solution by assigning to it a numerical value which in some way reflects its quality.

Search techniques can be divided into *weak* and *strong* methods [5]. Weak search techniques are general domain-independent methods and hence are widely applicable to different problems. On the other hand, strong search techniques are problem specific and they require some domain knowledge to guide the search. Domain knowledge allows strong methods to generally reach an acceptable solution faster than weak methods. Also, search techniques can be categorised as *trajectory* and *population–based*. In a trajectory approach, a single initial solution is the start of a trajectory through the space, the rest of the trajectory is achieved through iterative–based algorithms such as *local search*, *simulated annealing* [26] and *tabu search* [40]. In population–based approach an initial population of solutions are used to search the space, such as those used in evolutionary and other population–based search.

Evolutionary computation is a field of biologically-inspired research concerned with optimisation and search algorithms that imitate the principles of biological evolution [33]. *Evolutionary algorithms* (EAs) are weak, population–based, search techniques which use stochastic search procedures inspired from biological evolution [5]. EAs employ the *Darwinian* principle of survival of the fittest and natural selection, and hence the representation of solutions and operators is borrowed from the terminology used in natural evolution [10].

The origins of EAs can be traced back to the 1950's [34]. The first published work on simulating the evolution of genetic systems was first published by A. Fraser [36] and G. Box [14]. For the sake of brevity we will not concentrate on this early work

but will discuss four methodologies that have emerged in the last few decades:

Evolution Strategies (ESs), were developed by Rechenberg, Schwefel and Bienert at the Technical University of Berlin in 1964 [32, 7]. The aim of ESs is to solve continuous optimisation problems [31]. Basically, the phenotype of an individual is a vector containing the candidate values of the parameters being optimised. The genotype of each individual is a pair of real-valued vectors of the phenotypic vector and a vector of standard deviations used to apply the mutation operator.

Evolutionary Programming (EP) was proposed by Lawrence Fogel in 1962 [31]. EP is used for evolving programs that are capable of predicting their environment, and use those predictions to achieve some goal. In its most general approach, the environment is described as a sequence of symbols taken from a finite alphabet. With its knowledge of the environment the evolving entity is supposed to produce an output symbol that is related in some way to the next symbol appearing in the environment. The output symbol should maximise a payoff function, which measures the accuracy of the prediction. Finite state machines were selected to represent individuals in EP as they provide a meaningful representation of behavior based on interpretation of symbols.

Genetic Algorithms (GAs), first proposed by John Holland in 1975 [46, 41], are the most well-known paradigm of evolutionary computation. GAs operate on an encoding of the problem solution in a manner similar to the manipulation of natural genetic encodings. In the traditional GA, the solution to a problem is encoded as a fixed-length string of bits. Each bit represents a gene, and its value and position in the chromosome determines its meaning. This genotypic representation is decoded by a user-defined function to produce a phenotype, which is a solution to the problem. The fitness function is normally applied to the phenotype.

Genetic Programming (GP), as GP plays a major role in this Thesis, it is addressed separately in Section 4.3.

All evolutionary computation methods differ from one another in the manner that they represent their solutions, the genetic operators they use and the selection methods. However, they all have common elements, as shown in Figure 6.

- ◇ A population $P = \{s_1, s_2, .., s_N\}$ of N individuals, where each individual $s_i \in P$ being a solution to the problem.
- ◇ A user-defined fitness function $f : P \rightarrow \Re$ for assessing how well an individual solves the problem. This is non-trivial and dominates the computation time of the algorithm.
- ◇ A mechanism for creating the initial population, P_0, often generated at random.
- ◇ A probabilistic selection mechanism.
- ◇ Genetic operators similar to reproduction, mutation and sexual recombination in natural organisms for generating new populations.
- ◇ A termination (stopping) criteria.

Figure 6: Elements of an Evolutionary Algorithm.

EAs operate on a population of individuals that represent possible solutions to a problem in their chromosomes. Each individual can be as simple as a string of bits (zeroes and ones), or as complex as a computer program (parse trees). The initial population of individuals may be created entirely at random, or some knowledge about previously known solutions may be used to seed the population. The algorithm evaluates the individuals to determine how well they solve the problem at hand using a user–defined objective function, which is unique to each problem. Evolution

proceeds as the current population is iteratively transformed into a new population through one or more of the following procedures: selection, reproduction, recombination and mutation. Although transformation is different from one algorithm to another, it generally involves two stages:

- The fitness-proportionate selection of individuals from the population to form the mating pool.
- The application of genetic operators to individuals in the mating pool to produce offspring. These offspring are combined with the parent population in some way to make up the new population.

Each iteration of this transformation is called a *generation*. The offspring are evaluated, and the cycle of selection and creation of new individuals is iterated until a satisfactory solution is found or some termination criteria is met. This is usually when a predetermined limit on the number of generations (iterations) is reached.

The above describes one run of the algorithm. Since the results are based on a stochastic process, it is recommended that multiple runs be performed.

4.3 Genetic Programming

Genetic Programming (GP) was pioneered by John Koza in 1992. In his books [55, 56, 57], he successfully applied GP to many problems from different domains. GP is essentially a branch of GAs, with the primary difference being the representation of the structures undergoing adaptation. However, the concept of evolving computer programs was first introduced in 1985 by N. Cramer [22].

4.3.1 Problem Encoding and Control Parameters

The individuals in a GP population are not fixed-length binary strings, but rather *trees* which can vary in size and shape. Each tree is a *parse-tree* encoding of a computer program or functional expression. The reason for this representation is because most language compilers translate a given program into a parse–tree before generating a sequence of executable machine instructions. For example, the program $(x_1^2 + (\frac{x_3}{2.5})) * (x_2 - \sqrt{x_2})$ can be represented in the parse–tree (* (+ (* x_1 x_1) (/ x_3 2.5)) (- x_2 ($\sqrt{}$ x_2))), see Figure 7.

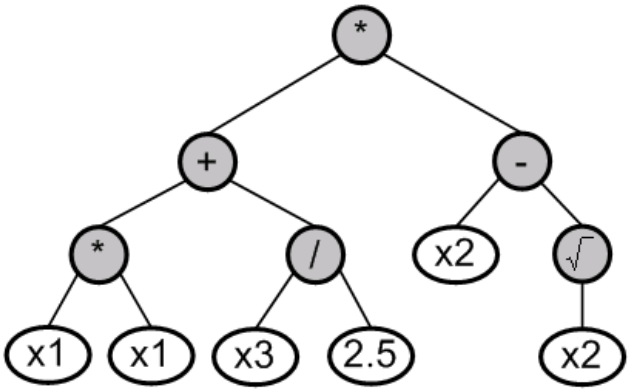


Figure 7: A parse tree of the program $(x_1^2 + (\frac{x_3}{2.5})) * (x_2 - \sqrt{x_2})$.

Figure 7 shows that a parse-tree consists of *branch nodes* and *leaf nodes*. The branch nodes (with children) are *functions* which take arguments from their child nodes. The leaf nodes (without children) are *terminals* which are the inputs to the overall program.

There are five essential elements that must be determined to make up a genetic program [55]:

- the set of *terminals*,
- the set of operators or *functions*,
- the *fitness function*,
- the *parameters controlling the run*,
- and the *termination criteria* of the GP.

4.3.1.1 The Terminal and Function Sets

The function and terminal sets define the search space for the problem. The function set contains N_F functions from $F = \{f_1, f_2, ..., f_{N_F}\}$ in the language such that each function f_i has *arity* (number of arguments) $arity(f_i) \geqslant 1$. The functions can be typical constructs found in programming languages, including those in Table 2. Alternatively, the function set can contain user–defined functions depending on the application domain.

Table 2: Examples of functions used in the function set.

Functions Type	**Examples**
Arithmetic	$+$, $-$, $\times$, $\div$
Mathematical	sin, cos, log, exp
Boolean	AND, OR, NOT
Conditional	IF, $IF-THEN-ELSE$
Loop	FOR, $WHILE$, $DO-WHILE$,
Comparative	$\geq$, $\leq$, $<$, $>$, $=$
:	:

Similarly, the terminal set contains N_T elements from $T = \{t_1, t_2, ..., t_{N_T}\}$. The terminals can be problem-specific variables, random constants or *zero-arity* (with no

arguments) functions. The random constants can be any real or integer numbers that are initialised randomly at the start of the run and maintain their values throughout the run. An example of zero-arity functions can be *GoLeft*, *PickUP*, .. etc, for a robot control problem. Recalling the example shown in Figure 7, the function set is $F = \{+, -, \times, \div, \surd\}$ and the terminal set is $T = \{x_1, x_2, x_3, 2.5\}$.

4.3.1.2 The Closure and Sufficiency Properties

For any problem, in order for the GP to work properly, F and T must consider the *closure* and *sufficiency* properties [55]. If all terminals are considered $0 - arity$ functions, then we can form the uniform function set $C = F \cup T$. The closure property requires that each function in F is able to accept, as its arguments, any values and data types that may be returned by any function in C. This avoids runtime errors. The sufficiency property ensures that the functions in C are capable of expressing the solutions to the problem.

4.3.1.3 The Fitness Function

The fitness function measures the success of an evolved individual and is obtained by executing the individual for a set of test cases and seeing how well it solves the problem. The measured fitness of a program determines whether it will take part in the creation of the next generation. There are four types of fitness measures; *raw fitness*, *standardised fitness*, *adjusted fitness* and *normalised fitness* [55]. These fitness measures are arranged in some hierarchy or sequence, i.e. each fitness measure is a transformation of the preceding measure. Raw fitness of an individual i, $r(i)$, measures the fitness that is defined in the natural terminology of the problem, usually evaluated over a set of fitness cases. For example, if the task is classification then the raw fitness is the classification accuracy or error rate. Standardised fitness, $s(i)$,

restates the raw fitness such that the lower numerical value is always the better value. As a result the fittest individual will have a fitness value of zero. Adjusted fitness, $a(i)$, is measured from the standardised fitness. The adjusted value for any individual in a population is always between zero and one. Adjusted fitness has the benefit over standardised fitness in that it exaggerates the importance of small differences in values of standardised fitness as the standardised fitness of multiple individuals approaches zero. Adjusted fitness can be calculated as: $a(i) = \frac{1}{1+s(i)}$. Normalised fitness, $n(i)$, of an individual, i, in a population of size M is within the range of zero and one and is calculated as: $n(i) = \frac{a(i)}{\sum_{k=1}^{M} a(k)}$. The sum of the normalised fitness of all the individuals in a generation is one. The fitter individuals have a larger value of normalised fitness.

4.3.1.4 The Run Parameters

Two of the most important parameters are the *population size* and the *maximum number of generations*. These parameters determine the amount of exploration and exploitation performed by the GP. A small population that is evolved for a large number of generations is biased for exploitation. Similarly, a large population evolved over a small number of generations is biased towards greater exploration. By carefully choosing the value of these parameters, one can balance exploration with exploitation.

The reproduction, crossover and mutation probabilities are used to select rates of the genetic operator to be used to create offsprings. They specify the probability with which each of the three genetic operators are chosen. Reproduction and mutation probabilities are usually set to be very small.

The shape and depth of the trees created for the initial population can be specified by the *creation method*, and the *maximum depth parameters* respectively. There are

three creation methods commonly in use, called the *full*, *grow*, and *ramped half-and-half* methods. The full method creates symmetric trees that are of maximum depth along any branch from the root. This is achieved by selecting the root node randomly from the function set, and then continuing recursively to select randomly from the function set for subnodes until the maximum allowable depth is reached (at which point a node is selected from the terminal set). Selecting the grow method creates asymmetric trees. As with full trees, the root is always chosen from the function set, this time however we continue recursively by selecting children from C, until either a terminal is selected or the maximum depth is reached (at which point a terminal is always selected). The ramped half-and-half is used to ensure population diversity in the initial population. The size of trees created is ramped up from 2 to the maximum allowable tree depth. Half of these trees are created using the full method, and the other half are created using the grow method.

4.3.1.5 The Termination Criteria

As the name suggests, the *termination criteria* determine the ending of the GP run. Typical termination criteria include:

- the fitness of the best individual scores the desired value, resulting in a successful run.
- or the maximum number of generations is reached, resulting in a less successful run.

4.3.2 The Initial Population

The initial population P_0 is typically randomly generated from C. Since it is the starting point for the search, it is important that a range of tree shapes and sizes be

present to avoid bias. Although functions and terminals absent from the population can be regenerated by mutation, this process should not be relied upon, and the initial population should contain an even distribution of the functions and terminals available. Genotypic duplicates should not be allowed so as to use the full potential of the population [55].

4.3.3 The Genetic Operations

The traditional GP operators are similar to those used in GAs, but have been adapted to work with trees. Each operator must ensure that the offspring do not exceed the maximum allowable tree depth.

Reproduction and *crossover* are viewed as the primary genetic operators, while mutation and other novel operators are secondary. Reproduction simply copies the individual to obtain an identical offspring. Crossover requires two parents and randomly selects a node in each as a crossover point. The sub-trees rooted at these crossover points are swapped to obtain the offspring, see Figure 8. There are a few important points about crossover in GP. Firstly, unlike GAs, crossover between two identical parents can result in different offspring. Secondly, if the crossover points are the roots of both parents, then crossover degenerates to reproduction. Thirdly, if the two crossover points are terminal nodes, crossover is like point mutation that simply changes a single node. Because of this latter point, internal points are typically chosen as crossover sites with a higher probability than terminals.

The most common form of *mutation* is *sub-tree* or *grow* mutation. A node is randomly selected from the tree, and the subtree rooted at the selected node is replaced with a new, randomly-generated sub-tree, see Figure 9. The usefulness of mutation

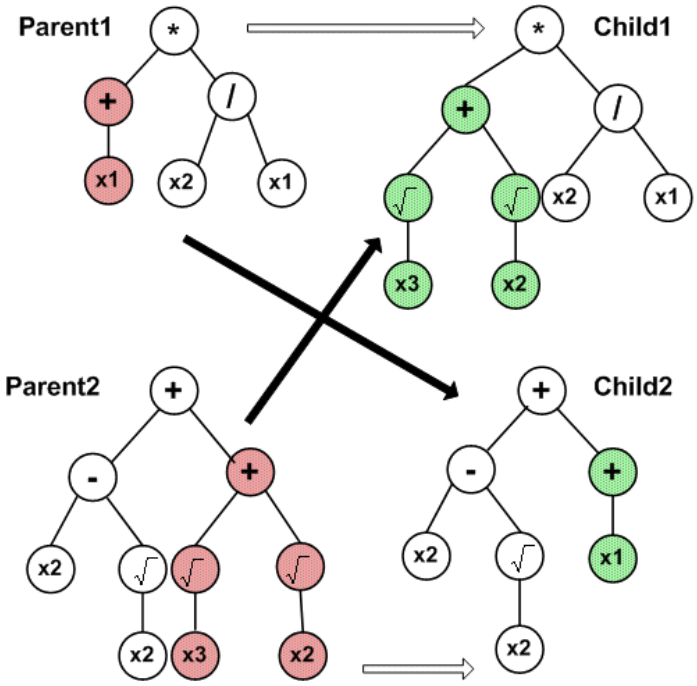


Figure 8: GP Crossover.

is questionable according to Koza: mutation is useful in GAs to relieve a search from stagnation and to re-introduce lost alleles into the population. Since in GP, identical parents can result in different offspring through crossover, mutation is not required to avoid stagnation. In GP the meaning of an allele is not fixed to its position, and since the sizes of the function and terminal sets are much smaller than the number of nodes in the population, it is unlikely that a symbol would disappear from the population.

Koza used several other secondary operators particular to GP, such as *permutation*, *editing* and *encapsulation*. For a fuller description of these operators, please refer to [55, 56].

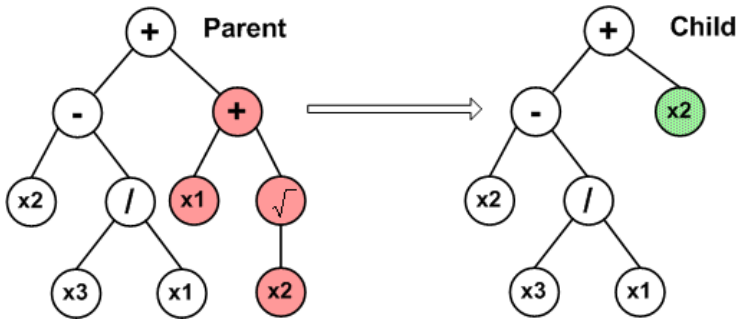


Figure 9: GP Mutation.

4.4 Applications of Evolutionary Algorithms

The versatility of the GP representation is its strength, which enables it to solve a wide variety of symbolic problems. Unlike other methods, GP allows the size and structure of the final solution to be left unspecified and undergo adaptation [55]. GP has achieved slightly superior performance than the best known human-generated solutions. The application areas of GP include *electrical circuit design*, *image processing*, *robotics*, *autonomous agents* and *pattern recognition*. A survey of some these applications can be found in [8, 55, 56, 57]. William Langdon maintains a bibliography of published material at the Genetic Programming Bibliography[1]. John Koza also maintains information of GP applications, and other events such as conferences, workshops, .. etc. at the genetic–progamming.com[2] and at the genetic–programming.org[3] websites.

In this Section, we survey the role that EAs, particularly GP and GA, play in the data mining task of classification and the preprocessing tasks of feature selection, extraction and construction.

[1]http://www.cs.bham.ac.uk/~wbl/biblio/README.html
[2]http://www.genetic-pogamming.com
[3]http://www.genetic-pogamming.org

4.4.1 Evolutionary Pre–processing

Since the main problem with the preprocessing tasks of feature selection, extraction and construction is how to cope with feature interaction, redundancy and correlation, local search-based approaches become less effective. Hence, there was a strong need to adapt new, more powerful methods in global search [38].

4.4.1.1 Feature Selection

There are many ways in which EAs can be used to address the problem of feature selection. The evolutionary approach most often applied for feature selection is to hybridise the selection method with the learning algorithm (i.e. the wrapper approach). In this approach, the learning algorithm is used to evaluate the fitness of the feature subsets obtained during the evolutionary computation. While this preserves any inductive and representational biases of the learning algorithm, it is more computationally intensive than selecting the features independently.

Siedlecki et al. [88] used GAs to select features for the*k–NN* algorithm. GAs assigned each individual in the population a binary (zero or one) value, where a one indicated that a feature was included in the classification, and a zero indicated that it was not. The KNN algorithm was then used to evaluate how good each individual is based on its classification accuracy and the number of the features used.

Others have applied the same basic binary encoding to select features in classification problems using neural networks [19, 18]. Punch et al. [78] extended the simple binary feature selection idea by representing an individual by a series of weights between zero and ten, thus weighting some features as more important than others. They found that their extension appeared to work better than the zero/one approach

of Siedlecki et al. [88] on noisy real world datasets.

Vafaie et al. [95] also investigated a similar approach to feature selection using decision trees for classification. However, in their work, instead of just weighting each feature, they allowed the combination of existing features to form new features through simple operations such as add, subtract, multiply, and divide. This adaptive feature–space transformation led to a significant reduction in the number of features and improved the classification accuracy. Other related work in this area is that of Yang et al. [103] who used neural networks as the classifier and a simple zero/one strategy for weighting each feature.

A very different use of genetic algorithms in feature selection is in the generation of ensembles of classifiers. Recent work by Dietterich [25] has shown that it was possible to improve classification accuracy by combining the prediction of multiple classifiers. These ensembles of classifiers differed in the ways in which the classifiers were generated and their results were combined. Early work of Ho [45], which used a random selection of features to create an ensemble, was extended by Guerra-Salcedo et al. [43]. They replaced the random selection with a more intelligent approach using genetic algorithms, and showed empirically that their method was more accurate.

4.4.1.2 Feature Extraction

EAs have proven to be effective in extracting new features to reduce the dimensionality of the feature space. Bot [13] proposed a GP framework for extracting features to improve the performance of the *KNN* classifier. The new features were constructed from the original variables by adding, subtracting, multiplying or dividing the original

variables by constants and/or the other variables. Besides these operators, the Mean function has also been added which takes two or three inputs and returns their mean. Individuals in the population are formulas that specify how a new feature should be calculated. For each additional feature, the GP is run. Suppose $n-1$ runs have already taken place. These previous $n-1$ runs have resulted in a new dataset of $n-1$ columns (the new features) plus the class attribute. In this run the n^{th} column will be evolved.

This work followed the interleaving approach to feature extraction, in which the fitness of the GP was based on the accuracy of the associated classifier on the training set. The author used three different fitness measures; the accuracy of the K–NN, the minimum distance to means (*MDM*), and the parallelepiped classifier. He compared the performance of the *KNN* classifier on the evolved features to those extracted using discriminant analysis, principal component analysis (*PCA*), and the add–one algorithm. The results showed that the GP performed best with the *MDM* accuracy as fitness function. The accuracy of GP was similar to that of the discriminant analysis but outperformed those of the *PCA* and the add–one algorithms. The GP reduced the feature set, on most datasets, to one, two, or three features. The results of this work are used in the comparison study, shown in Chapter 6.

Sherrah et al. [86] proposed a feature extraction system, called the *Evolutionary Pre–Preprocessor* (EPrep), which used GP to evolve new features to improve the classification task. EPrep employed the wrapper approach, where the central component of EPrep was the genetic program, and each individual in the population represented a pre–processor network and a standard classification algorithm (Generalised Linear Machine, *k–NN* or Maximum Likelihood Classifier). The pre–processor extracted new features, which are non-linear combinations of the original ones and passed them to

the specified classifier. The fitness function was based on the error rate of the classifier on the pre–processed data. The authors claimed that EPrep did not only extract features, it also chose the classifier that performed best on the pre–processed data. This claim can be questionable as both feature construction and classifier selection are random processes.

Experiments were carried out on 9 datasets and the error rate of the best classifier was compared to that of the EPrep [87]. The results, on the testing sets, showed significant improvement on only 2 datasets. On the rest, EPrep could not make significant improvement. In another experimental work [86], the performance of EPrep was compared to that of an MLP ANNs. The experiments were performed on 15 data sets from the UCI ML repository [68]. The results showed that EPrep outperformed ANNs on only 4 out of the 15 data sets and ANNs performed much better on the remaining 11 data sets. In Chapter 6 we compare the results of our experimental work to that of Sherrah's [86] on two common data sets, namely the Abalone and the Balance-scale.

Raymer et al. [82] applied GP to improve k-NN performance without feature reduction. For each attribute, they evolved a tree which was a function of the attribute itself and zero or more constants, e.g. $2.1x^3 + 1.5x$. They applied this to a biochemistry database and claimed improvement over a similar system with a GA. No comparisons were made to other feature extraction algorithms.

Masand et al. [66] used GP to find additional features for one historical customer dataset used for predicting future customer behavior. They reported improved performance of C4.5 for this dataset.

Early work by Tackett [93] identified targets in a cluttered image by combining simple features extracted from the segmented image through linear and non-linear operations. If the resulting single value at the root of the tree was greater than zero, then the object was classified as a target. Stanhope et al. [92] used a similar approach in their work on target classification using Synthetic Aperture Radar (SAR) images.

4.4.1.3 Feature Construction

There are essentially two approaches to constructing attributes in relation to data mining; one method is as a separate pre-processing stage, in which the new attribute(s) are constructed before any induction process, i.e. before the classification algorithm is applied to build the model. The second approach is an integration of construction and induction, in which new attributes are constructed within the induction process. The latter is therefore a hybrid induction algorithm.

