Feature Manipulation with Genetic Programming

by

Kourosh Neshatian

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2010

Abstract

Feature manipulation refers to the process by which the input space of a machine learning task is altered in order to improve the learning quality and performance. Three major aspects of feature manipulation are feature construction, feature ranking and feature selection. This thesis proposes a new *filter-based* methodology for feature manipulation in classification problems using genetic programming (GP). The goal is to modify the input representation of classification problems in order to improve classification performance and reduce the complexity of classification models.

The thesis regards classification problems as a collection of *variables* including *conditional* variables (input features) and *decision* variables (target class labels). GP is used to discover the relationships between these variables. The types of relationship and the ways in which they are discovered vary with the three aspects of feature manipulation.

In feature construction, the thesis proposes a GP-based method to construct high-level features in the form of functions of original input features. The functions are evolved by GP using an entropy-based fitness function that maximises the purity of class intervals. Unlike existing algorithms, the proposed GP-based method constructs multiple features and it can effectively perform transformational dimensionality reduction, using only a small number of GP-constructed features while preserving good classification performance.

In feature ranking, the thesis proposes two GP-based methods for ranking single features and subsets of features. In single-feature ranking, the proposed method measures the influence of individual features on the classification performance by using GP to evolve a collection of weak classification models, and then measures the contribution of input features to

the making of good models. In ranking of subsets of features, a virtual structure for GP trees and a new binary relevance function is proposed to measure the relationship between a subset of features and the target class labels. It is observed that the proposed method can discover complex relationships—such as multi-modal class distributions and multivariate correlations—that cannot be detected by traditional methods.

In feature selection, the thesis provides a novel multi-objective GP-based approach to measuring the goodness of subsets of features. The subsets are evaluated based on their cardinality and their relationship to target class labels. The selection is performed by choosing a subset of features from a GP-discovered Pareto front containing suboptimal solutions (subsets). The thesis also proposes a novel method for measuring the redundancy between input features. It is used to select a subset of relevant features that do not exhibit redundancy with respect to each other.

It is found that in all three aspects of feature manipulation, the proposed GP-based methodology is effective in discovering relationships between the features of a classification task. In the case of feature construction, the proposed GP-based methods evolve functions of conditional variables that can significantly improve the classification performance and reduce the complexity of the learned classifiers. In the case of feature ranking, the proposed GP-based methods can find complex relationships between conditional variables and decision variables. The resulted ranking shows a strong linear correlation with the actual classification performance. In the case of feature selection, the proposed GP-based method can find a set of sub-optimal subsets of features which provids a trade-off between the number of features and their relevance to the classification task. The proposed redundancy removal method can remove redundant features from a set of features. Both proposed feature selection methods can find an optimal subset of features that yields significantly better classification performance with a much smaller number of features than conventional classification methods.

Produced Publications

1. Kourosh Neshatian, Mengjie Zhang, and Mark Johnston. "Feature Construction and Dimension Reduction Using Genetic Programming". *Proceedings of the 20th Australian Joint Conference on Artificial Intelligence (AI'07), Lecture Notes in Artificial Intelligence*, Vol. 4830, Springer, Gold Coast, Australia, December 2007. pp 160-170.

2. Kourosh Neshatian and Mengjie Zhang. "Genetic Programming and Class-Wise Orthogonal Transformation for Dimension Reduction in Classification Problems". *Proceedings of the 11th European Conference on Genetic Programming (EuroGP 2008), Lecture Notes in Computer Science*, Vol. 4971, Springer, Napoli, Italy, March 2008. pp 242-253.

3. Kourosh Neshatian and Mengjie Zhang. "Genetic Programming for Performance Improvement and Dimensionality Reduction of Classification Problems". *Proceedings of the 2008 IEEE World Congress on Computational Intelligence (CEC'08)*, IEEE Press, Hong Kong, June 2008. pp 2811-2818.

4. Kourosh Neshatian, Mengjie Zhang, and Peter Andreae. "Genetic Programming for Feature Ranking in Classification Problems". *Proceedings of the seventh International Conference on Simulated Evolution and Learning (SEAL'08), Lecture Notes in Computer Science*, Vol. 5361, Springer, Melbourne, Australia, December 2008. pp 544-554.

5. Kourosh Neshatian and Mengjie Zhang. "Genetic Programming for Feature Subset Ranking in Binary Classification Problems". *Proceedings of the 12th European Conference on Genetic Programming (EuroGP 2009), Lecture Notes in Computer Science*, Vol. 5481, Springer, Tbingen, Germany, April 2009. pp 121-132.

6. Kourosh Neshatian and Mengjie Zhang. "Pareto Front Feature Selection: Using Genetic Programming to Explore the Feature Space[1]". *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO'09)*, ACM Press, Montreal, Qubec, Canada, July 2009. pp 1027-1034.

7. Kourosh Neshatian and Mengjie Zhang. "Unsupervised Elimination of Redundant Features Using Genetic Programming[2]". *Proceedings of the 22nd Australasian Joint Conference on Artificial Intelligence (AI'09), Lecture Notes in Artificial Intelligence*, Vol. 5866, Springer, December 2009. pp 432-442.

8. Kourosh Neshatian and Mengjie Zhang. "Dimensionality Reduction in Face Detection: A Genetic Programming Approach[3]". *Proceedings of the 24th International Conference on Image and Vision Computing*, IEEE Press, Wellington, New Zealand, November 2009. pp 391-396.

[1]The paper was nominated for the best paper award.

[2]The paper has received the best student paper award.

[3]The paper investigates one of the possible future directions of the thesis. The research is in a prelimenary stage, and is hence not included in the thesis.

Acknowledgments

I would like to express my gratitude to those who gave me the assistance and support I needed to complete this thesis.

My thanks goes to my advisers, A/Prof. Mengjie Zhang for his supervision, guidance and attention to detail throughout the course of my research, and Dr. Peter Andreae for his constructive and often motivating remarks and the many interesting discussions.

I must acknowledge the financial assistance of Victoria University of Wellington (Victoria PhD Scholarships and Marsden Fund VUW0806) without which the past three years would have been far more stressful and accompanied by far fewer Lattes and Mochaccinos.

And finally, my appreciation to all those, especially friends and fellow researchers from the School of Computer Science, who made my time at Victoria University of Wellington an experience that went beyond my expectations.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

This thesis proposes a new methodology that uses genetic programming for feature manipulation. Feature manipulation refers to the process by which the input space of a machine learning task is altered in order to improve the learning quality and performance. Alterations to the input space are made by means of constructing higher-level features, selecting informative features, and removing redundant or noisy features. Traditional solutions for feature manipulation such as linear transformation of the input space, single feature construction, individual feature ranking, and the likes are highly problem-dependent and domain-specific. They usually make assumptions that do not necessarily hold across different problems. The thesis addresses these issues by taking an evolutionary approach to searching through the space of functions and actions that can be applied to features using genetic programming.

1.1 Motivations

From an abstract point of view, machine learning solutions address two fundamental design commitments: representation and reasoning. A representation system provides a formalism to represent different aspects of the real world (problem domain) while a reasoning system deals with the

process of learning and predicting future states of the system [130]. Obviously, the quality of these two systems has a significant impact on the level of attainment in machine learning.

Inductive learning, where an agent learns from a set of observations, is a widely-practiced paradigm of machine learning. Increasing the quality of representation in this paradigm is usually carried out through some improvements to the input space [14]. In some learners, representational improvements are dealt with intrinsically as part of the learning process—for example, neural networks can implicitly build new features based on the input signals in their hidden layers. However, for many others like decision trees, there is no such implicit way of improving the representation as part of the learning process. For the latter category, therefore, explicit improvements to the input space are required to increase the learning performance.

The goal of *feature manipulation* is to improve the representation system by making changes to the feature space. Feature manipulation includes: feature construction, feature ranking and feature selection [89]. *Feature construction* is a means of enhancing the quality of representation by creating higher-level features as a function of the original features. *Feature selection* is the task of finding a minimal subset of the original features that is sufficient to describe the target concepts. Feature selection is a treatment for *the curse of dimensionality*. It leads to dimensionality reduction by eliminating unnecessary and redundant features from the problem, which in turn improves the learning performance. Learnt models induced by using smaller numbers of features are also easier to interpret. *Feature ranking* is an avenue to feature selection that imposes a ranking over features, representing their relative importance; the user usually chooses the desired number of features to be selected from a ranking scheme.

Genetic programming (GP) is an evolutionary search paradigm [71]. GP provides a flexible and expressive tool for dynamically building programs and functions. Given a set of primitive functions (or actions) and an objec-

tive function, GP is able to build different types of programs, ranging from mathematical expressions to complete classification models. This flexibility of GP in searching complicated search spaces motivates this paradigm a promising choice for non-trivial problems like feature manipulation.

There has been a growing trend in using GP for feature manipulation, particularly for feature construction, with very promising results. Unlike traditional feature construction algorithms—for example, principle component analysis—which come with certain assumptions and constraints and are limited to certain types of transformation, GP has been able to build a variety of transformations without being bound to any predefined templates. In the feature construction domain, GP has been successfully used to construct high-level features that boost the performance of classifiers. GP can be used in different feature construction scenarios: along with evolving a classifier [105, 80, 10], as a pre-processing phase [120], or in embedded solutions [30].

The current state of the art in using GP for feature manipulation is somewhat limited compared to the potential of GP. Most of the research in feature construction take a *wrapper* approach which is learner specific and computationally very intensive. Those research works that take a *filter* (non-wrapper) approach are limited to constructing one single feature [43, 104]. Research in feature selection and feature ranking using GP is quite young and limited to only a few works. Details of feature manipulation using GP are presented in Chapter 2. The aim of this thesis is to deal with the current challenges in this area.

1.2 Goals

The overarching goal of this thesis is to investigate a new approach to the use of GP for feature manipulation in classification tasks. This goal can be broken down to:

- using GP for feature transformation:

  - using GP for feature construction;
  - using GP for transformational dimensionality reduction;

- using GP for feature selection:
  - using GP for ranking single features and groups of features;
  - using GP for detecting feature redundancy.

Given a classification task presented with a set of *conditional features* (input features) and *decision features* (class labels), to achieve the above-mentioned goals, the thesis has to find possible answers to the following research question:

> **How can genetic programming be used to discover the relationships between the features of a classification task?**

When the goal is feature transformation, we need to use GP to discover the relationship between conditional features and decision features by constructing higher-level features. When the goal is feature selection, we need to use GP to discover the functional relationships between conditional features and decision features, and the mutual relationships amongst conditional features.

1.3 Major Contributions

The thesis has made the following major contributions:

- The thesis proposes a GP-based multiple feature construction system. Unlike many existing filter-based GP systems that can only construct a single feature, the proposed system is capable of making multiple high-level features without wrapping any other classification algorithms for fitness evaluation. The constructed features are evolved as functions of original features. A family of entropy-based

fitness functions are introduced which are used by the GP search. A new transformation technique that increases the chance of finding new high-level discriminating features is proposed. Our results on several benchmark problems show that the proposed method can significantly improve the classification performance of the problems while reducing the dimensionality and the complexity of the learnt models. These results have been partly published in [114, 109, 113].

- The thesis proposes a GP-based single feature ranking system. The system uses GP to find the relationship between features and target classes and then, based on the strength of the relationship, ranks individual features. The ranking provided by the system shows strong connections to the actual importance of features [108].

- The thesis proposes a GP-based system for measuring the relevance of subsets of features to target concepts in a binary classification task. A virtual program structure and an evaluation function are introduced in a way that constructed GP programs can measure the goodness of subsets of features. The proposed system can detect relevant subsets of features in situations where other ranking methods have difficulties, such as multimodal class distributions and mutually correlated features. The GP search results form a Pareto space in which feature selection is performed [110, 111].

- The thesis shows how GP can be used to find complex relationships between groups of features that cannot be found by traditional techniques. The method is used to measure the quotient of redundancy between features. We then introduce an algorithm that employs the GP-based redundancy detection system to perform feature selection by removing redundant features and irrelevant features [112].

1.4 Organisation of the Thesis

1.4.1 Structure

The main contributions of the thesis are presented in Chapters 3–7. Each chapter addresses some of the subgoals of the thesis by finding solutions for the central research questions of the thesis. All five chapters share the same high-level structure: each chapter starts by proposing some theoretical solutions followed by corresponding algorithms and diagrams. At the end of each chapter, the proposed system is tested and evaluated against a number of benchmark problems and the empirical results are analysed and discussed.

1.4.2 Outline

Chapter 2 carries out a review of the literature on feature manipulation, focusing on evolutionary approaches. The review covers the fundamental concepts of feature manipulation including feature construction, ranking and selection. It also visits the basics of evolutionary algorithms and genetic programming. It covers recent advances in feature manipulation using GP and discusses open questions and current challenges that form the motivations of the thesis.

Chapter 3 proposes a GP framework for constructing multiple high-level features. It investigates the notion of discriminative features and provides an entropy-based fitness function to measure this quality. It uses the proposed system to construct multiple features for some benchmark classification tasks and evaluates the system performance.

Chapter 4 proposes some advanced topics on feature construction that are used in transformational dimensionality reduction. It investigates how the construction process can be improved by enriching the variable terminal set of the GP search. Class-wise orthogonal transformation is introduced for making encapsulating features. A simplified fitness function is

proposed that makes the GP search over large input spaces computationally affordable. The empirical results are presented and discussed.

Chapter 5 proposes a GP-based method for single feature ranking. It introduces a scoring mechanism which is based on the frequency of the appearance of features in high-fitness GP programs. The scoring mechanism is then used to rank the individual (original) features. A number of classification tasks, and a variety of classifiers restricted to just high-rank features are tested and the performance is evaluated.

Chapter 6 proposes a new method for ranking and selection of subsets of features in binary classification problems. A virtual program structure and an evaluation function are defined in a way that constructed GP programs can measure the goodness of subsets of features. The outcomes of the GP search are presented in a Pareto space in which an optimum solution has a maximum relevance and a minimum cardinality. The chapter then investigate how the proposed ranking for each given subset of features is correlated to the actual classification performance using that subset. The performance of the system is then measured via measuring the classification performance using selected subsets of features.

Chapter 7 proposes an evolutionary way of feature selection by removing the redundant features from the result of a ranking algorithm. The chapter introduce a nonlinear redundancy measure which uses GP to find the redundancy quotient of a feature with respect to a subset of features. Then a forward selection algorithm is proposed which uses the proposed GP system as a redundancy measure. The effectiveness of the method is assessed by applying it to a dataset with a very large number of features.

Chapter 8 concludes the thesis by giving chapter-wise (goal-specific) conclusions and drawing overall conclusions regarding the research question. It also suggests some possible future research directions.

1.4.3 Navigation

To provide better navigation, Table 1.1 presents some information on the appearance of each of the aspects of feature manipulation and its dependency on chapters with the relevant background.

Table 1.1: The coverage and dependency of the content of the thesis

Chapter	Aspect of Feature Manipulation	Depends on
3	Construction	Chapter 2
4	Construction and Reduction	Chapters 2 and 3
5	Ranking	Chapters 2 and 3
6	Ranking and Selection	Chapters 2 and 5 (partly)
7	Selection	Chapters 2 and 5 (partly)

1.5 Benchmark Problems for Evaluation

Classification problems are the main applications to which the proposed methodology in the thesis can be applied. Therefore, evaluating the methodology involves testing it on a range of classifiers and benchmark classification problems. The choice of classifier and classification problem depends on the objective of the proposed algorithm. For example, in feature construction scenarios, the proposed algorithm is tested on a classifier that is not able to transform the input space by itself; in feature selection scenarios, where one needs to find a few good features from a large set of available features, the proposed algorithms is tested on classification problems that have a relatively large number of features.

Table 1.2 shows the classifiers and classification problems that have been used in the thesis. All the datasets are available from the UCI machine learning repository [6]. Appendix A provides detailed descriptions of the individual classification problems.

Table 1.2: The classification problems used throughout the thesis

Chapter	Classifier(s)	Benchmark Classification Problem(s)
3	Decision Tree (C4.5/J48)	Balance Scale, Glass Identification, Iris Plant, Liver Disorders, Pima Diabetes, Thyroid Disease, Wine Recognition, WBC-Original
4	Decision Tree (C4.5/J48)	JH Ionosphere, Sonar, Waveform, WBC-Diagnostic
5	Bayesian Net, Decision Tree (C4.5/J48), Naîve Bayes, SVM (SMO)	JH Ionosphere, Sonar, WBC-Diagnostic
6	Decision Tree (C4.5/J48), SVM (SMO)	JH Ionosphere, Sonar, WBC-Diagnostic
7	Decision Tree (C4.5/J48), SVM (SMO)	Isolet5

1.6 Notation

Throughout the thesis, we follow a certain mathematical notation. We use capital letters like X for random variables or when we are talking about features in abstract. Uppercase, boldfaced letters like $\mathbf{X}$ are used for matrices. Lowercase, boldfaced letters like $\mathbf{x}$ are used for vectors and $\mathbf{x}[i]$ rep-

resents the i-th element (or observation) of the vector. Calligraphic capital letters like $\mathcal{A}$ are used for sets. The unary operator $|.|$ is used to indicate the cardinality of a set. The list of special symbols used in the thesis follows:

- m The total number of original features in a classification problem.
- $m^\star$ The desired number of features to be selected.
- $\mathbb{F}$ The set of all original features in a classification problem; $|\mathbb{F}| = m$.
- n The number of instances in the dataset.
- L A scalar value showing the total number of classes (distinct class labels) in a classification problem[1].
- $\mathbb{C}$ The set of all class labels in a classification task; $\mathbb{C} = \{c_1, c_2, \ldots, c_L\}$ and therefore $|\mathbb{C}| = L$.
- $\mathbf{D}$ A dataset containing instances. Each instance has its values for the input features and the target class labels.
- ϕ A GP program that acts like a function; for example $y = \phi(x)$.
- I An interval of a class, where $I = (lower, upper)$ shows the lower and upper boundaries of the interval.

[1]This is the only exception to the convention of using lowercase letters for scalar values. The reason is that I has been frequently used in the equations and figures; a lowercase L could have been confusing, particularly in the figures.

Chapter 2

Literature Review

This chapter provides a review on the literature that forms the background and supports the motivations of the thesis. The chapter gives a brief introduction to Machine Learning and classification algorithms, the need for Feature Manipulation, and then an overview of Evolutionary Algorithms and Genetic Programming. The chapter, then, provides a detailed review of the literature on using Genetic Programming for feature manipulation. The review covers the potential and limitations of current methods for Feature Manipulation using Genetic Programming, which leads to the research direction adopted by the thesis.

2.1 Machine Learning

Machine learning is a major research area in *artificial intelligence* that is concerned with designing computer programs that are capable of learning in their environment [5, 14]. Machine learning systems are expected to be able to improve their performance as they gain more experience [100]. They should change their behaviour in a way that makes them act better in future [152]. Michalski et al. [94] state that:

> "Learning denotes changes in the system that are adaptive in

> the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time."

Machine learning algorithms use a feedback mechanism to change their behaviour (learn). Depending on the type of feedback, three cases can be distinguished: *supervised*, *unsupervised*, *reinforcement* learning [130]. In *supervised learning*, a set of examples in the form of different inputs and desired outputs are given and the goal is to learn a function that can do this input-output mapping. In *unsupervised learning*, only inputs are available; the learner has to find useful patterns in the input data. In *reinforcement learning*, the desired outputs are not directly provided; the learner instead has to learn based on rewards and punishments it receives for its actions (outputs).

2.1.1 Classification Algorithms

Inductive learning is perhaps the most common paradigm of learning. In inductive learning, learners generalise (from observations) patterns that can distinguish positive and negative examples. *Classification algorithms* are a major category of inductive learning algorithms. A *classifier* takes, as input, the description of an object and gives, as output, a label for the object. The set of class labels is defined as part of the problem (by users). A *classifier inducer*, is a supervised learning algorithm that uses a set of observations to learn a *hypothesis* (classifier) that can map inputs to correct class labels.

2.1.1.1 Training and Testing

The process by which a classifier inducer uses observations to learn a new classifier is called *training* [100]. During the training phase, the classifier inducer, is presented with observations from the problem domain called *instances*. The collection of instances used in the training phase is called the

training set. The algorithm learns important patterns in data by building models and adjusting the corresponding parameters. The performance of the algorithm is then tested on a collection of unseen instances, called the *test set*.

The learning capability of classifiers is usually evaluated by applying them to a set of benchmark problems. Benchmark problems are usually chosen from collections that are publicly accessible to researchers (e.g. UCI Machine Learning Repository [6]). Many benchmark problems do not have a specific test set. To evaluate the performance of a classifier on these problems, one should use k*-fold cross-validation* [100]. In k-fold cross-validation, the dataset is randomly partitioned into k folds (partitions). In a loop of k iterations, each time one of these folds is taken as the test set and the others are used together as a training set. The k results from the folds can then be averaged to produce a single estimate of classification performance. The advantage of this method over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once. In *stratified* k-fold cross-validation, the folds are selected so that the proportion of instances from different classes, remains the same in all folds.

2.1.1.2 Representation

Instances in the training and test sets are presented to algorithms using a *representation* system. In the majority of learning algorithms including classification algorithms, the quality of the representation is of key importance. The most common representation system is *feature-value*. In this system, each instance is represented in the form of a vector of values for the features defined in the problem domain. The datasets (including training and test set) are usually represented in the form of a table where each row is an instance and each column represents a different feature in the problem domain. The quality of the features defined in the problem domain, including their number and their relevance to the desired task, has

a significant effect on learning performance.

2.1.1.3 Decision Tree Classifiers

Decision tree learning is a method for approximating discrete-valued functions [100]. Decision trees classify instances by sorting them down the tree from the *root* to some *label nodes*. The tree is a hierarchy of nodes. For a given instance, the process starts at the root node; the value of the feature at the root node is tested and the process moves to one of the child nodes. Then the process is repeated for the subtree rooted at the new node.

There are different algorithms for learning a decision tree but the principles are the same [100]. The main question in learning a decision tree is which feature should be tested at each node of the tree. Most algorithms employ a top-down greedy search through the space of possible decision trees. Examples are the *ID3* algorithm [126], the *C4.5* algorithm [127], and its Java version, the *J48* algorithm [152]. These algorithms use an entropy function to measure the homogeneity of examples and choose the best node at each stage. The most important advantage of decision tree classifiers is their interpretability; learned decision trees can be translated to a set of 'if-then' rules to improve human readability. The most serious disadvantage of decision trees is perhaps their weakness in separating non-rectangular areas in the input space [127].

2.1.1.4 Support Vector Machines

Support vector machines (SVMs) form a category of statistical supervised learning algorithms. SVMs construct a number of hyperplanes in a high- or infinite-dimensional space, which are used for classification. Instances are categorised based on what side of these hyperplanes they fall on. SVMs maximise the distances between the hyperplanes and both the nearest positive and negative data points. The points that cause the boundary (hyperplane) to fix in a particular place are referred to as *support vectors*, and

a learning machine that uses such a boundary is therefore referred to as a support vector machine. The space between the boundary and the support vectors is called the *margin* [148].

2.1.1.5 Bayesian Classifiers

Bayesian classifiers provide a probabilistic approach to classification. Their assumption is that the behaviour of data (input-output relationships) can be captured in probability distributions [100]. Among these classifiers, *Naïve Bayes* classifiers are the most common and straightforward classifiers to learn. It has been shown that Naïve Bayes classifiers are quite competitive with other classifiers such as decision trees and neural networks [95]. Naïve Bayes classifiers make significant use of the assumption that all input features are conditionally independent. This assumption cannot be applied to many real world problems where there are some interdependency between input features. *Bayesian networks* have been proposed as a remedy to this problems [48, 54]. Bayesian networks allow conditional independence assumptions that only apply to a subset of features.

2.1.1.6 Other Classification Techniques

In addition to the above-mentioned classification algorithms—which are used in the experiments throughout the thesis—there are many other classification algorithms that are commonly used in data mining [100]. Two other important categories of classifiers are Artificial Neural Networks (ANNs) [13] and Case-based Reasoning (CBR) systems [1]. In ANNs, the information (usually in the form of numeric values) is transformed as it travels through the layers of the network. In classification problems, the network acts as a function which maps observations input space to target class labels. CBR systems are categorized as *lazy* learners because they do not induce any generalisation of training data until a query is received.

2.2 Feature Manipulation

Feature manipulation is an umbrella term that refers to the collection of methodologies and techniques that are practised to improve the input space of problems represented in feature-value systems [89]. This section reviews the most widely-known aspects of feature manipulation, namely *feature construction*, *feature ranking*, and *feature selection*.

2.2.1 Fundamental Concepts

This subsection first explains some basic concepts that are shared among all aspects of feature manipulation.

Definition A *feature* is a function that maps entities to one of their properties.

In this definition, entities are objects (observations) of the same type[1] and a feature represents a certain measurable property of the objects. Examples for objects of the same type are 'Ann', 'Ben', and 'Colin', all being from the 'Student' type. Examples for features are 'Height' and 'Gender' which correspondingly map these objects to numeric (the height of the person) and nominal (the gender of the person) values.

2.2.1.1 Basic Operations

All aspects of feature manipulation use one of the two following basic operations to make changes in the feature space of a problem.

Transformation. This process transforms the values of one or more features to a new set of values. The transformation functions are usually well-defined and deterministic. Examples of this operation are feature construction and transformational dimensionality reduction.

[1] In the context of learning by example and feature manipulation, the terms *observation*, *sample* and *instance* are often used with the same meaning in the literature.

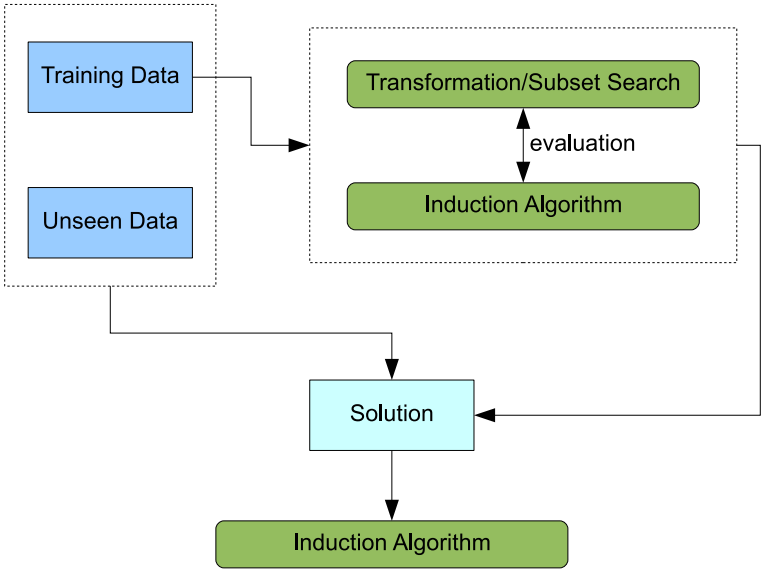


Figure 2.1: A feature manipulation system taking a wrapper approach.