In [59] and [9], the authors have developed a GP to evolve new Boolean attributes from the original Boolean attribute vector. This was applied as a pre-processing stage prior to testing the inclusion of the new attributes on parity problems, using C4.5 and backpropagation [9] and quick-prop [59].

In a recent study by Otero et al. [75], the authors used genetic programming as a pre–processing, attribute construction technique. For each data set used, and for each trial in a 10xCV test, a single new attribute was evolved using a GP with Information Gain Ratio as the fitness function of the GP. The GP constructed new attributes out of the continuous (real-valued) attributes of the data set being mined. Each individual corresponds to a candidate new attribute. The terminal set consisted of all

the continuous attributes in the data. The function set consisted of arithmetic and relational comparison operators. No restriction was applied on the depth or breadth of the GP trees, however, the overall size of the trees was restricted.

Classification using C4.5 was applied to the data set, both with and without the new attribute and the results showed an improvement in the performance of C4.5 with the use of the newly evolved attribute. However, in this study, it is notable that the objective function of the pre-processing technique, i.e. the attribute construction GP, and the induction technique C4.5, both use Information Gain Ratio. One is left asking, therefore, will a classifier whose induction process is not based on information gain benefit, or indeed benefit as much as C4.5, with the inclusion of an attribute which has been evolved by a GP with information gain as its fitness function? This question is addressed in Chapter 6, Section 6. 6.

Smith and Bull [90] proposed a system called *GAP*, which uses both GP and GAs to construct and select features. Their approach involves two stages; feature construction using GP and the feature selection using GAs. They used the WEKA [99] implemetation of C4.5, known as J48, to evaluate the fitness of both the constructed and selected features. In the feature construction stage, the population consisted of 101 genotypes, created at random. Each genotype consisted of n GP trees, where n is the number of numeric valued features in the data set. Each GP trees might contain 1 or more nodes. The fitness measure here was the accuracy of C4.5. Only one new feature was passed from the GP to the GA which selected from original features and the constructed one, those that are most predictive. The fittest individual from the feature creation stage (using GP) was analysed to see if any of the original features did not appear to be used. If there are any missing, sufficient trees were added to ensure that every original feature in the dataset appeared at least once (the new trees

are not randomly generated as in the feature construction stage, but have single nodes containing the required attribute). The extended genotype (up to twice as long as the fittest individual of the feature creation stage) replaced the initial individual and was used as the basis of the feature selection stage (using GA). A new data set was constructed with one attribute for every tree in the extended genotype.

The performance of GAP was compared to C4.5 and another four classifiers on ten datasets from the UCI repository. The results over ten-fold trials showed a slight improvement in the accuracy of eight datasets using GAP when compared C4.5. When compared with other classifiers, GAP was slightly better in only four. They extended their work to include a choice of another two classifiers, namely k–NN and Naïve Bayes, for evaluating the selected and constructed features [91]. This work was similar to Vafaie et al. [95] who used GAs to perform the feature selection for a face recognition dataset. The selected features were evaluated using the accuracy of C4.5. The feature subsets were then passed onto the GP to evolve new features which are evaluated again by C4.5. The performance of GAP is also included in the comparison study presented in Chapter 6.

4.4.2 Evolutionary Data Classification

In this Section we survey some of the applications of EAs in data classification. The main advantages in the use of EAs over the classical greedy induction algorithms are their ability to perform a global search and the way they treat the attribute interaction problem. Generally, there are two approaches to using EAs for classification; combined with another inductive algorithm (a hybrid approach) or as a stand–alone classifier. In the hybrid approach, EAs are used in conjunction with classification

algorithms such as neural networks and decision trees. In the stand-alone classifier, EAs are used to actually perform the classification.

In [104], Yao proposed a hybrid EAs–ANN framework (EANN). In this framework, EAs can be used to perform various tasks, such as finding a near–optimal ANN architecture automatically, and evolving connection weights which provides a global approach to connection weight training. EAs can also allow an ANN to adapt its learning rule to its environment. Wong et al. [101] proposed a *generic genetic programming* (GGP) framework to integrate GP and *inductive logic programming* (ILP). The developed a system called *LOGENPRO* (LOgic grammer based GEnetic PRogramming system) which was used to discover knowledge represented as functional programs and to induce knowledge represented as rules. A hybrid GP–decision tree approach for classification was proposed by Marmelstein et al. [65]. The authors developed a a hybrid GP-decision tree classifier for finding decision region boundaries, which differentiate one class from another. The proposed system incorporates GP classifiers into a decision tree structure. This was achieved by incrementally constructing a decision tree using GPs to implement the decision nodes. At the root node, each node's GP is evolved to separate the data into groups corresponding to a particular class and those corresponding to the remaining classes. Then each child node classifies a successively smaller subset of the data processed by its parent. After a parent node classifies its assigned data subset, this subset is passed to the child node to rectify any residual mistakes. The process continues until the error rate is below a predefined threshold, the maximum depth of the tree is reached or there is no fertile nodes. The fitness function was based on the addition of the error rates on the training and validation sets.

In the context of rule discovery, Freitas [37] proposed a GP framework for the

induction of both classification and generalised rules from databases. The framework outlined a method for classification using *tuple set descriptors* (TSD) in the form of WHERE clauses for SQL queries, and a count of rows of a goal attribute class matching the TSD. The fitness function corresponded to some rule quality evaluation measures. A selection procedure used the fitness value to choose the best rules. In Au et al. [6], the authors proposed an evolutionary learning algorithm for data mining (DMEL) to mine rules in databases. DMEL searches through huge rule spaces effectively using an evolutionary approach. The algorithm encodes a complete set of rules in one single chromosome. It performed its tasks by generating a set of initial first–order rules using a probabilistic induction technique in order to obtain higher order rules. DMEL evaluates the fitness of a chromosome using a function defined in terms of the probability that the attribute values of a record can be correctly determined using the rules it encodes. DMEL was applied to telecommunications data for churn prediction to mine rule representing the churn patterns and to predict the likelihood of a customer churning in the near future.

There is a wealth of resources on the applications of EAs in data mining. This includes international conferences such as the International Conference on Machine learning (ICML), the Genetic and Evolutionary Computation Conference (GECCO) and the Congress on Evolutionary Computation (CEC). The journals IEEE Transactions on Evolutionary Computation (EC) and the IEEE Transactions on Knowledge and Data Engineering (KDE) are also excellent resources.

4.5 Summary

This chapter has presented a brief introduction to the field of evolutionary computation, and in particular, genetic programming. The applications of GP in data pre-processing and classification were also addressed. The next chapter starts the novel content of this Thesis by addressing the problem of feature construction in classification and describing how genetic programming can be used to that end. The experimental results of the application of GP to feature construction are presented in Chapters 6 and 7. In Chapter 8 we present a commercial case study.

Chapter 5

Experimental Setup

5.1 Introduction

In this chapter, we describe the experimental set up and methodology of the research undertaken. The objective is to investigate the use of GP for the task of attribute construction (constructive induction) for decision tree classification. The intention is to study the effect of including a constructed feature in the original set of attributes on the performance of the classification models, particularly on the predictive accuracy and the size of the resultant trees (for decision tree models). The constructed attributes are analysed for potential hidden relationships among the original attributes as well as explaining physical properties about the data. We also compare the use of GP for classification to using it for feature construction.

In Section 5.2, we introduce the problem of feature construction for decision tree classification, outlining the objectives of the experimental work. The data sets used in the experiments are presented in Section 5.3. Section 5.4 outlines the sampling method of the data sets. In Section 5.5, we describe the GP algorithm and various algorithm parameter settings, including the four fitness measures used to evolve new

features. The experimental methodology is given in Section 5.6. Examples of the evolved features are shown in Section 5.7. Finally, Section 5.8 concludes this chapter.

5.2 The Problem

In data mining, when constructing classification models from data sets, the data is normally presented as a fixed number of features, or *attributes*, one of which is the discrete valued, dependent or *class* variable. The purpose of classification is to find a description of the class variable based on some or all of the other predicting variables. The representation of this description varies depending on the particular induction technique used, and includes decision trees, rules, artificial neural networks, Bayesian classifiers, and many others, see [99].

We are primarily addressing the performance of decision tree classifiers. A decision tree is typically constructed using a greedy, iterative process, wherein, during the induction stage, each internal decision node is associated with a test on one of the predicting attributes, the particular test being chosen to optimise a measure relating to the splitting criterion. Successors of the internal nodes are formed and the process is repeated at each successor node. In C4.5/C5, for instance, the splitting criterion is the *Information Gain Ratio* [80], whilst in CART (Classification and Regression Trees), the splitting criterion is the *Gini Index* [16], which is a measure of impurity. Another decision tree classifer, CHAID [51], uses the *Chi–squared* test of significance to evaluate proposed groupings of predictor values and to assess the quality of a proposed split.

The success of any classification algorithm depends on its ability to represent any

inherent patterns in the data set, and hence depends on the set of predictive attributes available, or its attribute vector. Greedy techniques such as tree induction, typically assess the predictive ability of attributes on a one–by–one basis. In other words, at each internal node of the tree, each attribute is analysed in turn to measure its predictive power in terms of the class output.

In this Thesis, we restrict our attention to the construction of new attributes, and in particular, to the use of GP [55] to construct/evolve new attributes. The aim of this research is to study the effect on the performance of a range of classification algorithms with the inclusion of the evolved attributes. Four different fitness functions are used in the genetic program; the information gain (IG), the gini index (GI), information gain+gini index (IG+GI) and the Chi–squared test (Chi^2). The classification algorithms used are three classification tree algorithms, namely C5, CART, CHAID and an MLP neural network.

We perform attribute construction with the aim of improving the performance of classification models in two aspects. The first is the classification accuracy, i.e. improving the classifiers error rates, these results are presented in Chapter 6. The second aspect is the size of the resulting classification tree models, this is shown in Chapter 7. The experiments presented in this Thesis also address the question of whether or not decision tree algorithms benefit more from the inclusion of an attribute evolved using a GP whose fitness function is based on the splitting criterion of the associated decision tree [72, 73, 71]. Another objective is to analyse the evolved (constructed) attributes to determine whether they reveal hidden relationships between the predictive attributes or other physical properties about the data, presented in Chapter 7. Finally, we also investigate the use of GP for classification and compare it to using it for feature construction, see Chapter 7.

5.3 The Data Sets

The experiments are performed on five data sets, all from the UCI data repository [68]. Table 3 shows the number of cases, classes and attributes for each data set. Note that our GP considers all attributes to be real-valued variables. Thus the single boolean attribute in the Abalone data set is considered as real-valued for the purposes of this study. A fuller description of each data set is given in Appendix A.

Table 3: Data Sets used in experimental work.

Data Set	Cases	Classes	Attributes
Abalone	4177	28	8
Balance-scale	625	3	4
BUPA Liver Disorder	345	2	6
Waveform	5000	3	21
Wine	178	3	13

Although the number of data sets used in our experiments is relatively small, these data sets were chosen to have one or more of the following desirable characteristics:

- Real, noisy data.
- Medium to large size to avoid issues concerning the statistical validity of results due to small sample sizes.
- Problems from different domains such as medical diagnosis, social science and image processing.
- Dimensionality of problems over a broad range.
- Number of classes of problems over a broad range.

- Non-trivial classification tasks.

5.4 Sampling

The problem of partitioning the data into training and testing sets suitable for classification has interested researchers for some time. The aim is to choose representative sets which give the particular learning method every opportunity to produce an effective classifier, based on the training set, and an accurate estimate of how well the classifier will generalise to unseen data, based on the performance on the test set. Simply partitioning the data randomly into two sets runs the risk of selecting non-representative sets. One method in particular has been established as a popular and effective selection method, namely *cross–validation* [97].

Cross-validation (CV) involves splitting the data into k equal or nearly equal sized sets. One of the k sets is taken as the test set, and the remaining k-1 sets are combined to form the training set. This is repeated, taking each of the k sets in turn as the test set. The learning algorithm is trained and tested on each of the combinations. Choosing the value of k can be done by guesswork or experiment. In our experiments, we randomly partition the data into ten folds, using each as a testing set, while the remaining nine folds form the training set.

5.5 The Genetic Program

In this section we present the technical issues related to the GP system used in the experiments, including the parameter settings, the initial population and the four fitness measures.

5.5.1 The Parameter Settings

The genetic program used in our experiments is designed to construct real-valued attributes from the original (assumed) real-valued attributes of the data set. Thus the terminal set consists of all the original attributes plus the constant 1, whilst the function set consists of the arithmetic operators $+, -, \times, /$. We use a GP system, GPSys[1], written in Java and developed at the University College of London by M. A. Qureshi [81].

The initial population is created using a ramped half-and-half method, and the size is fixed at 600. The GP was run for 100 iterations.

The selection method used is tournament, with a tournament size of 7. Mutation and crossover are fairly standard, with mutation replacing nodes with like nodes, and crossover swapping subtrees. The mutation rate is 50%, whilst the crossover rate is 50%. The fitness of an evolved attribute is measured as it would be at a branch node of the decision tree, whether IG, GI, IG+GI or the Chi^2 is used, see Subsection 5.5.3.

Finally, Otero et al. [75] showed that limiting the size of the tree, and hence the complexity of the constructed attribute, made little difference to the results. In [72, 73, 71], we limit the size of the constructed trees, choosing an upper limit of 30 or 40 nodes, depending on the number of features of the data set.

5.5.2 The Initial Population

The evolutionary process starting point is the random generation of the initial population. It is therefore crucial that the population be initialised with care, since biases

[1]http://www.cs.ucl.ac.uk/staff/A.Qureshi/gpsys.html

introduced in the initial population will inevitably bias the search algorithm. To obtain a wide range of shapes and sizes in the new trees, the ramped half-and-half generation method is used [55]. This method generates equal numbers of solutions with depths ramping from 2 up to a user-specified maximum initial depth, in this case 17. Half of the individuals are created using the full method, and half using the grow method. The full method ensures that all terminals occur at the maximum depth of the tree (ie: all nodes at less than the maximum depth are functions), and the grow method allows functions or terminals at any point in the tree. The algorithm is described in more detail in Chapter 4.

5.5.3 The Fitness Measures

For the fitness measures, we use the splitting criteria of the decision tree classifiers used in the experiments. The splitting criteria are the IG, the GI and the Chi^2 test. We also evolve features which have better IG + GI combination.

Since we only deal with numerical attributes, we use Quinlan's method for splitting the values of the test attribute [80]. This method is also used by Breiman [16] where the aim is to try to find appropriate thresholds against which to compare the values of the test attribute. Assuming the number of values of a given test attribute is m, the values of this attribute are first sorted in ascending order, $\{v_1, v_2, ..., v_m\}$. A threshold value lying between v_i and v_{i+1}, calculated as $\frac{v_i+v_{i+1}}{2}$, has the same effect of dividing the data set into cases with values lying in $\{v_1, v_2, ..., v_i\}$ and those with values lying in $\{v_{i+1}, v_{i+2}, ..., v_m\}$. Therefore, we can examine $m-1$ possible splits on the test attribute. This method is used for splitting the constructed attribute for all fitness measures in our experiments.

The following Subsections show the fitness measures used in our experiments. For a deeper description on the decision trees that incorporate these measures as their splitting criteria, the reader can revisit Chapter 3.

5.5.3.1 Information Gain (IG)

The first set of experiments use IG, used in C4.5 and C5, as the fitness function of the GP system. Given a set S of cases with c classes, $Entropy(S)$ measures the average information needed to identify the class of a case in S

$$Entropy(S) = -\sum_{j=1}^{c} p_j \times \log_2 p_j,$$

where p_j is the proportion of cases in S belonging to class j.

$Entropy_A(S)$ measures the average information needed to identify the class of a case in S when S is partitioned using attribute A

$$Entropy_A(S) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} \times Entropy(S_i)$$

where n is the number of partitions caused by attribute A, and S_i is the subset of cases in S belonging to partition i.

The expected reduction in entropy caused by partitioning the examples in accordance with attribute A is

$$IG(A) = Entropy(S) - Entropy_A(S)$$

By using IG as the fitness function, we aim at evolving an attribute, EA, which maximises the expected reduction in entropy, caused by using this attribute to partition the cases, i.e. minimising $Entropy_{EA}(S)$, and hence, maximising $IG(EA)$. Thus

$$fitness(EA) = Entropy_{EA}(S)$$

5.5.3.2 The Gini Index (GI)

The GI, used in CART, is the second fitness measure used in the experiments. The GI is referred to as an impurity measure [16]. A set of examples is said to be pure if all its records belong to one class. GI prefers splits that put the largest class in one pure node. For example, given a data set S with c classes, then the GI is defined by

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2$$

where p_i is the relative frequency of class i in S.

If a test attribute A splits S into two subsets S_1 and S_2 with sizes N_1 and N_2 respectively, then

$$Gini_A(S) = \frac{N_1}{N} \times Gini(S_1) + \frac{N_2}{N} \times Gini(S_2),$$

where N is the number of samples in the node.

The attribute that provides the highest $Gini_A(S)$ implies the least interesting information, whereas the attribute that gives the smallest $Gini_A(S)$ implies the most interesting information and is therefore chosen to split the node. When all records belong to one class, then $Gini_A(S) = 0$. On the other hand, when records are equally distributed among all classes, then $Gini_A(S) = 0.5$ (for a two class problem), and this number increases closer to 1 as the number of classes increase.

By using GI as the fitness function, we aim at evolving an attribute EA which minimises the GI.

$$fitness(EA) = Gini_{EA}(S)$$

5.5.3.3 Information gain + Gini Index (IG+GI)

The fitness measure used in the third set of experiments is the sum of the IG and GI. The concept behind using this measure is to construct new attributes with combined enhancement of IG and GI. Given a set S, when constructing a new attribute (EA), the aim is to minimise both the GI, $gini_{EA}(S)$, and the entropy of EA, $Entropy_{EA}(S)$. Thus

$$fitness(EA) = Entropy_{EA}(S) + Gini_{EA}(S) + (|Entropy_{EA}(S) - Gini_{EA}(S)| \times 0.5)$$

We use $|Entropy_{EA}(S) - Gini_{EA}(S)| \times 0.5$ to prevent any biases towards any of the measures. In other words, we penalise attributes that have a bigger gap between the two measures. If the fitness of an evolved attribute EA is just $Entropy_{EA}(S) + Gini_{EA}(S)$, then we could have a fit attribute with much better IG than GI, or vice versa.

The above formula has some deficiencies as the range of $Entropy_{EA}(S)$ is [0,1], whereas the range of $Gini_{EA}(S)$ varies, depending on the number of classes in the data. Nevertheless, the constructed features using this fitness function provided predictive power comparable to other approaches, see Chapter 6.

To overcome the difference in range between the possible values of the *Entropy* and the *Gini*, the fitness can be measured as

$$\begin{aligned} fitness(EA) = Entropy_{EA}(S) + Norm(Gini_{EA}(S)) + \\ (|Entropy_{EA}(S) - Norm(Gini_{EA}(S))| \times 0.5) \end{aligned}$$

where

$$Norm(Gini_{EA}(S)) = \frac{Gini_{EA}(S)}{Max((Gini_{EA}(S))}$$

The normalised GI, $Norm(Gini_{EA}(S))$, ensures that the value of the *Gini* is in the range [0,1], which is the same range of the *Entropy*. This is done by dividing $Gini_{EA}(S)$ by the maximum (worst) GI, $Max(Gini_{EA}(S))$, which occurs when all cases in a single node are equally distributed among all classes.

5.5.3.4 The Chi^2 Test

The fourth fitness measure used is the Chi^2 test, used in CHAID. When calculating the Chi^2 test, we use a contingency table which cross-tabulates the test attribute against the class attribute. If the continency table consists of x rows and y columns, where o_{ij} is the observed frequency of the i^{th} row in the j^{th} column, and e_{ij} is the expected frequency of the i^{th} row in the j^{th} column, such that

$$e_{ij} = \frac{RowTotal \times ColumnTotal}{GrandTotal},$$

then the Chi^2 test is

$$Chi^2 = \sum_{j=1}^{x}\sum_{i=1}^{y}\frac{o_{ij} - e_{ij}}{e_{ij}}.$$

If $Chi^2 > Chi^2_\alpha$ with $v = (x-1)(y-1)$ degrees of freedom, then we reject the null hypothesis that the variables are independent at the $\alpha = 0.05$ level of significance. This test forms the basis of CHAID's method of grouping categories and branching on predictors. Thus when using the Chi^2 test as the fitness function, the aim is to maximise the Chi^2 of the evolved attribute.

$$fitness(EA) = Chi^2(EA)$$

5.6 Experimental Methodology

The main aim of this work is to improve decision tree classification and to ascertain if classification using a decision tree is biased in any way by the inclusion of a constructed attribute which has been evolved by a GP whose fitness function is based on the splitting criteria of the decision tree. For the fitness function of the GP therefore, we use one of four measures, namely GI (the basis of the splitting criterion used in C5) and GI (the splitting criterion used in CART), IG+GI and Chi^2 (the splitting criterion used in CHAID).

The attribute construction stage is a pre-processing stage prior to the classification stage, see Figure 10. For the classification, we use four algorithms as in [72, 73, 71]:

1. C5, a descendant of C4.5, a decision tree algorithm whose splitting criterion is based on *Information Gain Ratio*, see [80];
2. CART, a decision tree algorithm whose splitting criterion is based on the *Gini Index*; see [16];
3. CHAID, a decision tree algorithm whose splitting criterion is based on achieving a threshold level of significance in a Chi^2 test, see [51].
4. MLP ANN. an ANN algorithm in which hidden layers use inner products to combine values coming from preceding layers [11].

For each data set we use 10-fold CV to compute the error rate. Thus the data set is firstly partitioned into 10 subsets. For each trial, 9 of the 10 subsets are used as the training set whilst the remaining set is used as the test set.

For each data set and for each 10-fold trial, the methodology is as follows:

Original Apply each classification algorithm to the training set (original attributes only) and evaluate resulting models on the test set.

Aug-IG Evolve a single, new attribute from the training set using GP with IG as the fitness function; apply each classification algorithm on the augmented training set (original attribute set plus IG-evolved attribute). Evaluate resulting model on the test set.

Aug-GI Evolve a single, new attribute from the training set using GP with GI as the fitness function; apply each classification algorithm on the augmented training set (original attribute set plus GI-evolved attribute). Evaluate resulting model on the test set.

Aug-IG+GI Evolve a single, new attribute from the training set using GP with IG plus GI as the fitness function; apply each classification algorithm on the augmented training set (original attribute set plus IG+GI-evolved attribute). Evaluate resulting model on the test set.

Aug-Chi^2 Evolve a single, new attribute from the training set using GP with Chi^2 as the fitness function; apply each classification algorithm on the augmented training set (original attribute set plus Chi^2-evolved attribute). Evaluate resulting model on the test set.

For both the classification using original attributes and those using the augmented sets, we determine the average error rate over the 10 trials for each algorithm and for each data set. These are referred to respectively as *Original*, *Aug-IG*, *Aug-GI*, *Aug-IG+GI* and *Aug-Chi^2* in the results tables in Chapter 6.

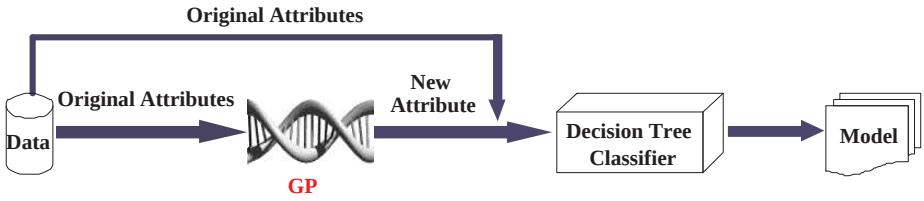


Figure 10: Attribute construction using GP.

The attribute construction is a pre-processing technique. Thus, in the current work, for each data set, the GP was run 40 times (10 folds $\times$ 4 fitness measures), making the total number of GP runs equal to 200 (5 data sets $\times$ 40). 50 classification models were created for each data set using each classifier (10 trials $\times$ (4 augmented + 1 original)), making the total number of models for the 4 classifiers on the 5 data sets equal to 1000 (5 sets $\times$ 4 classifiers $\times$ 50 models). Note also that, for each trial, a different attribute may be evolved by the GP, even when the same fitness function is used.

The GP running time was varied, and was dependent on the size of the data rather than the fitness measure. Hence, Table 4 shows the average time for a single GP run on each trial in (hh:mm:ss) format. The standard deviations show that running time of the GP was consistent on all ten-fold trials.

Table 4: Average time for a single GP run.

Data set	Average Time
Abalone	$25:26:42 \pm 0.11$
Balance	$00:31:24 \pm 0.00$
Bupa	$00:34:41 \pm 0.00$
Waveform	$45:38:26 \pm 0.03$
Wine	$00:46:58 \pm 0.00$

5.7 The Evolved Attribute

In this Section we outline the properties and nature of the evolved features. The fitness of the evolved features is a significant factor for the goodness of these features. However, if the GP evolves very complex features, then the understandability of the feature and hence the classification model becomes harder. The GP always evolves "fitter" features than the primitive ones, and this fitness ranges from marginal to significant improvement. Similarly, the complexity of the evolved features (features and operators) can be as simple as 3 or as complex as 39 (the maximum limit on feature complexity), depending on the dimensionality of the feature space (the number of original attributes). The evolved features are *symbolic expressions*, similar to those used in *LISP* [55]. Table 5 shows some examples of the features evolved using IG as the fitness measure. Figures 11 to 15 show the parse tree representation of the features listed in Table 5.

Table 5: Symbolic expressions of some of the features evolved using IG.

Data Set	Symbolic Expression of the Evolved Feature	Complexity
Abalone	(- (- (- f6 (* (* f1 (* f8 (/ f5 (/ f3 f5)))) (/ (+ f7 f2) (/ (+ (/ f1 f8) f8) f5)))) f4) (+ f7 f8))	29
Balance-Scale	(/ (* f3 f4) (* f1 f2))	7
Bupa-Liver Disorder	(/ (/ (- f2 (/ f2 f3)) (+ f5 (/ ((+ f2 f6) f3))) (/ f4 f3))	17
Waveform	(- (* f6 f8) (- (+ (* (- f15 f5) f10) (+ (* (- f7 (- (- f7 1) 1)) (+ (+ f13 f14) f12)) (+ (+ f11 f12) (- f16 (+ f7 (* f6 f7)))))) f7))	37
Wine	(* f7 (/ f11 f10))	5

In Tables 6 to 10, we show the fitness of the evolved attributes listed in Table 5. In each Table, and for each splitting measure, the attributes are listed in descending

or ascending order according to the values of the splitting measure, i.e. fitness. That is, the fitter attributes appear in the higher of the table.

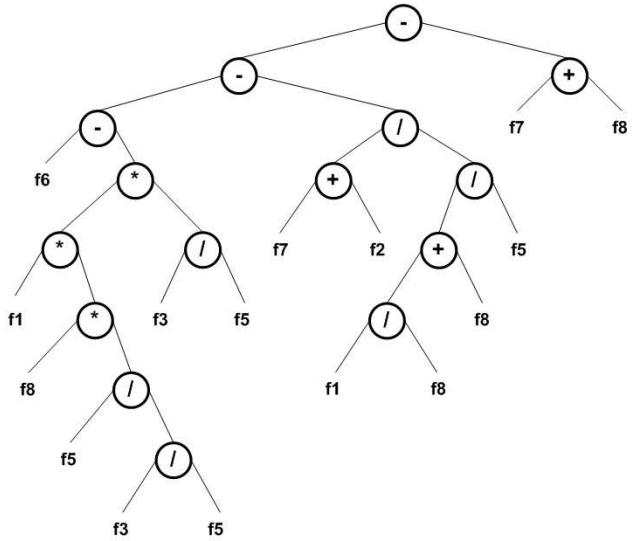


Figure 11: Parse tree of the evolved attribute using IG for the Abalone data set.

Table 6: Fitness of the evolved attributes in the Abalone data set.

IG			GI			Chi^2	
EA	**0.415**		*EA*	**0.858**		f8	**2183.755**
f8	0.362		f8	0.862		f3	2123.587
f4	0.328		f7	0.865		f2	2108.73
f7	0.324		f4	0.865		f5	2072.338
f5	0.324		f3	0.866		f7	2071.770
f3	0.323		f5	0.866		f6	2038.116
f2	0.314		f2	0.867		*EA*	1874.1821
f6	0.276		f6	0.870		f4	1837.006
f1	0.226		f1	0.873		f1	1152.333

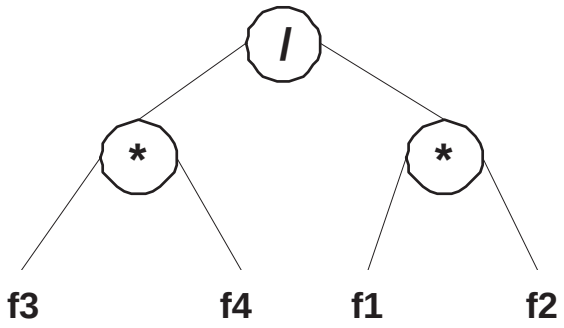


Figure 12: Parse tree of the evolved attribute using IG for the Balance-scale data set.

Table 7: Fitness of the evolved attributes in the Balance-scale data set.

IG			GI			Chi^2	
EA	**0.999**		*EA*	**0.133**		*EA*	**563.0**
f4	0.108		f4	0.502		f4	81.545
f1	0.108		f1	0.503		f1	80.944
f2	0.107		f2	0.503		f2	80.407
f3	0.099		f3	0.508		f3	74.804

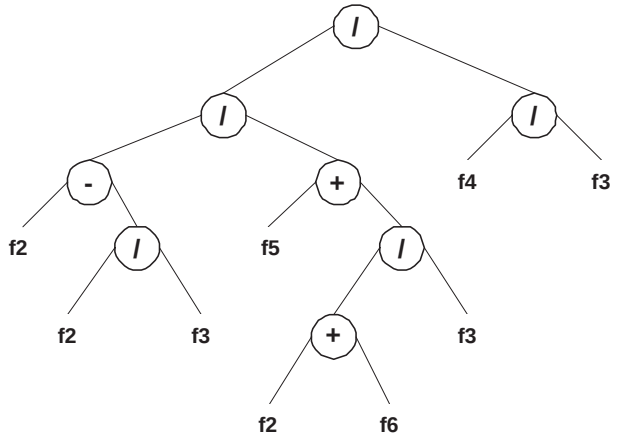


Figure 13: Parse tree of the evolved attribute using IG for the Bupa data set.

Table 8: Fitness of the evolved attributes in the Bupa data set.

IG			GI			Chi^2	
EA	**0.198**		*EA*	**0.358**		*EA*	**82.084**
f5	0.048		f5	0.454		f5	20.494
f3	0.027		f3	0.469		f3	11.404
f2	0.027		f2	0.469		f2	11.301
f4	0.021		f4	0.475		f4	7.238
f1	0.015		f1	0.477		f1	6.349
f6	0.011		f6	0.479		f6	4.605

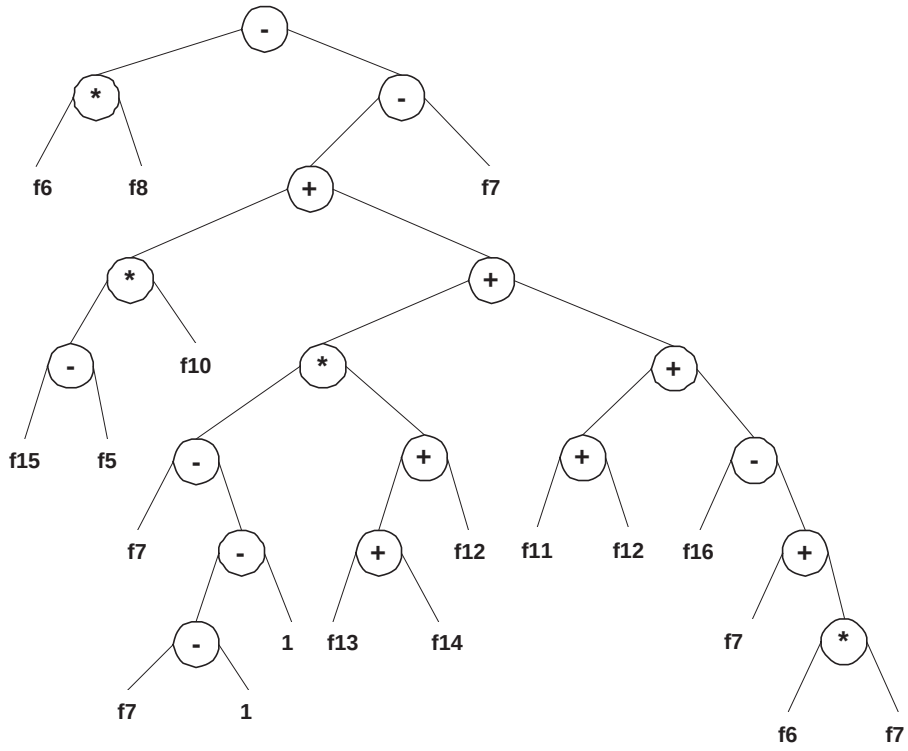


Figure 14: Parse tree of the evolved attribute using IG for the Wave data set.

Table 9: Fitness of the evolved attributes in the Wave data set.

IG			GI			Chi^2	
EA	**0.525**		*EA*	**0.456**		*EA*	**2814.706**
f7	0.296		f7	0.545		f7	1638.569
f15	0.287		f6	0.549		f15	1596.973
f6	0.2794		f15	0.549		f6	1575.146
f16	0.249		f5	0.562		f16	1405.492
f13	0.249		f13	0.563		f8	1403.545
f14	0.248		f8	0.563		f5	1390.479
f8	0.246		f16	0.564		f14	1388.7810
f5	0.245		f14	0.564		f13	1382.107
f9	0.234		f11	0.569		f11	1312.686
f11	0.232		f17	0.572		f17	1289.709
f17	0.224		f9	0.572		f9	1289.332
f12	0.206		f12	0.577		f12	1190.101
f10	0.198		f10	0.584		f10	1129.214
f18	0.176		f18	0.593		f18	1002.213
f4	0.157		f4	0.599		f4	905.849
f19	0.089		f3	0.628		f19	517.962
f3	0.082		f19	0.629		f3	511.170
f20	0.027		f20	0.654		f20	173.996
f2	0.025		f2	0.655		f2	152.488
f1	0.002		f1	0.666		f1	10.292
f21	0.001		f21	0.666		f21	8.798

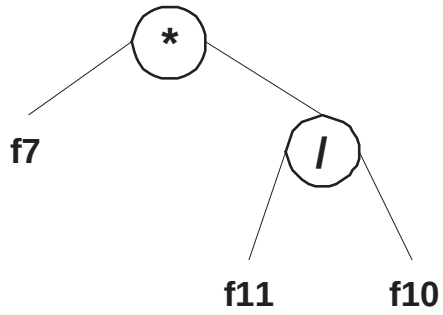


Figure 15: Parse tree of the evolved attribute using IG for the Wine data set.

Table 10: Fitness of the evolved attributes in the Wine data set.

IG			GI			Chi^2	
EA	**0.879**		*EA*	**0.350**		*EA*	**161.0**
f7	0.710		f7	0.414		f7	128.868
f12	0.669		f12	0.423		f12	124.319
f13	0.587		f13	0.425		f13	114.649
f10	0.571		f10	0.428		f11	108.634
f1	0.530		f1	0.448		f10	108.407
f11	0.521		f11	0.453		f1	98.288
f6	0.511		f6	0.488		f6	90.006
f9	0.294		f4	0.545		f2	60.535
f4	0.293		f2	0.546		f9	60.254
f2	0.289		f9	0.548		f4	59.124
f8	0.239		f5	0.569		f8	48.534
f5	0.233		f8	0.570		f5	43.053
f3	0.166		f3	0.594		f3	31.995

For some of the data sets, namely the Balance and the Wine, the evolved features do not only provide more predictive power, they also reveal physical properties about the data. They show important relationships between the predicting attributes and the class attribute. These properties and relationships are believed to reveal physical characteristics about the data. This is discussed further in Chapter 7.

5.8 Summary

In this chapter, we presented the experimental methodology for the experiments in this thesis. The data sets and GP system used are outlined. In Chapter 6 we show the error rates of classification and investigate any bias in the results. We also compare the results with those of other approaches. Chapter 7 presents further analysis of the resultant trees of the classification tree models and the evolved attributes. A comparison study on the use of GP for classification to its use for feature construction is also presented in Chapter 7. We extend the experimental work by applying GP for feature construction on a real–world commercial data.

Chapter 6

Comparing Error Rates

6.1 Introduction

In Chapter 5, we presented the experimental methodology and the GP setup used to construct features on a number of domains. In this chapter, we present the results of classification using four inductive algorithms, namely, C5, CHAID, CART and an MLP ANN, on the five data sets presented in previous chapter. The aim of this work is determine whether the performance of the inductive classifiers improve when a constructed attribute is included in the attribute set. We also check for any biases in the performance of decision tree classifiers towards the attribute sets which include a constructed attribute evolved using a GP system whose fitness function incorporates the same fundamental learning mechanism as the splitting criteria of the associated decision tree.

The performance of a decision tree classification model built using a training set can be measured by a number of methods, one of which is the model's prediction accuracy when applied to a testing set, known as the *error rate*. The error rate is the proportion of all the misclassifications made over the whole set of samples.

In Sections 6.2 to 6.5 we present the error rates for each classifier on all data sets, using original and augmented attribute sets. In Tables 11 to 30, we show and compare the performance of three decision tree classifiers and an MLP ANN classifier on each data sets, all outlined in Chapter 5. The experiments are performed using the original attribute set and the augmented attribute set (the original attributes plus the single evolved attribute)[1]. We present the error rates (for the testing sets) averaged over the 10xCV trials. The meanings of the columns and the rows in the Tables are:

The first column lists the classifiers used in the experiments. The remaining columns are explained as follows:

Original : shows the error rate achieved by the induction technique using the original attribute set. The figure after the ± is the standard deviation of the accuracies over the 10 trials.

Aug-* : the average error rates achieved by the induction technique on the augmented attribute set over the ten fold trials. The figure after the ± is the standard deviation of the error rates over the 10 trials. The * is one of (IG, GI, IG+GI, Chi^2)

Imp : shows the *absolute improvement* in error rate by the induction technique on the augmented attribute set.

R-Imp : shows the % *relative improvement* achieved by the induction technique on the augmented attribute set with regard to the error rate on the original attribute set.

[1]Note that, in each of the 10 runs necessary for 10xCV, a new attribute was evolved from scratch.

T-Test : is a statistical measure used to assess whether the means of two groups are statistically different from each other. The t-value will be positive if the first mean is larger than the second and negative if it is smaller. Once we compute the t-value we have to look it up in a table of significance to test whether the ratio is large enough to say that the difference between the groups is not likely to have been a chance finding. To test the significance, we need to set a risk level (called the alpha level). In most research, the "rule of thumb" is to set the alpha level at 0.05. This means that five times out of a hundred we would find a statistically significant difference between the means even if there was none (i.e., by "chance"). In our experiments, we use the T-Test with the assumption that we have a hypothesis that the original testing error rates are bigger than the augmented testing sets error rates. We compare the testing error rates of the ten folds of the original attribute sets with the testing error rates of the ten folds of the augmented attribute set. If the value of the T-Test is less than or equal to 0.05 then the difference of the two groups is significant (shown in bold font).

In Section 6.6, we investigate the performance of the all classifiers to check for any biases, particularly the performance of decision tree models towards attribute sets which include an additional attribute constructed using a GP whose fitness function incorporates the same splitting mechanism of the associated tree. In Section 6.7, compare the results with theresults of other approaches to feature construction. Section 6.8, summarises this chapter.

6.2 Information Gain (IG)

In this Section, we present the results using IG as the fitness measure of the GP. For the Abalone data set, Table 11 shows that, although all decision tree techniques show an improvement using the augmented attribute set, Aug-IG, the results are barely significant. In fact, it has to be said that all classification algorithms perform rather badly with this data set, even with the evolved attribute. Although the performance of the ANN is slightly better than the other inductive techniques using the original attribute set, its performance deteriorates slightly using Aug-IG.

Table 11: Error rates for the Abalone data set (Fitness: IG).

Abalone	Original	Aug-IG	Imp	R-Imp	T-Test
C5	79.26 ± 1.21	79.09 ± 2.06	0.17	0.21	0.42
CHAID	76.45 ± 1.61	74.77 ± 2.62	1.68	2.2	**0.025**
CART	74.69 ± 1.76	73.02 ± 2.55	1.67	2.24	**0.032**
ANN	72.45 ± 1.93	72.49 ± 2.74	-0.04	-0.06	0.468

On the other hand, when we look at the results for the Balance data set, in Table 12, we see a significant improvement in accuracy when we include the evolved attribute, particularly for C5 and CART, in which the error rate for the test set, as well as the training set, was 0 in all 10 trials. CHAID also achieves an 80.1% relative improvement. Indeed, the absolute accuracies of all the decision tree techniques are improved by almost 23%. The ANN, although having a better performance than the other classifiers with the original attributes, also improves performance with the augmented attribute set, but not so dramatically, and indeed less significantly if we take the standard deviations into account. It is also worth pointing out that, for this data set, not only are the error rates reduced significantly, but the standard deviations are

also reduced showing a more consistent performance, only for the decision tree models.

Table 12: Error rates for the Balance-scale data set (Fitness: IG).

Balance-scale	Original	Aug-IG	Imp	R-Imp	T-Test
C5	22.42 ± 6.20	0.00 ± 0.00	22.42	100	**0.00**
CHAID	28.39 ± 5.17	5.65 ± 2.55	22.74	80.1	**0.00**
CART	22.74 ± 5.01	0.00 ± 0.00	22.74	100	**0.00**
ANN	10.00 ± 3.79	9.36 ± 4.01	0.64	6.4	0.212

Table 13 shows the average error rates for the BUPA data set. Once again, all classifiers have improved their performance with the augmented set. Although it shows worse error rates than all other classifiers on the original attribute set, CHAID manages to achieve the best error reduction with the augmented attributes. Nevertheless, the relatively large standard deviations represent a more inconsistent performance all round, even with the augmented attribute set. The story is much the same with the Waveform data set, see Table 14, with all classifiers showing an improvement with the inclusion of an evolved attribute, although only barely significantly.

Table 13: Error rates for the BUPA data set (Fitness : IG).

BUPA	Original	Aug-IG	Imp	R-Imp	T-Test
C5	36.47 ± 7.99	32.35 ± 6.50	4.12	11.3	0.141
CHAID	40.00 ± 9.73	30.00 ± 6.17	10	25	**0.004**
CART	32.35 ± 7.21	30.29 ± 8.77	2.06	6.37	0.283
ANN	37.65 ± 7.18	35.29 ± 10.47	2.36	6.27	0.262

Finally, turning attention to the Wine data set, although not statistically significant, bearing in mind the standard deviations, the results in Table 15 show that,

Table 14: Error rates for the Waveform data set (Fitness : IG).

Waveform	Original	Aug-IG	Imp	R-Imp	T-Test
C5	22.94 ± 2.23	19.54 ± 1.64	3.4	14.82	**0.002**
CHAID	28.36 ± 1.81	24.92 ± 2.30	3.44	12.13	**0.00**
CART	24.58 ± 2.61	20.02 ± 1.71	4.56	18.55	**0.00**
ANN	17.15 ± 7.41	15.20 ± 2.29	1.95	11.37	0.189

while the performance of C5, CART and ANN has marginally improved with the augmented attribute, the performance of CHAID has deteriorated slightly.

Table 15: Error rates for the Wine data set (Fitness : IG).

Wine	Original	Aug-IG	Imp	R-Imp	T-Test
C5	6.47 ± 7.04	5.29 ± 7.04	1.18	18.24	0.222
CHAID	17.06 ± 8.06	17.65 ± 8.77	-0.59	-3.46	0.363
CART	10.59 ± 9.92	5.88 ± 8.32	4.71	44.48	**0.026**
ANN	3.53 ± 7.44	2.94 ± 5.72	0.59	16.71	0.295

The results shown in Tables 11 to 15 generally prove that attribute construction using GP can generate improvements in the performance of a classifier, sometimes significantly so, as in the case of the Balance-scale data set.

If we measure the absolute improvement in performance for each classifier, averaged out over all the data sets, see Figure 16, then, using the attributes evolved using IG, C5 manages an overall 6.26% improvement in accuracy, CHAID achieves a 7.45% improvement, CART 7.15% and ANN 1.10%. It cannot be concluded from these results, therefore, that C5 has any advantage over the other classifiers with the inclusion of a feature evolved using a GP with information gain as a fitness measure. However, it is obvious that decision tree classifiers benefit more from the inclusion of

a new attribute that has been evolved using a GP with information gain as a fitness function.

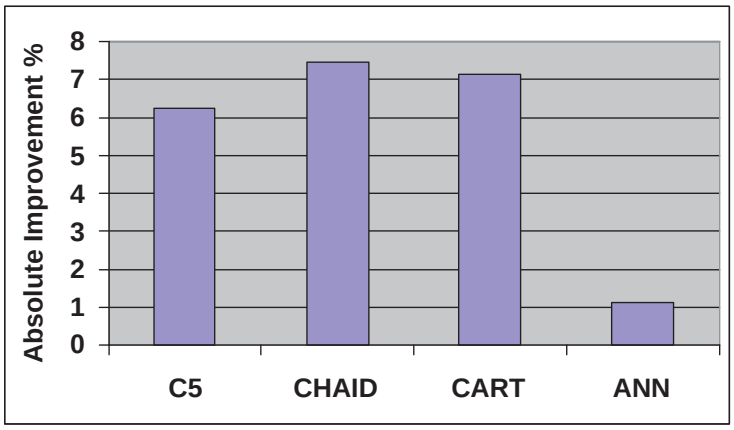


Figure 16: Absolute improvement in performance of each classifier averaged out over all data sets using Aug-IG.

6.3 Gini Index (GI)

In this Section, the GI is used as the fitness measure in the GP. It is clear from Table 16 that the Abalone data set continues to prove difficult. For all classification algorithms, the error rates in the test sets are significant, even with the evolved attribute. Arguably, the Aug-IG errors, see Table 11, are marginally better than the Aug-GI, but the standard deviations make this inconclusive.