Selection. This process selects a subset of available features in a problem. The selected features are usually used for both the training and the testing of classification algorithms.

2.2.1.2 Wrapper vs Filter Approach

In all feature manipulation problems, when a candidate solution is found, it should be evaluated to determine its goodness and find new search directions. For example, one has to know how much relevant information a set of constructed/selected features can provide. There are two major approaches to evaluating a solution: *wrapper* and *filter* (or non-wrapper) [66].

Figure 2.1 shows the diagram of a feature manipulation system taking a wrapper approach. In the wrapper approach, the performance of an induction algorithm (e.g. a classifier) is used to guide the search. The wrapper approach is computationally intensive; every evaluation involves training and testing an induction algorithm.

The filter approach on the other hand, does not use any learner's feedback to evaluate a solution. It instead uses other heuristics that are computationally more efficient. The diagram of the filter approach is very similar to that of the wrapper approach depicted in Figure 2.1. However, no induction algorithm is used to evaluate the solution.

2.2.2 Feature Construction

Many classification algorithms, particularly those based on symbolic learning (e.g. *decision rules* and *decision trees*), cannot achieve adequate predictive performance when faced with difficult real-world problems [77]. A known reason for this deficiency is the inability of these systems to make any transformations to their input spaces [100]. The issue can be partially alleviated by using feature construction as preprocessing.

2.2.2.1 Classical Methods for Feature Construction

Zheng [158] provides a review of *constructive induction* methods. In constructive induction, the original features are transformed into a new space in a way that the learning performance is improved [159]. Inductive logic programming is used to construct features that can model the behaviour of data. The newly constructed features can then be used by learning algorithms. The constructed features can also provide some structural information [142]. Hu [51] proposes a multi-strategy constructive inductive algorithm which is independent of learning algorithms.

2.2.3 Feature Selection

There are different definitions for feature selection in the literature [20]:

- Idealised: feature selection is defined to be the process of finding the minimally sized feature subset that is necessary and sufficient to model the target concept [63].

- Classical: feature selection is the process of selecting $m^\star$ features from m original features, such that $m^\star < m$ and the value of a criterion function is optimised over all subsets of size $m^\star$ [107].
- Improving predictive accuracy: feature selection is the process of finding a subset of features, using which either predictive performance is improved or the complexity of the model is reduced while the performance is maintained at an acceptable level [67].
- Approximating original class distribution: feature selection is the process of finding a subset of features such that the resulting class distribution, given only the selected features, approximates the original class distribution as closely as possible [67].

Overall, feature selection is the process of finding a minimal subset of features that is sufficient to solve a classification problem. Feature selection leads to dimensionality reduction by eliminating noisy and unnecessary features from the problem, which in turn improves the performance and makes the learning and execution processes faster. Models constructed using a smaller number of features are also easier to interpret.

2.2.3.1 Wrappers for Feature Selection

The search space of a feature selection problem has 2^m points where m is the number of original features in the problem. The search space grows exponentially with respect to m. Some wrapper approaches to feature selection use an external algorithm to explore this search space. The type of search algorithm could be anything from simple Hill-climbing to an evolutionary search [66]. The search can be towards growing an initial subset (e.g. *forward selection*) or towards shrinking an initial solution (e.g. *backward elimination*) [96].

Searching the collection of all 2^m possible combinations of features is computationally infeasible when m is large. Even if an algorithm does not

search the whole space exhaustively, as m grows, it needs to examine more points in order to find a near-optimal solution. In wrapper methods, evaluation of candidate solutions is costly—each evaluation needs a classifier to be trained and tested. Therefore, using wrapper methods on problems with a large number of original features is not always viable.

2.2.3.2 Filters for Feature Selection

Feature selection methods taking the filter approach use only data to find an optimal subset of features; they do not wrap any inductive learning algorithm (e.g. a classifier) to evaluate their solutions. *FOCUS* is a classical filter-based feature selection algorithm that was originally defined for noise-free Boolean domains [3, 4]. It exhaustively examines all subsets of features, selects the minimal subset of features that is sufficient to determine the label value for all instances in the training set.

The *Relief* algorithm is another filter method that assigns a "relevance" weight to each feature [64]. The algorithm attempts to find all relevant features. The Relief algorithm, however, does not help with redundant features [68]. Cardie [16] proposes a filter-based feature selection algorithm that uses a decision tree algorithm to select a subset of features for a nearest neighbourhood algorithm. Yu [155] proposes a feature selection algorithm that takes both relevance and redundancy into account. The algorithm, however, is limited to problems that only have discrete features.

2.2.4 Feature Ranking

Feature ranking is an avenue to feature selection [60]. In feature ranking, a score is assigned to each solution [45]. In single (univariate) feature ranking, a score is associated to each feature individually and independently from other features [129]. In single feature ranking, the user selects a number of high-rank features. Normally, the number is specified by the user [46]. There are also some analytical methods to determine the best number

of features [143].

Most feature ranking methods fall into the filter approach category, and can only measure the goodness of a single feature [129, 12, 88]. This includes all feature ranking measures from the information theoretic domain such as information gain (IG), gain ratio, mutual information and the likes [86].

2.2.4.1 Issues with Epistatic Features

Epistasis, a term originally from biology, is defined as interaction between genes [8]. It is used to describe how one gene can change (suppress or express) the phenotypical effect of another gene. Epistasis later entered Genetic Algorithms (GAs) and other computational evolutionary paradigms to indicate how changing a component of a candidate solution—for example changing a bit in a GA chromosome or changing a subtree in a GP program—can change the behaviour of other components in the solution [38, 150].

Epistasis happens frequently between the features of a classification task; that is, the contribution of a feature in predicting the class label will depend on the value of some other features. Many filter methods have difficulties in handling *epistatic* features. The difficulties are twofold:

- The majority of filter methods cannot provide any explicit way of measuring the goodness of a group (subset) of features. These methods are usually combined with a search technique to select a set of top-ranked features. However, since the features are examined individually, the selected subsets often suffer from the absence of groups of related features and the presence of redundant features.

- The majority of filter methods are limited to detecting only simple types of relationships between a feature and the target class. For example, in the logistic regression model [18], the relationship is assumed to be linear; in most of the information theoretic measures,

it is assumed that instances can be classified by setting a split point along the feature axis.

2.2.5 Transformational Dimensionality Reduction

Although, in a sense, all feature selection algorithms perform dimensionality reduction, the term dimensionality reduction is most often used to refer to *transformational dimensionality reduction*. In transformation dimensionality reduction, the original features are transformed into a new space (new features). Then a small number of these transformed features is used instead of the original features [147, 9, 128]. A successful reduction in dimensionality can help in building simpler classification models. The transformations can also be useful for interpretation.

Principle component analysis (PCA) is one of the dimensionality reduction techniques that is widely used in different applications. The goal of PCA is to linearly transform data into a more meaningful construct [37]. It can eliminate the redundancy between measurements (features), and reduce the noise by selecting more important components. This is done by diagonalizing the covariance matrix. However, as PCA is blind to the class labels in the training set, in many cases, it is not effective for classification problems. Another potential drawback of PCA is that, it makes the assumption that more diversity along the axis of a generated component (feature) is a sign of being more informative and therefore it ranks generated components based on this factor. However, this assumption is certainly not always true.

From a different perspective, the problem of dimensionality reduction can be seen as a feature construction problem in which the constructed features are functions of the original features and the total number of constructed features is sufficiently smaller than the number of original features in the problem. For example, the PCA method can be treated as a feature construction scenario in which all the constructed features are lin-

ear expressions and the objective is to find the coefficients of these polynomials so that PCA goals are satisfied. From this perspective, a limiting issue of PCA and many other classical dimensionality reduction methods is that they all have fixed models (e.g. linear, polynomial). These methods can only find the optimal value for the *parameters* (e.g. the coefficients in a polynomial model); they cannot find the right model for data by themselves [91, 17].

2.3 Genetic Programming

This section first gives an overview of evolutionary computation and hierarchy of algorithms in this field, and then provides a more detailed review of fundamental concepts in genetic programming.

2.3.1 Overview of Evolutionary Computation

Evolutionary Computation (EC) is an area of artificial intelligence that covers the majority of nature-inspired algorithms in this field. Two main classes of these algorithms are evolutionary algorithms and swarm intelligence.

2.3.1.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) refers to a subset of algorithms in evolutionary computation that are generic population-based metaheuristic optimisation algorithms. EAs use mechanisms inspired by biological evolution: reproduction, mutation, recombination, and selection. Each *individual* (member of the population) is a candidate solution. The goodness of individuals is determined by a *fitness* function. Evolutionary algorithms have been highly successful in solving complex problems in science and engineering [23]. Some important algorithms in this category follow.

Genetic Algorithms. *Genetic Algorithms* (GAs) are evolutionary search and optimisation techniques [39, 38, 49]. GAs evolve a population of chromosomes that can encode solutions in continuous and discrete domains. Compared to some analytical optimisation methods like gradient-based optimisation, they are less likely to be trapped in local optima. They, however, tend to be computationally expensive.

Evolutionary Programming. In *Evolutionary Programming* (EP) a population of chromosomes is used to evolve *finite-state machines* (FSMs) [34]. Each FSM is in fact a program. A sequence of symbols that have been observed up to the current time is fed to each FSM. The fitness of an individual is evaluated by its ability in predicting future symbols. Like other EAs, EP uses fitness values to select individuals and then applies some evolutionary operators to find other solutions. EP has been perhaps one of the first attempts to evolve computer programs. The structure of programs in EP is usually assumed to be fixed.

Evolutionary Strategies. In *Evolutionary Strategies* (ESs), each individual represents a fixed-length real-valued vector. The real values in the vector are parameters to a system/model that determine its behaviour. As with the bit-strings of GAs, each position in the vector corresponds to a feature of the individual. However, the features are considered to be behavioral rather than structural. Since all elements are real-valued, genetic operators can perform operations like averaging [141]. The selection of survivals in ESs is deterministic; that is, once the genetic operators are applied, a number of individuals with highest fitness are selected for the next generation.

Genetic Programming. Genetic programming (GP) is a sophisticated EA which is used to evolve a computer program that performs a desired task [71, 74, 73, 75]. GP has been highly successful as a technique for getting computers to automatically solve problems without having to tell them

explicitly how. It has proved to be applicable and effective in a variety of fields from planning and discovery of game-playing strategies to symbolic regressions and evolving classification systems.

2.3.1.2 Swarm Intelligence

Swarm Intelligence (SI) algorithms are inspired by the collective intelligence of social insects. SI systems are typically composed of simple interacting individuals and the intelligence lies in the networks of interactions among individuals, and between individuals and the environment [15]. Two main algorithms in SI are *ant colony optimisation* (ACO) and *particle swarm optimisation* (PSO). ACO is a class of optimisation algorithms modeled based on the behaviour of ant colonies [24]. ACO methods are useful in problems that need to find paths to goals. PSO is a simple search method in which solutions are represented by inter-communicating particles [62]. Particles are influenced by (and can influence) their neighbouring particles and *global* best-performing particles, depending on the topology. Like many other EC algorithms, PSO is derivative-free and does need specific information about the problem domain.

2.3.2 GP Algorithm

GP optimises a population of computer programs according to a fitness function that determines a program's ability to perform a given computational task [124]. The search space is explored using genetic operators [84]. Apart from the representation of genetic programs, the overall search process in GP is similar to other population-based EAs. The main steps in GP are as follows [7]:

1. *Initialise* a population of individual programs as solutions;
2. Assign a *fitness* to each individual program in the population;
3. While the *termination criterion* is not met, repeat the following:

(a) Select some individuals using a *selection method*;

(b) Produce new individuals by applying *genetic operators* to selected members;

(c) Place new individuals into the population of the next generation;

(d) Assign a fitness to each individual program in the population according to the fitness function;

4. Return the program with the highest fitness as the best solution.

2.3.3 Program Representation

The *phenotype* of each chromosome in a GP population is a program. The program is usually executed by a program interpreter. The *genotype* of a GP program (i.e. the way it is encoded) varies among different GP systems. The most common way of encoding a GP program is through using a tree structure [19, 70]. Other common representations include *Linear GP* [7, 115, 116, 52], *Cartesian GP* [97, 99, 98] and *Grammatical GP* [119, 151].

In tree-based GP, a program is represented by a hierarchy of nodes [36, 85]. Each node is either a *function* or a *terminal* [72]. Function nodes perform an operation. The children of a function node are the arguments of that function. Terminal nodes, as the name implies, do not have any children. They are either *variable terminals* which provide inputs to programs or *constant terminals* which are randomly generated values [7]. The set of all types of function nodes available to GP is called the *function set*. The set of all possible variable terminals is called the *variable terminal set*. In *strongly-typed GP*, functions and terminals can have different types and the GP search must take care of building valid program trees [101, 151]. Figure 2.2 shows a sample tree-based genetic program using elementary arithmetic functions and numeric terminals.

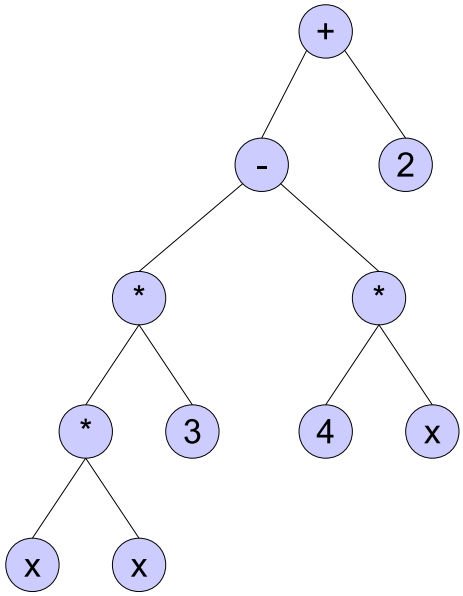


Figure 2.2: A sample tree-based genetic program representing the mathematical expression $3x^2 - 4x + 2$. The function nodes are addition, subtraction and multiplication. The only variable terminal node is x and constant nodes are 2, 3 and 4.

2.3.4 Creating Initial Populations

The first step in starting a GP search is to create an initial population. The initial population is usually created randomly. There are three commonly-used ways of creating the initial population, namely the *Grow* method, the *Full* method, and the *Ramped half-and-half* method [7].

In the Grow method, all nodes are chosen randomly, and therefore, the resulting program trees have very different shapes. To create an initial population using the Grow method, the following steps should be taken:

1. A node is randomly selected from the union of the function set and the terminal set as the root node;

2. Given n, the *arity* of the selected function, n functions or terminals are selected as child nodes;

3. For each child that is a function node, the previous step is repeated until *the maximum tree depth* is reached in which case the remaining child nodes are filled with terminals.

In the Full method, all the nodes except those at the maximum depth are functions. Therefore, the full capacity of the tree is used. The Ramped half-and-half method is used to enhance the diversity. In this method, half of the population is created using the Full method and the other half using the Grow method.

2.3.5 Genetic Operators

Genetic operators make changes to individuals in the population. They are the primary way of moving in the search space of programs. All the genetic operators work at the genotype level. Therefore, their implementation highly depends on the representation of GP programs [7].

2.3.5.1 Crossover

The primary function of the crossover operator is to share genetic material between individuals in the population. The crossover operator is typically applied to two individuals, called *parents*, and creates two new individuals as the result, called *children*. In the canonical tree-based representation [71], crossover is performed by simply swapping two randomly chosen subtrees in the parent programs. Figure 2.3 demonstrate this concept with an example. To the top of the figure, there are two GP individuals. One node is randomly selected in each individual. The subtrees at these nodes are then exchanged and two new individuals are created towards the bottom of the figure.

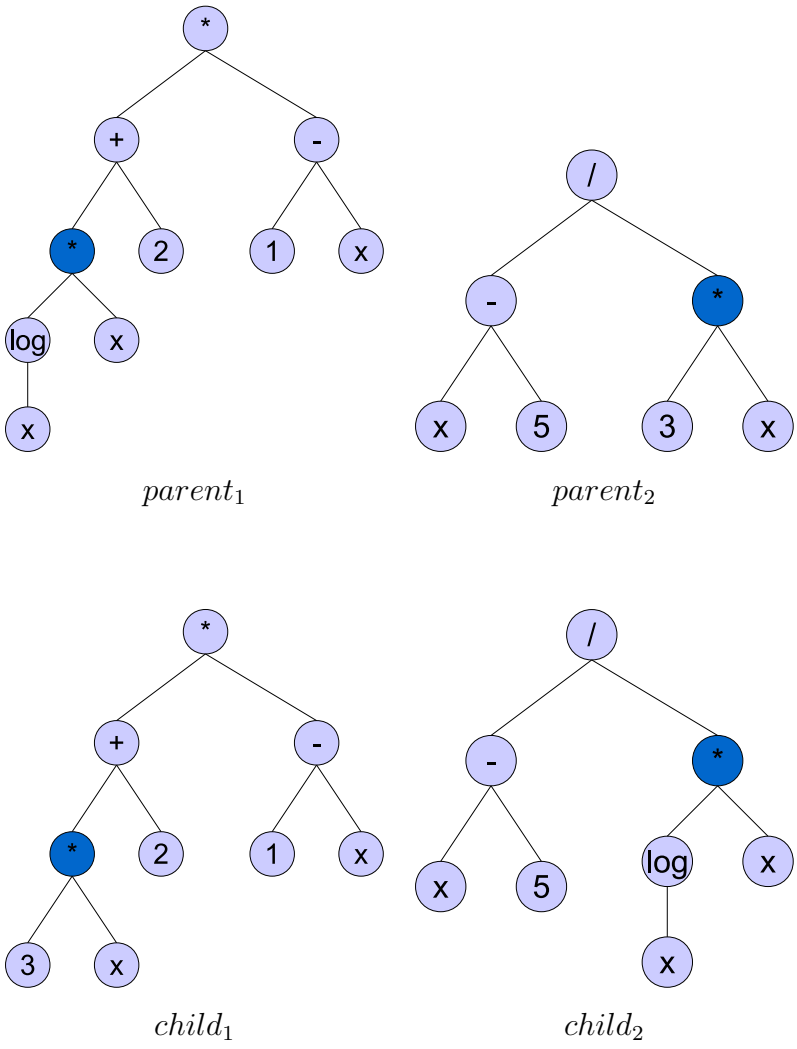


Figure 2.3: Crossover Operator in Tree-based GP: The operator is applied to two GP individuals, $parent_1$ and $parent_2$. Two nodes are randomly selected and their corresponding subtrees are exchanged. The results are two new individuals, $child_1$ and $child_2$.

2.3.5.2 Mutation

The primary function of the mutation operator is to bring new genetic material to the population. The operator is typically applied to one individual

at a time. It performs by randomly selecting a node in the tree and then replacing the subtree at that node with a randomly-created subtree. Figure 2.4 shows an example of applying the operator to a GP individual. In strongly-typed GP [101], to maintain integrity, the operation must be fail-safe—that is, the type of arguments of the parent node of the randomly selected node must be taken into account to create a random subtree of the correct type.

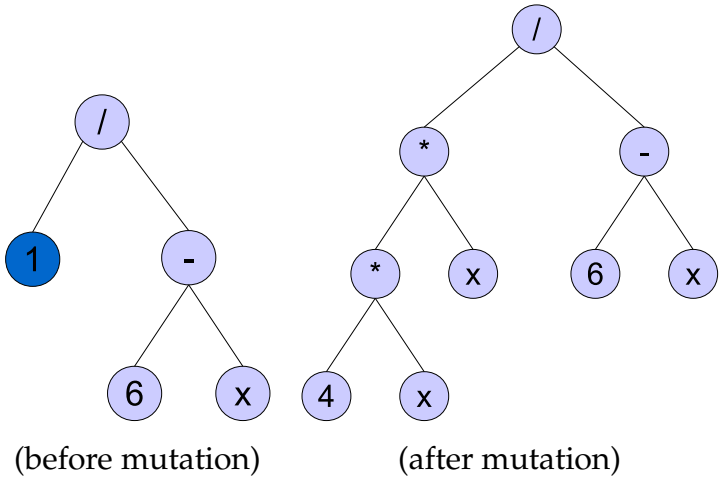


Figure 2.4: Mutation Operator in Tree-based GP: The operator is applied to the GP individual on the left. A node is randomly selected and then the subtree at that node is replaced with a new randomly-created subtree.

2.3.5.3 Reproduction

Reproduction is perhaps the most straightforward operator amongst others. It simply clones (copies) a GP individual to create a new one [124]. The reproduction operator is useful in two cases: i) for random preservation of genetic material—that is, some GP programs are randomly selected (using the selection operator) and copied to the next generation; ii) for elitism in which case the best performing individual(s) is (are) copied to the next

generation to make sure that the performance does not drop during evolution.

2.3.6 Fitness Function and Selection Mechanism

Fitness is a measure that determines the goodness of an individual (GP program) with respect to one or more objectives [7]. Fitness might show the quality/performance of a program in absolute terms or relative to other programs. The fitness of a program affects the probability of its selection and survival.

A GP algorithm might be used with different selection methods. In *Fitness-Proportional Selection*, the probability of selection of each individual is proportional to its fitness [7]. In other words, the fitter (better) a program, the more likely its selection.

In *Tournament Selection*, the competition is not among all individuals in the population but between a small set of randomly sampled individuals in the tournament [7]. This method has a parameter called *tournament size*. Given a tournament size t, t individuals are randomly sampled from the population. The individual with the highest fitness is then selected to be used with genetic operators. When t is 1, the selection mechanism becomes purely random—fitness is not considered. A large tournament size reduces the probability of selection of weak individuals. When t equals the population size, the selection mechanism becomes deterministic.

2.4 GP for Feature Manipulation

Evolutionary algorithms, particularly genetic algorithms (GAs), have been successfully used for feature selection problems. GA and neural networks have been used to rank input features in classification problems [88, 154]. Hybrid GA with local search operations has produced good results in feature selection [117]. GA has also been used to select features for Support

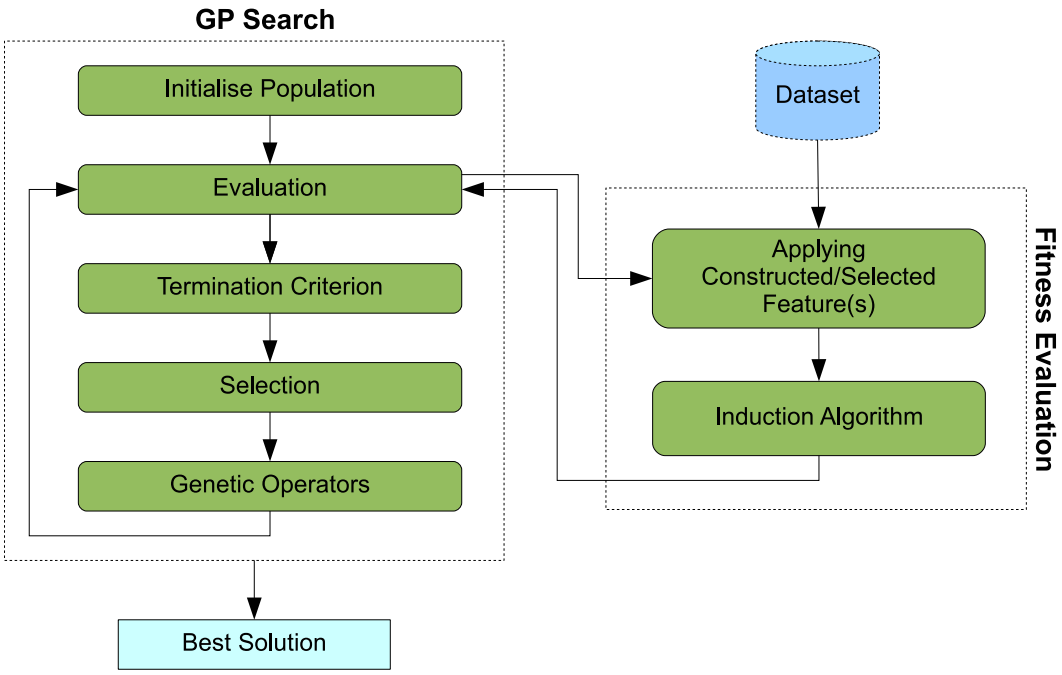


Figure 2.5: The outline of the feature creation process using GP.

Vector Machines [35].

The capability of GP in dynamically building logical and mathematical expressions [69] and classification models [65, 122, 134, 156, 157] has made it particularly a good choice for feature manipulation. Recently, there has been a new research trend in using GP for feature manipulation. In this section, we review some state-of-the-art developments in this area and some open problems which form the motivations of this thesis.

2.4.1 A Generic Outline

Figure 2.5 shows an overall architecture that is commonly used for feature manipulation using GP [11, 76, 82, 79]. The figure shows a wrapper system and the goal is to improve the performance of an inductive learner. The fitness function (the right block) measures the performance by training and testing the desired induction algorithm.

The GP component of this architecture is a typical evolutionary search module. It starts with initialising a new population of programs (solutions). Each solution in the population suggests a feature manipulation procedure (e.g. constructing or selecting features). A fitness value is assigned to each individual in the population through the *fitness evaluation* process. The evaluation process applies the individuals to the original dataset (genotype-phenotype mapping) and then uses the result to train and test an induction algorithm. The individuals are then ranked by the calculated fitness. Once the fitness values are assigned, the selection and other genetic operators are applied to the individuals to navigate the search.

2.4.2 GP for Feature Construction

GP has been used for making high-level features in the form of functions of original features. The GP-constructed features have been used to transform the input space of classification and object detection tasks.

Based on the application domain, the input space transformations might happen in different representation systems and have different objectives. Two common feature construction scenarios are: i) feature construction for classification problems represented in attribute-value system; ii) feature construction for object detection problems represented in raster graphics. In attribute-value representations, constructed features are scalar functions of the original features [43, 104, 30]. In the raster graphics representation (image processing and machine vision domain), constructed features are image filters (operators) acting on low level raw images [53, 56, 77, 80].

In terms of whether a classification algorithm is embedded into the system, research in using GP for feature construction can be divided into two areas: i) adopting a wrapper approach; that is, GP is used in conjunction with a target classifier [66]; ii) adopting a filter approach—that is, the fitness function does not depend on any other classification algorithm.

2.4.2.1 Wrapper Approaches to Using GP for Feature Construction

In some wrapper methods, each chromosome encodes only one candidate feature (constructed feature). These methods can create only a single feature [32]. In some others, each chromosome is an array of program trees, each of which represents a single constructed feature. The outcome of GP is a winning chromosome, which is a structure consisting of several constructed features (program trees) [76, 102, 137]. Another wrapper method for creating multiple features is to create multiple concurrent populations of features, and then, conduct a co-evolutionary search to find the optimal subset of chromosomes [10, 78, 81, 79].

As a sub-category of the wrapper approach, GP has been used as a complementary tool in the learning process of a classifier. It can be used in conjunction with a decision tree inducer to construct more discriminative features for decision stubs [26]. In the signal classification domain, GP has been used to provide synthetic artificial features for the k-nearest neighbour classifier [30, 31, 133] as well.

2.4.2.2 Filter Approaches to Using GP for feature Construction

In the filter approach, no classification algorithm is involved in the evaluation of constructed features and therefore, the search process is expected to be more efficient and the results are expected to be more general [136]. However, the requirement for a problem-independent and classifier-independent measure for the goodness of constructed features makes designing the fitness function a challenging task. Information theoretic measures like information gain (IG) and information gain ratio (IGR) [125, 92] have been used as fitness functions in filter-based GP systems for feature construction [120, 103, 104]. Fisher's distance has been another alternative for a fitness function [44, 50, 43].

Using functions like IG and IGR for comparing the goodness of constructed features, one could only tell which feature is better at splitting

up the data instances. When functions like these are used for fitness evaluation, the result of evolution is a constructed feature that provides the highest information gain; repeated GP runs (with different random seeds) produce very similar constructed features. This means that these methods can create only one feature per classification problem. Since using a single feature is not normally enough to achieve an acceptable classification performance, the constructed feature is often added to the set of original features and they are all fed to the classification algorithms [120, 104].

2.4.3 GP for Feature Selection

There are two categories of research work on GP for feature selection. In the first category, a filter-based selection method is used to remove some variable terminals from the search [106] or to bias the probabilities of selection of variable terminals [22, 118]. In this category, feature selection is used as an internal process to help the GP search achieve its objectives.

In the second category, GP is primarily used to evolve classification systems. The evolved classifiers are then analysed to find the features that have been used by the classifier [42, 83, 87, 139, 149]. The presence of a feature in a well-performing evolved classifier is considered selection. In a multi-objective approach, GP has been used to evolve classifiers with two objectives: maximising the classification performance and minimising the number of features being used in the classifier [105]. In that approach, the selection is in favour of subsets with smaller numbers of features.

2.5 Summary and Discussion

Genetic programming is a flexible and expressive tool in dynamically building mathematical models based on an objective function. GP expressions are not bound to any predefined template; they can have any type (linear, non-linear, trigonometric, etc.) given that an objective function is satisfied.

This feature has made GP an excellent choice for feature manipulation. Research in the area of using GP for feature manipulation has been rapidly growing. Despite recent developments in this area, there are still several open issues to be addressed:

Filter Approach to Multiple Feature Construction. When a wrapper approach is taken, GP systems for feature construction suffer from intense computation. In the filter approach, existing systems are limited to constructing one feature per classification problem. Adopting a filter approach to using GP for *multiple* feature construction is still an open issue.

Dimensionality Reduction. Existing filter-based GP systems for feature construction cannot achieve dimensionality reduction. Since a single feature (the output of current systems) is not typically enough to have an acceptable classification performance, the constructed and the original features are used together—that is, the existing filter-based methods, in fact, cause a slight increase in the dimensionality of problems. Taking a filter approach to using GP for dimensionality reduction is still an open issue.

Feature Ranking and Selection. Although GP has been successful in implicit feature selection, its potential for explicit feature ranking and selection has not yet been explored.

The next few chapters focus on proposing new GP methods that can address the above-mentioned issues.

Chapter 3

Multiple Feature Construction

3.1 Introduction

From an abstract viewpoint, there are two foundational design commitments for machine learning solutions: *representation* and *reasoning*. The quality of these two has a significant impact on the learning performance. The goal of *Feature construction* is to improve the quality of representation by transforming the input space using a set of one or more constructed features. In some learning systems, this can be achieved intrinsically as part of the learning process; for example, neural networks can implicitly build new constructed features in their hidden layers. In some other learners, like decision tree, original features are used directly, and this can present a problem [127]. Providing such learners with a higher quality representation requires an explicit external feature construction process.

3.1.1 Defining Feature Construction

Although there is, more or less, a consensus on the definition of feature construction in the existing literature, we give a formal definition for constructed features to avoid any ambiguity in the thesis.

Definition A *constructed feature* is a scalar function ϕ that transforms the

input space to a one-dimensional real value space. Given $(X_1, X_2, \ldots, X_m)$, the random vector corresponding to the set of original features in a classification problem, a constructed feature is a function of the form $\phi(X_1, X_2, \ldots, X_m)$. The term *Feature construction* refers to the process of producing constructed features.

3.1.2 The Appropriateness of Genetic Programming

Since constructed features are in fact mathematical (or logical) expressions of the original features, the capability of genetic programming (GP) in dynamically building programs and expressions, based on an objective function, makes it an excellent choice for automatic construction of new features. A critical design issue in using GP is to choose an appropriate fitness function. In the context of feature construction, there are two approaches to measuring the fitness of a GP-constructed feature: *wrapper* approach and *filter* (non-wrapper) approach.

3.1.3 Wrapper Approach vs Filter Approach

In the wrapper approach, the performance of another machine learning algorithm (usually a classification algorithm) is used as an indicator for the appropriateness of a constructed feature. For each fitness evaluation, the constructed feature (usually together with the original features) is fed into the classifier, then the classification performance is used to calculate the fitness of the constructed feature. Since every fitness evaluation involves training a classifier and then testing its performance, the search process is computationally very intensive.

In the filter approach[1], instead of wrapping a particular classifier in the fitness function, the fitness of an individual is evaluated by a function that acts as a surrogate classifier. The filter approach has some advantages over

[1]In this thesis, in parallel with the literature, we use the terms *non-wrapper* and *filter* interchangeably.

the wrapper approach. Since fitness evaluation does not involve training and testing a classifier, the search process can be performed faster or the gained computation time can be spent on exploring more candidate solutions (constructed features). Furthermore, as no particular classification algorithm is used in fitness evaluation, the constructed features are expected to be more general. However, designing a fitness function that is easy to evaluate and at the same time general enough to be applicable for different problems, is quite challenging.

While existing non-wrapper methods enjoy efficiency and generality, they have a drawback. Almost all the existing filter-based fitness functions have a fixed formulation (with no parameter) for evaluating the goodness of a feature. Therefore, given a fixed fitness function and a fixed dataset even multiple GP runs tend to converge to the same solution. This means that these methods can create only one feature per classification problem [see Section 2.4.2.2, page 34]. Designing a filter-based GP system that constructs multiple features is an open research question to be answered.

3.1.4 Chapter Goals

The research goal of this chapter is to devise a GP system to construct multiple high level features while adopting a non-wrapper approach. The central issue in this goal is to propose a non-wrapper measure to evaluate the goodness of features.

3.2 Developing a Measure of Goodness

The first step in constructing a feature is, of course, to have an idea of what constitutes a good feature and how one could measure this worth. To evaluate the goodness of a (constructed) feature, we need to find out how it can contribute to learning a good classifier. There are two main approaches to evaluating a feature. One approach is based on evaluating a feature in the

context of other features, taking account of the effect of other features. We will discuss this approach in feature selection topics. The other approach is based on the direct influence of the feature on learning quality, regardless of the presence or absence of other features. Unlike the former approach, the evaluation of a (constructed) feature regardless of the quality of other features can be achieved using simple models which are computationally cheap to build and evaluate. In this chapter, we take the latter approach, that is, evaluating the goodness of a standalone feature.

The goodness of a feature can be defined and measured in many different ways. For example for a C4.5 decision tree, a good feature is the one that can maximise the information gain (IG) [see Chapter 2]. In principle component analysis (PCA), a good feature (principle component) is the one with higher deviation. In the same way, from a statistical perspective, a good feature might be the one with higher correlation with the target class. Finally, in a wrapper approach a good feature is one that can improve the classification performance [see Chapter 2]. In fact, depending on one's perspective, the application domain and type of data, and the type of classification algorithm that is going to be used, the definition of a good feature may be different. However, despite the variation in the way the goodness of features is measured, a feature considered good based on one of these measures is usually good enough to satisfy a large group of classifiers.

3.2.1 Decision Stump and Its Limitation

Our study on finding a non-wraper measure for the goodness of individual features starts with decision stumps. A *decision stump* is a simple machine learning model that is constructed by comparing the value of a feature against a constant value called *split point*. In more concrete terms, a model of the form "`if` $x < \alpha$ `then A; otherwise B`" is a decision stump which checks the feature x against the split point α and decides

whether to take the action A or B. These actions might be assigning (predicting) a class label or branching to other decision stumps.

Decision stumps are not usually used on their own because many real world classification problems are not solvable using a single decision stump. However, they constitute the building blocks of a category of classification algorithms called *decision trees*[2]. Assuming that the optimal split point can be found by a helper algorithm, the learning performance of a decision stump can be used as a measure of class separability which in turn can be used as a measure for the goodness of a feature. In fact, the majority of filter-based measures, particularly those coming from information theory like Information Gain (IG) and Gain ratio, share the same basis.

Figure 3.1 illustrates three cases in which a decision stump has been used to judge the quality of a feature. Each case is a binary or multi-class classification problem with positive and negative instances represented along a feature x. The judgement is based on how the value of the feature can assist in separating negative and positive instances from each other. In case (a), positive and negative instances are mixed together along the feature axis. Even the best split point cannot separate the instances of the two classes, causing the performance of the decision stump to be quite low. The poor performance of a decision stump on this feature is considered as an indication of the low quality of the feature (assuming that the feature is used alone). In case (b), a split point can perfectly separate positive and negative instances indicating the high quality of the feature.

In case (c), although the instances are spread out in a clearly-distinguishable pattern, one split point is not enough to separate all the instances and as a result a decision stump would perform quite poorly on this feature. In this case, the feature is obviously good—there is only one chunk of negative instances which, can easily be separated by two split points. However, as one single split point does not provide enough discrimination, using a decision stump model, the quality of the feature is considered low. The

[2]A hierarchy of decision stumps in the form of a tree, makes a decision tree.

situation presented in case (c) is actually quite common in classification tasks—the distribution of a class is surrounded by the distribution of other classes.

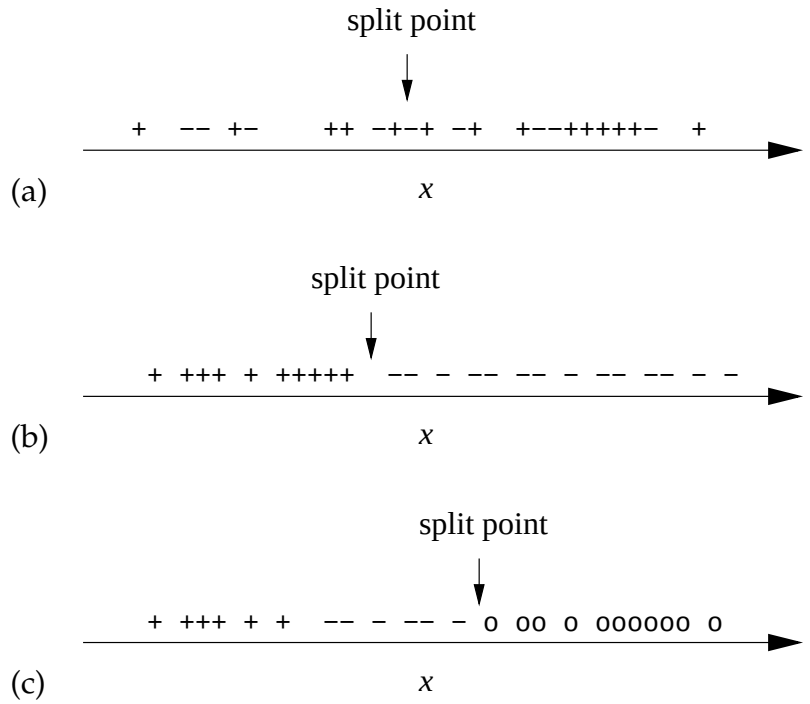


Figure 3.1: The goodness of a feature x form the viewpoint of a decision stump: (a) the feature is poor and positive and negative instances cannot be separated using a split point; (b) the feature is good and the instances are separable using the split point; (c) the feature is good, but one split point is not enough to separate positive and negative instances.

3.2.2 Extending Decision Stumps to Class Intervals

To address the issues like the one depicted in Figure 3.1(c) where a simple decision stump is not able to determine the goodness of a feature, a more sophisticated model is needed. Sophistication however, although it might

compensate for the limitation of decision stumps, might compromise the simplicity and efficiency of the calculation as well. Here, the aim is to steer a middle course; we propose a measure that can address the above-mentioned issue without much computational overhead.

We introduce the idea of class intervals. In abstract terms, a class interval along a feature is the span in which the instances of that class are scattered. We shall later give a more concrete definition of class intervals but first we see how the concept of class intervals can improve our judgment about features. Figure 3.2 shows a single feature in three different cases where the distribution of positive and negative instances matches those in Figure 3.1. Class intervals along each axis are rough areas with the most occurrence of the instances of that class. The dotted double arrow lines on top of the feature axis show the class intervals along each axis.

In Figure 3.2(a), where instances of the positive and negative classes are all mixed together, there are two overlapping intervals, indicating that the feature (on its own) cannot be used to separate the instances. In case (b) there is a clear boundary between the two classes and consequently, there are tow non-overlapping intervals. In case (c), where the decision stump method was not able to evaluate the goodness of the feature using one split point, there are three non-overlapping class intervals which is an indication of a good discriminating feature.

It is observable that when class intervals overlap, they contain instances from the other class (e.g. case (a)). By contrast, non-overlapping intervals are quite good at separating instances from different classes. This suggests that the quotient of overlap between class intervals could be a good indication for the goodness of a feature. Overlap can be indirectly measured by taking account of the occurrence of the other class instances in the interval of one class. This is in fact a measure of *purity*. A pure class interval contains a minimum number of instances from the other classes. In other words, a good feature has a pure class interval. Therefore, to make this measure quantitative, one should model two components: a class interval

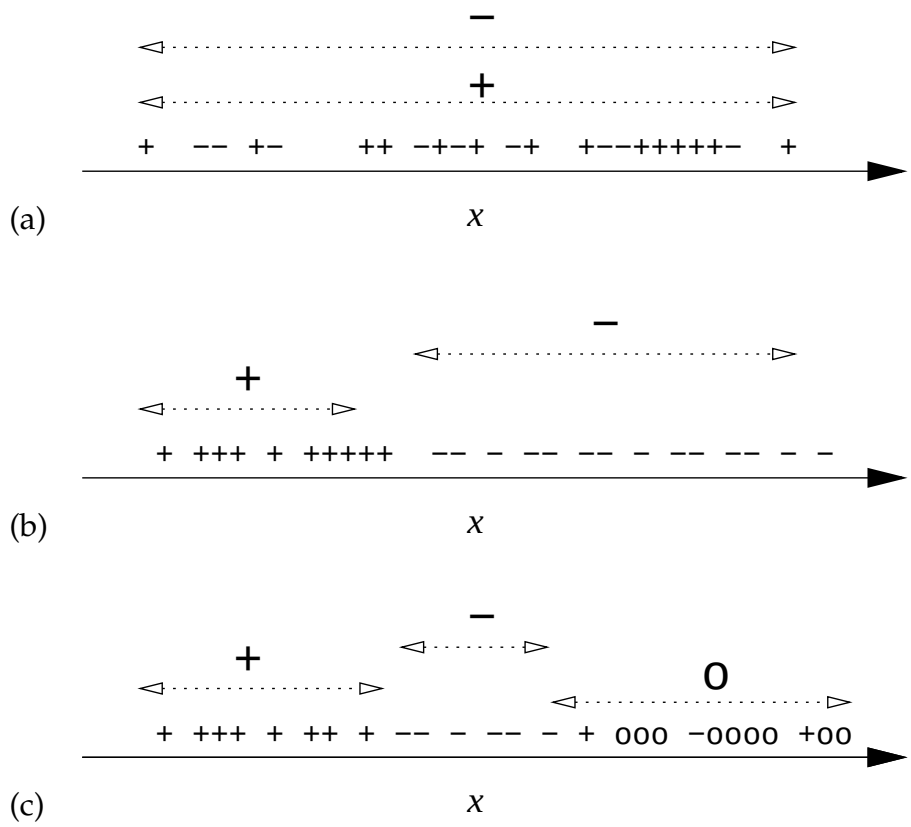


Figure 3.2: The goodness of a feature x from the viewpoint of a class interval: (a) the feature is poor and the intervals are completely overlapping; (b) and (c) the feature is good and the instances can be separated using class intervals.

and a purity measure.

3.3 Proposing a Non-Wrapper Fitness Function

To have a quantitative measure for the goodness of features, the above-mentioned abstract concept of pure class intervals should be expressed in more concrete terms. This measure, can then be used as a fitness function

in a GP search to evaluate the goodness of a candidate constructed feature. In this section, we address the problem of quantifying the purity of a class interval by decomposing the problem into finding a class interval and then measuring its purity.

3.3.1 Class Intervals: A Mathematical Model

The interval of a class along a feature is determined by the dispersion of the instances of that class along the feature axis. The dispersion of instances itself is related to the distribution of data points in that class. An interval I is represented with a pair $(lower, upper)$ which shows the lower and upper boundaries of the interval. I_c is used to indicate an interval for class c. In this section, we define an interval for two different cases: a) the distribution of the class is normal, and b) the distribution of the class is unknown.

3.3.1.1 Intervals of Classes with Normally Distributed Instances

Assuming that the distribution of a class along a feature x is normal, the mean and standard deviation of the distribution can be used to find the boundaries of the interval. Given μ_c and σ_c , the estimated mean and standard deviation of a normally distributed class along feature x, the following interval covers 99% of the class instances:

$$I_c = [\mu_c - 3\sigma_c, \mu_c + 3\sigma_c] \tag{3.1}$$

In other words, the mean of the class is the center of the interval and the standard deviation of the class determines the width of the interval. The values of μ and σ can be estimated using a set of observations.

3.3.1.2 Intervals of Classes with Unknown Distributions

Class instances do not necessarily follow a normal distribution in all classification problems and there might not be enough observations and com-

putation time to discover whether their distribution is normal. Besides, even if a class is normally distributed along one of the original features in the problem, a non-linear constructed feature can easily transform the input space in a way that changes the class distribution.

A simple solution to this problem is to consider a class interval as the smallest interval that can cover the maximum and minimum of the class data points. This solution, however, might include undesired outliers[3]. A primitive way to deal with the issue of outliers is to exclude a few observations from both ends of the data range. In other words, all the instances should be sorted along the new constructed feature and then a small portion of them should be removed from both the left and right sides. Since sorting is a relatively costly algorithm (at least for a fitness function), we simplify this even further by performing only a partial sort by which the first half of the data points before the first percentile and the second half of the data points after the last percentile can be excluded so that the resulting interval covers 99% of samples. The details of determining the class interval in this way will be given when we give the fitness function algorithm.

3.3.2 Purity Measure: A Mathematical Model

In the previous subsection, a mathematical model for a class interval was defined. To quantify the purity of the interval to determine the goodness of the associated feature. Information theory has a measure of purity called *information entropy*, which is commonly used by decision trees. Entropy is mainly used to measure the information content (aka *uncertainty*) of a communication channel. To use entropy for measuring the purity of a class interval, one must see the interval as an information channel where different symbols (class labels) may occur with different probabilities. In

[3]Outliers are noisy samples (caused by measurement error, etc) which are not based on the real characteristic of the class distribution

feature construction, we are looking for class intervals with a very low entropy—that is, the instances of other classes are unlikely to occur in the interval.

3.3.2.1 Using Shannon's Entropy

The most common way of measuring entropy is perhaps *Shannon's entropy*. Given a discrete or categorical random variable C that can take values $c_1, c_2, \ldots, c_L$ with probabilities $p(c_1), p(c_2), \ldots, p(c_L)$, the Shannon entropy of C is defined by

$$H(C) = -\sum_{i=1}^{L} p(c_i) \log_b p(c_i) \tag{3.2}$$

where b is base of the logarithm and is usually 2. A class interval establishes a new probability space. Therefore, the probability of classes in equation 3.2 should be conditioned on the values of the feature that fall in the interval. Given X, a feature, $\mathbb{C}$, the set of all class labels, and $c^\star$, the class of interest with corresponding interval $I_{c^\star}$, the Shannon entropy of the interval of class $c^\star$ is

$$H(I_{c^\star}) = -\sum_{c \in \mathbb{C}} p(c|X \in I_{c^\star}) \log_2 p(c|X \in I_{c^\star}) \tag{3.3}$$

The conditional probability of classes can be estimated by using a set of observations and measuring the frequency of each class in the interval. Since a lower entropy implies higher purity in an interval, the lower the entropy in the interval, the better the quality of the feature.

Figure 3.3 illustrates the concept of entropy in a class interval. The interval for class '+' is represented by a rectangle which includes the majority of instances from this class. Since the interval includes only a few instances from other classes (low entropy), it is considered a fairly pure interval, indicating that the feature x can be used to discriminate the instances. If a class interval includes a lot of instances from other classes, then there is not a good separation between classes, meaning that the feature cannot be used (individually) to separate the classes.

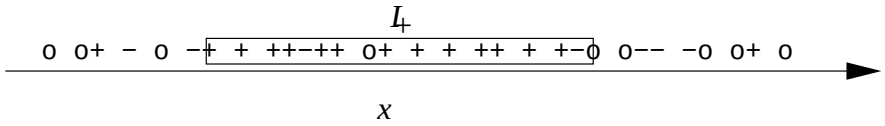


Figure 3.3: An example feature x and an interval for class $+$. The class interval creates a new probability space in which the Shannon entropy is measured.

3.3.3 The Fitness Function

In feature construction, GP individuals are interpreted as mathematical expressions that map the original features to a constructed one-dimensional feature. To run the GP process, a fitness function is required to evaluate the goodness of constructed features based on their discrimination power. The result of the evaluation, depending on the selection method, is used in ranking or selecting the individuals.

The proposed measure in equation (3.3) is one way (out of possibly many) of looking at the characteristics of good features. According to the measure, the entropy of a class interval indicates the probability of occurrence of instances from other classes in the interval. The measure is not meant to be used directly on the original features, as in real world problems a single original feature can hardly discriminate instances of a class completely. However, the measure can establish a fitness landscape for GP-constructed high level features which are in fact a combination of several original features. The measure can guide the search towards constructing features that are more discriminative.

Given a GP individual, there are two main steps in calculating its fitness: a) finding an interval for the class for which the individual has been created, and b) measuring the entropy of the interval.

3.3.3.1 Finding a Class Interval: The Algorithm

The steps toward finding an interval for a class have been laid out in Algorithm 1. Given a desired class label for which the interval should be found, the algorithm finds the lower and upper boundary of an interval that contains 99% of the class instances. The first half of the data points before the first percentile and the second half of the data points after the last percentile are excluded to compensate for possible outliers. During the course of a GP run, the algorithm is called (as a function) by the fitness function to find the requested class interval.

The algorithm takes, as input, $\mathbf{y}$ the values of a constructed feature (genetic program) and $\mathbf{c}$ its corresponding vector of class labels. Assuming there are n samples in the training dataset, the size of these two vectors is n. In other words, the two vectors constitute a dataset with two columns and n rows. At row i, $\mathbf{y}[i]$ is the value of the constructed feature and $\mathbf{c}[i]$ is its corresponding class label. $c^\star$ is the class label for which a high-level feature is being constructed, thus the algorithm should find the interval of class $c^\star$. The output of the algorithm is a pair (l, u) indicating the lower and upper boundaries of the interval.

To find the lower and upper boundaries, the algorithm finds 99% of the instances of class $c^\star$ that fall in the middle of the range of all the instances of class $c^\star$. In other words, the algorithm excludes 1% of instances of the class with extreme values—that is, 0.5% of instances having the lowest and 0.5% of instances having the highest values.

To exclude the extreme values the algorithm defines two sets $Left$ and $Right$ that store the lowest and highest values of the instances of class $c^\star$ correspondingly. The sets are initialised with one element each: $+\infty$, the highest possible values for $Left$, and $-\infty$, the lowest possible value for $Right$. The sets can grow to have up to $\lceil \frac{n_{c^\star}}{200} \rceil$ elements (0.5% of instances). The loop at line 4 iterates over all the instances and fills these two sets. The loop resembles a partial sorting by the end of which 0.5% of instances having lowest and 0.5% of instances having highest values are stored in

Algorithm 1: $Find\text{-}Interval(\mathbf{y}, \mathbf{c}, c^\star)$

```
/* Given a set of observations (instances) of a
   single scalar feature along with their class
   labels, and a desired class label, the
   algorithm finds an interval that covers 99% of
   instances in the middle.                      */
```

Input: $\mathbf{y}$, a vector of n observations of a feature
Input: $\mathbf{c}$, a vector containing the class label of each observation in x
Input: $c^\star$, the class label for which the interval should be found
Output: (l, u), a pair indicating the lower and upper boundaries of the interval

$n_{c^\star} = |\{\mathbf{c}[i] : i \in \{1, 2, \ldots, n\}, \mathbf{c}[i] = c^\star\}|$; // no. of $c^\star$ instances
$Left \leftarrow \{+\infty\}$; // left half percentile
$Right \leftarrow \{-\infty\}$; // right half percentile
for $i \leftarrow 1$ **to** n **do**
  if $\mathbf{c}[i] = c^\star$ **then**
    if $\mathbf{y}[i] < \max Left$ **then**
      if $|Left| \geq \lceil \frac{n_{c^\star}}{200} \rceil$ **then**
        $Left \leftarrow Left \setminus \{\max Left\}$;
      $Left \leftarrow Left \cup \{\mathbf{y}[i]\}$;
    if $\mathbf{y}[i] > \min Right$ **then**
      if $|Right| \geq \lceil \frac{n_{c^\star}}{200} \rceil$ **then**
        $Right \leftarrow Right \setminus \{\min Right\}$;
      $Right \leftarrow Right \cup \{\mathbf{y}[i]\}$;
$(l, u) \leftarrow (\max Left, \min Right)$;
return (l, u);

$Left$ and $Right$. Once $Left$ and $Right$ are determined, the lower and upper boundaries of the interval can be obtained by finding the maximum

value in $Left$ and the minimum value in $Right$.