The results for the Balance data set, in Table 17, are consistent with those shown in Tables 12 with C5 and CART, achieving 0% error (on all training and test sets) with the inclusion of the evolved attribute. The ANN, although having a better

Table 16: Error rates for the Abalone data set (Fitness : GI).

Abalone	Original	Aug-GI	Imp	R-Imp	T-Test
C5	79.26 ± 1.21	80.07 ± 2.40	-0.81	-1.02	0.182
CHAID	76.45 ± 1.61	75.40 ± 2.34	1.05	1.37	**0.042**
CART	74.69 ± 1.76	73.86 ± 2.43	0.83	1.11	0.17
ANN	72.45 ± 1.93	73.14 ± 2.79	-0.69	-0.95	0.166

performance than the other classifiers with the original attributes, also improves performance with the augmented attribute set, but not so dramatically, and indeed less significantly if we take the standard deviations into account.

Table 17: Error rates for the Balance-scale data set (Fitness : GI).

Balance-scale	Original	Aug-GI	Imp	R-Imp	T-Test
C5	22.42 ± 6.20	0.00 ± 0.00	22.42	100	**0.00**
CHAID	28.39 ± 5.17	5.49 ± 2.30	22.9	80.66	**0.00**
CART	22.74 ± 5.01	0.00 ± 0.00	22.74	100	**0.00**
ANN	10.00 ± 3.79	9.19 ± 3.65	0.81	8.1	0.149

Table 18 shows the average error rates for the BUPA data set. Once again, all classifiers have improved their performance with the addition of the evolved attributes. However, it is notable that CART does not appear to be benefiting more from the addition of the attributes evolved using GI. The story is much the same with the Waveform data set, see Table 19, with all classifiers showing an improvement with the inclusion of an evolved attribute, however, CART achieves a slightly better improvement.

Finally, turning attention to the Wine data set, the results in Table 20 once again indicate the potential for any classification algorithm to benefit from the inclusion

Table 18: Error rates for the BUPA data set (Fitness : GI).

BUPA	Original	Aug-GI	Imp	R-Imp	T-Test
C5	36.47 ± 7.99	32.94 ± 8.75	3.53	9.68	0.122
CHAID	40.00 ± 9.73	32.06 ± 8.14	7.94	19.85	**0.01**
CART	32.35 ± 7.21	31.18 ± 7.87	1.17	3.62	0.303
ANN	37.65 ± 7.18	31.47 ± 8.66	6.18	16.41	**0.034**

Table 19: Error rates for the Waveform data set (Fitness : GI).

Waveform	Original	Aug-GI	Imp	R-Imp	T-Test
C5	22.94 ± 2.23	19.64 ± 2.37	3.3	14.39	**0.003**
CHAID	28.36 ± 1.81	24.64 ± 2.12	3.72	13.12	**0.001**
CART	24.58 ± 2.61	19.54 ± 1.76	5.04	20.5	**0.00**
ANN	17.15 ± 7.41	15.77 ± 2.28	1.38	8.05	0.274

of evolved, highly predictive variables. The ANN, despite achieving around 96.5% accuracy with the original attribute vector, still manages to improve its performance using the augmented attribute set, attaining 100% accuracy over a number of the 10-fold test sets. However, the *Aug-GI* set, in Table 20, appears to be giving marginally better results than the *Aug-IG* set (Table 15), but the standard deviations are too large to make this significant.

Table 20: Error rates for the Wine data set (Fitness: GI).

Wine	Original	Aug-GI	Imp	R-Imp	T-Test
C5	6.47 ± 7.04	4.12 ± 4.84	2.35	36.32	0.155
CHAID	17.06 ± 8.06	15.29 ± 7.94	1.77	10.38	0.139
CART	10.59 ± 9.92	3.53 ± 3.04	7.06	66.67	**0.026**
ANN	3.53 ± 7.44	1.76 ± 2.84	1.77	50.14	0.197

The results shown in Tables 16 to 20 generally prove, again, that attribute construction using GP can generate improvements in the performance of a classifier, sometimes significantly so, as in the case of the Balance-scale data set.

However, we can claim here that the improvement in performance, if any, of the classifiers is not significantly dependent on which measure is used in the GP to evolve the attributes. If we measure the absolute improvement in performance for each classifier, averaged out over all the data sets, see Figure 17, then, using the attributes evolved using the GI, C5 achieves a 6.16% improvement, CHAID achieves a 7.48% improvement, CART 7.37% and ANN 1.89%. It cannot be concluded from these results, therefore, that C5 (resp. CART) has an advantage over the other classifiers with the inclusion of a feature evolved using a GP with IG (resp. GI) as a fitness measure. Nevertheless, it is still obvious that decision tree classifiers benefit more from the inclusion of a new attribute that has been evolved using a GP with information gain or gini index as fitness functions.

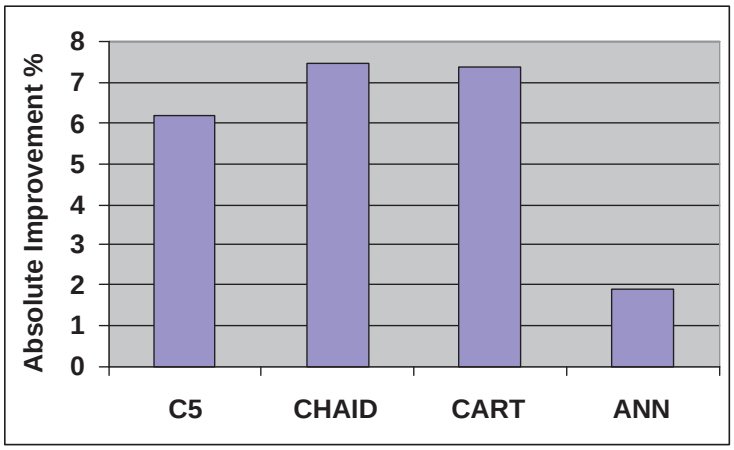


Figure 17: Absolute improvement in performance of each classifier averaged out over all data sets using Aug-GI.

6.4 IG + GI

In this Section, the sum of IG and GI is used as the fitness measure in the GP. Once again, the results shown in Table 21, for the Abalone data set, are consistent with those in Tables 11 and 16. It shows the highest deterioration, so far, of the ANN's performance using the augmented attribute set, Aug-IG+GI.

Table 21: Error rates for the Abalone data set (Fitness : IG+GI).

Abalone	Original	Aug-IG+GI	Imp	R-Imp	T-Test
C5	79.26 ± 1.21	79.28 ± 2.4	-0.02	-0.06	0.485
CHAID	76.45 ± 1.61	74.61 ± 1.88	1.84	2.41	**0.005**
CART	74.69 ± 1.76	73.76 ± 1.85	0.93	1.26	**0.05**
ANN	72.45 ± 1.93	73.64 ± 1.99	-1.19	-1.64	**0.03**

The results for the Balance data set, in Table 22, are consistent with those shown in Tables 12 and 17 for C5 and CART, achieving 0% error (on all training and test sets) with the inclusion of the evolved attribute. Furthermore, the results of CHAID and ANN show smaller error rates than those in Tables 12 and 17.

Table 22: Error rates for the Balance-scale data set (Fitness : IG+GI).

Balance-scale	Original	Aug-IG+GI	Imp	R-Imp	T-Test
C5	22.42 ± 6.20	0.00 ± 0.00	22.42	100	**0.00**
CHAID	28.39 ± 5.17	5.16 ± 2.92	23.23	81.82	**0.00**
CART	22.74 ± 5.01	0.00 ± 0.00	22.74	100	**0.00**
ANN	10.00 ± 3.79	7.42 ± 4.57	2.58	25.8	0.06

Table 23 shows the average error rates for the BUPA data set. Although all classifiers have improved their performance with the addition of the evolved attributes,

decision tree classifiers appear to benefit the most, particularly CHAID. It is therefore clear that Aug-IG+GI is superior to Aug-IG and Aug-GI on this data set. However, these results are not statistically significant if standard deviations are taken into account.

Table 23: Error rates for the BUPA data set (Fitness : IG+GI).

BUPA	Original	Aug-IG+GI	Imp	R-Imp	T-Test
C5	36.47 ± 7.99	26.47 ± 10.65	10	27.42	**0.022**
CHAID	40.00 ± 9.73	27.94 ± 6.83	12.06	30.15	**0.006**
CART	32.35 ± 7.21	27.65 ± 8.34	4.7	14.53	0.084
ANN	37.65 ± 7.18	31.18 ± 8.46	6.47	17.18	**0.025**

The situation is different with the Waveform data set, see Table 24, with all classifiers showing an improvement with the inclusion of an evolved attribute. Decision tree error rates do not improve as much as those in Tables 14 and 19, however, although the ANN achieves a better error rate.

Table 24: Error rates for the Waveform data set (Fitness : IG+GI).

Waveform	Original	Aug-IG+GI	Imp	R-Imp	T-Test
C5	22.94 ± 2.23	20.22 ± 1.77	2.72	11.86	**0.00**
CHAID	28.36 ± 1.81	25.3 ± 1.92	3.06	10.79	**0.00**
CART	24.58 ± 2.61	20.04 ± 1.41	4.54	18.47	**0.00**
ANN	17.15 ± 7.41	14.4 ± 1.53	2.75	16.03	0.126

With the Wine data set, see Table 25, while other classification algorithms indicate potential improvement in their performance, C5 deteriorates with the inclusion of an evolved attribute. On the other hand, CHAID achieves additional enhancement.

Table 25: Error rates for the Wine data set (Fitness : IG+GI).

Wine	Original	Aug-IG+GI	Imp	R-Imp	T-Test
C5	6.47 ± 7.04	7.06 ± 7.75	-0.59	-9.12	0.411
CHAID	17.06 ± 8.06	14.12 ± 7.44	2.94	17.23	0.069
CART	10.59 ± 9.92	7.77 ± 9.92	2.82	26.63	0.170
ANN	3.53 ± 7.44	2.35 ± 5.68	1.18	33.43	0.084

The results shown in Tables 21 to 25 generally prove, again, that attribute construction using GP can generate improvements in the performance of a classifier, sometimes significantly so, as in the case of the Balance-scale data set.

However, we can claim here that the improvement in performance, if any, of the classifiers is not significantly dependent on which measure is used in the GP to evolve the attributes. If we measure the absolute improvement in performance for each classifier, averaged out over all the data sets, see Figure 18, then, using the attributes evolved using the IG and GI, C5 achieves a 6.91% improvement, CHAID achieves a 8.63% improvement, CART 7.15% and ANN 2.36%. It can be concluded from these results, therefore, that decision tree classifiers have an advantage over the other classifiers with the inclusion of a feature evolved using a GP with IG + GI as a fitness measure.

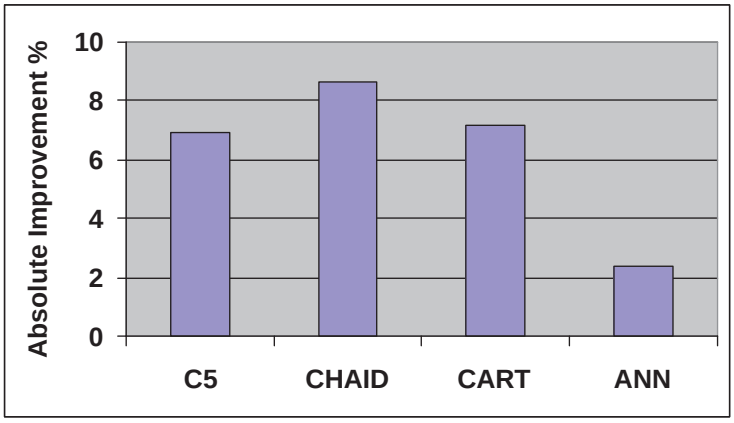


Figure 18: Absolute improvement in performance of each classifier averaged out over all data sets using Aug-IG+GI.

6.5 Chi^2

In this Section, the Chi^2 test is used as the fitness measure in the GP. The results in Table 26 confirm that the ANN does not gain any improvement on the Abalone data set, using Aug-Chi^2. Indeed this result is in keeping with those of Tables 11, 16 and 21, as they all show deterioration in the ANN's performance on the augmented Abalone data set.

Table 26: Error rates for the Abalone data set (Fitness : chi^2).

Abalone	Original	Aug-Chi^2	Imp	R-Imp	T-Test
C5	79.26 ± 1.21	79.26 ± 1.16	0	0	0.499
CHAID	76.45 ± 1.61	75.33 ± 0.99	1.12	1.47	**0.021**
CART	74.69 ± 1.76	74.56 ± 1.87	0.13	0.17	0.353
ANN	72.45 ± 1.93	72.88 ± 3.08	-0.43	-0.59	0.289

The results for the Balance data set, in Table 27, are consistent with those shown

in Tables 12 and 17 for C5 and CART, achieving 0% error (on all training and test sets) with the Aug-Chi^2. An interesting observation is the result of CHAID, where it shows the best error rate so far, when compared to those in Tables 12, 17 and 22. The ANN, on the other hand, shows the least improvement of all results.

Table 27: Error rates for the Balance-scale data set (Fitness : chi^2).

Balance-scale	Original	Aug-Chi^2	Imp	R-Imp	T-Test
C5	22.42 ± 6.20	0.00 ± 0.00	22.42	100	**0.00**
CHAID	28.39 ± 5.17	4.19 ± 2.04	24.2	85.24	**0.00**
CART	22.74 ± 5.01	0.00 ± 0.00	22.74	100	**0.00**
ANN	10.00 ± 3.79	9.68 ± 4.93	0.32	3.2	0.352

As for the Bupa data set, see Table 28, all errors rates improve, however, the results of CHAID, which uses Chi^2 as its splitting criterion, show less improvement compared with the results in Tables 13, 18, and 23.

Table 28: Error rates for the BUPA data set (Fitness : chi^2).

BUPA	Original	Aug-Chi^2	Imp	R-Imp	T-Test
C5	36.47 ± 7.99	26.77 ± 7.26	9.7	26.6	**0.007**
CHAID	40.00 ± 9.73	33.24 ± 6.05	6.76	16.9	**0.035**
CART	32.35 ± 7.21	29.71 ± 7.52	2.64	8.16	0.169
ANN	37.65 ± 7.18	32.06 ± 7.9	5.59	14.85	**0.031**

The results for the Waveform data set, see Table 29, show similar improvement rates on all decision tree classifiers, ranging between 3.72% to 5.1%. The ANN, however, while it shows better error rates on both the original and the augmented attribute sets, the rate of improvement is very marginal.

Table 29: Error rates for the Waveform data set (Fitness : chi^2).

Waveform	Original	Aug-Chi^2	Imp	R-Imp	T-Test
C5	22.94 ± 2.23	19.22 ± 2.03	3.72	16.22	**0.00**
CHAID	28.36 ± 1.81	24.36 ± 2.79	4	14.1	**0.00**
CART	24.58 ± 2.61	19.48 ± 2.46	5.1	20.75	**0.00**
ANN	17.15 ± 7.41	15.44 ± 2.03	1.71	9.97	0.256

Finally, if we look at the Wine data set, see Table 30, we notice the results of C5 do not improve at all from the inclusion of evolved attributes. On the other hand, other classifiers improve but only marginally.

Table 30: Error rates for the Wine data set (Fitness : chi^2).

Wine	Original	Aug-Chi^2	Imp	R-Imp	T-Test
C5	6.47 ± 7.04	6.47 ± 8.06	0	0	0.5
CHAID	17.06 ± 8.06	14.12 ± 9.28	2.94	17.23	0.122
CART	10.59 ± 9.92	5.88 ± 8.32	4.71	44.48	**0.035**
ANN	3.53 ± 7.44	2.35 ± 4.11	1.18	33.43	0.172

The results shown in Tables 26 to 30 generally prove, again, that attribute construction using GP can generate improvements in the performance of a classifier, sometimes significantly so, as in the case of the Balance-scale data set.

However, we can claim here that the improvement in performance, if any, of the classifiers is not significantly dependent on which measure is used in the GP to evolve the attributes. If we measure the absolute improvement in performance for each classifier, averaged out over all the data sets, see Figure 19 then, using the attributes evolved using the Chi^2, C5 achieves a 7.17% improvement, CHAID achieves a 7.80% improvement, CART 7.06% and ANN 1.67%. It can be concluded from these results,

therefore, that neither C5 nor CART nor CHAID have an advantage over the other classifiers with the inclusion of a feature evolved using a GP with IG, GI or Chi^2, respectively, as a fitness measure. Nevertheless, it is still obvious that decision tree classifiers benefit more from the inclusion of a new attribute that has been evolved using a GP with IG, GI or Chi^2 as the fitness measure.

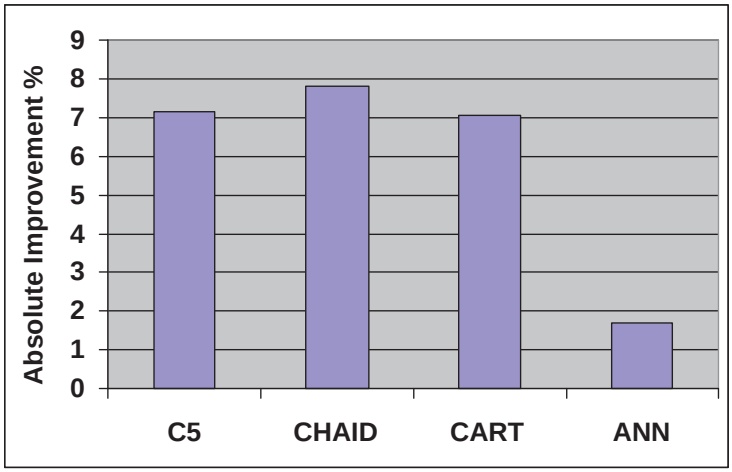


Figure 19: Absolute improvement in performance of each classifier averaged out over all data sets using Aug-Chi^2.

The results presented in Sections 6.2 to 6.5 to show that all classifiers in general, and decision tree classifiers in particular improve the classification accuracy using the augmented attribute sets. This conclusion is proven to be statistically valid as shown by the values of the T-Tests.

6.6 Bias Check

The above Sections presented the error rates of the classification algorithms on five data sets. We discussed the performance of these classifiers on the original and the augmented attribute sets. The augmented attribute set contains the original attributes and one feature constructed using GP.

In this Section, we compare the error rates of each classifier using the original and the augmented attribute sets to examine whether there are biases in performance when the augmented attribute set includes an attribute constructed using a GP whose fitness function incorporates the same fundamental learning mechanism as the associated decision tree.

Table 31 shows the average error rates of C5 on the five data sets using the original attribute sets and all other augmented attribute sets. The results show no indication that C5 benefits more using Aug-IG, except for the Abalone and the Balance data sets. However, using the Bupa data set, C5 shows the lowest error rate using Aug-IG+GI. Nevertheless, for each data set, the performance of C5 is very similar on all augmented attribute sets.

Table 31: Error rates of C5.

C5	Original	Aug-IG	Aug-GI	Aug-IG+GI	Aug-Chi^2
Abalone	79.26	**79.09**	80.07	79.28	79.26
Balance	22.42	**0.00**	**0.00**	**0.00**	**0.00**
Bupa	36.47	32.35	32.94	**26.47**	26.77
Waveform	22.94	19.54	19.64	20.22	**19.22**
Wine	6.47	5.29	**4.12**	7.057	6.471

Table 32 shows the average error rates of CHAID on the 5 data sets. Although, once again, the error rates are very similar using the augmented attribute sets, Table 32 shows that CHAID performs best using Aug-IG+GI and Aug-Chi^2. Therefore, the results show no clear bias towards Aug-Chi^2. Recall that Chi^2 is the basis of the splitting measure for CHAID.

Table 32: Error rates of CHAID.

CHAID	Original	Aug-IG	Aug-GI	Aug-IG+GI	Aug-Chi^2
Abalone	76.45	74.77	75.4	**74.61**	75.32
Balance	28.39	5.65	5.49	5.16	**4.19**
Bupa	40	30	32.06	**27.94**	33.24
Waveform	28.36	24.92	24.64	25.3	**24.36**
Wine	17.06	17.65	15.29	**14.12**	**14.12**

The error rates of CART, shown in Table 33, tell a similar story to that of C5. The results show no indication that CART benefits more using Aug-GI, except for the Wine and the Balance data sets. Once again, on the Bupa data set, CART shows the lowest error rate using Aug-IG+GI. Thus, it can be concluded that the CART performance was not biased towards Aug-GI.

Table 33: Error rates of CART.

CART	Original	Aug-IG	Aug-GI	Aug-IG+GI	Aug-Chi^2
Abalone	74.69	**73.02**	73.86	73.76	74.56
Balance	22.257	**0.00**	**0.00**	**0.00**	**0.00**
Bupa	32.35	30.29	31.18	**27.65**	29.71
Waveform	24.58	20.02	19.54	20.04	**19.48**
Wine	10.59	5.88	**3.53**	7.77	5.88

If we compare the error rates of the ANN, see Table 34, with those in Tables 31

to 33, we will see that the ANN has the smallest error rate baseline than the other classifiers using the original attributes, except for the Bupa data set. Although there is not significant improvement in the error rates using the augmented attribute sets, the ANN seems to perform best using Aug-IG+GI.

Table 34: Error rates of ANN.

ANN	Original	Aug-IG	Aug-GI	Aug-IG+GI	Aug-Chi^2
Abalone	**72.45**	72.49	73.14	73.64	72.88
Balance	10.00	9.36	9.19	**7.42**	9.68
Bupa	37.65	35.29	31.47	**31.18**	32.06
Waveform	17.15	15.2	15.77	**14.4**	15.44
Wine	3.53	2.94	**1.76**	2.35	2.35

By measuring the absolute error rate of each classifier over all data sets, we can conclude that, while all classifiers benefit from the inclusion of the constructed attribute, none show any clear bias towards any of the augmented attribute sets, see Table 35. This is also clearly shown in Figure 20.

Table 35: Absolute performance over all data sets.

	Original	Aug-IG	Aug-GI	Aug-IG+GI	Aug-Chi^2
C5	33.51	27.25	27.35	26.61	**26.34**
CHAID	38.05	30.6	30.58	**29.43**	30.25
CART	32.99	25.84	**25.62**	25.84	25.93
ANN	28.16	27.6	26.27	**25.8**	26.48

Figure 21 shows the absolute improvement in error rates of all classifiers using the augmented attribute sets. It is clear that all decision tree classifiers show higher improvement than the ANN. However, we should not forget that, initially, the ANN

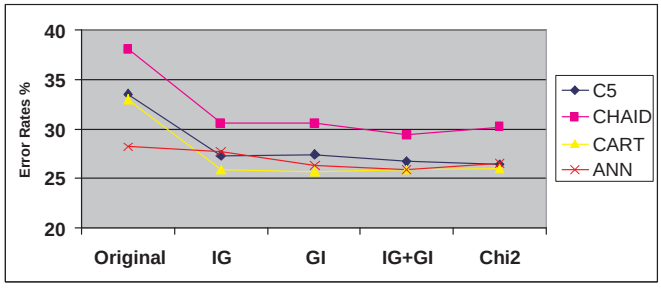


Figure 20: Absolute error rate of all classifiers over all data sets.

had a better error rate baseline on most data sets, see Figure 20.