3.3.3.2 Fitness Evaluation: Measuring the Entropy

The next step in determining the fitness of a GP program (constructed feature) is to find the purity of the obtained interval through the proposed entropy measure. A GP program, as discussed earlier, can be thought of as a function of a number of variables (the original features). The function maps the original features to a single feature, called the constructed feature. The value of the constructed feature is evaluated at the root of the corresponding GP program.

Algorithm 2 depicts the process of evaluating the fitness of a GP program. We use ϕ to represent the program for which the fitness should be evaluated. The program maps m (or less) original real-valued input feature vectors to a scalar (one-dimensional) feature vector. This is done in the first few lines of the algorithm where each example in the training set (stored in $\mathbf{X}$) is transformed to a new vector, $\mathbf{y}$. The values of the constructed feature $\mathbf{y}$ along with the vector of class labels $\mathbf{c}$, constitute a new dataset with 2 columns, including 1 high-level feature and n observations. Once the new dataset is ready, if instances of class $c^\star$ can be well-discriminated from the others, then ϕ (representing the newly-constructed feature) is good and should receive a better fitness (low entropy).

At line 2 the algorithm finds the interval of class $c^\star$ using Algorithm 1. Then we need to know the probability of the occurrence of each class in the interval. We use the prior probability of each class that is estimated by the frequency of the occurrence of each class label in the interval. To keep track of occurrences of different class labels within the interval, we use the vector $\mathbf{c}'$. On lines 7–8, the frequency of each class is calculated by dividing the number of occurrences of that class in the interval of class $c^\star$ by the total number of instances within the interval. The fitness is then calculated on lines 9–11. If most of the instances falling into an interval belong to a single class (the class for which the interval has been found),

Algorithm 2: $MFC\text{-}Fitness(\mathbb{X}, \phi, c^{\star})$

```
/* Given a training dataset, a desired class label
   for which a feature is being constructed, and a
   GP program (a candidate feature), the algorithm
   evaluates the fitness of the GP program.      */
```

Input: $\mathbf{D}$, a dataset of the form $\mathbf{D} = (\mathbf{X}, \mathbf{c})$ where $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$ is a set of vectors of length n containing samples from the m original features in the problem and $\mathbf{c}$ is a vector of class labels for the corresponding observations in $\mathbf{X}$

Input: ϕ, a GP program, which acts as a function $\mathbb{R}^m \mapsto \mathbb{R}$

Input: $c^{\star}$, the label of the class for which a feature is being constructed

Output: $fitness$, a real value showing the fitness of the program (the lower, the better and the minimum is zero)

$\mathbf{y}[i] \leftarrow \phi(\mathbf{x_1}[i], \mathbf{x_2}[i], \ldots, \mathbf{x_m}[i])$, $\forall i \in \{1, 2, \ldots, n\}$;
// transformation
$(l, u) \leftarrow Find\text{-}Interval(\mathbf{y}, \mathbf{c}, c^{\star})$;
$\mathbf{c}' \leftarrow ()$; // an empty vector
for $i \leftarrow 1$ **to** n **do**
    if $\mathbf{y}[i] \in (l, u)$ **then**
        $\mathbf{c}' \leftarrow (\mathbf{c}', \mathbf{c}[i])$;
foreach $label \in \mathbb{C}$ **do**
    $p_{label} = \frac{|\{i:\ i \in \{1,2,\ldots,|\mathbf{c}'|\},\ \mathbf{c}'[i] = label\}|}{|\mathbf{c}'|}$; // frequency of class
$fitness \leftarrow 0$;
foreach $label \in \mathbb{C}$ **do**
    $fitness \leftarrow fitness - p_{label} \log(p_{label})$;
return $fitness$;

then the fitness will be quite low. Therefore, the smaller the fitness, the better the program, and consequently the better the constructed feature.

3.4 A GP System for Feature Construction

Now that the main component of a GP-based feature construction system, the fitness function, is available, we propose our GP system by which multiple features can be constructed.

3.4.1 System Diagram

Figure 3.4 shows a top level view of the proposed GP-based multiple-feature construction system. The GP system uses the training data to construct a set of high-level features. The constructed features specify how the input space should be transferred. They are used to transform the training and test data. The transformed data is used to induce and test a new classifier. The original data (features) might be fed to the classification algorithm as well in the form of an augmented dataset.

3.4.2 The GP Search

The main body of our GP-based feature construction algorithm is presented in Algorithm 3. The algorithm constructs one feature per class label in the problem. The input to the algorithm is a dataset with m original feature vectors $\mathbf{x_1}$ to $\mathbf{x_m}$ and one decision variable vector (the target class) $\mathbf{c}$ which takes its values from $\mathbb{C} = \{c_1, c_2, \ldots, c_L\}$. We use $\mathcal{F}$ to denote the set of constructed features. At the beginning of the algorithm $\mathcal{F}$ is empty, but by the end of the algorithm it will contain L constructed features, one for each class in the problem.

The outer loop in the algorithm iterates over all the class labels in the problem. For each class label, a separate *GP run* is conducted and the

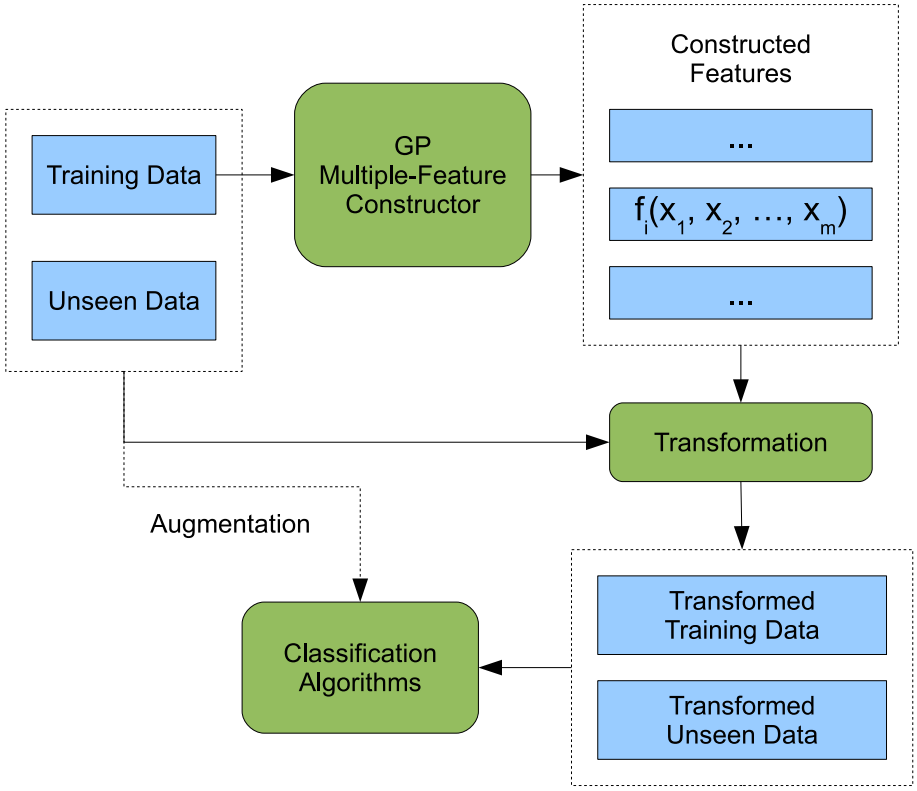


Figure 3.4: The system diagram of the proposed GP-based multiple-feature construction system and its relation to a classification algorithm.

resulting program (the best constructed feature) is added to $\mathcal{F}$. In each GP run, the fitness function focuses on one specific class label, optimising GP programs to best separate the instances of that class from others. The best program is the one with the lowest fitness value (i.e. with the lowest entropy or maximum purity). The algorithm keeps track of the best program by updating the value of the variable $best\text{-}fitness$. The inner loop, implementing the GP search, will terminate either when the maximum number of generations is reached or when the best possible fitness, zero, is achieved.

Algorithm 3: $GP\text{-}Multiple\text{-}Feature\text{-}Constructor(\mathbf{D})$

```
/* Given a training dataset the algorithm uses GP
   to construct as many features as the number of
   class labels in the problem.  Each constructed
   feature is a GP program which acts as a
   function to transform the input space.        */
```

Input: $\mathbf{D}$, a dataset of the form $\mathbf{D} = (\mathbf{X}, \mathbf{c})$ where $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$ is a set of vectors of length n of observations of the m original features in the problem and $\mathbf{c}$ is a vector of class labels for the corresponding observations in $\mathbf{X}$

Output: $\mathcal{F}$, the set of constructed features

```
1  F ← {} ;      // initialising the set of constructed features
2  foreach c* ∈ ℂ = {c_1, c_2, ..., c_L} do
3      P ← create a new initial population;
4      best-fitness ← +∞; // initialising the best fitness to be
       the worst possible fitness.
5      while ¬max-generations ∧ best-fitness ≠ 0 do
6          foreach φ ∈ P do
7              φ_fitness ← FindFitness(𝕏, φ, c*);
8              if φ_fitness < best-fitness then
9                  best-program ← φ;
10                 best-fitness ← φ_fitness;
11         perform selection;
12         perform genetic operators;
13     F ← F ∪ {best-program};
14 return F;
```

3.5 Empirical Results

3.5.1 Design of Experiments

The performance of the proposed GP-based multiple feature construction system is measured by conducting two major experiments: In the first experiment, we measure the changes in classification performance by augmenting the original dataset—that is, by adding the newly constructed features to the original dataset. In the second experiment, the classification performance is measured after feeding only the constructed features to the classifier. We also carry out two minor experiments, one to study the effect of using the constructed features on the size of decision tree classifiers, and the other to analyse how and why a constructed feature can actually improve the classification performance. The details of the experiments follow.

3.5.1.1 Datasets

We use 8 classification datasets from the UCI machine learning repository [6]. They include binary and multiple class classification tasks. Table 3.1 summarises the main characteristics of these datasets. A common property of these datasets is that they only have numerical features. This is because we define a constructed feature to be a mathematical function of the original features. In some datasets—for example, the Wisconsin Breast Cancer dataset (WBC-Original)—instances with missing values have been removed [see Appendix A].

3.5.1.2 GP Settings

We use the standard tree-based GP model [71]. In this model, each program produces a single floating-point number at its root for each observation in the dataset. Table 3.2 shows various settings of the GP system we developed for the experiments. The four standard arithmetic operators

Table 3.1: Specification of datasets used in experiments.

Problem	# Features	# Instances	# Classes
Balance Scale	4	625	3
Glass Identification	9	214	6
Iris Plant	4	150	3
Liver Disorders	6	345	2
Pima Diabetes	8	768	2
Thyroid Disease	5	215	3
Wine Recognition	13	178	3
WBC-Original	9	683	2

were used to form the function set. The division operator is *protected*—that is, it returns zero for division by zero. All the members of the function set are binary—they take two parameters. During the search process we use a heavy dynamic limit on tree depth [132] to control the code bloating. The initial maximum program tree depth is set to 3, but it can increase to 6 during evolution. More discussions on the sensitivity of the algorithm to the maximum tree depth is presented in 6.3.2 and 6.3.3. The population size is 512 individuals [4]. This is common and reasonable value for the population size. The probability of the crossover and mutation operators are adapted automatically at runtime [21] and finally, an elitist approach is taken to keep the best individual of the generation.

[4] When the value of a parameter is not precisely known, some computer scientists tend to use the nearest power of two (e.g. 512 instead of 500). In certain situations, the powers of two or variables of these sizes are easier and more efficient to store and handle. We have been following this convention for some of the GP parameters.

Table 3.2: GP Settings

Function Set:	$+, -, \times, \div$ (protected division)
Variable Terminals:	The original features ($\{x_1, x_2, \ldots, x_m\}$)
Constant Terminals:	Randomly Generated
Population Size:	512
Number of Generations:	50
Initialisation:	Ramped half and half
Mutation:	Subtree creation
Selection:	Tournament (size=5)
Initial Tree Depth:	3
Maximum Tree Depth:	6
Mutation Probability:	Adaptive[21]
Cross-over Probability:	Adaptive[21]
Elitism:	Yes

3.5.1.3 Evaluation Process

Since none of the datasets that are used in our experiments comes with a specific test dataset, we adopt a 10-fold cross-validation approach. At the start of each 10-fold cross-validation, the seed of a random number generator is initialised and the following steps are performed:

1. Shuffle the dataset;
2. Create 10 stratified partitions (folds);
3. For each fold repeat the following:
   (a) Take the current fold as the test set and the others as the training set;
   (b) Run Algorithm 3;
   (c) Transform the training and test set through the constructed features;

(d) Perform classifier learning and testing.

The dataset is shuffled and stratified to 10 folds. Stratified folds have the same proportion of instances form different classes. Each time one of the folds is taken as the test set and the remaining as the training set and then Algorithm 3 is executed. The shuffling process and the inner GP algorithm all depend on the random number generator.

We consider each execution of Algorithm 3 as one *GP job* and each job involves L *GP runs*, where L is the number of distinct class labels in the problem. There are 10 GP jobs in each 10-fold cross-validation. Since GP is a stochastic process, we need to have a number of GP runs before being able to extract any reliable statistics. We repeat the above-mentioned process three times. This gives us $3 \times 10 \times L$ GP runs in total. In all experiments, the J48 implementation of C4.5 decision tree inducer [127, 152] is used to evaluate the quality of the constructed features. Table 3.3 summarises different parameters involved in the evaluation.

Table 3.3: Evaluation Settings

Validation:	10-fold cross-validation with stratified folds
GP jobs:	30
Total GP runs:	$3 \times 10 \times L$ (L is the number of class labels)
Classifier:	C4.5 Decision tree (J48 version)
Evaluation Modes:	a) Using augmented datasets and
	b) Using constructed features only

3.5.2 Results and Analysis

3.5.2.1 Classification Performance Using Augmented Datasets

The first group of experiments examines whether adding the newly constructed features to the original features—that is, making an augmented dataset—can improve the classification performance of the decision tree

classifier. The number of features in the augmented dataset is shown in Table 3.4. The number is simply the sum of the number of original features and the number of constructed features. The number of constructed features, as discussed earlier, is equal to the number of classes in the given classification task.

Table 3.4: Number of features at different stages

Problem	Original	Constructed	Augmented
Balance Scale	4	3	7
Glass Identification	9	6	15
Iris Plant	4	3	7
Liver Disorders	6	2	8
Pima Diabetes	8	2	10
Thyroid Disease	5	3	8
Wine Recognition	13	3	16
WBC-Original	9	2	11

Table 3.5 shows the J48 decision tree accuracy using the original and augmented datasets for the eight classification problems. The classification accuracy using the original dataset is obtained via 10-fold cross validation (using test folds). Since the decision tree classification process is deterministic and no other major stochastic processes are involved[5], the process is repeated only once (10-fold cross-validation) and the result is reported in the second column of the table. For augmented datasets where GP-constructed features are involved, the process is repeated as many times as specified in Table 3.3. The mean and standard error of the classification accuracy are reported in the table. The t value is calculated by $t = \frac{\bar{X}-\mu_0}{s/\sqrt{n}}$ where μ_0 indicate the accuracy using the original features, $\bar{X}$

[5]We assume that the effect of random assignment of instances to different folds on classification accuracy is negligible.

indicates the estimated mean of accuracy using augmented features, s is the standard deviation and n is the number of repetitions which is 30. The probability values come from a T distribution with 29 degrees of freedom and show the confidence level at which the accuracy using augmented features outperform the accuracy using the original features.

Table 3.5: Classification accuracy over the original and augmented dataset

Problem	Original	Augmented Dataset		t-test	
	Accuracy	Accuracy	s	t	$P\{T \leq t\}$
Balance Scale	0.775	**0.976**	0.014	78.6	99.99%
Glass Identification	0.678	**0.725**	0.022	11.7	99.99%
Iris Plant	0.947	0.947	0.006	0.00	-
Liver Disorders	0.647	**0.675**	0.019	8.07	99.99%
Pima Diabetes	0.746	0.725	0.087	-1.3	-
Thyroid Disease	0.921	**0.949**	0.017	9.02	99.99%
Wine Recognition	0.910	0.910	0.003	0.00	-
WBC-Original	0.958	**0.964**	0.010	3.28	99.87%

According to Table 3.5, compared to the performance obtained using the original datasets, the augmented datasets can improve the classification accuracy in 5 out of the 8 classification tasks. The dominating values are printed in bold face. For the Balance Scale dataset, the improvement is quite significant. In 2 of the 3 datasets in which the performance has not improved, the performance is the same, and in only one case, Pima Diabetes, the performance has deteriorated. For almost all cases the standard error is quite low, suggesting that the results of different GP runs are quite consistent. Overall, the results suggest that GP-constructed features play a positive role in improving the classification performance.

3.5.2.2 Classification Performance Using Only Constructed Features

In the second group of experiments, we compare the performance of the J48 decision tree using the original features with the J48 decision tree using only the GP-constructed features. Table 3.6 shows the performance results. The accuracy values are calculated using the test folds. The means and standard deviaitons of the accuracy using only GP-constructed features have been reported in the table. The t value is calculated by $t = \frac{\bar{X}-\mu_0}{s/\sqrt{n}}$ where μ_0 indicate the accuracy using the original features, $\bar{X}$ indicates the estimated mean of accuracy using only GP-constructed features, s is the standard deviation and n is the number of repetitions which is 30. The probability values come from a T distribution with 29 degrees of freedom and show the confidence level at which the accuracy using only GP-constructed features outperforms the accuracy using the original features.

We explain the table by taking the Wine Recognition dataset as an example. The number of original features in this dataset is 13. GP has constructed 3 high-level features for this problem, one for each class label. Comparing the classification accuracy when using original features (0.910) with the classification accuracy when using only the three newly-constructed features (0.944), there is over 3% improvement. A small standard error of 0.013 suggests that the improvement is quite significant. The last columns shows the maximum accuracy of 0.967 can be reached by using the constructed features.

According to Table 3.6, in terms of average performance, for 7 datasets out of 8, classifiers using the constructed features can outperform those using the original features. In terms of maximum performance (the last column, which is obtained by using the best constructed features), they can outperform in all 8 datasets. A low standard error in almost all datasets suggests that the improvement is quite consistent. Even for the 3 datasets where the augmented datasets could not improve the performance in the previous experiment, the performance has been improved using only the high-level constructed features. Although augmented feature sets are su-

Table 3.6: Results of the proposed approach and the basic decision tree approach.

Problem	# Features		Org.	Cnstd. Features Acc.			t-test	
	Org.	Cstd.	Acc.	mean	s	max	t	$P\{T \leq t\}$
Balance Scale	4	3	0.775	**0.972**	0.021	**1.000**	51.38	99.99%
Glass	9	6	0.678	**0.718**	0.009	**0.730**	24.34	99.99%
Iris Plant	4	3	0.947	**0.952**	0.002	**0.960**	13.69	99.99%
Liver Disorders	6	2	0.647	**0.688**	0.007	**0.704**	32.08	99.99%
Pima Diabetes	8	2	0.746	0.744	0.005	**0.754**	-2.19	-
Thyroid Disease	5	3	0.921	**0.941**	0.006	**0.953**	18.25	99.99%
Wine Recognition	13	3	0.910	**0.944**	0.013	**0.967**	14.32	99.99%
WBC-Original	9	2	0.958	**0.966**	0.004	**0.972**	10.95	99.99%

persets of constructed features, in 5 tasks out of 8, namely Iris, Liver, Pima, Wine and WBC, the performance of classifiers using only constructed features is higher than those using the augmented sets. This suggests that in many cases GP-constructed features carry all important information required for classification and augmentation might cause the inclusion of some noisy or redundant features that are not required for classification and can actually lead to deterioration. It also raises questions about decision trees' capability in feature selection, but this is not in the scope of the thesis.

3.5.2.3 Effect of Constructed Features on Decision Tree Complexity

To see how constructed features can affect the complexity of decision tree classifiers, we study the changes in the size of decision trees using the original and constructed features. The size of decision trees are measured by counting the number of decision nodes they contain. The complexity of decision trees has a direct effect on their generalisation capability and the extent to which they can be interpreted. The less complex a decision

tree, the more its generalisation capability and the easier its interpretation.

Table 3.7: Changes in Complexity of Decision Trees

Problem	# Features		Max	# of nodes in DT		
	Org	Cnstd.	Improvement	Org.	Cnstd.	Shrink
Balance Scale	4	3	29.0%	86.0	5.0	94.2%
Glass Identification	9	6	7.7%	46.6	37.6	19.3%
Iris Plant	4	3	1.4%	8.4	8.4	0.0%
Liver Disorders	6	2	8.8%	44.6	8.8	80.3%
Pima Diabetes	8	2	1.1%	40.6	3.0	92.6%
Thyroid Disease	5	3	3.5%	15.4	11.2	27.3%
Wine Recognition	13	3	6.3%	9.2	8.6	6.5%
WBC-Original	9	2	1.5%	22.0	6.2	71.8%

Table 3.7 shows the results of this study. The number of original features, the number of GP-constructed features, and the maximum improvement achieved by using the GP-constructed features (instead of the original features) are reported in the left columns. The maximum improvement is obtained by $\frac{accuracy_{max} - accuracy_{original}}{accuracy_{original}} \times 100\%$ where $accuracy_{max}$ is the maximum accuracy obtained by using the constructed features and $accuracy_{original}$ is the accuracy obtained by using the original features. The average number of nodes in the decision tree using the original features and the GP-constructed features are in columns 5 to 6. The last column shows how much the decision tree classifier has shrunk when only GP-constructed features have been used. For example, the first row shows that by using the 3 GP-constructed features instead of the 4 original features in the Balance Scale problem, the classification performance has increased by 29% and the average decision tree size has reduced from 86.0 nodes to 5.0, i.e. a 94.2% shrinkage. This pattern is the same for almost all the eight datasets; using the GP-constructed features, the classification

performance increases and the classifier complexity decreases.

3.5.2.4 Analysis of A GP-Constructed Feature

To have a better picture of how GP-constructed features can actually improve the classification performance, we analyse one of these constructed features as an example. One of the cases that has shown a considerable improvement after using the GP-constructed features is the Balance Scale dataset. The dataset has three classes, namely *Left* (L), *Right* (R), and *Balance* (B). We analyse one of the GP-constructed features for the class B of this dataset. The constructed feature, y_B, is the non-linear Lisp expression `(/ x3 (* x2 (/ x1 x4)))`, which can be mathematically expressed as $y_B = \frac{x_3 x_4}{x_1 x_2}$, where x_i is the i-th original feature. The fitness of this constructed feature is zero indicating that along the axis of the constructed feature, the interval of the class B contains only instances of B, demonstrating a perfect separation.

Figure 3.5 shows a learnt decision tree induced by the J48 algorithm using the constructed feature. Although only one constructed feature has been used, the performance of the classifier on all of the test folds is 100%. In fact, the decision tree benefits from the fact that the instances of the three classes form three non-overlapping bands along the axis of the constructed features. All the instances of B have been squeezed into a narrow band approximately between 3.28 and 3.5. The instances of the other two classes, L and R, are at the right and left hand side of this band. The bands are illustrated at the bottom of Figure 3.5.

3.6 Discussion

The proposed GP-based feature construction method has the capability of making multiple features while using a filter-based fitness function. We achieve this by making the feature evaluation measure (the fitness

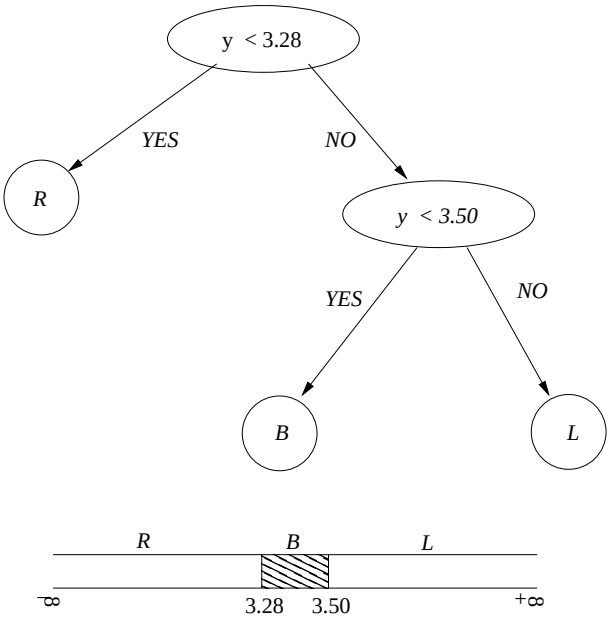


Figure 3.5: A learnt decision tree using a constructed feature, y_B. The feature has been constructed for the class B of the Balance Scale problem. Although using one feature, the decision tree can perfectly separate all the instances of the three classes.

function) class-centric—that is, it evaluates the goodness of a feature with respect to its power to discriminate between the instances of different classes. This is an important advantage because almost all the existing filter-based methods in the literature can construct only one feature. Since using a single feature is usually not enough for successful classification, the only available option when using these methods is to feed the original features along with the constructed feature to the classification algorithm. This is in fact a limitation of the traditional GP-based feature construction systems, which makes them unsuitable for certain purposes like dimensionality reduction.

Our observations on the eight classification tasks show that in most cases, augmented feature sets (the union of constructed features and original features) improve the classification performance over using the orig-

inal features (the standard approach). When comparing with using constructed features alone, however, the classification results achieved using the augmented datasets were lower in most cases. Besides, due to the larger number of features in the augmented datasets, there is a slight increase in dimensionality. A possible reason for this phenomenon might be that the constructed features and the original features are redundant. However, if decision tree classifier inducers had good feature selection ability—and this is what is expected according to the literature—then using augmented feature sets should not lead to decrease in performance when compared to using only constructed features. This might suggest that C4.5 algorithm does not really have as good feature selection ability as mentioned in the literature.

Since constructed features are able to express the original input space in a more concise form, the learnt decision trees using only these features tend to be simpler (having fewer decision nodes). For example, the learnt decision tree classifier for the Balance Scale problem, using the constructed features, has far fewer nodes than the learnt decision tree classifier using only the original features. The decrease in the complexity of decision trees is due to the richness of GP-constructed features. Smaller decision trees are easier to interpret and faster in execution. However, the constructed features themselves might be difficult to interpret meaningfully.

As the fitness measure approaches zero, the instances of the class for which a feature is being constructed are the only occupants of the class interval. The instances of this class gather together in the form of a distinguishable band, that is easy to separate from the instance of other classes. While the objective of this feature is to discriminate the instances of the class of interest, it can sometimes group the instances of other classes on either side of the interval of the class of interest. We saw an example of this when analysing a GP-constructed feature for the Balance Scale dataset. This suggests that sometimes GP-constructed features are potentially able to perform the actual classification task by grouping the class instances in

separate bands.

Overall, our results suggest that GP can be effectively used for constructing multiple high-level features for classification problems. The constructed features, either in the form of an augmented dataset or on their own, can significantly improve the performance of classifiers. The newly-constructed features seem to be able to give more generalisation capability to classifiers than the original features. Therefore, the improvement in classification usually coincides with a decrease in the complexity of classifiers.

Chapter 4

Dimensionality Reduction

4.1 Introduction

The number of dimensions of a classification problem is a decisive factor in the performance of a classifier. A high dimensional dataset might severely suffer from the *curse of dimensionality*; the search space of possible models for the data is huge and there might be a lot of redundancy. Generally, the lower the number of dimensions, the easier to learn a system and the higher the performance. In most cases, reducing the dimensionality of a problem, as long as important information is not lost, makes learnt models simpler and more general and therefore, easier to interpret. Consequently, dimensionality reduction is a major task in feature manipulation.

4.1.1 Transformational Reduction

Transformational reduction is an approach to dimensionality reduction. In this approach, the input space is transferred to a new input space with lower dimensionality where each dimension (feature) in the new input space is in fact a function of a number of dimensions in the original input space. Principle Component Analysis (PCA) is a classical example of transformational reduction in which the new dimensions (principle com-

ponents) are a linear combination of the original features.

Transformational reduction can be regarded as a special case of feature construction in which the number of constructed features is considerably less than the number of original features in the problem. In the previous chapter, we used a GP-based feature construction algorithm in which the number of constructed features equals the number of classes in a classification problem. Since the number of classes in a high dimensional classification problem is usually less than the number of features, the algorithm can implicitly provide some degree of reduction depending on the ratio of the number of classes to the number of features.

4.1.2 Challenges in Dimensionality Reduction using GP

While in principle, the algorithm proposed in the previous chapter can be used for dimensionality reduction for a broad category of classification problems in which the ratio of the number of class labels to the number of features is low, there is a subtle but very important issue that should be taken into account before using the algorithm for dimensionality reduction. Dimensionality reduction techniques are usually applied to problems with fairly large number of features. For a GP-based algorithm, where normally there is one variable terminal per feature in the problem, the size of the variable terminal set grows with the number of original features in the problem. The size of the GP search space (the space of all possible programs), however, grows exponentially with respect to the size of the variable terminal set.

In general, given a constant amount of computational resources, the probability of success of an evolutionary algorithm decreases as the size of the search space increases. The algorithm proposed in the previous chapter is, of course, no exception. Classification problems with a larger number of original features are generally more difficult to solve[1]. One

[1]We will see an example of this phenomenon in Chapter 6.

remedy for big search spaces is to have large population sizes. Since having larger populations means more fitness evaluations, one would like to have a fairly simple fitness function that is computationally affordable when evaluated for a large number of times. The other remedy might be to increase the probability of success by using some good heuristics.

4.1.3 Chapter Goals

This chapter aims to develop a non-wrapper GP-based approach to dimensionality reduction in classification problems. To deal with the GP difficulty in searching enormous search spaces created by classification problems with a large number of features, we aim to make two critical improvements in the algorithm proposed in the previous chapter by:

- increasing the chance of finding (constructing) a discriminative feature in a large search space by employing some types of heuristics; and
- making the search process computationally affordable by proposing a new fitness function that is easy to evaluate.

We address the first objective by introducing *class-wise orthogonal transformation* and *encapsulating terminals* ideas influenced by some classical dimensionality reduction methods. We achieve the second objective by first providing a general form of entropy-based fitness functions and then deriving a simplified version of that. The performance of the proposed method is measured in terms of reduction ratio and improvements in classification accuracy. The method is also compared with a classical transformational dimensionality reduction method.

4.2 Enriching GP Material through Transformations

An important factor for GP to achieve success in finding desired solutions is the quality of the available building blocks. A piece of genetic material, in tree-based GP, is a subtree[2]. Good genetic material is a partial solution that occurs in the structure of desired solution trees. Providing the search with good genetic material can considerably increase the likelihood of success.

4.2.1 Finding a Promising Transformation

In the GP-based approach to feature construction and dimensionality reduction, a subtree is itself a transformation. Therefore, one way of finding heuristics to produce good genetic material is to look for promising transformations. To do this, we first have a look at the problem of non-orthogonal class boundaries, a common phenomenon in decision trees. Next, we see how PCA, a largely practised method for dimensionality reduction, attempts to deal with non-orthogonal datasets and discuss its shortcomings. Then, we propose an alternative transformation that will be used, in the next section, to enrich genetic material in our proposed GP-based dimensionality reduction system.

4.2.1.1 The Issue of Oblique Class Boundaries in Decision Trees

A decision tree is a hierarchy of decision nodes (decision stumps) that are triggered in a top-down manner. Each node examines the value of an individual feature and branches to one of its two child-nodes, which are either a prediction or another decision node. In an n-dimensional input space, this process looks like partitioning the space into different areas where

[2]It can also be thought of as a building block in its very basic syntactic form, without any wild-card node.

each partition is associated with a class label and is formed by a collection of a finite number of hyper-rectangles. The decision tree classification process works quite well when the boundaries of classes are orthogonal to one of the features.

Now consider an n-dimensional classification problem in which instances of a particular class make a hyper-ellipsoid cloud of data in a way that its boundaries with other classes are not orthogonal to the input features. This is in fact a very common phenomenon in real classification problems. A two-dimensional example of this phenomenon is depicted in Figure 4.1. The figure shows a binary classification problem with normally distributed classes. The solid lines show the direction of deviation in each class. The dashed line shows the boundary between two classes. The boundary between the two classes is neither perpendicular to x_1 nor x_2. Since decision tree learners find the class areas by dividing the input space into some rectangular regions, an angled class boundary like this causes the decision tree learner to make several rectangles to include the desired class instances and exclude the unwanted ones. This phenomenon makes learnt trees quite big and complicated which consequently affects their generalisation capability, classification performance, and execution time. Some aspects of this issue have been discussed in [127] as well.

4.2.1.2 Limitations of PCA in Classification Problems

PCA is one of the transformational dimensionality reduction techniques that is widely used in different applications. PCA performs a linear transformation. The objective of the transformation is to extract features with high variability (principle components) and no correlation (orthogonality). To understand the advantages and disadvantages of PCA in a classification context, we first need to see how principle components are extracted.

PCA diagonalises the covariance matrix by linearly transforming data to a new space where the axes of the data distribution are orthogonal to

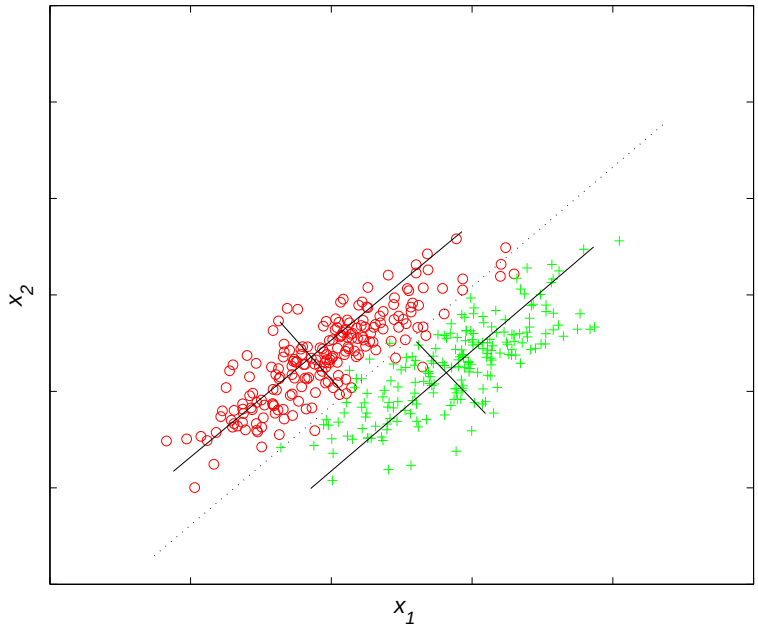


Figure 4.1: An artificial dataset with two original features, x_1 and x_2, and two classes, $+$ and $\circ$. Although there is a clear linear boundary between the two classes, since the boundary between the classes is oblique, a decision tree classifier has to separate the two classes by making several rectangular regions.

the axes of the new space [59]. The axes of the new space are called *principle components*. Based on our terminology, principle components are constructed features that map data to a new coordinate system. The locations of instances in the new coordinate are obtained by multiplying the location in the original coordinate by eigen vectors of the covariance matrix. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible [37].

Figure 4.2 shows the PCA transformation of data presented in Figure 4.1. The solid lines show the direction of deviation of instances of all classes. PCA has rotated the data cloud in a way that the new axes (prin-

ciple components) , pc_1 and pc_2, are parallel to the direction of deviation of all instances (including the two classes). According to PCA assumptions, the principle component with larger deviation (i.e. greater eigen values) is more significant. That is, in Figure 4.2 the pc_1 component is more important (informative) than pc_2.

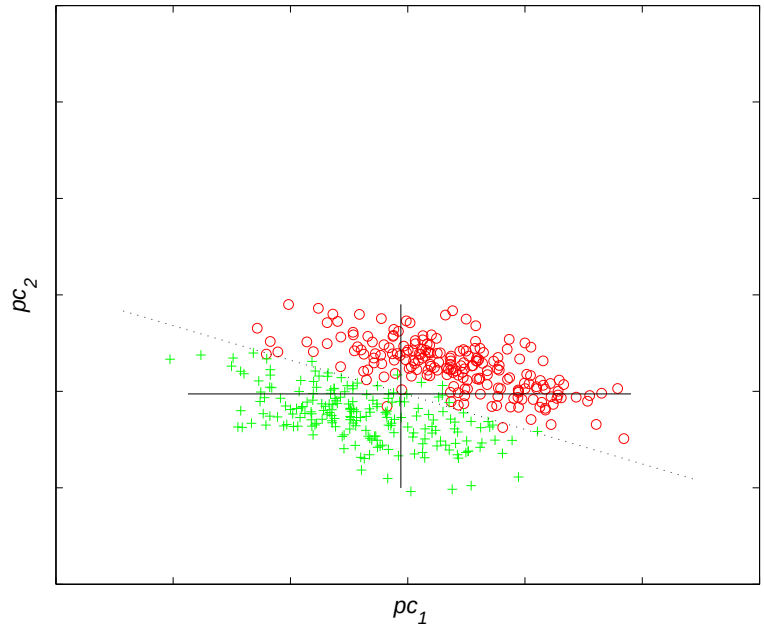


Figure 4.2: The PCA transformation of the data displayed in the previous figure.

Looking at Figure 4.2, although the new axes are now the principle components, the boundary between the two classes (the dashed line) is still at an angle. The new space is not still favourable from the standpoint of a decision tree inducer. This is due to the fact that the PCA procedure is blind to the class labels in a training set. PCA considers data as a whole regardless of the distribution of different classes. Since the class boundary in Figure 4.2 is not orthogonal to either of the axes, when training a decision tree in this new input space, difficulties similar to those of training a decision tree for the data in Figure 4.1 arise. That is, a decision tree inducer

has to create a large number of partitions to separate the classes. This suggests that although PCA is useful for dimensionality reduction, in some classification problems when certain classifiers used, it might not achieve the desired effects.

4.2.1.3 Class-wise Orthogonal Transformation

In relation to the above-mentioned limitation, we propose a transformation that takes into account the class information (class labels in the training set). As mentioned earlier, the transformation will be used as heuristics to increase the chance of finding a few high-level features that provide good class separation. These high-level features, when used instead of the original features in a classification problem, result in dimensionality reduction.

The Concept The PCA transformation has two main elements: an orthogonal transformation and a ranking mechanism. The orthogonal transformation makes the axes of the data distribution (the directions of deviation) parallel to those of the new coordinate system after transformation. The ranking mechanism is not useful in our feature construction scenario. Besides, it comes with the general assumption that higher-ranked components (features) are those with more deviation, which is not necessarily true for classification problems. Thus, we disregard the ranking mechanism of PCA, but consider how class information can be incorporated into the orthogonal transformation process.

Looking at the example illustrated in Figure 4.2, we see that it would have been better if a transformation could rotate the dataset in a way that the boundary between the two classes was perpendicular to one of the axes. This could have possibly been achieved by considering the centre of a class as the centre of rotation rather than using the centre of the whole dataset. That is, data points are transformed in a way that the axes of a certain class are parallel to the axes of the input space. This can increase

the chance of having a class boundary perpendicular to one of the axes[3].

To achieve this, we propose a modified version of orthogonal transformation that is class-wise. That is, a dataset is analysed class by class, and for each class a new n-dimensional space is obtained (by transformation) in which the axes are along the axes of the class distribution. Such a transformation for the example given in Figure 4.1 is presented in Figure 4.3. In this figure, after the dataset has been rotated around the centre of class '$\circ$', the class boundary is perpendicular to $cwoc_1$. After transformation, a single feature like $cwoc_1$ is sufficient to learn the resulting space. This effect can be very helpful in improving the search process of a GP-based dimensionality reduction system.

Mathematical Model Let $\mathbf{D} = (\mathbf{X}, \mathbf{c})$ be a training dataset, where $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$ is a set of vectors, each of length n, of observations of the m original features in the problem and $\mathbf{c}$ is a vector of class labels for the corresponding observations in $\mathbf{X}$. $\mathbf{x_i}[j] \in \mathbb{R}$ and $\mathbf{c}[j] \in \mathbb{C}$ for $i \in \{1, 2, \ldots m\}$ and $j \in \{1, 2, \ldots, n\}$ where m is the number of original features, n is the number of observations, $\mathbb{C} = \{c_1, c_2, \ldots, c_L\}$ is the set of class labels, and L is the number of classes in the classification task. The training set is divided into L partitions of the form $\mathbf{P}_k$ each of which containing only instances from one of the classes.

$$\mathbf{P}_k = \{(x_1, x_2, \ldots, x_m) \mid (x_1, x_2, \ldots, x_m, c) \in \mathbf{D}, c = c_k\} \tag{4.1}$$

[3]Linear Discriminant Analysis (LDA) and Fisher's measure [93, 25] could have been other options to be used as heuristics. Although these two methods provide linear transformations that maximise class separation, we assume that our proposed entropy-based fitness function can explicitly address this objective by finding pure class intervals. Therefore, we rather focus on orthogonal transformations that are not directly addressed (as an objective) by the fitness function and therefore may take a long time to be found during an evolutionary process.

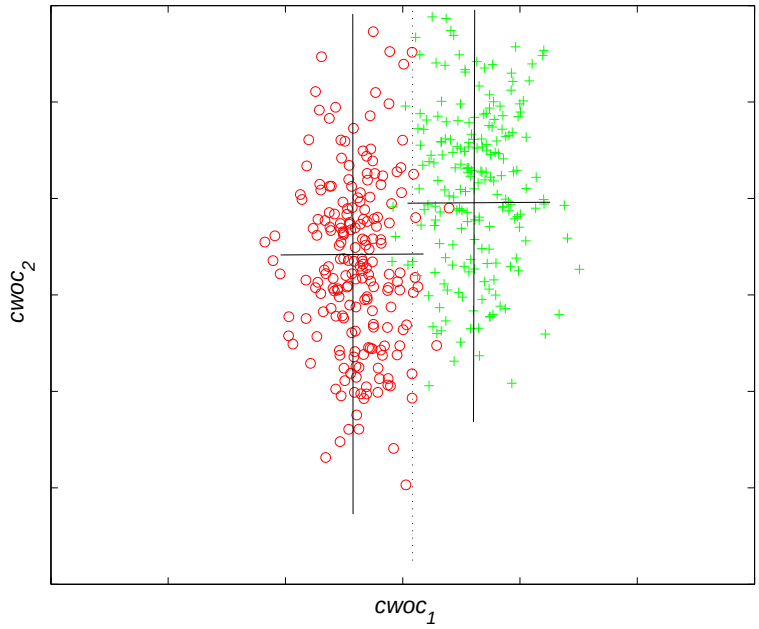


Figure 4.3: Transformed input space using class-wise orthogonal transformations

where $k \in \{1, 2, \ldots, L\}$ and

$$\bigcup_{k=1}^{L} \mathbf{P}_k = \mathbf{X} \,. \tag{4.2}$$

Each partition represents a hyper-ellipsoid cloud. To have the axes of deviation in the hyper-spheroids (axes along which the cloud of instances are mostly scattered) perpendicular to the features in the newly transformed space, one should diagonalise the covariance matrix of the data in each partition. The covariance matrix of each partition $\mathbf{P}_k$ is

$$\mathbf{\Sigma_k} = E[(\mathbf{P}_k - E(\mathbf{P}_k))(\mathbf{P}_k - E(\mathbf{P}_k))^T] \tag{4.3}$$

where E denotes the expected value and $\mathbf{\Sigma}_k$ is an $m \times m$ square matrix containing the covariances (and variances along the diagonal) of features based on the data instances observed in the k-th partition. The axes of this

partition can be obtained by finding corresponding eigen vectors of the covariance matrix of the partition [37]:

$$\mathbf{A_k} = \text{eigen}(\mathbf{\Sigma}_k) \tag{4.4}$$

where each row of this matrix shows a vector in an m-dimensional space. A transformation via these vectors makes the axes of the resulting dataset orthogonal to the new coordinate system. Since the goal is to find a new space in which class boundaries are perpendicular to the axes, the eigen values are disregarded (despite what PCA does for ranking the resulting components). The whole dataset is transformed by

$$\mathbf{X'}_k = \mathbf{A_k}\mathbf{X} \tag{4.5}$$

where $\mathbf{X'_k}$ is the transformed version of $\mathbf{X}$ (the whole dataset) using information from the k-th class. Note that while the calculations of the covariances and the eigen vectors are based on the partitioned data, the transformation is applied to the whole data set. So for each partition (distinct class in the problem), one transformation is applied. In other words, having L classes in a classification problem, there will be L different transformations of the original dataset, each having m dimensions[4]. For example, for the dataset in Figure 4.1, two 2-dimensional transformations are created. Figure 4.3 shows one of these two transformations. The boundary of the two classes (dashed line) is now perpendicular to the new component (horizontal axis).

4.2.2 A Real World Example

Dimensionality reduction is usually expected to improve or retain classification performance. To see how a single feature in the transformed

[4]In fact, $L - 1$ transformations would have been sufficient if it was certain that the transformations discover the real boundary between classes. However, as it was pointed out, the L transformations are distinct from each other and are *only likely* to transform the boundary between classes orthogonal to the axes.

space can carry the information provided by a number of features from the original space while improving the classification performance, an example from a real dataset is presented. Figure 4.4 depicts a two dimensional representation of the Thyroid Grand problem [6] based on its first two attributes: T3-resin and total Serum Thyroxin. There are three different classes in this problem, namely 'normal', 'hyper', and 'hypo' which are presented by different symbols. The boundaries of these classes are neither linear nor perpendicular to the coordinate axes.

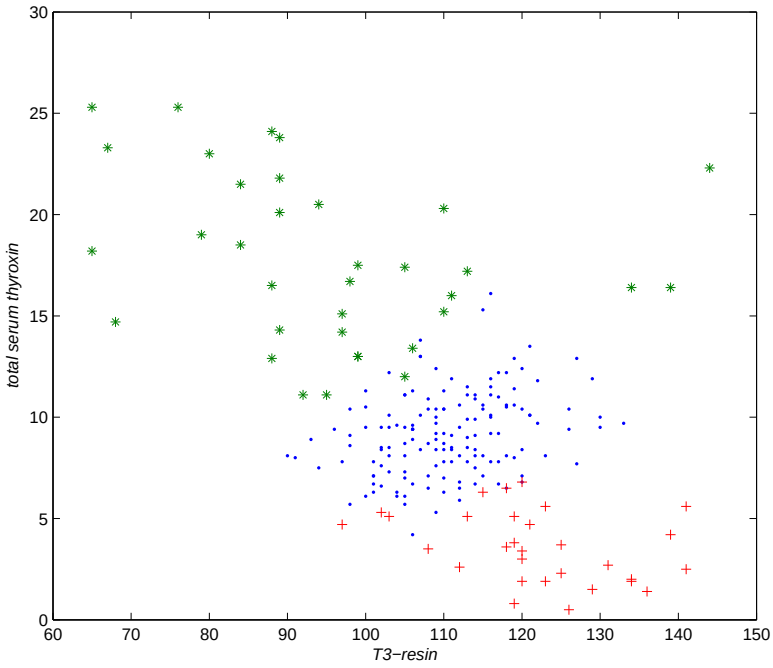


Figure 4.4: Thyroid disease dataset represented in two dimensions. The axes are two attributes from the dataset, T3-resin and total Serum thyroxin. There are three classes in this dataset: normal, hyper, and hypo which have been represented by '.', '⋆', and '+' respectively.

For a decision tree inducer to learn a classifier on this space, it has to divide the input space to some rectangular regions to find the class areas. Thus, the non-orthogonal class boundaries in the Thyroid problem

cause the decision tree inducer to make several such rectangular regions to include the desired class instances and exclude the unwanted ones. To quantify the separability of classes in an input space, we consider the performance of a learnt decision tree on this space. For the dataset in Figure 4.4, a J48 decision tree inducer [152] is used. Only the two illustrated dimensions are used for training and testing the decision tree. 10-fold cross-validation is used to evaluate the classification accuracy. The classification performance measures 91.6% in the original input space. This performance is used as a baseline to see how different transformations affect the classification performance.

Figure 4.5 shows the result of applying PCA to the first two dimensions of the thyroid grand problem. As it is seen, the shape of the whole dataset (all three classes together) has been straightened along the new coordinates. However, because PCA cannot distinguish between different classes labels, the class boundaries are still at an angle. Training a decision tree in this new input space has deficiencies similar to those of the original input space. Using a J48 decision tree inducer to learn and classify this new PCA-created space, the classification performance using the first component alone is 77.2%, using the second component alone is 73.0%, and using both components (no dimension reduction) is 86.5%; all of them cause some decrease in classification performance. As anticipated, PCA might not be a suitable transformation for classification problems.

Figure 4.6 shows the result of applying class-wise orthogonal transformation to the first two features of the Thyroid problem depicted in Figure 4.4. After transformation, six new dimensions are generated, two for each class (partition) in the problem. Two of these newly created dimensions are shown in Figure 4.6. The dashed lines around the middle class show boundaries of this class perpendicular to the first constructed dimension. If we induce a J48 decision tree classifier using the second component alone, the average classification performance using 10-fold cross-validation is 92.1%. Compared to the classification performance using the

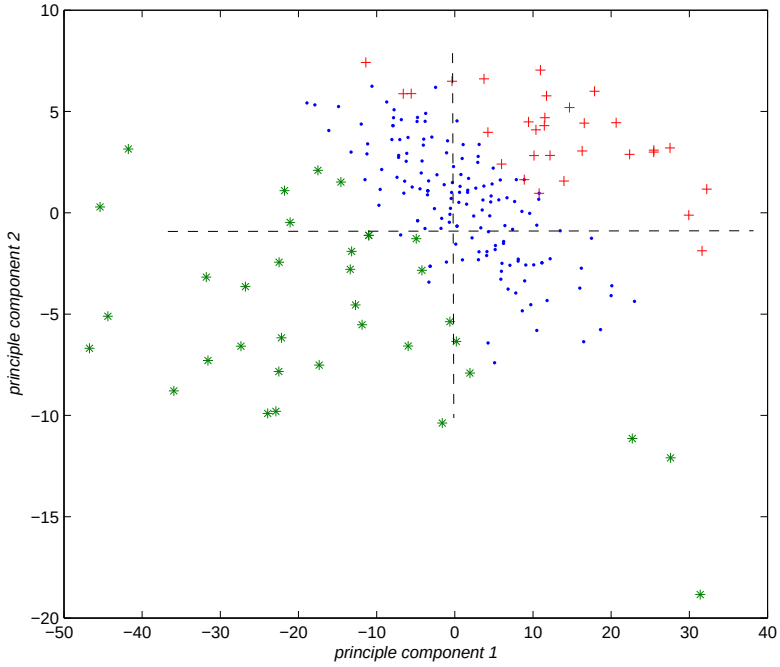


Figure 4.5: Transformed input space of the Thyroid problem using PCA. The axes are the two principle components.

original features, although the dimensionality of the problem is reduced from two to one, the classification performance has slightly increased.

4.2.3 The Enrichment Process

We use the proposed class-wise orthogonal transformation as heuristics to enrich the genetic material available to GP for transformational dimensionality reduction.

4.2.3.1 Two Options for Using Transformations as Genetic Material

One way to add transformations, as genetic material, to GP search is to build their equivalent program trees and enter these new trees into the GP population. The transformation is the inner product of an eigen vector

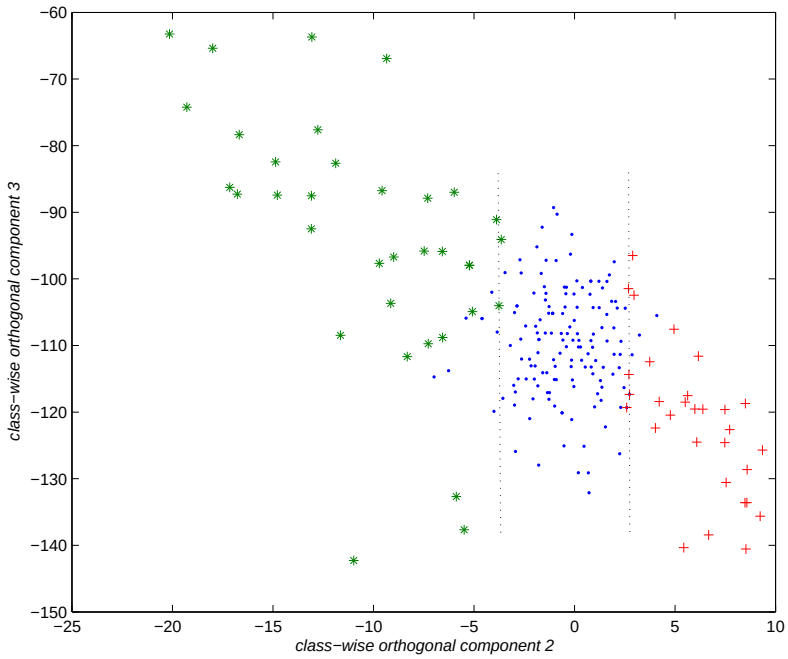


Figure 4.6: Two (out of six) dimensions presented after transforming the original input space of the Thyroid Gland problem using class-wise orthogonal transformations.

and the input vector. Therefore, given a set of binary functions (functions taking two arguments), the equivalent program tree for a transformation will have the original features and their coefficients (elements of the eigen vector) at the leaf level (the lowest), the binary multiplication function at the next level (for multiplying the original features with their coefficients), and the binary addition function at higher levels (for adding all the terms together). This way of adding genetic material has two disadvantages: A) due to the nature of evolution in GP, the genetic material from the transformation may soon be destroyed or disappear during the evolution before having a chance of being used in a good program, B) the transformation takes a large number of nodes and creates deep program trees, particularly with binary primitive functions and in problems with a large number of

original features.