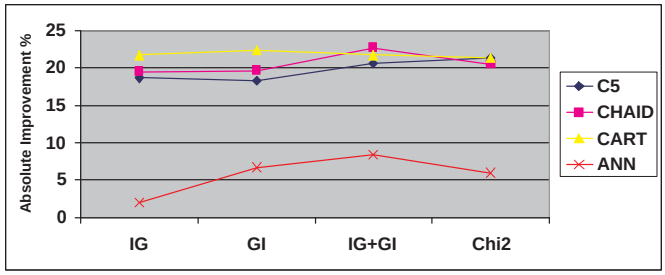


Figure 21: Absolute improvement in error rate of all classifiers over all data sets.

6.7 A Comparative Study

In this Section, we compare our results with those of other approaches to feature construction, namely, PCA (principle components analysis) and other evolutionary techniques discussed in Chapter 4.

6.7.1 Principle Components Analysis (PCA)

We perform principle components analysis on the ten-fold trials of each data set. The principle components are created by linearly combining the original set of attributes (using SPSS) with the aim of finding a smaller new subset with similar/better predictive power.

SPSS is used to extract the *component scores* which are used to calculate the principle components. The rotation method used is *varimax* as it minimises the number of variables that have high loadings on each component. The number of extracted components is determined by their eigenvalues. For this comparative study, only components with eigenvalues greater than 1 are extracted. Table 36 shows the average number of components for each data set along with their eigenvalues and percentages of variance. The number of principle components ranges from a minimum of 1 component, for the Abalone, to a maximum of 3 components, for the Wave and Wine, see Table 36. For each data set, two augmented attribute sets are created for each trial. In the first attribute set we add the principle components to the original set of attributes to form *aug-PC* (feature construction). The second attribute set consists of the principle components only, *PC* (feature extraction). The error rates of classification using the GP augmented attribute sets and PCA augmented attribute sets (bolded) are shown in Table 37. C5, CART and ANN are used in this comparative study.

The results in Table 37 show that the error rates of classification using the GP augmented attribute sets are generally better than those using the PCA augmented sets, particularly on the Balance, Bupa and Wine data sets. In fact, on the Wine data set, the performance of classifiers using all PCA augmented sets is even worse

Table 36: The average number of principle components, their eigenvalues and % of variance.

Data set	Number of components	Eigenvalues	% of Variance
Abalone	1	6.61	82.57
Balance	2	1.03, 1.01	25.66, 25.18
Bupa	2	2.50, 1.07	41.71, 17.91
Wave	3	7.90, 3.26, 1.02	37.64, 15.53, 4.88
Wine	3	4.71, 2.50, 1.45	36.22, 19.21, 11.16

than using the original attribute sets. The performance using the *PC* sets is bad all around, except on the Wave data set. The error rates using the Abalone data set, once again, do not show much benefit from all augmented attribute sets. Only for the Waveform data set, do we see better error rates using the two PCA augmented attribute sets. This could be due to the fact that there is a linear relationship between the predicting attributes and the class attribute, which is exploited by the principle components. However, with no size limitation, the GP should be able to find this linear combination of the original attributes. Furthermore, as the only constant in the GP terminal set is the number 1, it would require the GP to construct large subtrees to find the appropriate eigenvalues.

6.7.2 Other Evolutionary Approaches

In Table 37 we also compare error rates of our approach with those using other evolutionary work presented in Chapter 4, particularly with that of Bot [13] (*GP-KNN Classifier*), Sherrah et al. [86] (*EPrep*), and Smith et al. [90, 91] (*GAP*). Although the classifiers used in the other approaches are different from the ones used in our experimental work, it is thought that such a comparison would provide the reader with a broader idea of existing work on evolutionary feature construction/extraction.

The results on common data sets (shown in Table 37) are outlined as follows:

- Although the error rates of C5, CHAID, CART and ANN using original attributes were lower than those of the KNN's [13] on the Bupa, Wave and Wine data sets, generally, the rate of improvement using the augmented attribute sets in our approach was not only higher than that of [13], it was more consistent all around. We can also see that the error rates of the KNN using the Wave augmented attribute sets of [13] are worse than those of the original's. Only the performance of CHAID on the Wine data set was worse than the KNN's performance.

- It is not very clear what classifier Eprep [86] used to get the error rates using the Abalone and the Balance data sets, however, our experiments show better error rates than those of EPrep's, particularly on the Balance.

- Compared to decision tree classifiers, GAP [91] showed a slightly better performance using all augmented attribute sets on the Wine data set. Nevertheless, ANN in our approach performed better. As for the Bupa, all classifiers showed superior performance than that of GAP's.

Table 37: Comparison study: error rates (averaged over 10-fold trials) of the GP augmented attribute sets, the PCA augmented attribute sets, GP-KNN classifier, EPrep, and GAP.

Classifer	Attribute set	Abalone	Balance	Bupa	Wave	Wine
C5	Original	79.26 ± 1.21	22.42 ± 6.20	36.47 ± 7.99	22.94 ± 2.23	6.47 ± 7.04
	Aug-IG	79.09 ± 2.06	0.00 ± 0.00	32.35 ± 6.50	19.54 ± 1.64	5.29 ± 7.04
	Aug-GI	80.07 ± 2.40	0.00 ± 0.00	32.94 ± 8.75	19.64 ± 2.37	4.12 ± 4.84
	Aug-IG+GI	79.28 ± 2.40	0.00 ± 0.00	26.47 ± 10.65	20.22 ± 1.77	7.06 ± 7.75
	Aug-Chi^2	79.26 ± 1.16	0.00 ± 0.00	26.77 ± 7.26	19.22 ± 2.03	6.47 ± 8.06
	Aug-PC	79.139 ± 2.21	15.00 ± 6.36	35.00 ± 8.14	14.44 ± 2.10	8.82 ± 7.47
	PC	79.523 ± 2.47	26.29 ± 11.1	41.77 ± 7.94	13.48 ± 1.25	21.77 ± 12.10
CHAID	Original	76.45 ± 1.61	28.39 ± 5.17	40.00 ± 9.73	28.36 ± 1.81	17.06 ± 8.06
	Aug-IG	74.77 ± 2.62	5.65 ± 2.55	30.00 ± 6.17	24.92 ± 2.30	17.65 ± 8.77
	Aug-GI	75.40 ± 2.34	5.49 ± 2.30	32.06 ± 8.14	24.64 ± 2.12	15.29 ± 7.94
	Aug-IG+GI	74.61 ± 1.88	5.16 ± 2.92	27.94 ± 6.83	25.30 ± 1.92	14.12 ± 7.44
	Aug-Chi^2	75.32 ± 1.00	4.19 ± 2.04	33.24 ± 6.05	24.36 ± 2.79	14.12 ± 9.28
CART	Original	74.69 ± 1.76	22.26 ± 4.74	32.35 ± 7.21	24.58 ± 2.61	10.59 ± 9.92
	Aug-IG	73.02 ± 2.55	0.00 ± 0.00	30.29 ± 8.77	20.02 ± 1.71	5.88 ± 8.32
	Aug-GI	73.86 ± 2.43	0.00 ± 0.00	31.18 ± 7.87	19.54 ± 1.76	3.53 ± 3.04
	Aug-IG+GI	73.76 ± 1.85	0.00 ± 0.00	27.65 ± 8.34	20.04 ± 1.41	7.77 ± 9.92
	Aug-Chi^2	74.56 ± 1.87	0.00 ± 0.00	29.71 ± 7.52	19.48 ± 2.46	5.88 ± 8.32
	Aug-PC	73.16 ± 2.18	18.23 ± 5.27	31.18 ± 6.82	14.00 ± 1.01	11.18 ± 8.53
	PC	74.84 ± 1.80	29.68 ± 10.54	45.59 ± 11.02	13.56 ± 1.17	21.76 ± 10.76
ANN	Original	72.45 ± 1.93	10.00 ± 3.79	37.65 ± 7.18	17.15 ± 7.41	3.53 ± 7.44
	Aug-IG	72.49 ± 2.74	9.36 ± 4.01	35.29 ± 10.47	15.20 ± 2.29	2.94 ± 5.72
	Aug-GI	73.14 ± 2.79	9.19 ± 3.65	31.47 ± 8.66	15.77 ± 2.28	1.76 ± 2.84
	Aug-IG+GI	73.64 ± 1.99	7.42 ± 4.57	31.18 ± 8.46	14.40 ± 1.53	2.35 ± 5.68
	Aug-Chi^2	72.88 ± 3.08	9.68 ± 4.93	32.06 ± 7.90	15.44 ± 2.03	2.35 ± 4.11
	Aug-PC	73.164 ± 2.18	10.324 ± 4.18	42.35 ± 10.85	15.32 ± 1.60	3.53 ± 7.44
	PC	74.844 ± 1.80	32.742 ± 13.26	48.24 ± 10.21	13.46 ± 1.35	31.18 ± 19.81
KNN [13]	Original	-	-	49.6 ± 10.2	31.2 ± 11.3	29.00 ± 14.4
	Aug-KNN	-	-	41.6 ± 11.53	38.2 ± 14.87	12.4 ± 9.53
	Aug-mdm	-	-	38.3 ± 11.51	31.9 ± 13.74	11.9 ± 7.64
	Aug-ppd	-	-	42.6 ± 11.5	34.3 ± 13.07	14.00 ± 11.10
EPrep [86]	-	76.98 ± 1.6	9.28 ± 11.25	-	-	-
C4.5 [90]	GAP-C4.5	-	-	34.23 ± 8.67	-	4.71 ± 5.64
IBK [91]	GAP-IBK	-	-	39.95 ± 11.23	-	3.45 ± 4.75
NB [91]	GAP-NB	-	-	29.56 ± 6.33	-	3.64 ± 5.00

Generally, using GP for feature construction has proven to very effective in terms of improving the classification accuracy of various classifiers on different domains. The results of our experimental work support such claim and show a great potential, particularly for decision tree classification. Most evolutionary approaches to feature construction, including those used in the comparison, however, employ the hybrid or interleaving approach, where a classifier is run every time the fitness is calculated. One of the strengths of our approach is the use of the node-splitting criteria as the fitness function of the GP. This approach avoids embedding the classification technique within the GP and using its performance as the fitness measure, causing overfitting and longer execution time. However, there are some limitations with the existing work that need to be tackled, which are addressed in Chapter 9.

6.8 Summary

This chapter presents the results of classification of a number of inductive algorithms, namely C5, CHAID, CART, and an MLP ANN, on different domains. We performed attribute construction with the aim of improving the error rates of classification on five data sets. The presented error rates are on the testing sets using the augmented attribute sets which include an additional attribute evolved using GP. Four fitness measures were used, individually, in the GP. They include information gain, gini index, combined information gain and gini index, and the Chi^2 test. The reason for choosing to evolve attributes using these fitness measures is to improve the classification error rate and to investigate whether any of the decision tree classifiers have a bigger advantage over attribute sets that include a constructed attribute whose GP fitness function incorporates the classifier's splitting mechanism.

The results show a clear improvement in error rate, particularly for all decision tree classifiers. The improvement in performance was statistically validated, as shown in the T-Tests. However, no obvious biases were found in the performance of any of the classifiers.

The results of the current show the robustness and consistency in performance when constructing attribute using GP whose fitness function is based on the splitting criteria of decision tree classifiers. When compared to other approaches, the current work showed lots of strength and proved competitive, particularly with those employing an evolutionary approach.

Chapter 7

Further Analysis

7.1 Introduction

In the previous chapter, we presented the accuracies of four classification techniques, namely, C5, CHAID, CART and ANN on a number of data sets. We compared their performance on the original attribute sets and on the augmented attribute sets which includes an additional attribute constructed using a GP whose fitness function incorporates the splitting mechanism of one of the decision trees used. In this chapter, we perform further analysis on the size of the decision trees and the evolved attributes. Furthermore, we examine the effectiveness of GP for classification to GP for attribute construction by comparing its classification performance to the performance of the classification techniques used in Chapter 6 on the Balance–scale data set.

In Section 7.2 we turn our attention to the issue of the size of the resultant decision tree when an evolved attribute is included in the attribute set. Further illustration of the decision trees and the evolved attributes of two data sets is given in Section 7.3. We show the error rates of GP classification and compare the results to those achieved by other classifiers before and after the evolved attributes were

included in the attribute set in Section 7.4. In Section 7.5, we present a summary of this chapter.

7.2 Tree Size

In this Section, for four data sets, we have a look at the characteristics of the resultant decision trees with the original and augmented attribute sets. These characteristics are presented for the Balance, BUPA, Wave and Wine data sets, in Tables 38 to 41, respectively. The decision trees for the Abalone data set are not discussed in this chapter as they are computationally expensive to analyse, particularly C5 trees. This is because C5 produces very large trees, where the number of nodes for one trial is 2120 and the depth of the tree is 22. In Tables 38 to 41, for each data set, and for each tree induction algorithm (C5, CHAID, CART), there are four groups of entries. The terms Original, Aug-IG, Aug-GI and Aug-Chi^2 correspond to the constructed decision trees using the original attribute set, the augmented attribute sets. The meanings of the columns and the rows in the table are:

D : the average depth of all decision trees over the 10 fold trials.

N : the average number of nodes in the trees over the 10 fold trials.

Occur. : the average number of times that the evolved attribute appears in the tree over the 10 fold trials.

Av. Complexity : this is the last row and it shows the average complexity of the evolved attribute over the 10 fold trials. The complexity represents the number of original attributes, constants and the number of operators used to form the

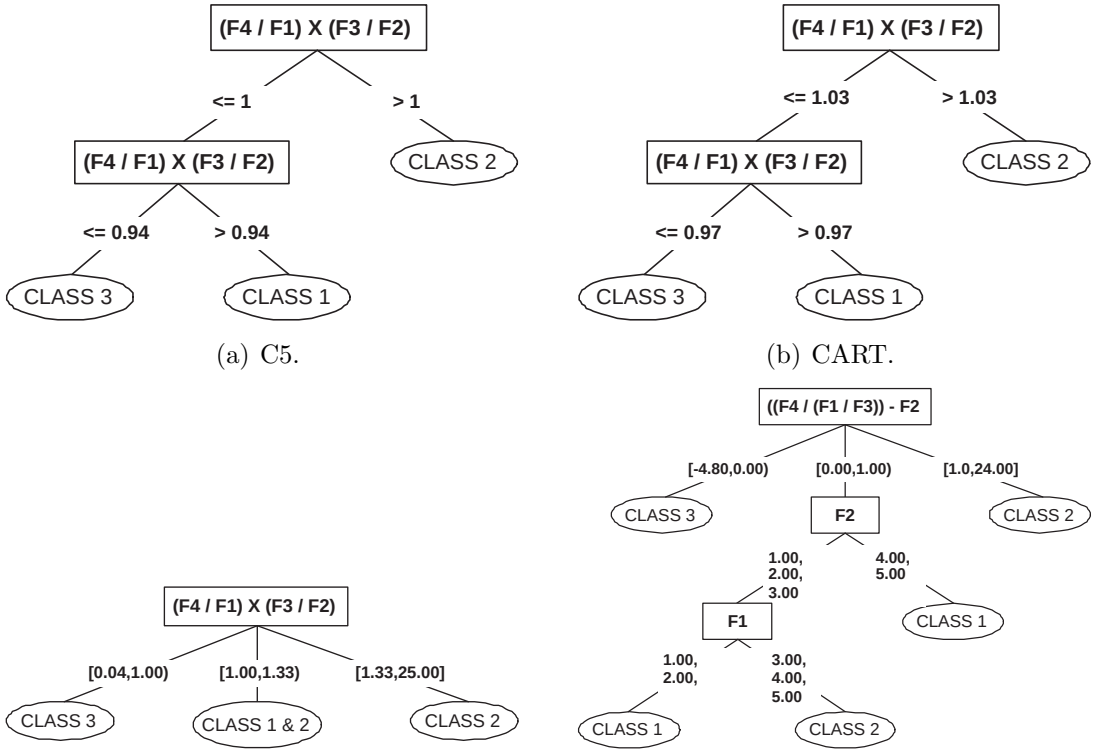
new feature.

Table 38: Average tree size for the Balance-scale classification by C5, CART and CHAID.

Balance	Original		Aug-IG			Aug-GI			Aug-Chi^2		
	D	N	D	N	Occur.	D	N	Occur.	D	N	Occur.
C5	8.7	80.8	2	5	2	2	5	2	2	5	2
CHAID	4.4	43.7	1.4	4.8	1	1.6	5.5	1	1.4	4.8	1
CART	7.2	82.2	2	5	2	2	5	2	2	5	2
Av. Complexity			7.2			7			7		

For the Balance data set, the tree sizes are presented in Table 38 and it shows that all classifiers produce much smaller trees with all augmented attribute sets compared to the trees produced using the original attribute sets. The depths, *D*, of the trees, for C5 and CART on the augmented attribute sets is reduced by between 72% to 77% of D using the original attribute sets and the number of nodes, *N*, is around 6% of N using the original sets. CHAID trees using the augmented attribute sets are also much smaller than original trees with D showing between 64% to 68% reduction and N reduced to around 11.5% of the original N. By relating D, the average tree depth, and the number of times the evolved feature appears on the trees of the augmented attribute sets, *Occur*, we see that for C5 and CART, the evolved feature is the only attribute used in the branch nodes of the trees, see Figures 22(a) and 22(b). On the other hand, for CHAID, the evolved feature appears once at the top of the tree, which indicates that, when the depth of a tree is greater than 1, some of the original attributes are used in that tree, see Figures 22(c) and 22(d).

It is therefore very clear that all classifiers produce significantly smaller trees, however, no algorithm shows a clear bias towards the set that has an additional feature



(a) C5. (b) CART.

(c) CHAID(1). (d) CHAID(2).

Figure 22: Examples of the decision trees for the Balance data set.

which embeds the same splitting mechanism as the associated decision tree.

We now look at the decision trees for the BUPA data set, shown in Table 39. CHAID trees are generally smaller on the original and augmented attribute sets. The depths using the augmented attributes are reduced by between 26.3% to 29% and the reduction in the number of nodes, N, ranges from 15.7% to 27%. However, if we take into account, *Occur*, the number of times the evolved attribute appears in a tree, and the complexity of the evolved attributes, we find that CHAID does not benefit in terms of the overall size of the trees. In other words, the computation time for creating a new attribute is not justified by CHAID as it is only used once in the trees

and does not result in significant reduction in the tree size. C5 shows a significant improvement as the number of nodes is reduced to about 50% of the original number. However, the depths of the trees decrease only slightly, ranging from 11% to 21%. CART trees, on the other hand, do not show much reduction on the augmented attribute sets. In fact, the range of increase of the depths of the trees is 6% to 11% while the number of nodes decreases by between 26% to 34.5%. By taking into account *Av. Complexity*, the average complexity of the evolved attributes, together with *Occur*, the average number of times they appear in the trees, we see that the overall number of nodes $(N - Occur) + (Av.complexity \times Occur))$ using the augmented attribute sets is much higher than the number of nodes of the original trees. Nevertheless, *Occur* shows that C5 and CART uses the constructed attribute more times in the trees, which means that they both benefit from the inclusion of the constructed attributes.

Table 39: Average tree size for the BUPA classification by C5, CART and CHAID.

BUPA	Original		Aug-IG			Aug-GI			Aug-Chi^2		
	D	N	D	N	Occur.	D	N	Occur.	D	N	Occur.
C5	9.1	61.9	7.5	28	3.3	7.2	26.4	2.3	8.1	32.8	2.5
CHAID	3.8	13.4	2.8	10.5	1	2.8	9.8	1	2.7	11.3	1
CART	6.7	45.2	7.5	33	4.5	6.5	29.6	2.3	7.1	33.6	3.9
Av. Complexity			24.2			25.2			24.7		

The decision trees on the Wave data set are incredibly large, specially for C5 and CHAID, see Table 40. The trees using the augmented attribute sets show hardly any reduction in the size from the original trees, except for C5, which shows a reduction in the number of nodes, N, by around 25%. The average complexity of the constructed attributes are the highest so far, however, *Occur* shows that the constructed attributes appear between 10 to 19 times on the C5 and CHAID trees, and around

3.4 times on CART trees, which means that they all benefit from the inclusion of the constructed attributes.

Table 40: Average tree size for the Waveform classification by C5, CART and CHAID.

Wave	Original		Aug-IG			Aug-GI			Aug-Chi^2		
	D	N	D	N	Occur.	D	N	Occur.	D	N	Occur.
C5	15.2	353.4	14.5	275.6	10.2	14.9	259.2	11.1	15.6	278	9.6
CHAID	7.2	377.8	6.2	336.8	13.1	7.5	338.9	16	7.4	349.5	18.8
CART	7.5	53.4	6.3	48.2	3.8	6.5	44.8	3.4	6.5	47.6	3.1
Av. Complexity			32.8			32.7			30.3		

Finally, we look into the decision trees using the Wine data set, shown in Table 41. Originally, the tree sizes are much smaller than those of the other data sets. If we measure relative performance for CART, we see that it achieves the highest reduction in size with D reduced to between 57% and 90% and N between 44% and 73%. C5 trees show less improvement with N reduced to between 57% and 92% and D decreased to between 72% and 87% using the augmented attribute sets IG and GI but increases using Aug-Chi^2 by 12%. On the other hand, CHAID trees show the worst deterioration using the augmented attribute sets except for Aug-IG where there is slight improvement. By looking into the columns in Table 41, we see that all classifiers achieve the best reduction in tree sizes using Aug-IG attribute set.

The last row in Tables 38 to 41, *Av. Complexity*, shows the average complexity of the evolved attributes using the three fitness measures, IG, GI and Chi^2. By comparing these values, we see that the GP produces smaller attributes when Chi^2 is used as the fitness measure. In Table 41, for example, the average complexity of the evolved feature in Aug-Chi^2 is as little as 5.2, which is no more than 40% of the

Table 41: Average tree size for the Wine classification by C5, CART and CHAID.

Wine	Original		Aug-IG			Aug-GI			Aug-Chi^2		
	D	N	D	N	Occur.	D	N	Occur.	D	N	Occur.
C5	3.2	9.8	2.3	5.6	1	2.8	7.2	1.1	3.6	9	1
CHAID	2.1	11.6	1.8	10.1	1.1	2.2	11.9	1.2	2.3	10.7	0.8
CART	4	14	2.3	6.2	1	2.5	6.4	1	3.6	10.2	1
Av. Complexity			12.8			14.2			5.2		

average complexities of the other evolved features using other fitness functions. We explore this further in the next Section.

7.3 A Closer Look at the Evolved Attributes

In this Section we further discuss the characteristics of two data sets, the artificial data set, Balance, and the real world data set, Wine. Further details on these two and all other data sets used in the thesis are available in [68] and are shown in Appendix A. We analyse the properties of the evolved features used to achieve 100% accuracy on both the training and testing sets. We try to find similarities and relationships among features evolved using the same fitness measure over all trials.

7.3.1 The Balance-Scale

The Balance data set was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight ($f1$), the left distance ($f2$), the right weight ($f3$), and the right distance ($f4$). The correct way to find the class is the greater of (left-distance $\times$ left-weight), i.e. ($f1 \times f2$) and (right-distance $\times$ right-weight), i.e. ($f3 \times f4$). If they are equal, it is balanced. In other words, if ($f1 \times f2$)

is **greater** than $(f3 \times f4)$, then the class is Left (L), else if $(f1 \times f2)$ is **less** than $(f3 \times f4)$ then the class is Right (R) otherwise, the class is Balanced (B).

For all fitness measures, the GP managed to evolve a feature, which is a combination of all primitive features in the data set, that satisfies the above property. Table 42 shows some of the most frequently evolved features.