Another method to add the transformations (as genetic material) to GP is to pack each transformation into a single virtual node. The virtual node, when it is evaluated, applies the corresponding transformation to the input data. This way of enrichment has the advantage of taking only one node in a program tree. Virtual nodes are immutable and do not change during the evolution process. They can be evaluated once for every data instance in the problem and cached for future references. Since virtual nodes can serve our purpose without adding any disadvantages, they are utilised to enrich the genetic material.

4.2.3.2 Extended Variable Terminal Sets

A terminal set is one of the ingredients that GP uses to make programs. A variable terminal is a type of terminal that GP uses to read input data (features). Commonly, a variable terminal is connected to an original feature in the dataset and for each instance it returns the value of that feature. We define an *encapsulating terminal* to be a special type of a variable terminal that is virtually connected to all the m original features and encapsulates an m-to-1 transformation. It returns a scalar real value for each instance that is obtained by applying an m-to-1 transformation to the m feature values of the instance. With regard to program structure, an encapsulating terminal is like any other node in a GP program; it returns a single real value.

Having variable terminals that carry useful information for separating data instances can improve the success rate of a GP search. In the previous subsection, we saw that class-wise orthogonal transformation demonstrates a high potential for improving the classification performance via increasing the chance of class separation. We use class-wise orthogonal transformation in encapsulating terminals to enrich the genetic material of our GP search. An extended variable terminal set is then formed that is the union of the encapsulating terminals and standard variable terminals.

This is an attempt to increase the chance of finding (constructing) high-level discriminative features that are informative enough to serve in place of a large number of original features.

Figure 4.7 shows the overall process of genetic material enrichment. The flow of data is from left to right. The process starts with the original features of a classification problem. Class-wise orthogonal transformation is applied to the original features to obtain encapsulating terminals. The encapsulating terminals are then used together with the original variable terminals to constitute an *extended variable terminal set*. The proposed GP system uses the extended variable terminal set to construct a few high-level features that can serve instead of a number of original features. The reduction occurs when the number of constructed features is smaller than the number of original features.

4.2.3.3 Enrichment Algorithm

The enrichment process can be performed in a pre-processing phase. Like the way ordinary variable terminals are looked up from a dataset, encapsulating variable terminals are cached and stored in a table to gain efficiency. The class-wise orthogonal transformation process is repeated for each class in the problem, so that with m original features and L distinct class labels in the problem, $L \times m$ encapsulating variable terminals are created. All the ordinary and extended variable terminals are then stored in a matrix called *extended dataset*. The extended dataset has $m \times (L + 1)$ columns, one for each variable terminal (including ordinary and encapsulating). Our proposed GP method uses the extended variable terminal set (and the corresponding extended dataset) to build high-level features and achieve dimensionality reduction. Algorithm 4 shows the steps taken to prepare an extended dataset.

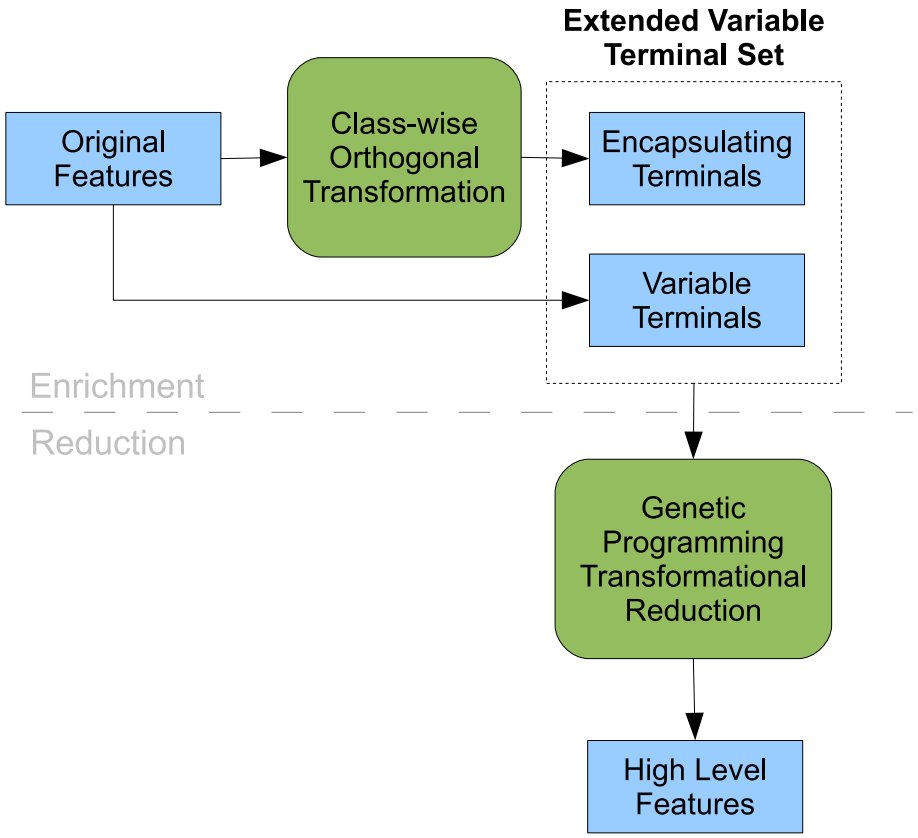


Figure 4.7: An overview of the genetic material enrichment process and dimensionality reduction.

4.3 The Fitness Function

Another critical concept in designing a GP system is the fitness function. The basic idea of the fitness function is a non-wrapper function similar to that of the previous chapter; the function measures the purity of a class interval. For dimensionality reduction, however, it is desired to simplify the fitness function so that more fitness evaluations can be performed and the expansion in the search space be compensated.

Algorithm 4: $Create\text{-}Extended\text{-}Dataset(\mathbf{D})$

/* Given a training dataset with m features, the algorithm uses class-wise orthogonal transformation to construct m encapsulating terminals for each class in the problem. The encapsulating terminals together with original variables are stored in an extended dataset */

Input: $\mathbf{D}$, a dataset of the form $\mathbf{D} = (\mathbf{X}, \mathbf{c})$ where $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$ is a set of vectors containing n observations of the m original features and $\mathbf{c}$ is a vector of class labels for the corresponding observations in $\mathbf{X}$

Output: $\mathbf{D}^+$, an extended dataset containing the values of the original and encapsulating variable terminals

$\mathbf{D}^+ \leftarrow \mathbf{X}$; // add the original features to the new dataset

foreach $c^\star \in \mathbb{C} = \{c_1, c_2, \ldots, c_L\}$ **do**

  $\mathbf{P} \leftarrow \{(x_1, x_2, \ldots, x_m) : (x_1, x_2, \ldots, x_m, c) \in \mathbf{D}, c = c^\star\}$; // find the class partition

  $\mathbf{\Sigma} \leftarrow E[(\mathbf{P} - E(\mathbf{P}))(\mathbf{P} - E(\mathbf{P}))^T]$; // calculate the covariance of the partition

  $\mathbf{A} \leftarrow$ eigen-vectors$(\mathbf{\Sigma})$;

  $\mathbf{X}^+ \leftarrow \mathbf{XA}$; // make encapsulating terminals by applying a class-wise orthogonal transformation to $\mathbf{X}$ for the current class

  $\mathbf{D}^+ \leftarrow (\mathbf{D}^+, \mathbf{X}^+)$; // append the encapsulating terminals to the new dataset

$\mathbf{D}^+ \leftarrow (\mathbf{D}^+, \mathbf{c})$; // adding the class label vector

return $\mathbf{D}^+$; // returning the extended dataset

4.3.1 A Generalised Model: Renyi's Entropy

To simplify the fitness function introduced in the previous chapter, the concept of Shanon's entropy is revisited. The Shannon entropy is a specific form of a more general entropy function, called Renyi's entropy [28, 125] which is defined as

$$H_\alpha(I_{c^\star}) = \frac{1}{1-\alpha} log_2 \sum_{c \in C} p^\alpha(c|X \in I_{c^\star}) \tag{4.6}$$

where $\alpha > 0$ is the order of entropy, $\alpha \neq 1$, I is the interval being investigated, $\mathbb{C}$ is the set of all class labels, and $p_I(c)$ is the probability of class c in interval I, which is calculated by measuring the frequency of occurrences of class c in the interval. The relationship between Renyi's entropy and Shannon's entropy can be expressed as

$$\lim_{\alpha \to 1} H_\alpha(I_{c^\star}) = \sum_{c \in \mathbb{C}} -p(c|X \in I_{c^\star}) \log_2 p(c|X \in I_{c^\star}) = H(I_{c^\star})$$

That is, when α approaches 1, Renyi's entropy is equal to Shannon's entropy in the limit.

4.3.2 A Simple and Efficient Model

Thinking of the whole dimensionality reduction algorithm as an optimisation algorithm, the goal of GP is to find (construct) a high-level feature that maximises the purity in a class interval. To measure the purity of a class interval, one could use any order of the Renyi entropy. Here, we use the second order, which can be further simplified.

$$H_2(I_{c^\star}) = log_2 \frac{1}{\sum_{c \in \mathbb{C}} p^2(c|X \in I_{c^\star})} \tag{4.7}$$

The purity is maximised when the entropy function, $H_2(I_{c^\star})$, is minimised. $H_2(I_{c^\star})$ is minimised when the term $\sum_{c \in C} p^2(c|X \in I_{c^\star})$ is maximised. We know that $0 \leq p(c) \leq 1$ and $\sum_{c \in \mathbb{C}} p(c|X \in I_{c^\star}) = 1$. Thus

$$0 \leq \sum_{c \in C} p^2(c|X \in I_{c^\star}) \leq 1 \ , \tag{4.8}$$

which implies that the upper bound value of this function is 1 and it is only reached when the probability of one of the classes is 1 and the probabilities of the rest of the classes are zero. Since the interval of class $c^\star$ will always contain instances of that class, $p(c^\star|X \in I_{c^\star})$ cannot be zero. Therefore, the sum term is maximised when the probability of the occurrence of instances of other classes in the interval $I_{c^\star}$ approaches zero. In other words, we are interested in having the minimum occurrence of instances of other classes in a class interval. This is the basis for defining a simple fitness function that counts the number of instances of other classes in a class interval. Based on this fitness measure, a constructed feature is fitter than another one if the value of this function is lower for that feature—that is, a constructed feature for class $c^\star$ is better if the interval $I_{c^\star}$ of the class along this feature has a smaller number of instances from other classes.

4.3.3 Algorithm

Algorithm 5 shows the steps towards calculating the fitness of a GP program. Based on the simplified fitness model, the calculated fitness is simply the number of instances of other classes in the interval of a class. As long as one is not concerned about the fitness of a program in absolute terms—which is the exact entropy quotient in the class interval—the simplified model is enough to find the relative fitness of GP individuals. Since in our proposed GP algorithm we use tournament selection, we only need the relative fitness of individuals to compare them against one another and therefore, the proposed simplified fitness function would suffice.

The algorithm starts with using the given GP program ϕ to transform the input space—which has $m \times (L + 1)$ dimensions after enrichment—to a one-dimensional real-valued high-level constructed feature. The values of the constructed feature are stored in $\mathbf{y}$, a vector with n elements, one for each instance in the dataset. It then uses Algorithm 1, introduced in the previous chapter to find the class interval. The algorithm, then, counts the

Algorithm 5: $DR\text{-}Fitness(\mathbf{D}^{+}, \phi, c^{\star})$

```
/* Given a training dataset and a GP program (as a
   candidate constructed feature) and a desired
   class label for which a feature is being
   constructed, the algorithm evaluates the
   fitness of the GP program.                    */
```

Input: $\mathbf{D}^{+}$, an extended dataset of the form $\mathbf{D}^{+} = (\mathbf{X}^{+}, \mathbf{c})$ where $\mathbf{X}^{+} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_{m\times(L+1)}}\}$ is a set of vectors of length n of observations of the m original features and $m \times L$ encapsulating terminals, and $\mathbf{c}$ is a vector of class labels for the corresponding observations in $\mathbf{X}^{+}$

Input: ϕ, a GP program which acts as a function $\mathbb{R}^{m\times(L+1)} \mapsto \mathbb{R}$

Input: $c^{\star}$, the label of the desired class for which a feature is being constructed

Output: $fitness$, a real value showing the fitness of the program (the lower the better, and the minimum is zero)

```
1 y[i] ← φ(x_1[i], x_2[i], ..., x_{m×(L+1)}[i]) , ∀i ∈ {1, 2, ..., n} ; // using
  the GP program to transform the data
2 (l, u) ← Find-Interval(y, c, c*);    // finding the class interval
                  // based on the algorithm proposed in Chapter 3
3 fitness ← 0 ;                        // initialising the fitness
4 for i ← 1 to n do
5     if y[i] ∈ (l, u) then
6         if c[i] ≠ c* then
7             fitness ← fitness + 1;
8 return fitness;
```

number of instances from undesired classes in the interval and returns it as the fitness of the individual. The fitness value is an implicit indication

of impurity in the class interval and it is better when it is lower.

4.4 A GP System for Dimensionality Reduction

Having two remedies for the explosion of a GP search space in high-dimensional problems—enrichment of genetic material and an efficient fitness function—we propose a new GP system for transformational dimensionality reduction. The output of the system will be a few high-level constructed features that can carry the important information of a large number of original features. The overall design of the system is similar to that of the previous chapter; however we use the above mentioned techniques to compensate for the computation time required in large dimensionality reduction problems.

Figure 4.8 shows the overall process of the proposed system. The dataset with the original features is divided into the training and test sets. The training set is first used for enrichment of genetic material; a class-wise orthogonal transformation is applied to the original features to make encapsulating terminals. The original variable terminals and the encapsulating terminals constitute the extended variable terminal set. The GP search is then conducted to construct a set of high-level features for each target class in the problem. As the number of distinct class labels in a classification problem is usually much smaller than the number of original features in the problem, the dimensionality of the problem is indirectly decreased after the GP run is completed. Based on the constructed features, the training set and test set are then transformed into a new training set and a new test set. If a constructed feature uses an encapsulating terminal, the corresponding class-wise orthogonal transformation will be applied to the test data to calculate the value of the constructed feature.

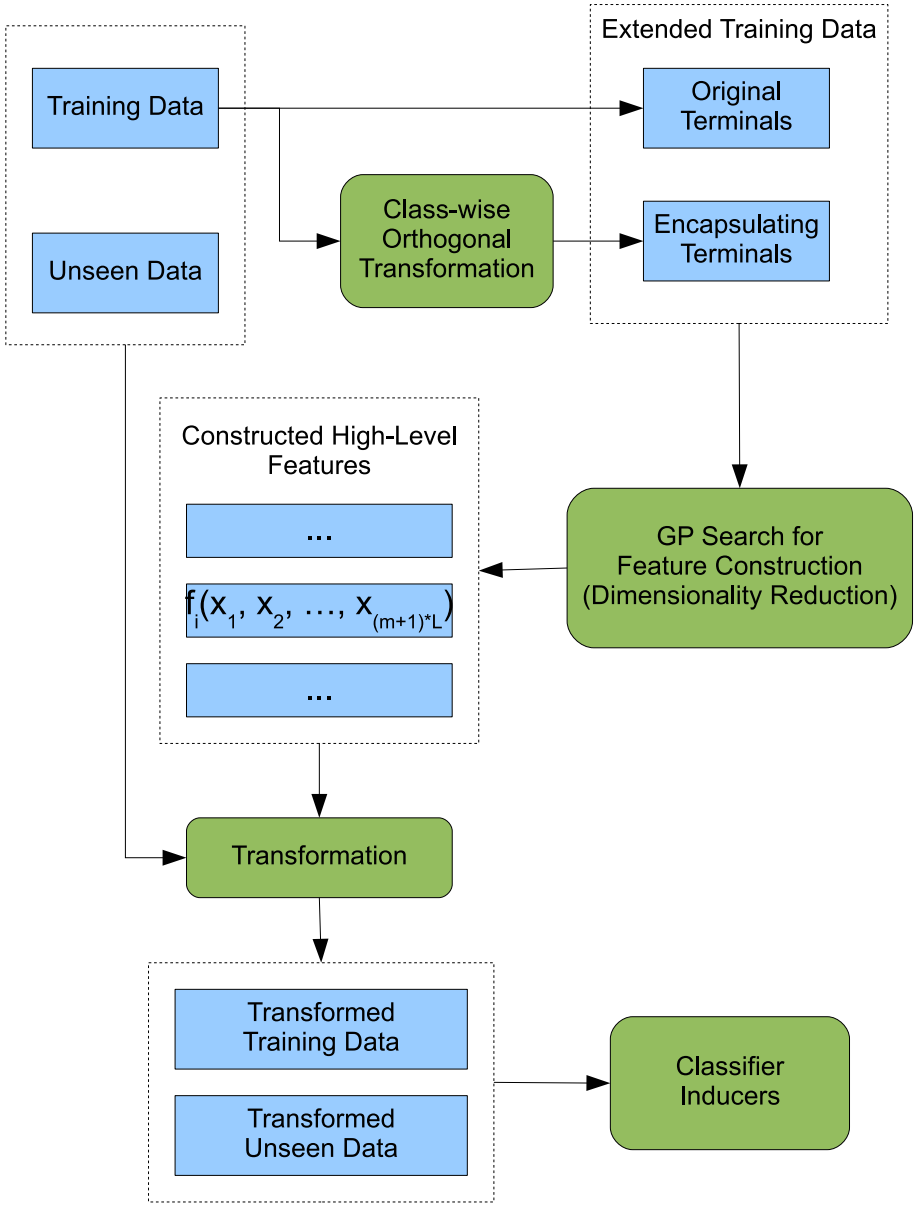


Figure 4.8: Overview of the proposed GP-based dimensionality reduction system.

4.4.1 Algorithm

The main body of our proposed GP-based dimensionality is represented in Algorithm 6.

The basic principles of the algorithm are the same as those of the multiple-feature construction algorithm presented in the previous chapter. The major differences, however, are using an extended dataset and a more efficient fitness function. The GP search is conducted for every target class in the problem. With L class labels in a classification problem there will be L high-level constructed features as the result of the algorithm. However, since m, the number of original features in a classification problem, is usually far larger than L, there will be some degree of dimensionality reduction as a result of the construction process.

4.5 Empirical Results

4.5.1 Design of Experiments

The experiments are designed to evaluate the proposed system from two complementary aspects: changes in dimensionality and changes in classification performance. One cannot investigate either of these two aspects individually; a mere measure of dimensionality reduction regardless of changes in classification performance is not very meaningful as any ratio of dimensionality reduction can be obtained by any applicable algorithm if performance is not an important factor. On the other hand, measuring only classification performance is a good measure for feature construction rather than dimensionality reduction. Therefore the two factors are considered together in the experiments and it is studied how they affect each other. We will conduct two sets of experiments.

The goal of the first set of experiments is to evaluate the effectiveness of the proposed GP-based transformational dimensionality reduction system. That is, we want to know, using the proposed algorithm, how

Algorithm 6: $GP\text{-}based\text{-}Dimensionality\text{-}Reduction(\mathbf{D})$

```
/* Given a training dataset the algorithm uses GP
   to construct a few features to transform the
   input space to a new space.                    */
```

Input: $\mathbf{D}$, a dataset of the form $\mathbf{D} = (\mathbf{X}, \mathbf{c})$ where $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$ is a set of vectors of length n of observations of the m original features in the problem and $\mathbf{c}$ is a vector of class labels for the corresponding observations in $\mathbf{X}$

Output: $\mathcal{F}$, the set of constructed features

```
1  F ← {} ;        // initialising the set of constructed features
2  D+ ← Create-Extended-Dataset(D) ;          // Algorithm 4
3  foreach c* ∈ C = {c_1, c_2, ..., c_L} do
4      P ← create a new initial population;
5      best-fitness ← +∞;     // initialising the best fitness to
       the worst.
6      while ¬max-generations ∧ best-fitness ≠ 0 do
7          foreach φ ∈ P do
8              φ_fitness ← DR-Fitness(X+, φ, c*) ;        // Algorithm 5
9              if φ_fitness < best-fitness then
10                 best-program ← φ;
11                 best-fitness ← φ_fitness;
12         perform selection;
13         perform genetic operators;
14     F ← F ∪ {best-program};
15 return F;
```

much dimensionality reduction can be achieved and compared to the ordinary approach, where all the original features are fed to the classifier, how the resulting transformational reduction affects the classification performance. For this purpose, all the original features are fed to the decision tree classifier and the classification performance is recorded for each problem. This gives a baseline for each problem without dimensionality reduction. Then the proposed algorithm is applied and changes in dimensionality and performance are compared to the baseline.

The goal of the second set of experiments is to find out how well our proposed algorithm can perform in comparison with a widely-used classical transformational dimensionality reduction method such as PCA. For this purpose, the datasets are transformed using PCA and then in two batches: first all the generated components and then only high-ranked components are fed to the classification algorithm. The classification performances are then compared to those from the first set of experiments.

4.5.1.1 Datasets

Four classification problems (datasets) are used in the experiments. The datasets are collected from the UCI machine learning repository [6]. All the problems have relatively large number of features. Table 4.1 summarises the main characteristics of these datasets. They include two-class and multiple-class classification problems. The Waveform dataset is an artificial dataset whose instances can be created by a program. So the number of instances is arbitrary. We used 500 instances, which were distributed evenly over three classes.

4.5.1.2 GP Settings

The standard tree-based genetic programming model is used [71]. In this model, each program produces a single floating-point number at its root as the result of its evaluation (output). Table 4.2 shows various settings of

Table 4.1: Specification of datasets used in experiments

Problem	# Features	# Instances	# Classes
JH Ionosphere	34	351	2
Sonar	60	208	2
Waveform	21	500	3
WBC-Diagnostic	30	569	2

the GP system we developed for the experiments. There is one variable terminal for each feature in the problem. A number of randomly generated constants are also used as terminals. The four standard arithmetic operators were used to form the function set. The division operator is *protected*—that is, it returns zero for division by zero. All the members of the function set are binary—they take two parameters.

The ramped half-and-half method [71] is used for generating programs in the initial population and for the mutation operator. The initial maximum program tree depth is set to 4, but it can increase to 8 during evolution. During the search process we use a heavy dynamic limit on tree depth [132] to control the code bloating. The probability of the crossover and mutation operators are adapted automatically at run time [21]. An elitist approach has been taken to keep the best individual of the generation. The initial maximum program tree depth is set to 4, but it can increase to 8 during evolution.

4.5.1.3 Evaluation Process

As shown in the system diagram, the proposed algorithm only uses the training data, and the test data is used only to measure the classification performance. Since none of the datasets that are used in our experiments come with a specific test dataset, we adopt a 10-fold cross-validation approach. At the start of each 10-fold cross-validation, the seed of a random

Table 4.2: GP Settings

Function Set:	$+, -, \times, \div$ (protected division)
Variable Terminals:	The original features ($\{x_1, x_2, \ldots, x_m\}$)
Constant Terminals:	Randomly Generated
Population Size:	2048
Number of Generations:	50
Initialisation:	Ramped half and half
Mutation:	Subtree creation
Selection:	Tournament (size=5)
Initial Tree Depth:	4
Maximum Tree Depth:	8
Mutation Probability:	Adaptive [21]
Cross-over Probability:	Adaptive [21]
Elitism:	Yes

number generator is initialised and the following steps are performed:

1. Shuffle the dataset;
2. Create 10 stratified partitions (folds);
3. For each fold repeat the following:
   (a) Take the current fold as the test set and the others as the training set;
   (b) Run Algorithm 6;
   (c) Transform the training and test set through the constructed features;
   (d) Perform classifier learning and testing.

The dataset is shuffled and stratified to 10 folds. Stratified folds have the same proportion of instances from different classes. Each time one of the

folds is taken as the test set and the remaining as the training set and then Algorithm 6 is executed. The shuffling process and the inner GP algorithm all depend on the random number generator.

We consider each execution of Algorithm 6 as one *GP job* and each job involves L *GP runs*, where L is the number of distinct class labels in the problem. There are 10 GP jobs in each 10-fold cross-validation. Since GP is a stochastic process, we need to have a number of GP runs before being able to extract any reliable statistics. We repeat the above-mentioned process three times. This gives us $3 \times 10 \times L$ GP runs in total. In all experiments, the J48 implementation of the C4.5 decision tree inducer [127, 152] is used for classification. The decision tree inducer is used to learn a new decision tree classifier based on the transformed training set. The classifier is then applied to the transformed test set and the performance is measured. Table 4.3 summarises different parameters involved in the evaluation.

Table 4.3: Evaluation Settings

Validation:	10-fold cross-validation with stratified folds
GP jobs:	30
Total GP runs:	$3 \times 10 \times L$ (L is the number of class labels)
Classifier:	C4.5 Decision tree (J48 version)
Evaluation Modes:	a) Using augmented datasets and b) Using constructed features only

4.5.2 Results and Analysis

4.5.2.1 Effectiveness of the Algorithm

Table 4.4 shows the number of features in different stages of the proposed GP process. The first column shows the number of original features in the problem. The second column is the number of features in the extended terminal set after the encapsulating terminals are added. Note that the

Table 4.4: Dimensionality reduction

Problem	# of Features			Reduction
	Original	Extended	Constructed	Rate
JH Ionosphere	34	102	2	94.1%
Sonar	60	180	2	96.7%
Waveform	21	84	3	85.7%
WBC-Diagnostic	30	90	2	93.3%

number of features at this stage is $m \times (L + 1)$ where m and L are respectively the number of original features and the number of distinct class labels in a given classification problem. For example, in the JH Ionosphere problem, there are $34 \times (2 + 1) = 102$ features in the extended terminal set.

The third column in Table 4.4 is the number of constructed features (output of the GP system), which is equal to the number of distinct classes in the problem. The fourth column shows the dimension reduction ratio for each problem, which is calculated using $\frac{\#Original - \#Constructed}{\#Original}$. The reduction is of course due to the fact that the number of classes in these problems is less than the number of features. In all the problems, the dimensionality has been decreased. The reduction rate, however, is different from one problem to another. The average reduction rate is 92% and it is higher than 85% in all the four problems.

Since the reduction rate on its own, without considering the changes in the classification problem, is not very meaningful, changes in the classification performance are considered with relation to dimensionality reduction. In general, one would like to have as much reduction as possible without a considerable deterioration in the classification performance.

Table 4.5 shows the classification performance of the J48 decision tree before and after using the proposed dimensionality reduction algorithm. The first column after the problem names shows the dimensionality re-

duction rates from the previous table. The second column shows the classification performance using the original features set. The next column shows the average and the standard error of the classification accuracy when the GP-constructed features are used to reduce the dimensionality. In each row, the numbers printed in boldface are the highest performance. The t value is calculated by $t = \frac{\bar{X}-\mu_0}{s/\sqrt{n}}$ where μ_0 indicate the accuracy using the original features, $\bar{X}$ indicates the estimated mean of accuracy using transformed features, s is the standard deviation and n is the number of repetitions which is 30. The probability values come from a T distribution with 29 degrees of freedom and show the confidence level at which the accuracy using augmented features outperform the accuracy using the original features.

Table 4.5: Classification performance before and after dimensionality reduction

Problem	Reduction Rate	Classification Accuracy			t test	
		Original	GP	s	t	$P\{T \leq t\}$
JH Ionosphere	94.1%	0.896	**0.914**	0.014	7.042	99.99%
Sonar	96.7%	0.732	**0.803**	0.023	16.90	99.99%
Waveform	85.7%	0.764	**0.857**	0.018	28.29	99.99%
WBC-Diagnostic	93.3%	0.935	**0.967**	0.006	29.21	99.99%

Comparing the classification performance achieved by the GP-constructed features with the classification performance when all the original features are used, we find that for all the problems, the new system has been able to improve the performance while considerably reducing the number of dimensions. The standard error in all the four problems is quite low suggesting that the GP results are fairly consistent from run to run and statistically significant.

4.5.2.2 Comparison with the PCA method

In this section we intend to compare our proposed system to PCA. The outputs of applying PCA transformation to a problem with m features are m components (another term for constructed-feature in PCA terminology). PCA ranks the resulting components so the user can choose as many high-ranked components as required. Since decision trees are used for classification and decision trees come with their own feature selection algorithm, in one of the experiments all the resulting components are fed to the decision tree classifier. In the next experiment, however, to make sure that the comparisons are fair, we feed only as many high-ranked features to the classifier to keep the dimensionality reduction ratio the same as that achieved by our proposed algorithm.

Table 4.6 shows the outcome of our experiments. The first and the last columns under "Classification Performance" are like those in Table 4.5, showing the classification performance before and after using the proposed dimensionality reduction method. The second column shows the classification performance when all the features are transformed by the PCA method to a new set of components. This includes all the generated components, which are as many as the number of the original features. The third column (PCA-DR) shows the classification performance when only high-ranked components are selected from the PCA transformation. The number of selected components is equal to the number of features generated by the GP system. Outperforming performances are printed in boldface.

Comparing the classification performance obtained by using all the components generated by PCA (PCA column), with the classification performance obtained by using the L GP-constructed features (GP column), in all the problems, the proposed GP system outperforms the PCA method. When comparing the classification performance obtained by using L top components generated by PCA (PCA-DR column), with the classification performance obtained by using the L GP-constructed features (GP col-

Table 4.6: Classification Performance

Problem	Classification Performance				
	Original	PCA	PCA-DR	GP	s.e.
JH Ionosphere	0.896	0.866	0.818	**0.914**	0.014
Sonar	0.732	0.749	0.485	**0.803**	0.023
Waveform	0.764	0.769	**0.862**	0.857	0.018
WBC-Diagnostic	0.935	0.926	0.931	**0.967**	0.006

umn), in 3 problems out of 4, the proposed GP system outperforms the PCA method. Only in one problem (Waveform dataset) the PCA method perform slightly better than the proposed method; the difference, however, is very small. Overall, considering the stand error of the GP method, the proposed algorithm demonstrates a significant superiority over PCA.

4.6 Discussions

The goal of this chapter was to develop a GP approach to transformation dimensionality reduction to reduce the dimensionality of classification problems and improve the classification performance. The goal has been achieved by proposing a GP-based system that transforms the original input space into a new space via a set of GP-constructed features. The number of dimensions in the new input space is equal to the number of classes in the problem. Therefore, dimensionality reduction is achieved by taking advantage of a natural characteristic of the majority of classification problems which is having larger number of features than number of distinct classes.

Since the space of possible transformations (GP programs) grows exponentially with respect to the number of original features in a classification task, we had to introduce some heuristics in order to be able to perform

a GP search that is likely to succeed in finding acceptable solutions in a reasonable amount of time. A class-wise orthogonal transformation was proposed to enrich the genetic material of the GP search by adding some encapsulating terminals to the variable terminal set. We also introduced an entropy-based fitness function that is computationally inexpensive.

The proposed GP system was evaluated and compared with the standard decision tree approach, and a combination of PCA and the decision tree approach. The results show that the proposed system is able to achieve significant dimensionality reduction and performance improvement in most classification problems. The results also show that, in most cases, the proposed system can outperform the PCA method in terms of dimensionality reduction and classification performance. This suggests that GP is an effective approach to transformational dimensionality reduction in classification problems.

The ratio of dimensionality reduction cannot be directly controlled via the proposed algorithm; it depends on the number of original features and the number of distinct classes in the problem. This could be a disadvantage if the user needs to have an arbitrary number of constructed features (or reduction ratio). There are some remedies for this limitation, however. If the user needs more features than the number of classes in the problem, the algorithm can be modified to construct features for a combination of classes [109]. If the user needs fewer constructed features (dimensions) than the number of classes in the problem, the algorithm can construct features for certain classes in the problem. As we saw in a sample constructed feature in the previous chapter, if a constructed feature is good at separating instances of a certain class, it might be good at separating instances of other classes as well. Therefore, constructing fewer features than the number of classes in the problem might be achievable without a significant loss in classification performance.

Chapter 5

Single-Feature Ranking

5.1 Introduction

Feature ranking is a common approach to feature selection and dimensionality reduction. It provides a measure of usefulness for the conditional variables (input features) of a classification task. Dimensionality reduction can be achieved by ranking features and then using only a few high-rank features for classification. In this chapter, we use GP to rank the input features of a classification task.

5.1.1 Motivations

Most existing feature ranking methods rank single features—that is, they measure the relative importance of a single feature in predicting target concepts (class labels). An advantage of single-feature ranking is that users have the freedom to have their required dimensionality reduction ratio; users can select any number of high-rank features that gives them a desired balance of accuracy, interpretation, dimensionality reduction and execution time [see Section 2.2.4, page 20]. Single-feature ranking can also be helpful in studying the underlying nature of classification problems. Compared to subset-feature ranking algorithms, single-feature ranking al-

gorithm have the advantage of being computationally inexpensive [146].

Almost all the existing single-feature ranking methods are filter-based. Although these methods enjoy the typical advantages of the non-wrapper approach, they have a common flaw. In many real-world classification problems, an original input feature alone may not show any relevance to target classes. The feature, nonetheless, might be quite relevant in the presence of some other features. Therefore, for a single-feature ranking method to be able to evaluate the importance of a feature properly, it should take into account the context (the presence or absence of some other features) in which the feature might provide useful information.

5.1.2 GP Suitability for Feature Ranking

GP's expressiveness and superiority in dynamically finding mathematical functions based on an objective function make it a promising choice for discovering the relationship between conditional and decision variables of a classification task. Because GP programs/expressions are not bound to any predefined template and can be of any type (linear, non-linear, trigonometric, logical, etc), they can reveal a wide variety of relationships between the input features and target classes. In this chapter, GP is used to discover existing functional dependencies between features and target classes and then rank the input features based on their influence on the discovered dependencies.

5.1.3 Chapter Goals

The goal of this chapter is to devise a non-wrapper GP-based method for ranking the individual input features of a classification problem. The ranking must be in a way that important/informative features get higher ranks while noisy and irrelevant features get lower ranks. As far as possible, it is desired to have a system that considers the influence of other features (context) when finding the rank of a feature. By selecting a number of

high-rank features, we aim to reduce the dimensionality while maintaining/improving the classification performance.

5.2 GP-based Single-Feature Ranking

A top-down approach is taken in this chapter. We first propose the general idea for using GP for single-feature ranking and then work out the details.

5.2.1 The Main Idea

One of the desired characteristics for a single-feature ranking method is to be context-sensitive. That is, if a feature cannot perform well individually—and this is the case in majority of real-world problems—the feature should be ranked considering the presence of other features that can possibly increase its performance. Of course, there is always a trade-off between the extent to which this goal can be satisfied and the computational effort; to find the perfect solution, one could perform an exhaustive search, which is clearly computationally infeasible.

To find the relationship between a feature and target classes, a multivariate model is needed; the model should map a number of input features to target classes. If a model performs well, then one can infer that the quality of at least a subset of the input features used in the model is high. Of course, there might be cases where some of the features used in the model do not actually have any effect on the performance of the model (e.g. a feature multiplied by a zero coefficient in a linear model). Therefore, in addition to finding a good model that embodies the mathematical relationship between features and class labels, one should distinguish between the influential and non-influential features used in the model.

GP is used to find a good multivariate model. GP has had a very successful history in evolving classifiers. In this chapter, GP is used to evolve some very simple classification models called **weak classifiers**, each of

which can only separate instances of one class from instances of other classes. We then consider the use of a subset of features in a good weak classifier as an indication of the subset being promising.

To distinguish influential features from those that may enter a GP-constructed model randomly (e.g. features in an intron), we construct a large enough number of models (weak classifiers) to have sufficient statistics on the frequency of occurrence of features in good models. Frequent participation of a feature in several good classification models indicates the potential importance of the feature. Even if an irrelevant feature enters a good model by chance, it is very unlikely that it can appear in a large number of other models as well. Therefore, in the long run, the frequency of appearance of relevant features in good models will be higher than less relevant ones.

A scoring mechanism is defined for the single (original) features based on their frequency of appearance in good models. Features that are more frequent in good classification models score more. The features are then re-ordered based on the yielded scores and users can choose a few high-rank features to achieve dimensionality reduction. To evaluate the effectiveness of the proposed method, different numbers of high-rank features are fed to different classifiers and analyse the performance.

5.2.2 Overall System Diagram

Figure 5.1 depicts the abstract diagram of the system. Given a dataset for a classification task, we conduct a number of GP runs each of which produces a high-fitness weak classifier. The classification performance of the weak classifier is used as the fitness [see Section 5.3]. The best programs (weak classifiers) are stored in a **program collection**. The input features score points for their appearance in weak classifiers. The score is proportional to the fitness of the corresponding classifier. The features are then ranked based on their score [see Section 5.4]. A projected dataset is created

by selecting a number of high-rank feature from the original dataset. The projected dataset will be fed to a classifier inducer to train a classifier.

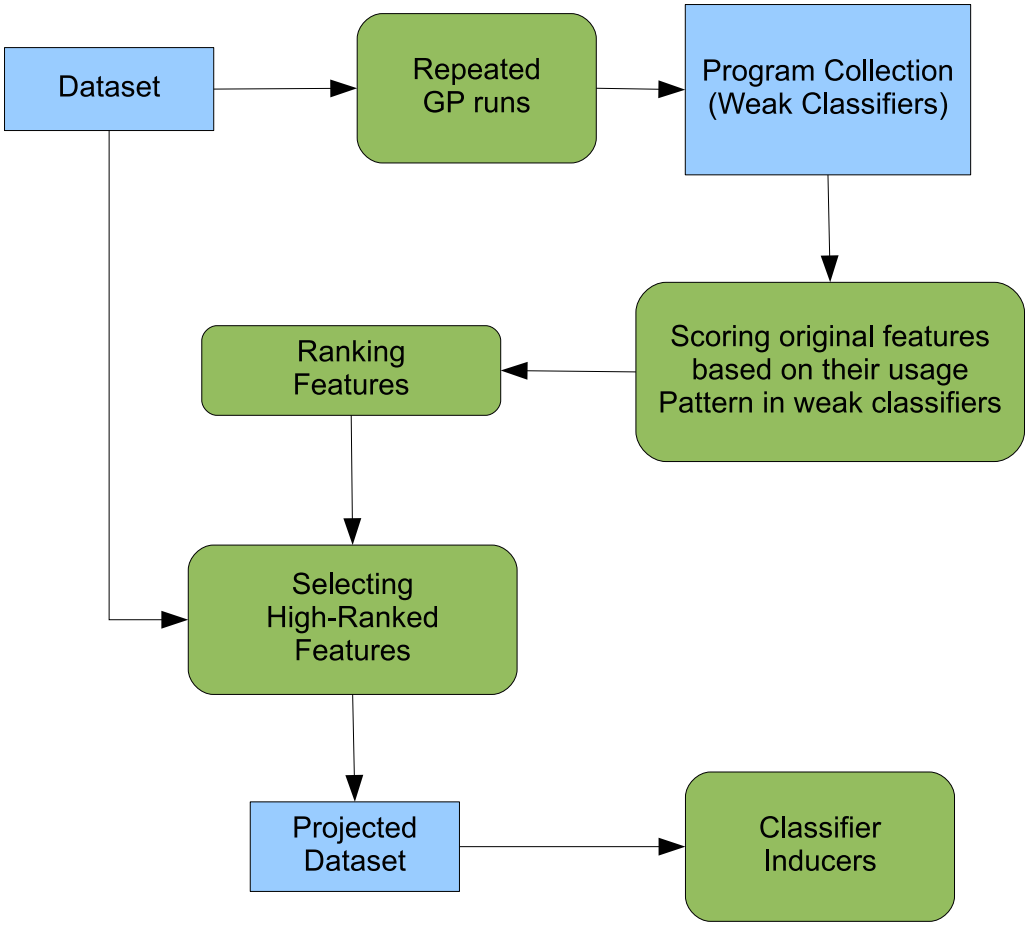


Figure 5.1: Overview of the system.

5.3 Using GP to Build Weak Classifiers

A weak classifier is a simple learning model that can perform slightly better than random guessing (e.g. more than 50% performance in binary classification) [55]. Weak classifiers are generally not good as standalone classifiers, but their weighted combination is usually used to build a complete

classification model (e.g. boosting). Since weak classifiers do not necessarily have to separate all the instances, they are much less complex compared to complete classification systems and therefore, they are computationally less expensive to learn/evolve. Here, we use GP to build weak classifiers.

5.3.1 Classification Model

In Chapter 3, it was observed that some of the constructed features can potentially perform simple classification by placing instances of different classes in different bands along the axis of a constructed feature. We can use GP to construct a transformation that gathers the majority of instances of a particular class in a continuous interval that contains as little occurrence of instances from other classes as possible. Given such an interval, one can build a weak classifier by checking the result of the transformation against the boundaries of the interval. Similar to the case of multiple feature construction, for a classification task with L distinct class labels, L weak classifiers can be evolved[1].

The dataset of a classification task is of the form $\mathbf{D} = (\mathbf{X}, \mathbf{c})$, where $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$ is a set of vectors of length n containing samples from the m original features in the problem and $\mathbf{c}$ is a vector of class labels for the corresponding observations in $\mathbf{X}$. We also have $\mathbf{c}[i] \in \mathbb{C}$ for $i \in \{1, 2, \ldots, n\}$ where $\mathbb{C} = \{c_1, c_2, \ldots, c_L\}$. Suppose $\phi_{c^\star}$ is a GP program that has been evolved for a weak classifier that separates the instance of class $c^\star$ from other classes. The GP program generates a mapping of the form $\phi_{c^\star} : \mathbb{R}^m \mapsto \mathbb{R}$ that transforms the multi-dimensional input matrix $\mathbf{X}$ to a one dimensional vector $\mathbf{y}$. Consider a continuous interval $I_{c^\star} = (l_{c^\star}, u_{c^\star})$ on $\mathbf{y}$ that covers the majority of instances from class $c^\star$. We define a binary

[1] As mentioned in Chapter 4, one could build more than L weak classifiers by considering weak classifiers that separate different combinations of classes from each other.

weak classifier $WK_{c^\star}$ as

$$WK_{c^\star}(x_1, x_2, \ldots, x_m) = \begin{cases} positive, & \phi_{c^\star}(x_1, x_2, \ldots, x_m) \in I_{c^\star} \\ negative, & \text{otherwise.} \end{cases} \tag{5.1}$$

The classifier treats $c^\star$ as positive and other classes as negative. An instance is classified as $c^\star$ (positive) if $y = \phi_{c^\star}(x_1, x_2, \ldots, x_m)$ falls in the interval of the class; and negative otherwise.

5.3.2 GP Algorithm and the Fitness Function

For each GP program (transformation), there is a corresponding binary weak classifier. To find $WK_{c^\star}$, we first have to find the interval of the given class $c^\star$ for the GP individual (transformation). In fact, one can think of the upper and lower bounds of the interval as the parameters of the weak classifier $WK_{c^\star}$. To find the interval of class $c^\star$, we use Algorithm 1 ($Find\text{-}Interval$) in Chapter 3 (page 50). The algorithm finds a continuous interval that covers the majority of the instances of class $c^\star$ while excluding instances at extreme left and right to diminish the effect of possible outliers or noisy observations.

Once the interval is found, $WK_{c^\star}$ is built and its classification performance is used to determine the fitness of the corresponding GP program. Since the interval of $c^\star$ covers the majority of the instances from that class (usually 99%), the true positive rate of the classifier is always very high (close to 1). The false positive rate, however, depends on how many instances of other classes fall in the interval of class $c^\star$. Therefore, to improve the classification performance, the false positive rate should be minimised. The fitness of a GP program is defined to be

$$fitness = 1 - FPR = TNR = \frac{TN}{TN + FP} \tag{5.2}$$

where FPR is the false positive rate, TNR is the true negative rate, TN is the number of instances correctly rejected (true negative) and FP is the number of instances incorrectly accepted (false positive).

Algorithm 7: $Evolve\text{-}Weak\text{-}Classifier(\mathbf{D}, c^{\star})$

```
/* The algorithm returns the created GP program
   whose weak classifier for class c* performs the
   best among others.                                */
```

Input: $\mathbf{D}$, a dataset of the form $\mathbf{D} = (\mathbf{X}, \mathbf{c})$ where $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$ is a set of vectors of length n of observations of the m original features in the problem and $\mathbf{c}$ is a vector containing the class labels of the observations

Input: $c^{\star}$, the class label for which a weak classifier should be built

Output: $(\phi, fitness_{\phi})$, a pair containing the best performing GP individual, program ϕ, and its fitness

1 $\mathcal{P} \leftarrow$ create a new initial population;
2 $best\text{-}fitness \leftarrow 0$; // initialising the best fitness
3 **while** $\neg max\text{-}generations \wedge best\text{-}fitness \neq 1$ **do**
4 **foreach** $\phi \in \mathcal{P}$ **do**
5 $\mathbf{y}[i] \leftarrow \phi(\mathbf{x_1}[i], \mathbf{x_2}[i], \ldots, \mathbf{x_{m\times(L+1)}}[i])$, $\forall i \in \{1, 2, \ldots, n\}$;
  // using the GP program to transform the data
6 $(l, u) \leftarrow Find\text{-}Interval(\mathbf{y}, \mathbf{c}, c^{\star})$; // finding class interval
7 $TN \leftarrow 0$;
8 **for** $i \leftarrow 1$ **to** n **do**
9 **if** $(\mathbf{y}[i] < l \vee \mathbf{y}[i] > u) \wedge \mathbf{c}[i] \neq c^{\star}$ **then**
10 $TN \leftarrow TN + 1$; // it is correctly rejected
11 $fitness_{\phi} \leftarrow \frac{TN}{\#negative\text{-}instances}$;
12 **if** $fitness_{\phi} > best\text{-}fitness$ **then**
13 $best\text{-}program \leftarrow \phi$;
14 $best\text{-}fitness \leftarrow fitness_{\phi}$;
15 perform selection and genetic operators;
16 **return** $(best\text{-}program, best\text{-}fitness)$;

Algorithm 7 shows the steps involved in evolving a GP program that has a good weak classifier for class $c^\star$. The main loop, implementing the GP search, will terminate either when the maximum number of generations is reached or when the best possible fitness, 1, is achieved. The algorithm keeps track of the best program by updating the value of the variable $best\text{-}fitness$. At line 5, to find the fitness of each program in the population, the program is first used to transform the dataset; for each instance, the program uses the feature values from $\mathbf{X}$ and produces a single floating point value that is stored in vector $\mathbf{y}$. At line 6, (l, u), the lower and upper boundaries of the interval of class $c^\star$ along $\mathbf{y}$, is determined. Then at lines 8–11, the fitness of the program is determined by measuring the performance of the corresponding weak classifier on the transformed dataset. The best GP program (weak classifier) along its fitness (true negative rate) will be returned as the result.

5.4 Ranking Features

A scoring mechanism is defined by which features receive credit for their appearance in a GP program. The amount of credit will be proportional to the performance of the corresponding weak classifier. The score gained by a feature f due to its appearance in the GP program ϕ is

$$score_{f,\phi} = \begin{cases} \frac{fitness_\phi}{|terminals\text{-}of(\phi)|}, & f \in terminals\text{-}of(\phi) \\ 0, & \text{otherwise} \end{cases} \tag{5.3}$$

where $f \in \mathbb{F} = \{f_1, f_2, \ldots, f_m\}$, $fitness_\phi$ is the fitness of program ϕ which is determined by equation (5.2), $terminals\text{-}of(.)$ is a function that returns a set of variable terminals (features) used in a the given GP program, and $|.|$ is the set cardinality. Effectively, this equation divides the fitness of a program equally among the features used in the program.

GP programs may contain *introns*—that is, there are some portions in the program tree with zero or very little contribution towards the acquired

fitness. In other words, there might be some features used in a weak classifier that do not have any effect on the performance of the classifier. Using equation (5.3), however, non-contributing features receive as much score as contributing features. To compensate for this effect, instead of using one GP program, a collection of GP programs is used to determine the score of features. Each program in the collection is the result of a GP run and has a well-performing corresponding weak classifier.

Features accumulate the scores they receive for each GP program in the collection. So the features that are used more frequently in the GP programs gain higher scores. On the other hand, since the appearance of non-contributing features in introns is completely random and they might be different from one program to another, at the end non-contributing features receive lower scores. The normalised score of feature f after considering all the GP programs in the collection is obtained by

$$score_f = \frac{\sum\limits_{\phi \in \Phi} score_{f,\phi}}{\sum\limits_{f \in \mathbb{F}} \sum\limits_{\phi \in \Phi} score_{f,\phi}} \tag{5.4}$$

where Φ is the collection of GP programs. The denominator normalises the final score relative to the total score gained by all the features. The normalised score shows the relative importance of features in a classification problem.

5.4.1 Algorithm

Algorithm 8 shows the main algorithm of the proposed GP-based single-feature ranking system. The algorithm conducts a number of GP jobs to create a collection of GP programs that have well-performing corresponding weak classifiers. Each GP job includes L GP runs to create L weak classifiers (GP programs), one for each class in the problem. The number of jobs is presented by $\#jobs$, thus in total there will be $\#jobs \times L$ weak classifiers. The algorithm calls Algorithm 7 ($Evolve\text{-}Weak\text{-}Classifier$) to

create weak classifiers.

Algorithm 8: $GP\text{-}based\text{-}Single\text{-}Feature\text{-}Ranking(\mathbf{D})$

```
/* Given a training dataset, the algorithm uses GP
   to build a collection of binary weak
   classifiers and then rank the original features
   based on their influence.                        */
```

Input: $\mathbf{D}$, a dataset of the form $\mathbf{D} = (\mathbf{X}, \mathbf{c})$ where $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$ is a set of vectors of length n of observations of the m original features in the problem and $\mathbf{c}$ is a vector containing the class label of observations

Output: $(\mathbf{s}, \mathbf{r})$, a pair containing the scores and rank of the features

$\Phi \leftarrow \{\}$; // the collection of GP programs

$current\text{-}job \leftarrow 1$;

repeat

  foreach $c^\star \in \mathbb{C} = \{c_1, c_2, \ldots, c_L\}$ **do**

    $\Phi \leftarrow \Phi \cup \{Evolve\text{-}Weak\text{-}Classifier(\mathbf{D}, c^\star)\}$;

until $current\text{-}job = \#jobs$;

$\mathbf{s} \leftarrow \mathbf{0}_{1\times m}$; // vector of scores initialised to zero

$sum\text{-}of\text{-}scores \leftarrow 0$;

foreach $(\phi, fitness_\phi) \in \Phi$ **do**

  for $i \in \{1, 2, \ldots, m\}$ **do**

    if $f_i \in terminals\text{-}of(\phi)$ **then**

      $\mathbf{s}[i] \leftarrow \mathbf{s}[i] + \frac{fitness_\phi}{|terminals\text{-}of(\phi)|}$;

      $sum\text{-}of\text{-}scores \leftarrow sum\text{-}of\text{-}scores + \frac{fitness_\phi}{|terminals\text{-}of(\phi)|}$;

for $i \in \{1, 2, \ldots, m\}$ **do**

  $\mathbf{s}[i] \leftarrow \frac{\mathbf{s}[i]}{sum\text{-}of\text{-}scores}$; // normalising the scores

$\mathbf{r} \leftarrow indexed\text{-}descending\text{-}sort(\mathbf{s})$;

return $(\mathbf{s}, \mathbf{r})$;

At line 7 of the algorithm, a vector $\mathbf{s}$ of size m is defined that keeps the scores of the features. For each program, the scores of the features is calculated and accumulated in this vector. At line 15, the vector is then normalised by dividing the score of each feature by the sum of the scores of all the features. Finally at line 16, the features are ranked by sorting them in the descending order of their score. The ranking result is stored in vector $\mathbf{r}$ whose elements are indexes to the original features. The feature with the highest score is considered the best and its index is the first element of $\mathbf{r}$.