Table 42: Examples of the most frequent evolved features on the Balance-Scale data set.

Fitness Measure	Evolved Feature
IG	$(f3 \times f4) \div (f1 \times f2)$ or $(f1 \times f2) \div (f3 \times f4)$
GI	$(f3 \times f4) \div (f1 \times f2)$ or $(f1 \times f2) \div (f3 \times f4)$
IG+GI	$(f3 \times f4) \div (f1 \times f2)$ or $(f1 \times f2) \div (f3 \times f4)$
Chi^2	$(f3 \times f4) \div (f1 \times f2)$ or $(f1 \times f2) \div (f3 \times f4)$

Figures 22(a) and 22(b) reveal one interesting finding, that both C5 and CART produce identical trees on all augmented attribute sets, with some variation on the splitting values.

7.3.2 The Wine

The Wine data set represents a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. This data set has three classes (wine types) and thirteen features representing the quantities of constituents found in each type.

Although this data set has 13 features, only 4 to 6 original features appeared in any single evolved feature when IG and GI were used as fitness measures. The

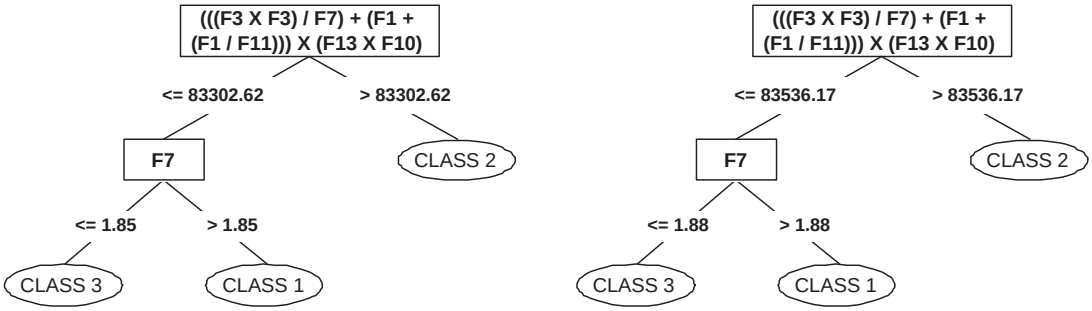
features that appeared regularly are $f1$, the alcohol content, $f7$, the flavanoids, $f10$, the colour intensity, $f11$, the hue, and $f13$, the proline. When Chi^2 was used as the fitness measure, the GP evolved features with a maximum complexity of 5, the features appearing in this single evolved feature are $f7$, $f10$ and $f11$, see Table 43.

In Table 43, we show some of the evolved features that, when added to the original attributes set, C5 and CART achieve 100% accuracy on both the training and testing sets, i.e. score 100% accuracy on the whole data set. Interestingly enough, Chi^2 seems to evolve the smallest feature that scores 100% accuracy on both C5 and CART.

Table 43: Some of the evolved features used to achieve 100% accuracy on the Wine data set.

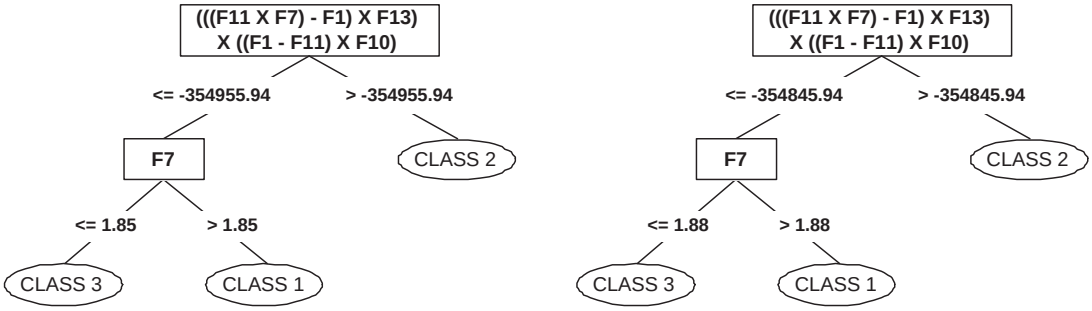
Fitness Measure	Evolved Feature	Classifiers
IG	$(((f3 \times f3) \div f7) + (f1 + (f1 \div f11))) \times (f13 \times f10)$	C5, CART
	$((1 - (f10 - f11)) \times ((f3 \times f5) \div (f10 + f4))) \times (f12 \times ((f13 - f5) \div f7))$	C5, CART
	$f7 \times (f11 \div f10)$	CART
GI	$(((f11 \times f7) - f1) \times f13) \times ((f1 - f11) \times f10)$	C5, CART
	$f13 \times ((f1 + (f1 \times f10)) \times ((f1 \div f11) - (f9 - f3))$	C5, CART
Chi^2	$f10 \div (f7 \times f11)$	C5, CART
	$f7 \times (f11 \div f10)$	CART

Figures 23 to 25 show some of the decision trees built using the augmented attribute sets where accuracies of 100% are scored on both the training and testing sets. As is the case with most of the other trees, we notice that C5 and CART construct identical trees, with some minor variation in the splitting values. The decision trees built using Aug-IG and Aug-GI, see Figure 23 and 24, first split on the evolved feature then, almost always, split on $f7$. This is interesting because the evolved feature, almost always, separates the samples that belong to *Class 2* from the rest, and then



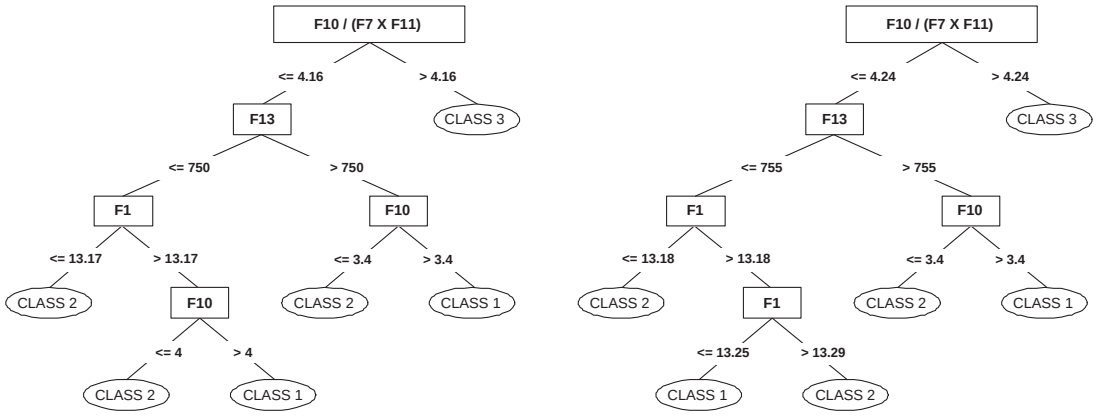
(a) C5. (b) CART.

Figure 23: The decision trees for the Wine data set using Aug-IG.



(a) C5. (b) CART.

Figure 24: The decision trees for the Wine data set using Aug-GI.



(a) C5. (b) CART.

Figure 25: The decision trees for the Wine data set using Aug-Chi^2.

$f7$ perfectly separates samples belonging to *Class 1* from *Class 3*.

Using Aug-Chi^2, although the evolved features are much smaller than those evolved using other fitness measures, all classifiers produce slightly bigger decision trees than those built using other augmented attribute sets, see Figure 25. Both C5 and CART first split on the evolved features which, this time, separate samples belonging to *Class 3* from the rest. They then split on $f13$, $f1$ and $f10$ to separate between *Class 1* and *Class 2*. It is noticed that $f7$ was never used in any of the subsequent splits. It is therefore clear that two separate attributes are evolved.

7.4 GP for Classification vs Constructive Induction

Since for the Balance data set, the GP managed to find a feature that, when added to the original attribute set, all decision tree classifiers managed to improve the classification accuracy, sometimes dramatically, as in the case of C5 and CART, we investigate

the performance of classification using GP as the inductive algorithm. In this Section, we address the question of whether GP is as powerful, when used for classification as it is for feature construction.

We perform two sets of experiments, where the GP is used for classifying the Balance data set. We use the same trials that were used in the feature construction process. The outcome of the evolution process is a rule of the form *"if <condition> then <classA> else <classB>"*, since we know that the rule which perfectly classifies the samples in this Balance data is something like:

if (f1 × f2) > (f3 × f4) then Class = L
else if (f1 × f2) < (f3 × f4) then Class = R
else Class = B

For this reason the function set is extended to include the logical operators $\{if, =, >, <, AND, OR\}$, in addition to the arithmetic operators $\{+, -, \times, /\}$. The terminal set includes the original four attribute and the values of the class $1, 2, 3$, denoting B, L and R. We apply the same GP parameters as those used in the feature construction process, see Table 44.

Table 44: The GP parameters.

Parameter	Value
Population Size	600
Generations	100
Crossover Probability	50%
Mutation Probability	50%
Selection Method	Tournament Selection of 7
Initial Population Generation	Ramped Half–and–Half

The fitness function of the GP is the classification error rate on the training data. We applied the same restriction on rule size as the one used in the attribute construction, i.e a maximum of 30. This means that any rule exceeding this predefined size is penalised. Ideally, one would expect the GP to evolve a rule of the form *(if (> (* (f1 f2)) (* (f3 f4))) 1 (if (< (* (f1 f2))(*(f3 f4))) 2 3))* that achieves 100% accuracy on the Balance–scale data set. This rules would have a complexity of 19.

In the two experiments, the GP produces relatively similar rules on all 10 trials (in terms of rule complexity and error rate). For simplicity, only the rules produced by one set of experiments are shown in this Section. The rules, shown as symbolic expressions, and the error rates on the associated training and testing sets are shown in Table 45. As can be seen in the Table, the evolved rules contain combinations of all primitive attributes. We also notice that the evolved rules manage to classify samples belonging to class 2 (scale to the left) and class 3 (scale to the right) only. Samples belonging to class 1 (scale balanced) are not classified by any of the rules. This may be due to the fact that samples belonging to classes 2 and 3 form 92.16% of the whole data. In fact, the training and testing error rates shown in Table 45 represent exactly the proportion of samples that belong to class 1 in each trial. The evolved rules shown in Table 45 can be represented in three trees, see Figure 26.

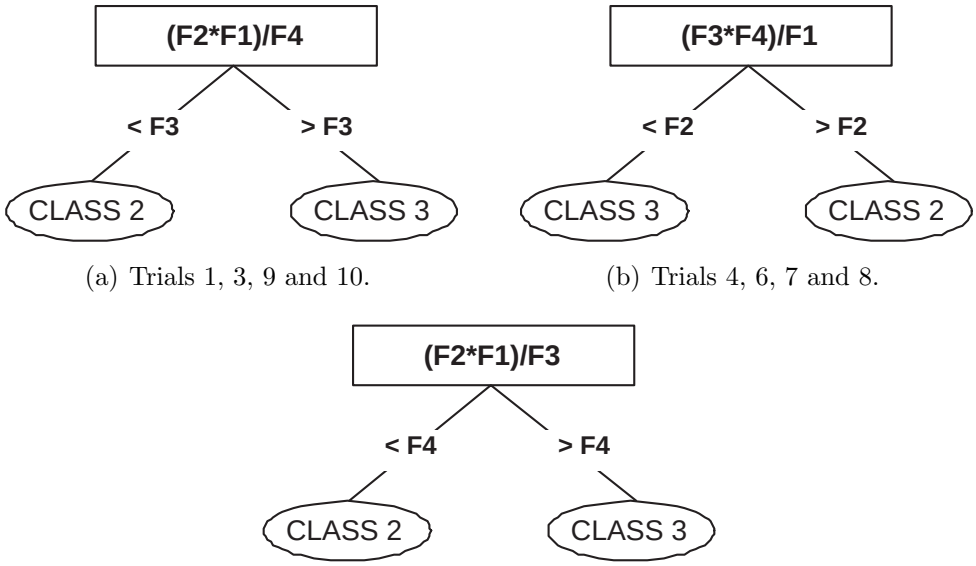
Table 45: Symbolic Expression of the Evolved Rules and Error Rates on the Associated Testing sets.

Trial #	Produced Rules	Error Rates	
		Training	Testing
1	(If (> (/ (* f2 f1) f4) f3) 3 2)	8.17	4.84
2	(If (< (* (/ f1 f3) f2) f4) 2 3)	7.28	12.9
3	(If (> (/ (* f2 f1) f4) f3) 3 2)	7.82	8.06
4	(If (< f2 (/ f4 (/ f1 f3))) 2 3))	8.17	4.84
5	(If (< f4 (/ (+ 1 (* f2 f1)) f3)) 3 2))	7.28	12.9
6	(If (> f2 (/ f3 (/ f1 f4))) 3 2)	7.64	9.68
7	(If (> (* f4 (/ f3 f1)) f2) 2 3)	8.35	3.23
8	(If (< f2 (* (/ f4 f1) f3)) 2 3)	8.35	3.23
9	((If (< (/ f1 (/ f4 f2)) f3) 2 3)	7.99	6.45
10	((If (> (/ (* f1 f2) f4) f3) 3 2)	7.28	12.9

In Table 46, we show the error rates of all classifiers using the original and augmented attribute sets, and the classification results of the GP. The Table shows that GP is superior to all decision tree classifiers on the original attributes, however, on the augmented attribute sets, all decision tree classifiers outperform the GP, showing error rates as small as 0%. On the other hand, the ANN shows a similar performance to the GP using the original and augmented attribute sets.

Table 46: Error rates for the Balance-scale data set.

Balance	Original	IG	GI	IG+GI	Chi^2
C5	22.42 ± 6.20	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
CHAID	28.39 ± 5.17	5.65 ± 2.55	5.49 ± 2.30	5.16 ± 2.92	4.19 ± 2.04
CART	22.74 ± 5.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
ANN	10.00 ± 3.79	9.36 ± 4.01	9.19 ± 3.65	7.42 ± 4.57	9.68 ± 4.93
GP	7.903 ± 3.98	-	-	-	-



(a) Trials 1, 3, 9 and 10.

(b) Trials 4, 6, 7 and 8.

(c) Trials 2 and 5.

Figure 26: The GP rules represented as trees.

7.5 Summary

In this chapter, we performed in depth analysis of the resultant decision trees. We presented and compared the sizes of the trees using the original and augmented attribute sets. Generally, all decision tree classifiers produced smaller trees, sometimes significantly, using the augmented attribute sets, as in the case of the Balance data set. The complexity of the evolved attributes was variable, and its use in the decision tree was dependent on the data as well as the classifier.

We also investigated the nature of the evolved attributes that, when included in the attribute sets, the classifiers perform perfectly. We found that, for the Balance, on all trials, the GP evolves an attribute that perfectly classifies all samples in the data set, using C5 and CART. On the Wine data set, the GP also evolved two attributes that enabled the classifiers to achieve the optimum accuracy. Our findings on the Balance data set encouraged us to further investigate the power of GP and use it for classification. Although GP outperformed all classifiers using the original attributes, it failed to perform better than all decision tree classifiers using the augmented attribute sets.

Chapter 8

A Commercial Case Study

8.1 Introduction

In this chapter we present an experimental study on the Apple NIR (near infra red) data, a real-world commercial data set. The data set contains 231 samples (151 training and 80 testing) and represented by 213 attributes. They include a sample number attribute, 211 attributes representing NIR wavelengths, and a class attribute (*Brix*). The initial overall aim of the exercise was to explore the data and to ascertain if it is feasible to use some or all of the 211 input fields to predict the target Brix value for the samples. A range of exploratory and investigatory analyses were performed to achieve these objectives:

- Applying statistical techniques such as correlation and collinearity which allow more familiarity with the data. They are used to test the data for strong predictors (wavelengths).
- Applying visualisation techniques allows us to explore the data in a much more visual manner, ranging from individual field analysis, through scattergrams of key predictors, to high dimensionality reduction visualisation techniques.

- Preprocessing the data using feature reduction techniques, such as feature selection.
- Applying data mining techniques on the pre–processed data to predict the Brix values. These include approximation methods and classification methods.

The purpose of this chapter is to extend the work performed in [89]. We perform attribute construction using the original and pre–processed data to enhance the performance of the classification models. Section 8.2 provides an overview of the data. In Section 8.3 previous work is presented. In Section 8.4, we present and compare the classification error–rates before and after attribute construction. Section 8.5 summarises this chapter.

8.2 An Overview

The Apple NIR data consists of 231 samples represented by 213 numerical attributes. The first attribute is the sample number and is eliminated from the data. The last attribute, *Brix*, is the class attribute with values in the range [9.4,17.9]. The remaining 211 attributes represent the NIR values measured at wavelengths ranging from 680 to 1100 in steps of 2, and denoted by *wv680, wv682,..., wv1098, wv1100*, respectively.

The data was provided in two sets, a training set and a testing set. The training set consists of 151 samples with the Brix values in the range [10.2,15.9]. The testing set consists of 80 samples with the Brix values in the range [9.4,17.9].

The overall aim of the exercise is to explore the data and to ascertain if it is feasible to construct attributes from some of the 211 to predict the target Brix value

for the samples. The sample number played no role in the analysis.

A number of models have been built and tested according to the standard methodology, i.e. each model is constructed using the samples in the training data set and the model is subsequently tested by applying it to the samples in the testing data set and measuring its accuracy.

8.3 Previous Work

In this Section, we outline previous work performed on the Apple NIR data. The following Subsection present the preliminary analysis and the prediction models performed on the data [89].

8.3.1 Preliminary Analysis

In this Section we present the preliminary analysis and previous work on the Apple NIR data set. A range of techniques was applied to explore the data initially. Note that some techniques apply only to numerical data, whilst others can be applied once the Brix values have been discretised.

Statistical analysis of NIR data was performed to determine the characteristics of the data. Figure 27 shows the min, max and mean value of the values for each wavelength in the training set. We note at this stage that there appears to be a wider variation in values for the lower wavelengths indicating that they may be more powerful predictors.

This wider variation is confirmed when we look at the values of the standard

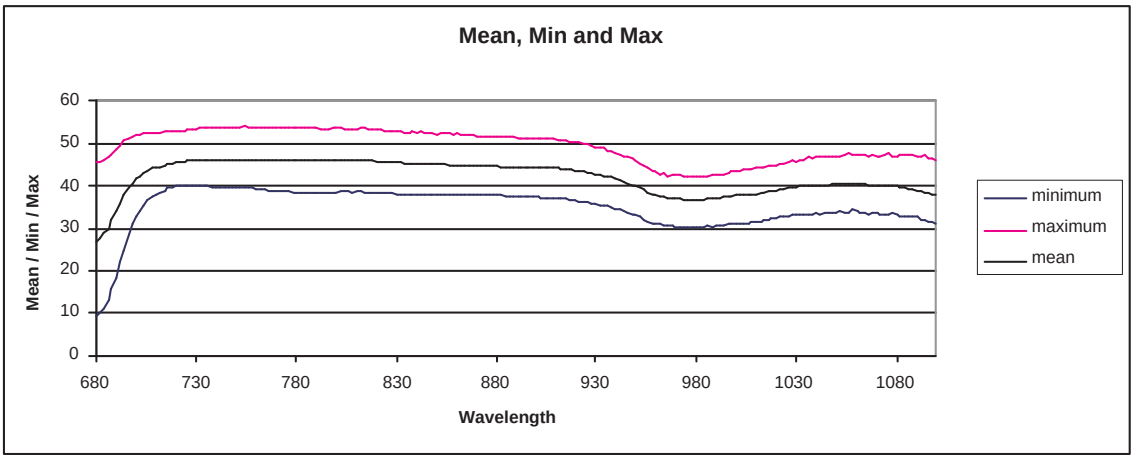


Figure 27: Mean, min and max values for wavelength data in training set.

deviation for each wavelength, once again for the training set, see Figure 28.

Comparison of training and testing sets to ensure that the sets are statistically similar revealed two facts:

- The range of Brix values for the testing sample [9.4, 17.9] is slightly larger than the range [10.2, 15.9] of Brix values of samples in the training set. This may cause a slight problem for models in that they are typically less well trained on data at the extreme edges of the Brix range.

- There are relatively few samples of high Brix value compared to the number of samples with low to medium Brix values. The resulting models may therefore be biased towards the lower Brix valued samples.

Identification of strong predictive fields using correlation analysis of predictive attributes (wavelength NIR data) with the class attribute (Brix), see Figure 29, for approximation models revealed the following:

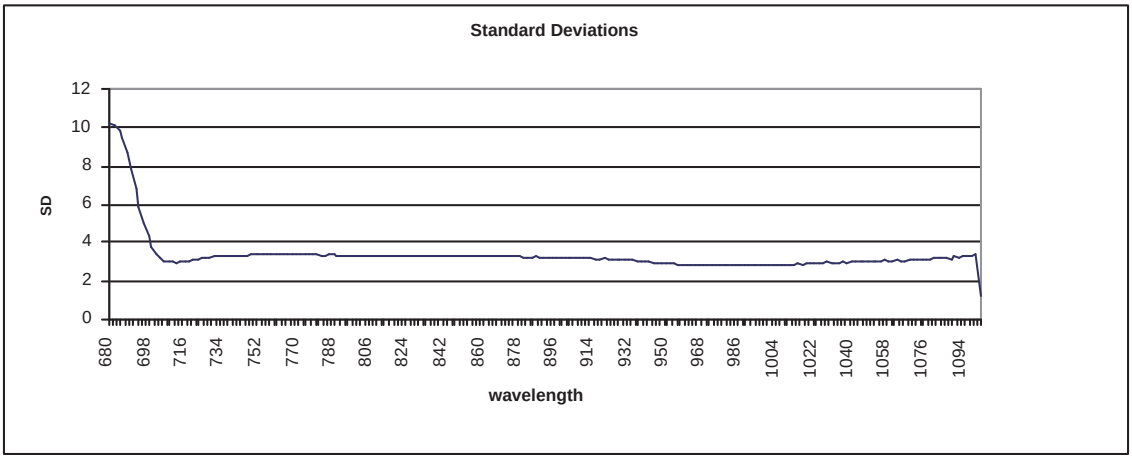


Figure 28: Standard deviations for wavelength data in training set.

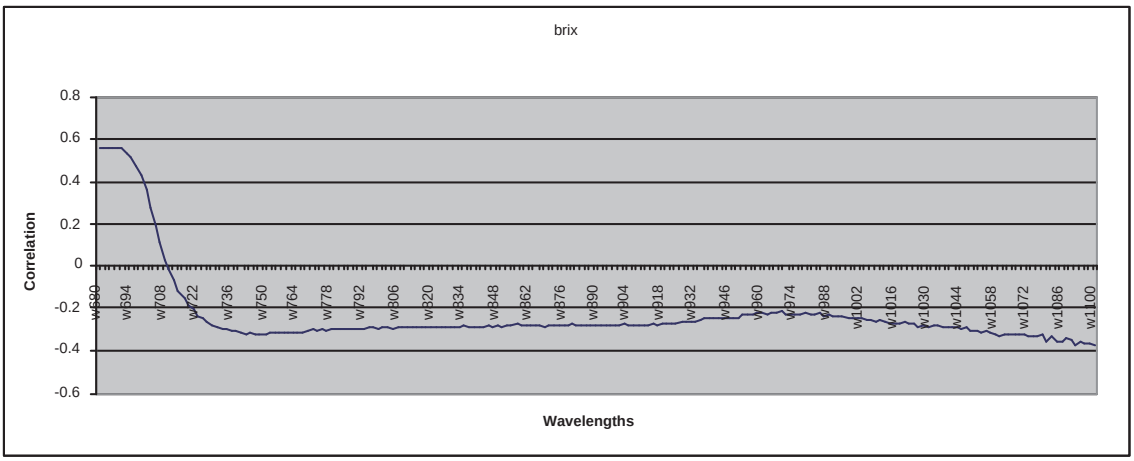
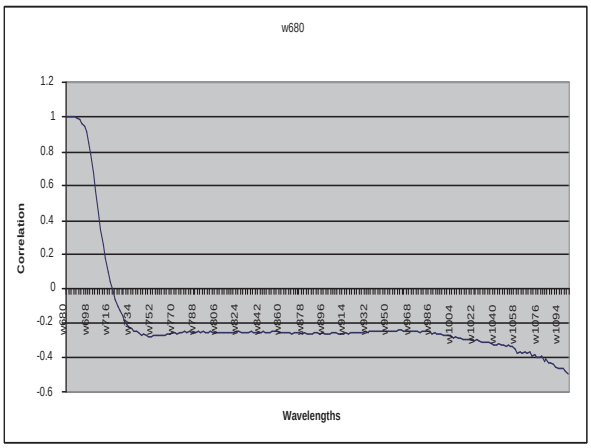


Figure 29: Correlation between the wavelengths and the Brix.

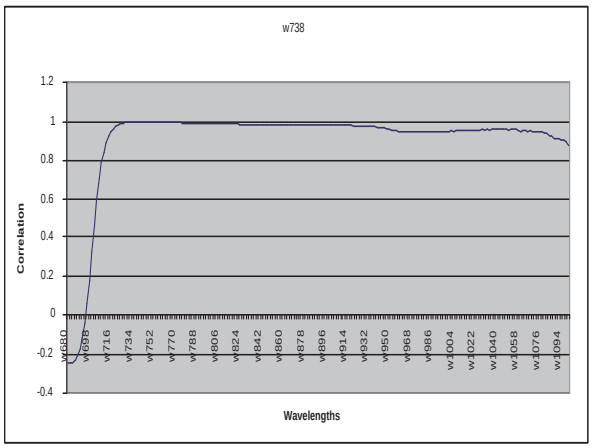
- There is a relatively high correlation between the lower wavelengths (WV680-WV696) and the Brix value (correlation values 0.57 - 0.43).
- There is little correlation between the medium valued wavelengths (WV700-WV1090) and the Brix value (absolute correlation values less than 0.35).
- There is a slight increase in (absolute value of) correlation between the higher wavelengths (WV1092-WV1100) and the Brix value (absolute correlation values between -0.35 and -0.4).
- By applying cross-correlation analysis on the input fields alone, there is high correlation (in excess of 0.9) between the wavelengths WV680-WV696, but the correlation between this subset and the remaining wavelengths drops off rapidly, see Figure 30(a). However, there is stronger correlation (almost 1.0) between the middle wavelengths, see Figures 30(b) and 30(c). At high wavelengths, although they are still highly correlated with the middle wavelengths, there is an increase in correlation (negative) with the lower wavelengths, see Figure 30(d).
- By looking at sample profiles of particular Brix values plotted over wavelength, there appears to be a distinction between low Brix value samples (<13) and high Brix value samples (>13) only at the lower wavelengths, see Figure 31.

Thus the subset WV680-WV696 is likely to be dominant at predicting the Brix value. However, because of the high cross-correlation, only one wavelength need be used to construct the model (this is confirmed by comparing the accuracies from different models constructed using different sets of wavelengths).

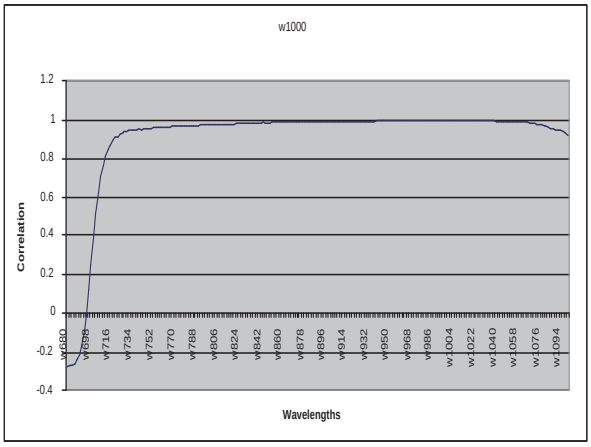
Based on the above analysis, we performed feature subset selection, in which we select the wavelengths that will be used to construct the models. A range of five different feature subsets was chosen:



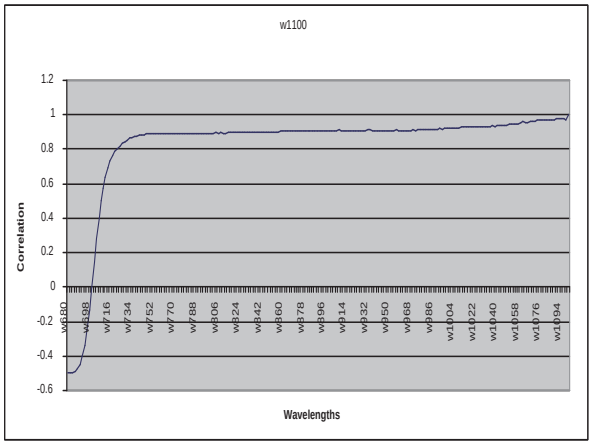
(a) Correlation of WV680 data with all other wavelengths.



(b) Correlation of WV738 data with all other wavelengths.

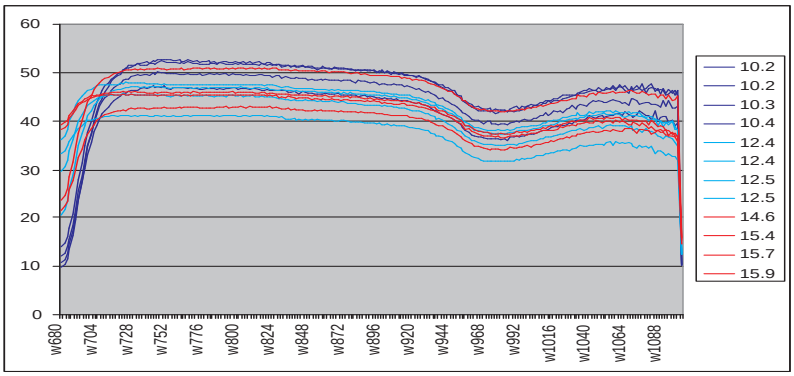


(c) Correlation of WV1000 data with all other wavelengths.

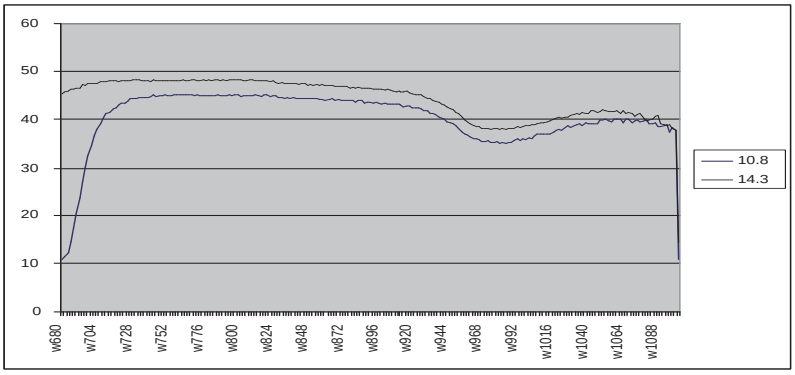


(d) Correlation of WV1100 data with all other wavelengths.

Figure 30: Correlation of some wavelengths with all other wavelengths.



(a) Profile of 12 samples (4 low Brix value, 4 medium and 4 high).



(b) Two sample profiles showing the potential difficulty of using higher wavelength data to predict the Brix value.

Figure 31: sample profiles of particular Brix values plotted over wavelength.

- *FS1*: the wavelengths 680-696 and wavelengths 1092-110, chosen because of relatively high correlation with the Brix values.
- *FS2*: the wavelengths 680-696, chosen for their high correlation with the Brix values.
- *FS3*: the wavelengths 680, 682, chosen to illustrate the fact that, because of high correlation between the wavelengths 680-696, there is no need to use the whole set.
- *FS4*: the single wavelength 680, chosen to represent the group of wavelengths (680-696) with relatively high collinearity.
- *FS5* the three wavelengths 686, 716 and 1100, chosen because of their relatively low correlation with each other.

8.3.2 The prediction models

Two approaches were used to predict the Brix values. The first is approximation models. In these models, the output is a real-valued variable approximating the Brix value of each sample. The models constructed were based on regression and artificial neural networks. Models are constructed for each feature subset. The results show that neither approach appeared to provide very accurate models, with correlations between actual and predicted Brix values of around 0.64-0.68 for the various feature subsets used.

The second approach is classification. In order to perform classification, the Brix values were discretised prior to model construction. A variety of different discetisations have been used for both the information gain analysis and subsequent model construction:

- *D6*: the Brix values are discretised into 6 equally sized bins (< 11, (11,12], (12,13], (13,14], (14,15], ≥ 15).
- *D4*: the Brix values are discretised into 4 bins ($< 11.5, (11.5, 13], (13, 14.5], \geq 14.5$).
- *D2*: the Brix values are discretised into 2 bins (≤ 13, > 13).

By measuring the information gain of the discretised data, we notice that the feature subsets FS1, FS2 and FS5 appear to mostly have the strongest predictors. We also use FS6, a feature subset created using stepwise reduction technique. FS6 includes $wv680$, $wv700$, $wv718$, $wv750$, $wv820$, $wv834$, $wv908$, $wv936$, $wv1002$, $wv1060$ and $wv1092$.

In the classification models, the output is an integer value approximating the bin associated with the Brix value of each sample. The models constructed were based on tree induction (C5), rule induction and multi-discriminant analysis. Models are constructed for each discretisation/bin set and for some of the feature subset sets. Information gain measures were used for classification models. The results varied more than in the approximation models showing that no one technique may be robust enough to rely upon. However, tree induction appears to have a slight edge over the others. Tables 47 to 49 show the accuracy of each model.

Table 47: Comparison of classification accuracies for discretisation set D6.

Method	No FSS	FS1	FS2	FS3	FS4	FS5	FS6
Tree Induction	33.75	37.50	-	-	-	-	-
All-rules model	28.75	-	-	-	-	-	-
MDA-direct	-	35.8	35.8	27.2	27.2	33.8	48.3
MDA-stepwise	-	33.1	29.1	27.2	-	31.8	-

Table 48: Comparison of classification accuracies for discretisation set D4.

Method	No FSS	FS1	FS2	FS3	FS4	FS5	FS6
Tree Induction	52.50	53.75	-	-	-	-	-
All-rules model	48.75	-	-	-	-	-	-
MDA-direct	-	53.60	53.60	44.40	44.40	52.30	60.90
MDA-stepwise	-	49.00	46.40	43.70	-	51.70	-

Table 49: Comparison of classification accuracies for discretisation set D2.

Method	No FSS	FS1	FS2	FS3	FS4	FS5	FS6
Tree Induction	68.75	67.50	-	-	-	-	-
All-rules model	63.75	-	-	-	-	-	-
MDA-direct	-	65.6	-	-	-	-	74.80
MDA-stepwise	-	62.3	-	-	-	-	-

8.4 Attribute Construction Using GP

In this Section, we focus on the performance of the decision tree models, built using C5. Based on the results from previous work, it is clear that classification models show superior performance using the discretised data set D2. Therefore, D2 is selected for further experimentation.

We use GP for constructing new features in the same manner used in the previous chapters, i.e. the constructed feature is added to the original feature set then classification is performed using the augmented feature set. As mentioned earlier, the training and testing set were provided by the client, therefore, no cross–validation was performed.

8.4.1 Feature Subsets

The feature subsets FS1, FS2, FS5 and FS6 are used for feature construction as well as the following subset:

- NoFS, which includes the whole set features.
- FS90, which includes the first 30, middle 30, and last 30 attributes. These are chosen to promote diversity in selection and to represent the middle attributes as well as the extreme sides.
- FS52, which includes the 52 attributes where the information gain does not vary. This set is a good representative of the whole data set, in terms of information gain.
- FSDiff1, which includes the first difference of neighbouring attributes.
- FSDiff2, FSDiff3 and FSDiff4, which include the second, third and fourth difference of the attributes, respectively. These are not used in the feature construction process.

8.4.2 The GP

Since we used C5 for the decision tree classification models, the fitness function used is information gain (IG). The terminal set consists of all the original attributes plus the constant 1, whilst the function set consists of the arithmetic operators $+, -, \times, /$.

The initial population is created using a ramped half-and-half method, and the size is fixed at 600. The GP was run for 100 iterations.

The selection method used is tournament, with a tournament size of 7. Mutation and crossover are fairly standard, with mutation replacing nodes with like nodes, and crossover swapping subtrees. The mutation rate is 50%, whilst the crossover rate is 50%.

8.4.3 The Results

A total of 16 experiments were carried out using the *original* and *augmented* feature subsets of the discretised data, D2. The results of the experiments conducted to assess the performance of C5 models on the original and augmented feature subsets are presented in Table 50. The *Orginal* column shows the classification accuracies using the associated feature subsets. The *augmented* column shows the accuracies after an evolved attribute is included in the associated feature subset. Table 50 shows that the performance using the augmented attribute sets is superior to that of the original feature subsets, particularly on the whole attribute set (No FSS) and on the subset created using stepwise reduction (FS6), with a relative improvement of almost 11%. Only using FSDiff1 does C5 perform better on the original subset than on the augmented subset.

Since the predicting attributes are basically NIR wavelengths measured in steps of 2, we transform the data by approximating the gradient of the different wavelengths using differences. We investigate the performance of C5 on the first, second, third and fourth difference of the wavelengths. Table 51 show the classification accuracies on the training and testing set. The results show a dramatic increase in accuracy using the training sets to between 97.35% and 99.34%, whereas the testing sets do not show much improvement, except for the first difference. One reason for this big difference in accuracies between the training and testing sets is because of overfitting.

Table 50: Classification accuracies for discretisation set D2 using original and augmented feature subsets.

Feature subset	Originial	Augmented
No FSS	68.75	76.25
FS1	67.5	68.75
FS2	67.5	67.5
FS5	67.5	67.5
FS6	67.5	76.25
FS90	68.75	68.75
FS52	70.00	72.5
FSDiff1	73.75	68.75

It could also be because the number of samples in the data is too small, represented by a large number of attributes.

Table 51: Classification accuracies for discretisation set D2 using the first, second, third and third differences of wavelengths.

Feature subset	Training	Testing
FSDiff1	99.34	73.75
FSDiff2	98.68	61.25
FSDiff3	97.35	50.00
FSDiff4	97.35	51.25

8.5 Summary

The work presented in this chapter extends the work performed by Smith et al. [89] on the Apple NIR data, a commercial data. We performed attribute construction with the aim of improving the performance of classification. The nature of the data has led us to believe that the accuracy of classification could improve when evolved features using GP are included in the attribute set. There are two reasons for this

assumption. The first reason is the fact that the Apple NIR data is numeric and the attributes represent NIR values measured at wavelengths ranging from 680 to 1100 in steps of 2. The second is because the GP finds non–linear combinations of the attributes that could explain physical traits in the data, as is the case with the Balance-scale and the Wine data sets, see Chapter 6 and 7.

There were two approaches to transforming the data. The first was to construct attributes using the feature subsets listed in Section 8.4. The second approach to transforming the data was by approximating the gradient of the different wavelengths using their differences. This included four data transformations using the first, second, third and fourth difference. Attribute construction was performed only on the first difference.

The results of the experiments of the first approach generally showed slight improvement in classification accuracy using the attribute sets which include an evolved attribute compared to the results using the original attribute sets. Only using the augmented attribute set of the first difference did we see a slight deterioration.

The results using the second approach show that the accuracies on the training data increases significantly (more than 97%), whereas, only using the first difference that the accuracies on the testing show superior performance.

Chapter 9

Conclusions, Limitations and Further Suggestions

9.1 Conclusions

This Thesis has presented an approach for restructuring feature spaces to improve decision tree classification. The developed model is based on using genetic programming as the search procedure for performing attribute construction. The constructed attributed is added to the original attributes to form an augmented attribute set which is presented to a classifier.

Chapter 1 described the background to the problem, the major motivations and contributions of this research. Chapter 2 outlined the field of KDD, major tasks and relevant subjects. We presented related previous work on data pre–processing and listed a number of approaches to data classification. In Chapter 3, we described three decision tree algorithms used in the experimental work. Chapter 4 introduced evolutionary computation techniques, addressing four well known algorithms. We focused on genetic programming which was used in our work. We also surveyed some of the applications of genetic programming in data mining.

The general approach and experimental methodology was presented in Chapter 5. It is based on the construction of a new attribute using genetic programming, which is then added to the original attribute set to form an augmented attribute set. The general aim is to improve the performance of classification on a number of data sets. In this chapter, the data sets, the genetic program and its fitness measures, and the experimental methodology were described some detail. We also presented some of the evolved attributes and show their fitness to give the reader an idea about the nature and properties of the evolved attributes.

The remaining chapters presented analysis of the results pertaining to the objectives of the research:

1. **Successful use of the decision tree splitting criteria as the fitness function of a GP system for feature constuction**.
2. **Improve classification accuracy** of decision tree and non–decision tree classification models (Chapter 6).
3. **Perform bias analysis** of the classifiers' performance (Chapter 6).
4. **Compare with existing approaches**, the classification accuracy of the experimental work (Chapter 6).
5. **Perform tree size analysis** on the resultant trees of the decision tree models (Chapter 7).
6. **Ascertain physical properties of data**, which may be explained by the introduction of an evolved attribute (Chapter 7).

7. **Compare the performance of GP** for classification with the performance of GP for constructive induction (Chapter 7).

8. **Undertake a commercial case study** involving the application of GP for constructive induction on real–world commercial data (Chapter 8).

In Chapter 6, we addressed objectives 1 and 2. We examined the performance of three decision tree classifiers, C5, CHAID, and CART and an MLP ANN on five data sets using the original and augmented attribute sets. The results showed that, for all data sets, the error rates of all classifiers improved using the augmented attribute set. Obviously, decision tree models showed a much better improvement rate than the ANN. The bias check presented in Chapter 6 showed that, generally, no decision tree model had more advantage using an augmented attribute set which included an attribute evolved using the GP when the fitness incorporates the associated splitting measure. We also performed a comparative study of the results of our experimental work with those of PCA and other evolutionary approaches. The comparison showed that our current work was robust and performed consistently on all data sets. It also showed that our approach was comparable to existing approaches.

In Chapter 7, we performed further analyses which cover objectives 3, 4 and 5. Analysis of the size of the resultant trees showed that, when the complexity of the evolved attribute is not taken into account, the models produce smaller, sometimes very concise trees using the augmented attribute sets. The structure of the evolved attributes had led us to believe that they could reveal interesting characteristics about the data as well as improving the performance of classification. For instance, on the Balance data set, the evolved attributes exposed how the data was created. On the Wine data set, where accuracy on the training and testing sets was 100%, the

evolved attributes were almost always constructed of 5 out of 13 of the original attributes (constituents). In some trials, the evolved attribute was made of 3 attributes only. When we used GP for classification, it outperformed all the other four techniques using the original attribute sets. However, using the augmented attribute sets, all other techniques were superior.

In Chapter 8, two data transformation approaches were carried out to improve the classification of C5 on a real–world commercial data. This work was an extension to the work in [89]. The first approach was to perform attribute construction on all original attributes and a number of feature subsets. The results showed that the C5 model performed better using all augmented attribute sets. In the second approach, we approximated the gradient of the different attributes (wavelengths) using their differences (first, second, third and fourth). The results showed that the C5 model scored accuracies of more than 97% on the training data, whereas, on the testing data, the accuracy drops dramatically.

9.2 Limitations and Further Suggestions

There are few limitations of the GP System. The first is that, although it has proved to be robust, the current system is computationally expensive and is sensitive to the dimensionality of input data. The increase in the number of attributes as well as the number of samples would bring about a significant increase in computation time. More computational resources are needed and effective methods for sampling the data, rather than random sampling, have to be developed to deal with these problems. To reduce the computational overhead of local optimisation, algorithms could be developed to apply different amounts of computational resources to different indiduals. For instance, individuals whose ancestors have been optimised and that

are similar to those ancestors could be considered relatively well optimised and hence, receive few optimisation trials in future.

Secondly, the current system only deals with numerical data and could be extended to deal with categorical data through *strong typing* and extending the function set to cope with such data.

Thirdly, the Chi^2 test used in the GP uses binary splitting of attribute values similar to that used in information gain and the gini index. We do not merge and group values in the same manner used in CHAID.

Most importantly, rather than evolving a single attribute that generates the best split at the root node of the tree, there are plans to extend the current work to evolve a set of attribute that can also split subsequent nodes, and hence, evolving a whole decision tree.

Finally, further research is required to enhance the fitness measure of the GP to suit a wider range of classifiers. Also, since the current GP system is concerned with constructing attributes to improve the classification task, we are looking into extending this work to include other data mining tasks.

Bibliography

[1] KDnuggets: Data Mining, Web Mining, and Knowledge Discovery Guide. `http://www.kdnuggets.com`.

[2] R. Agrawal, T. Imielinski, and A. Swami. Database Mining: A performance Perspective. In *IEEE Transactions on Knowledge and data Engineering*, volume 5(6), pages 914–925. IEEE Computer Society, 1993.

[3] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. Fast Discovery of Association Rules. In *Advances in Knowledge Discovery and Data Mining, U.M. Fayyad, G. Piatetski-Shapiro, P. Smyth and R. Uthurusamy (Eds)*, pages 307–328. AAAI Press, 1996.

[4] H. Almuallim and T. Dietterich. Learning with Many Irrelevant Features", booktitle ="Proceedings of the Ninth National Conference on Artificial Intelligence. pages 547–552. MIT Press, 1991.

[5] P. Angeline. Genetic Programming and Emergent Intelligence. In *Advances in Genetic Programming*, pages 75–98. MIT Press, 1994.

[6] W. Au, K. Chan, and X. Yao. A Novel Evolutionary Data Mining Algorithm With Applications to Churn Prediction. *IEEE Transactions on Evolutionary Computation*, 7, 2003.

[7] T. Back. Evolutionary Algorithms. *SIGBIO Newsletter*, 12(2):26–31, June 1992.

[8] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming – An Introduction: On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, 1998.

[9] H. Bensusan and I. Kuscu. Constructive Induction using Genetic Programming. In *Proceedings of International Conference on Machine Learning, Evolutionary Computing and Machine Learning Workshop, Fogarty, T. and Venturini, G. (Eds)*, 1996.

[10] H. Beyer, E. Brucherseifer, W. Jakob, H. Pohlheim, B. Sendhoff, and T. Binh To. Glossary: Evolutionary Algorithms - Terms and Definitions. 2002. `http://ls11-www.cs.uni-dortmund.de/people/beyer/EA-glossary/def-engl-html.html`.

[11] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[12] A. Blum and P. Langley. Selection of Relevant Features and Examples in Machine Learning. *Artificial Intelligence*, pages 245–271, 1997.

[13] M. Bot. Feature Extraction for the k-Nearest Neighbour Classifier with Genetic Programming. In *EuroGP 2001, Computer Science, 4th European Genetic Programming Conference, Lake Como, Italy, Lecture Notesin Computer Science, LNCS, no 2038, J. Miller, M. Tomassini, P.L. Lanzi, C. Ryan, A.G.B. Tetamanzi, W.B. Langdon (Eds)*, pages 256–267. Springer-Verlag, April 18-20, 2001.

[14] G. E. P. Box. Evolutionary operation: a method of increasing industrial productivity. *Applied Statistics*, 6:81–101, 1957.

[15] R. Brachman and T. Anand. The Process of Knowledge Discovery in Databases: A Human Centred Approach. In *Advances in Knowledge Discovery and Data Mining, Fayyad, U., Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, R. (eds)*. AAAI/MIT Press, 1996.

[16] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Inc. Belmont, California, 1984.

[17] P. Brierley and B. Batty. Data Mining With Neural Networks: An Applied Example in Understanding Electricity Consumption Patterns. *IEE, Knowledge Discovery and Data mining, Bramer, M. (eds)*, 1999.

[18] F. Brill, D. Brown, and W. Martin. Genetic algorithms for feature selection for counterpropagation networks. *Tech. Rep. No. IPC-TR-90-004*, 1990.

[19] T. Brotherton and P. Simpson. Dynamic feature set training of neural nets for classification. In *Evolutionary Programming IV, McDonnell, J. and Reynolds, R. and Fogel, D.(Eds.)*, pages 83–94, Cambridge, MA, 1995. MIT Press.

[20] R. Caruana and D. Freitag. Greedy Attribute Selection. In *Machine Learning: Proceedings of the Eleventh International Conference, W. Cohen and H. Hirsh (eds)*. Morgan Kaufmann, 1994.

[21] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth. CRISP-DM 1.0: Step–By–Step Data Mining Guide. 2000.

[22] N. L. Cramer. A Representation for the Adaptive Generation of Simple Sequential Programs. In *ICGA85: Proceeding of the International Conference on Genetic Algorithms*, pages 183–187, 1985.

[23] J. Debuse, B. de la Iglesia, C. Howard, and V. Rayward-Smith. A Methodology for Knowledge Discovery: a KDD Roadmap. Sys-c99-01, School of Computing Sciences, University of East Anglia, 1999.

[24] J. Debuse, B. de la Iglesia, C. Howard, and V. Rayward-Smith. Building the KDD Roadmap: A Methodology for Knowledge Discovery. *Industrial Knowledge Management, R. Roy (Ed.)*, Springer-Verlag, London.:179–196, 2000.

[25] T. Dietterich. Experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, 40 (2):139–158, 2000.

[26] K. Dowsland. Simulated Annealing. In *Modern Heuristic Techniques for Combinitorial Problems, C. R. Reeves (eds)*, pages 20–69, Oxford, UK, 1993. Blackwell Scientific.

[27] F. Farnstrom, J. Lewis, and C. Elkan. Scalability for Clustering Algorithms Revisited. *ACM SIGKDD*, 2:1:5151–57, Jul 2000.

[28] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge Discovery and data mining: Towards a Unifying Framework. In *Proceedings of the second International Conference on Knowledge Discovery and data Mining (KDD-96)*, pages 82–88. AAAI Press, 1996.

[29] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The KDD Process for Extracting Useful Knowledge from Volumes of Data. *Communications of the ACM*, 39(2):27–34, 1996. ACM, 1996.

[30] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. (eds) Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI Press / The MIT Press, 1996.

[31] D. Fogel. *Evolutionary Computation Toward a New Philosophy of Machine Intelligence*. IEEE Press, 1995.

[32] D. Fogel. The Advantages of Evolutionary Computation. *Biocomputing and Emergent Computation*, pages 1–11, 1997.

[33] D. Fogel. An Introduction to Simulated Evolutionary Optimization. *IEEE Transactions on Neural Networks*, pages 3–14, January 1994.

[34] D. B. (Editor) Fogel. *Evolutionary Computation: The Fossil Record*. Wiley-IEEE Press, 1998.

[35] L. Foulds. *Optimization Techniques : An Introduction*. Springer-Verlag, New York, 1981.

[36] A. S. Fraser. Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Science*, 10:484–491, 1957.

[37] A. Freitas. A Genetic Programming Framework for Two Data Mining Tasks: Classification and Generalized Rule Induction. In *GP97: Proceedings of the second Annual Conference*, pages 96–101. Morgan Kaufmann, 1997.

[38] A. Freitas. Understanding the Crucial Role of Attribute Interaction in Data Mining. *Artificial Intelligence Review 16(3)*, pages 177–199, 2001.

[39] A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, 2002.

[40] F. Glover and M. Laguna. Tabu Search. In *In Modern Heuristic Techniques for Combinitorial Problems, C. R. Reeves (eds)*, pages 70–151, Oxford, UK, 1993. Blackwell Scientific.

[41] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, 1989.

[42] M. Gonzales. Data Mining: A Call To Action. Intelligent Enterprise, (2003, April). `http://www.intelligententerprise.com/030405/606feat2_1.shtml`.

[43] C. Guerra-Salcedo and D. Whitley. Genetic Approach to Feature Selection For Ensemble Creation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 236–243, 1999.

[44] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, San Francisco, 2001.

[45] T. Ho. The Random Subspace Method for Constructing Decision Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20 (8):832–844, 1998.

[46] J. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 1995.

[47] H. Hotelling. Analysis of a Complex of Statistical Variables into Principal Components. *Journal of Educational Psychology*, 24:417–441, 1933.

[48] A. Jain, M. Murty, and P. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31:30:264–323, 1999.

[49] G. John, R. Kohavi, and K. Pfleger. Irrelevant Features and Subset Selection Problem. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 121–129. Morgan Kaufmann Publishers, 1994.

[50] I. Jolliffe. *Principal Component Analysis.* Springer-Verlag, second edition edition, 2002.

[51] G. Kass. An Exploratory Technique for Investigating Large Quantities of Categorical Data. *Applied Statistics*, 29:119–127, 1980.

[52] K. Kira and L. Rendell. The Feature Selection Problem: Traditional Method and a New Algorithm. In *Proceedings of the Tenth National Conference on Artificial Intillengence*, pages 129–134. MIT Press, 1992.

[53] R. Kohavi and G. John. Wrappers for Feature Subset Selection. *Artificial Intellengence 97*, pages 273–324, 1997.

[54] I. Kononenko. Estimating Attributes: Analysis and Extensions of Relief. In *Proceedings of the Seventh European Conference on Machine Learning*, pages 171–182. Springer-Verlag, 1994.

[55] J. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

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

[57] J. Koza, F. Bennett III, D. Andre, and M. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, San Francisco, CA, 1999.

[58] M. Kubat, I. Bratko, and R. Michalski. *A Review of Machine Learning Methods.* John Wiley and Sons Ltd., 1998.

[59] I. Kuscu. A Genetic Constructive Induction Model. In *Proceedings of Congress on Evolutionary Computation, Angeline, P. J. and Michalewicz, Z. and Schoenauer, M. and Yao, X. and Zalzala, A. (Eds)*, volume 1, pages 212–217. IEEE Press, 1999.

[60] P. Langley and S. Sage. Oblivious Decision Trees and Abstract Cases. *In Working Notes of the AAAI94 Workshop on CaseBased Reasoning*, 1994.

[61] C. Lee and D. Landgrebe. Feature Extraction Based on Decision Boundaries. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15 (4), 1993.

[62] H. Liu and H. Motoda. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers, Boston/Dordrecht/London, Boston, 1998.

[63] H. Liu, H. Motoda, and L. Yu. Feature Extraction, Selection, and Construction. In *Handbook of Data Mining (Human Factors And Ergonomics), Nong Ye (ed)*, pages 409 – 423. Lawrence Erlbaum Associates, Inc., 2003.

[64] W. Loh and Y. Shih. Split Selection Methods for Classification Trees. *Statistica Sinica*, 7:815–840, 1997.

[65] R. Marmelstein and G. Lamont. Pattern Classification using a Hybrid Genetic Program Decision Tree Approach. In *Genetic Programming 1998: Proceedings of the Third Annual Conference, Koza, J. and Banzhaf, W. and Chellapilla, K. and Deb, K. and Dorigo, M. and Fogel, D. and Garzon M. and Goldberg, D. and Iba H. and Riolo, R. (eds)*, pages 223–231, San Francisco, CA, USA, 1998. Morgan Kaufmann.

[66] B. Masand and G. Piatetsky-Shapiro. Discovering Time Oriented Abstractions in Historical Data to Optimize Decision Tree Classification. In *Advances in Genetic Programming, Angeline, P. and Kinnear Jr, E. (Eds)*, volume 2, pages 489–498. MIT Press, 1996.

[67] C. Matheus and L. Rendell. Constructive Induction on Decision Trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 645–650, San Mateo, CA, 1989. Morgan Kaufmann.

[68] C. Merz and P. Murphy. UCI Repository of Machine Learning Databases. 1996. `http://www.ics.uci.edu/\simmlearn/MLRepository`.

[69] T. Mitchell. *Machine Learning*. WCB/McGraw-Hill, 1997.

[70] W. Moore and S. Lee. Efficient Algorithms for Minimizing Cross Validation Error. In *Machine Learning: Proceedings of the Eleventh International Conference, W. Cohen and H. Hirsh (eds)*. Morgan Kaufmann, 1994.

[71] M. Muharram and Smith. G. A Comparison of GP Fitness Functions in Evolutionary Feature Construction. *Accepted in IEEE Transaction on Knowledge and Data Engineering (TKDE)*, 2005. A Special Issue on Intelligent Data Preparation.

[72] M. Muharram and G. Smith. The Effect of Evolved Attributes on Classification. In *AI 2003, Advances in Artificial Intelligence, 16th Australian Conference on AI, Perth 2003 Lecture Notes in Artifical Intelligence, LNAI, no 2903, T.D. Gedeon and L. C. C. Fung (Eds)*, pages 933–941. Springer-Verlag, 2003.

[73] M. Muharram and G. Smith. Evolutionary Feature Construction using Information Gain and Gini Index. In *EUROGP2004, Proceedings of the Seventh European Conference on Genetic Programming, Coimbra 2004, Lecture Notes in Artificial Intelligence, LNAI, no 3003*, pages 379–288. Springer-Verlag, 2004.

[74] S. Murthy and S. Salzberg. A System for Induction of Oblique Decision Trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[75] F. Otero, M Silva, M. Freitas, and J. Nievola. Genetic Programming For Attribute onstruction in Data Mining. In *Genetic Programming: Proceeding of the Sixth European Conference (EuroGP-2003), LNCS.*, volume 2610, pages 384–393, 2003.

[76] G. Pagallo and D. Haussler. Boolean Feature Disovery in Empirical Learning. *Machine Learning*, 5:199, 1990.

[77] K. Pearson. On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine*, 2:559–572, 1901.

[78] W. Punch, E. Goodman, M. Pei, C. Lai, P. Hovland, and R. Enbody. Further Research on Feature Selection and Classification Using Genetic Algorithms. In *Proceedings Fifth International Conference on Genetic Algorithms*, pages 557–564, 1993.

[79] J. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, 1986.

[80] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, 1993.

[81] M. Qureshi. The Evolution of Agents. *PhD Thesis*, 2001.

[82] M. Raymer, W. Punch, E. Goodman, and L. Kuhn. Genetic Programming for Improved Data Mining: An Application to the Biochemistry of Protein Interactions. In *Proceedings GP 1996*, pages 375–380. MIT Press, 1996.

[83] RuleQuest Research. See5: An Informal Tutorial. `http://www.rulequest.com`.

[84] G. Richards, V. Rayward-Smith, P. Snksen, S. Carey, and C. Weng. Data mining For Indicators of Early Mortality in a Database of Clinical Records. In *Artificial Intelligence in Medicine*, volume 22(3), pages 215–231. Elsevier, 2001.

[85] R. Setiono and H. Liu. Feature Extraction via Neural Networks. In *Feature Extraction, Construction and Selection: A Data Mining Perspective. H. Liu and H. Motoda (eds) (1998). 2nd Printing*, pages 191–204, Boston, 2001. Kluwer Academic Publishers.

[86] J. Sherrah. Automatic Feature Extraction for Pattern Recognition. *PhD thesis, University of Adelaide, South Australia*, 1998.

[87] J. Sherrah, R. Bogner, and A. Bouzerdoum. The Evolutionary Pre–Processor: Automatic Feature Extraction for Supervised Classification using Genetic Programming. In *GP-97, Proceedings of the Second Annual Genetic Programming Conference, University of Stanford, CA, Koza, J., Deb, K., Dorigo, M., Fogel, D., Garzon, M., Iba, H., and Riolo, R. (Eds)*, pages 304–312. Morgan Kaufmann, 1997.

[88] W. Siedlecki and J. Sklansky. A Note on Genetic Algorithms For Large–Scale Feature Selection. *Pattern Recognition Letters*, 10:335–347, 1989.

[89] G. Smith and G. Richards. A Report on Apple NIR Data for Sinclair International Ltd. *A Confidential, Commercial Report*, 2000.

[90] M. Smith and L. Bull. Feature Construction and Selection Using Genetic Programming and a Genetic Algorithms. In *EuroGP 2003, Proceedings of 6th European Conference on Genetic Programming, C. Ryan, T. Soule, E. Tsang, R. Poli and E. Costa (Eds)*, pages 229–237, Essex, UK, 2003. Springer-Verlag.

[91] M. Smith and L. Bull. Using Genetic Programming for Feature Creation with a Genetic Algorithm Feature Selector. In *In Parallel Problem Solving from Nature - PPSN VIII, X. Yao et al. (Eds)*. Springer-Verlag, 2004.

[92] S. Stanhope and J. Daida. Genetic Programming For Automatic Target Classification and Recognition in Synthetic Aperture Radar Imagery. In *Evolutionary Programming VII, Proceedings of the Seventh Annual Conference on Evolutionary Programming, V.W. Porto, N. Saravan, D. Waagen, and A.E. Eiben (Eds), Berlin*, pages 735–744. Springer-Verlag, 1998.

[93] W. Tackett. Genetic Programming for Feature Discovery and Image Discrimination. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 303–309. Morgan Kaufmann, 1993.

[94] D. Treigueiros and R. Berry. The Application of Neural Network Based Methods to the Extraction of Knowledge From Accounting Reports. In *Proceedings of 24th Annual Hawaii International Conference on System Sciences IV*, pages 137–146, 1991.

[95] H. Vafaie and K. DeJong. Feature Tpace Transformation Using Genetic Algorithms. *IEEE Intelligent Systems and their Applications*, 13(2):57–65, 1998.

[96] S. Weiss and N. Indurkhya. *Predictive Data Mining, a Practical Guide*. Morgan Kauffmann, San Francisco, California, 1998.

[97] S. Weiss and C. Kulikowski. *Computer Systems That Learn: Classification and Prediction Methods From Statistics, Neural Nets, Machine Learning and Expert Systems*. Morgan Kaufmann, 1991.

[98] R. Wirth and J. Hipp. Crisp-dm: Towards a Standard Process Model For Data Mining. In *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*, Manchester, UK, April 2000.

[99] I. Witten and E. Frank. *Data Mining: Pracitcal Machine Learning Tools and Techniques with JAVA Implementations*. Morgan Kaufmann Publishers, 2000.

[100] J. Wnek and R. Michalski. Hypothesis-Driven Constructive Induction in AQ17-HCI: A Method and Experiments. *Machine Learning*, 14:139–168, 1994.

[101] M. Wong and K. Leung. *Data Mining Using Grammar Based Genetic Progamming and Applications*. Kluwer Academic Publishers, USA, 2000.

[102] J. Wyatt and D. Altman. Prognostic Models: Clinically useful or Quickly Forgotten? *BMJ*, 311:1539–1541, 1995.

[103] J. Yang and V. Honavar. Feature Subset Selection Using a Genetic Algorithm. *IEEE Intellegent Systems*, 1998.

[104] X. Yao. Evolving Artificial Neural Networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[105] Z. Zheng. Constructing X–of–N Attributes for Decision Tree Learning. In *Machine Learning*, pages 1–43. Kluwer Academic Publishers, 1998.

Appendix A

This Appendix contains a description of each of the 5 public-domain data sets used in the experiments of Chapter 5. Each data set has a subsection with a brief description of the classification problem, and three tables containing the details of the data set. The fields of the tables are:

Dimensions: number of input attributes from which to predict the class label.

Classes: number of distinct classes.

Samples: number of examples in the database.

Preparation: steps carried out on the data to prepare it for use with GP and for the classification models.

Partition: number of samples of the training and testing sets, and the sampling method used for partitioning.

Source: where the data came from.

Missing values: describes how missing values, if any, were handled by the original collectors of the data.

The input attributes Table includes the name, type and range of values for each input attribute from the database. The class distribution Table includes the name

of each class, along with the number and proportion of samples from the database belonging to that class.

The Abalone Data Set

Predicting the age of abalone from physical measurements. The age (class) of abalone is determined by the long process of cutting the shell through the cone, staining it, and counting the number of rings through a microscope. Other measurements, which are easier to obtain, are used to predict the age. Further information, such as weather patterns and location (hence food availability) may be required to solve the problem.

From the original data, examples with missing values are removed (the majority having the predicted value missing), and the ranges of the continuous values have been scaled by dividing by 200.

Description of the data:	
Dimensions	8
Classes	28
Samples	4177
Preparation	For attribute 1, replaced M with 1, F with 2, and I with 3.
Partition	10–fold cross–validation
Source	UCI Machine Learning Repository
Missing values	None

The input attributes:			
Attr.	Name	Type	Values/Range
1	Sex	categorical	male, female, infant (converted to 1, 2 and 3)
2	Length	numerical	0.075–0.815
3	Diameter	numerical	0.055–0.65
4	Height	numerical	0.00–1.13
5	Whole weight	numerical	0.002–2.826
6	Shucked weight	numerical	0.001–1.488
7	Viscera weight	numerical	0.001–0.76
8	Shell weight	numerical	0.002–1.005

The class distribution:		
Class	Samples	Proportion (%)
1	1	0.0239
2	1	0.0239
3	15	0.3591
4	57	1.3646
5	115	2.7532
6	259	6.2006
7	391	9.3608
8	568	13.5983
9	689	16.4951
10	634	15.1784
11	487	11.6591
12	267	6.3921
13	203	4.8599
14	126	3.0165
15	103	2.4659
16	67	1.604
17	58	1.3886
18	42	1.0055
19	32	0.7661
20	26	0.6225
21	14	0.3352
22	6	0.1436
23	9	0.2155
24	2	0.0479
25	1	0.0239
26	1	0.0239
27	2	0.0479
29	1	0.0239
Total	4177	100.00

The Balance-Scale Data Set

This data set was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight, and the right distance. The correct way to find the class is the greater of (leftdistance × leftweight) and (rightdistance × rightweight). If they are equal, it is balanced.

Description of the data:

Description of the data:	
Dimensions	4
Classes	3
Samples	625
Preparation	Replaced letters: L with 0, B with 1, R with 2
Partition	10–fold cross–validation
Source	UCI Machine Learning Repository
Missing values	None

The input attributes:			
Attr.	Name	Type	Values/Range
1	Left-Weight	numerical	1, 2, 3, 4, 5
2	Left-Distance	numerical	1, 2, 3, 4, 5
3	Right-Weight	numerical	1, 2, 3, 4, 5
4	Right-Distance	numerical	1, 2, 3, 4, 5

The class distribution:		
Class	Samples	Proportion (%)
Left	288	46.08
Balanced	49	7.84
Right	288	46.08
Total	625	100.00

The Bupa Liver Disorders Data Set

Each line in the data constitutes the record of a single male individual. The first 5 variables are all blood tests which are thought to be sensitive to liver disorders that might arise from excessive alcohol consumption. The final attribute was changed to a nominal attribute by discretising the number of units consumed into two classes > 5 and $<= 5$.

Description of the data:	
Dimensions	6
Classes	2
Samples	345
Preparation	None
Partition	10–fold cross–validation
Source	UCI Machine Learning Repository
Missing values	None

The input attributes:			
Attr.	Name	Type	Values/Range
1	mcv (mean corpuscular volume)	numerical	65–103
2	alkphos (alkaline phosphotase)	numerical	23–138
3	sgpt (alamine aminotransferase)	numerical	4–155
4	sgot (aspartate aminotransferase)	numerical	5–82
5	gammagt (gamma–glutamyl transpeptidase)	numerical	5–297
6	drinks	numerical	0–20

The class distribution:		
Class	Samples	Proportion (%)
1 (> 5)	88	25.5
2 ($<= 5$)	257	74.5
Total	345	100.00

The Waveform Data Set

This example is a three class problem based on linear waveforms described in details by [16]. Each class consists of a random convex combination of two of these waveforms sampled at the integers with noise added. The attributes are the values of the measurement vectors (cases or instances).

Description of the data:	
Dimensions	21
Classes	3
Samples	5000
Preparation	None
Partition	10–fold cross–validation
Source	UCI Machine Learning Repository
Missing values	None

The input attributes:			
Attr.	Name	Type	Values/Range
1	f1	numerical	-3.34–3.94
2	f2	numerical	-3.25–3.88
3	f3	numerical	-4.20–4.72
4	f4	numerical	-3.84–5.75
5	f5	numerical	-3.48–6.50
6	f6	numerical	-2.76–7.62
7	f7	numerical	-3.32–8.76
8	f8	numerical	-3.52–7.84
9	f9	numerical	-3.38–7.90
10	f10	numerical	-1.79–7.63
11	f11	numerical	-1.48–9.06
12	f12	numerical	-1.69–7.40
13	f13	numerical	-2.61–7.50
14	f14	numerical	-2.82–7.75
15	f15	numerical	-2.56–8.72
16	f16	numerical	-2.99–7.86
17	f17	numerical	-3.56–6.74
18	f18	numerical	-4.08–6.20
19	f19	numerical	-3.50–5.28
20	f20	numerical	-3.57–4.65
21	f21	numerical	-3.88–4.01

The class distribution:		
Class	Samples	Proportion (%)
0	1657	33.14
1	1647	32.94
2	1696	33.92
Total	5000	100.00

The Wine Data Set

These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

Description of the data:	
Dimensions	13
Classes	3
Samples	178
Preparation	None
Partition	10–fold cross–validation
Source	UCI Machine Learning Repository
Missing values	None

The input attributes:			
Attr.	Name	Type	Values/Range
1	Alcohol	numerical	-3.34–3.94
2	Malic acid	numerical	-3.25–3.88
3	Ash	numerical	-4.20–4.72
4	Alcalinity of ash	numerical	-3.84–5.75
5	Magnesium	numerical	-3.48–6.50
6	Total phenols	numerical	-2.76–7.62
7	Flavanoids	numerical	-3.32–8.76
8	Nonflavanoid phenols	numerical	-3.52–7.84
9	Proanthocyanins	numerical	-3.38–7.90
10	Color intensity	numerical	-1.79–7.63
11	Hue	numerical	-1.48–9.06
12	OD280/OD315 of diluted wines	numerical	-1.69–7.40
13	Proline	numerical	-2.61–7.50

The class distribution:		
Class	Samples	Proportion (%)
1	59	33.15
2	71	39.89
3	48	26.96
Total	178	100.00