5.5 Empirical Results

5.5.1 Design of Experiments

A set of experiments have been designed to evaluate the effectiveness of our proposed GP-based single-feature ranking method. Since there is no direct way to measure the performance of a feature ranking system, the system is evaluated by analysing changes in classification performance caused by using highly ranked features. For each classification task, the proposed algorithm is used to calculate the scores of the features and rank them. The proposed algorithm is then evaluated from two perspectives: its effectiveness in feature selection and its utility in dimensionality reduction.

To evaluate the effectiveness of the proposed GP-based method, we have to find out if the provided ranking reflects the actual importance of features. We consider a situation where one needs to select m' features out of the m original features to use with a classification algorithm. With no knowledge about the importance of features, features will be selected randomly. However, if a ranking mechanism showed the true importance of features, one could select m' highest-ranked features to achieve a better classification performance. Therefore, the proposed feature ranking

system is evaluated by comparing two different ways of selecting m' features: one is selecting randomly and the other is selecting features ranked as high by GP. For each given value of m', the classification performance using the two different selection methods is compared.

The other aspect of feature ranking is its utility in dimensionality reduction. In particular, we are interested to know whether a high classification performance can be achieved by using just a few highest-ranked features. If a ranking is good, we expect to see a quick rise in classification performance by adding a few highest-ranked features. On the other hand, low-ranked features are not expected to have a considerable effect on classification performance and therefore, one should be able to remove them without much deterioration in performance.

5.5.1.1 Datasets

Three datasets are used with a relatively large number of features from the UCI machine learning repository [6] in the experiments. Table 5.1 summarises the main characteristics of these datasets [see Appendix A].

Table 5.1: Specifications of datasets used in experiments

Problem	# Features	# Instances	# Classes
JH Ionosphere	34	351	2
Sonar	60	208	2
WBC-Diagnostic (WBCD)	30	569	2

5.5.1.2 GP Settings

The standard tree-based GP model is used [71]. In this model, each program produces a single floating-point number at its root as the result of its evaluation (output). Table 5.2 shows various settings of the proposed

GP-based system. There is one variable terminal for each feature in the problem. A number of randomly generated constants are also used as terminals. The four standard arithmetic operators were used to form the function set. The division operator is *protected*—that is, it returns zero for division by zero. All the members of the function set are binary—they take two parameters.

The ramped half-and-half method [71] is used for generating programs in the initial population and for the mutation operator. The initial maximum program tree depth is set to 4, but it can increase to 8 during evolution. During the search process, we use a heavy dynamic limit on tree depth [132] to control code bloating. The probability of the crossover and mutation operators are adapted automatically at runtime [21]. An elitist approach has been taken to keep the best individual of the generation. The platform is implemented in Java and grid computing is used to have parallel GP runs.

Table 5.2: GP Settings

Function Set:	$+$, $-$, $\times$, $\div$ (protected division)
Variable Terminals:	The original features ($\{x_1, x_2, \ldots, x_m\}$)
Constant Terminals:	Randomly Generated
Population Size:	1024
Number of Generations:	50
Initialisation:	Ramped half and half
Mutation:	Subtree creation
Selection:	Tournament (size=5)
Initial Tree Depth:	4
Maximum Tree Depth:	8
Mutation Probability:	Adaptive [21]
Cross-over Probability:	Adaptive [21]
Elitism:	Yes

5.5.1.3 Evaluation Process

To create a program collection, Φ, that is large enough to extract statistics required by feature ranking, 300 GP jobs are conducted. GP runs are started with a different random seed to have a variety of programs in the collection. Since all the datasets used in this chapter are binary classification problems ($L = 2$), for each dataset there will be 600 GP programs in Φ.

Table 5.3: Evaluation Settings

Validation:	10-fold cross-validation with stratified folds
GP jobs:	300
Size of Collection Φ:	$300 \times 2 = 600$ (binary classification problems)
Classifiers:	Decision Tree (J48 version of C4.5), Naïve Bayes, SVM (SMO version), Bayesian Network
Evaluation Modes:	Limiting classifiers to highest-ranked features

Table 5.3 shows the settings involved in the evaluation process. Four types of classifiers are used in our experiments, namely the J48 implementation of C4.5 decision tree [127, 152], Bayesian Networks, Naïve Bayes [57], and the SMO version of the SVM classifier [61]. Since none of the datasets that are used in our experiments come with a specific test set, we adopt a 10-fold cross-validation approach. The dataset is shuffled and stratified to 10 folds. Stratified folds have the same proportion of instances from different classes. Each time one of the folds is taken as the test set and the remaining as the training set. Weka [152] library is used for the classification and evaluation processes.

5.5.2 Results

5.5.2.1 Scores and Ranks

First the scores obtained by the input features in each of the three problems are reported. Bar charts of the scores are shown in Figure 5.2. In each chart, the horizontal axis shows the feature index, starting from 1, and the vertical axis shows the score of each feature calculated by equation (5.4). Note that the scores are relative and the absolute values are not important. Table 5.4 shows the ranks of the features in each classification task. The features are listed in the order of importance starting with the most important one.

Table 5.4: Feature ranks

Problem	Order of features
JH Ionosphere	5, 1, 3, 6, 8, 14, 4, 7, 9, 16, 25, 2, 21, 10, 15, 27, 17, 33, 34, 18, 11, 23, 13, 22, 28, 29, 20, 24, 31, 12, 19, 32, 30, 26
Sonar Dataset	11, 47, 49, 12, 45, 28, 46, 9, 27, 48, 19, 10, 36, 17, 26, 22, 13, 16, 44, 35, 34, 58, 4, 52, 5, 37, 43, 42, 54, 21, 18, 25, 38, 39, 20, 23, 41, 50, 15, 29, 8, 40, 3, 55, 32, 1, 51, 31, 30, 59, 14, 7, 2, 33, 6, 24, 56, 53, 57, 60
WBC-Diagnostic	24, 22, 28, 25, 8, 14, 2, 21, 23, 5, 29, 4, 10, 30, 11, 1, 18, 15, 7, 16, 19, 20, 27, 9, 13, 3, 26, 17, 6, 12

5.5.2.2 Effectiveness of GP-based Ranking: Comparison to the Baseline

For creating the baseline, where no ranking is available, we repeat the process of random selection of m' features several times and measure the

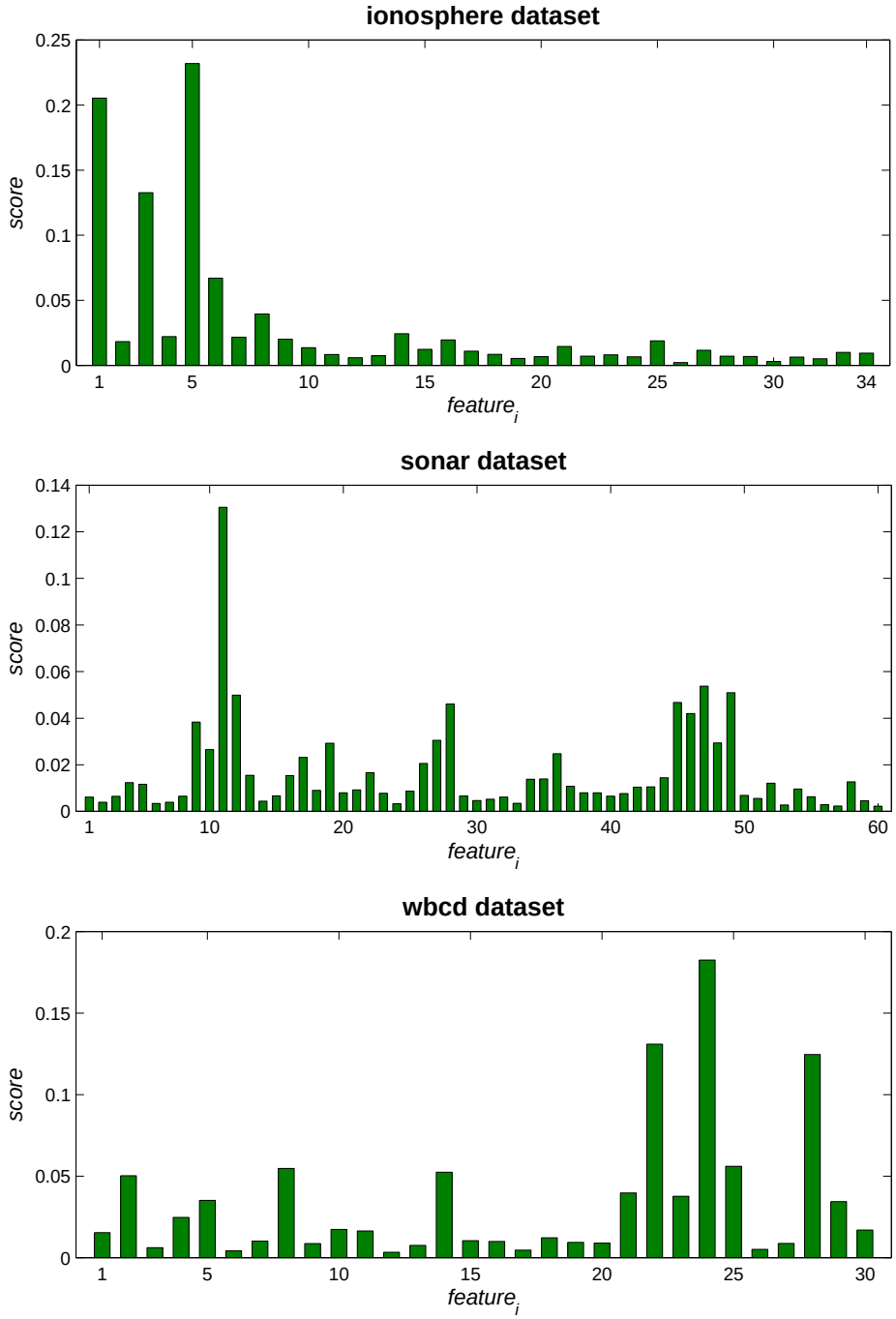


Figure 5.2: Score of features in the Ionosphere, Sonar and WBC-Diagnostic

average classification performance. The number of times we select a subset of size m' is $\min(50, \binom{m}{m'})$—that is, we will have 50 different subsets of size m' and their corresponding classification performance as long as the number of possible combinations of m' chosen from m is greater than 50. To select m' features using the ranking provided by GP, we make a subset that contains m' features with the highest ranks; that is, the feature at rank 1 (the best), the feature at rank 2, and so on up to the feature at rank m'.

Figures 5.3, 5.4 and 5.5 compare the performance obtained by the GP-based ranking and the baseline performance of the four classifiers for the three datasets respectively. Each figure has four plots corresponding to the four classifiers used in the experiments. In each plot, the horizontal axis shows the number of features used in the classification and the vertical axis shows the classification accuracy. The classification accuracy is obtained by 10-fold cross-validation. For each given subset of size m', we compare the accuracy in the baseline with that obtained by selecting features via GP-based ranking.

In all the figures, as the number of selected features increases, the performance curve of the GP ranking and the baseline get closer to each other. This is because as the number of selected features in the two method grows, the likelihood of them sharing similar features increases. In the limit when $m' = m$—that is, when all the available features are used for classification—the two curves meet each other. The performance difference between the two curves varies depending on the problem, the selected subset, and the type of classifier. However, it is noticeable that regardless of the dataset and the type of classifier, in almost all cases, the classification is higher when selection is based on the provided GP ranking.

5.5.2.3 Utility in Dimensionality Reduction

In Figures 5.6, 5.7 and 5.8 the performance of all the four classifiers are studied together. Each figure corresponds to one of the datasets used in

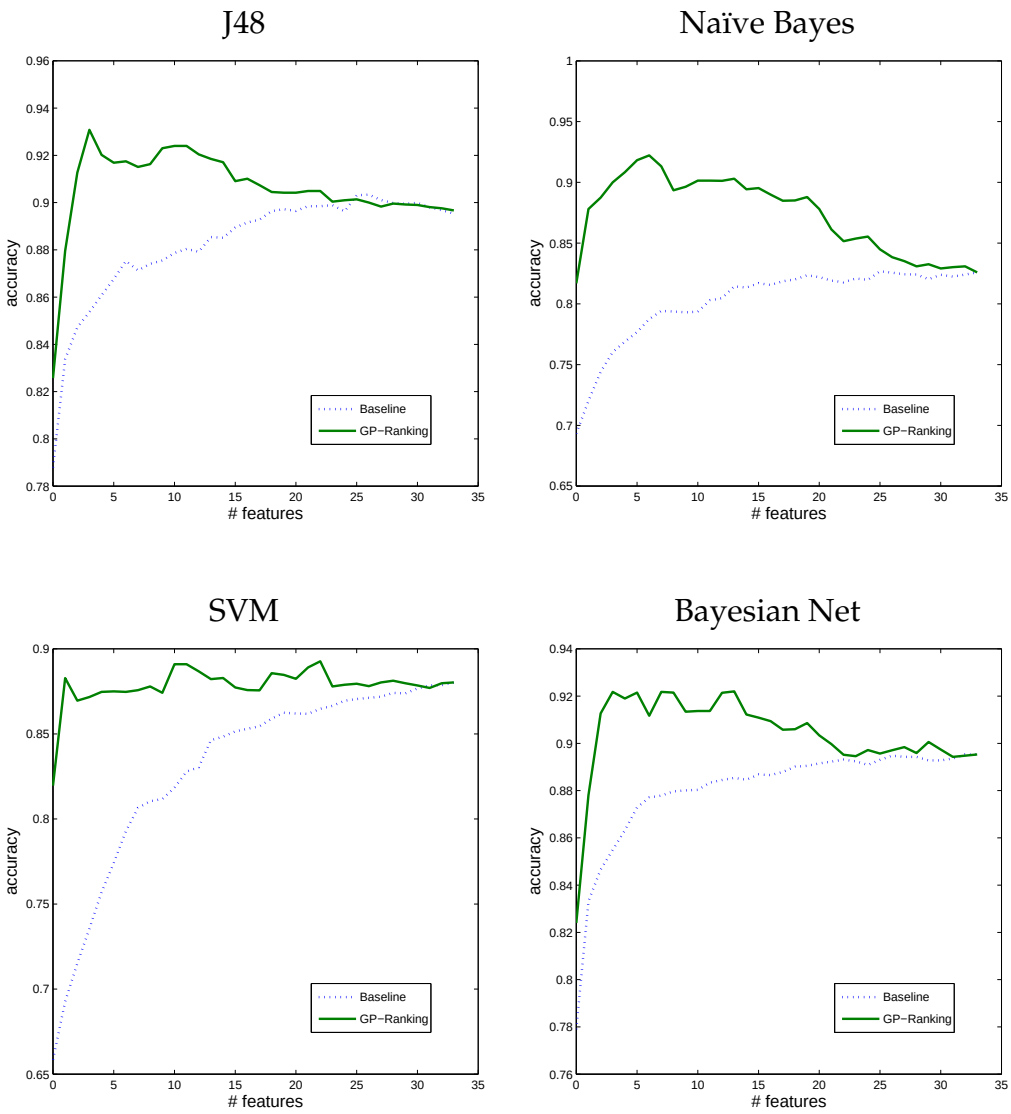


Figure 5.3: Comparison of the proposed ranking with the baseline (random selection) in the Ionosphere dataset.

the experiments. In each figure, the horizontal axis shows the number of features used in the classification and the vertical axis shows the classification accuracy. The classification accuracy is obtained by 10-fold cross-validation. In each dataset, one wants to find out with how many selected

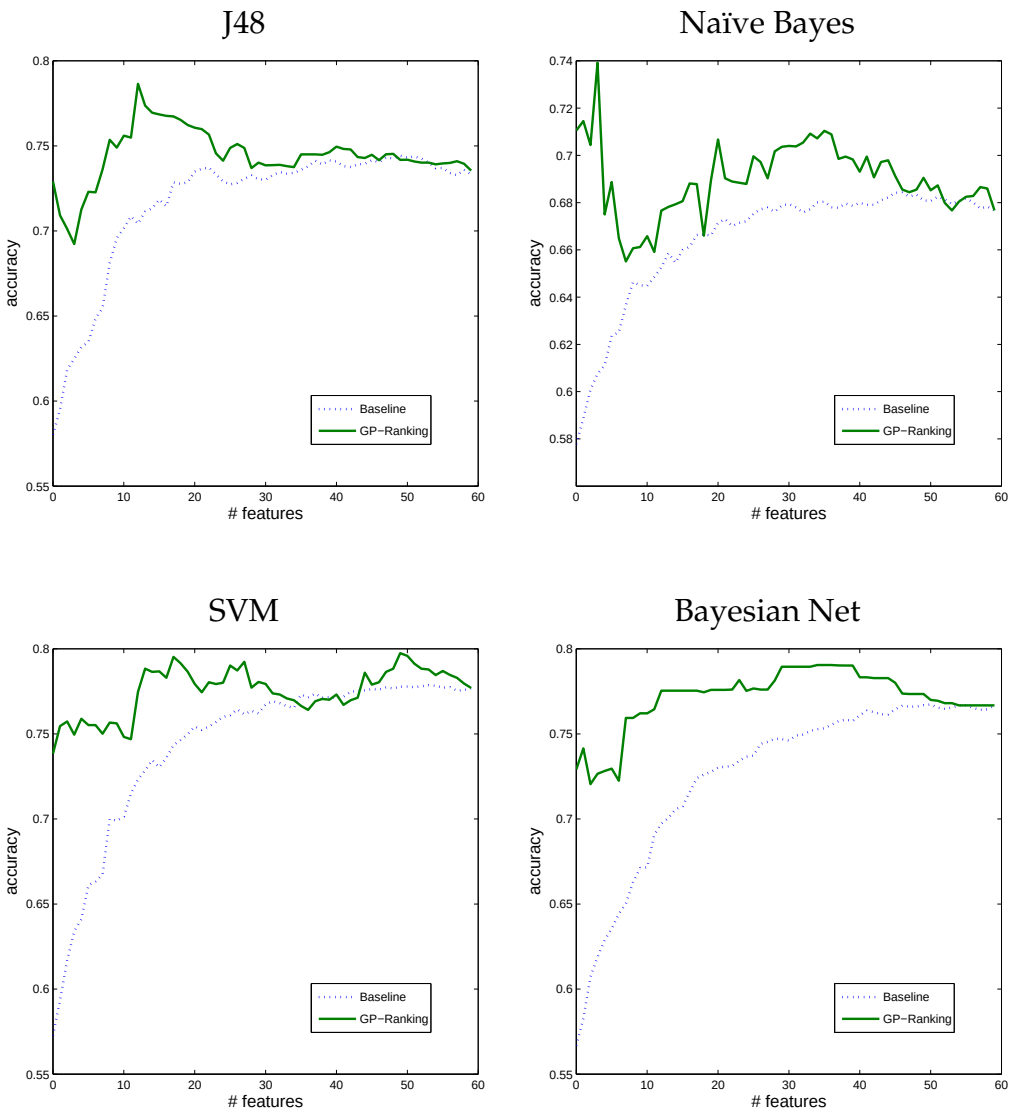


Figure 5.4: Comparison of the proposed ranking with the baseline (random selection) in the Sonar dataset.

features a classifier can achieve a performance close to (or even better than) when all the available features are used.

In the Ionosphere dataset, compared to the situation where all the 34 features are used, Naïve Bayes and SVM can do better by using only 2

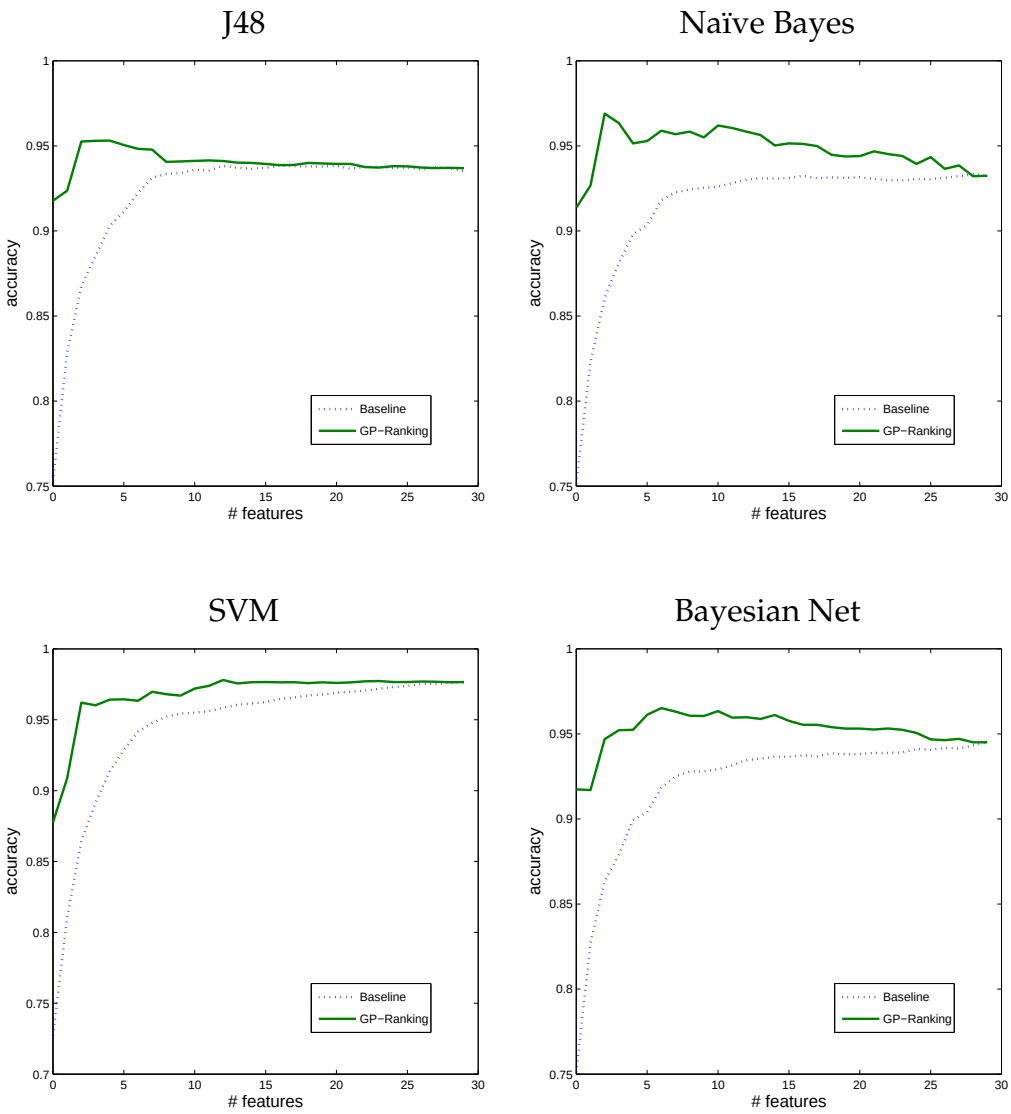


Figure 5.5: Comparison of the proposed ranking with the baseline (random selection) in the WBC-Diagnostic dataset.

features, and Decision Tree and Bayesian Network can do better by using just 3 features. In the Sonar dataset, Decision Tree, Naïve Bayes, SVM and Bayesian Network can, by using 8, 1, 14 and 13 feature(s) respectively, perform better than situations in which all 60 features are used. It is almost

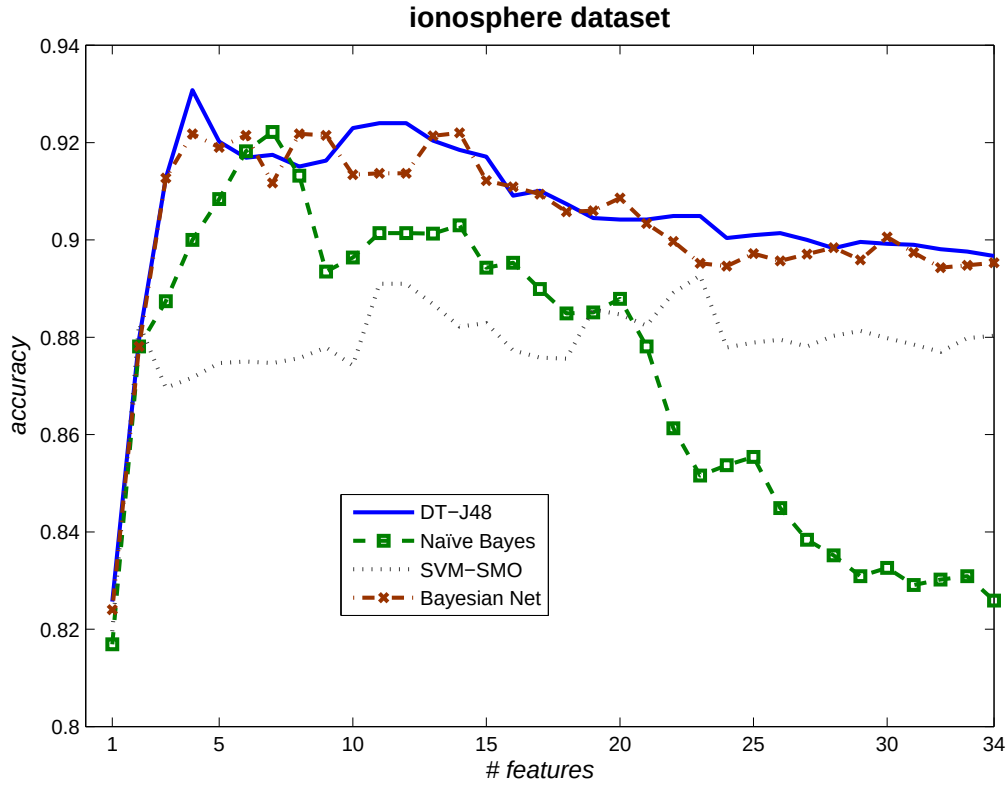


Figure 5.6: Accuracy of different classifiers in the John Hopkins University Ionosphere classification task by using different numbers of ranked features.

the same in the WBC-Diagnostic dataset where for all classifiers (except SVM) the performance obtained by using all features can be obtained by using less than 4 features out of 30.

Considering the performance figures of the three classification tasks, it is observed that classification performance increases very quickly as the first few highest-ranked features are added. For almost all the classifiers, by using less than 10% of the features, one can obtain a similar performance or better than that obtained by using all the features. Looking at the trends in the classification performance, it is revealed that in most cases, using more features with these classifiers, not only does not increase the

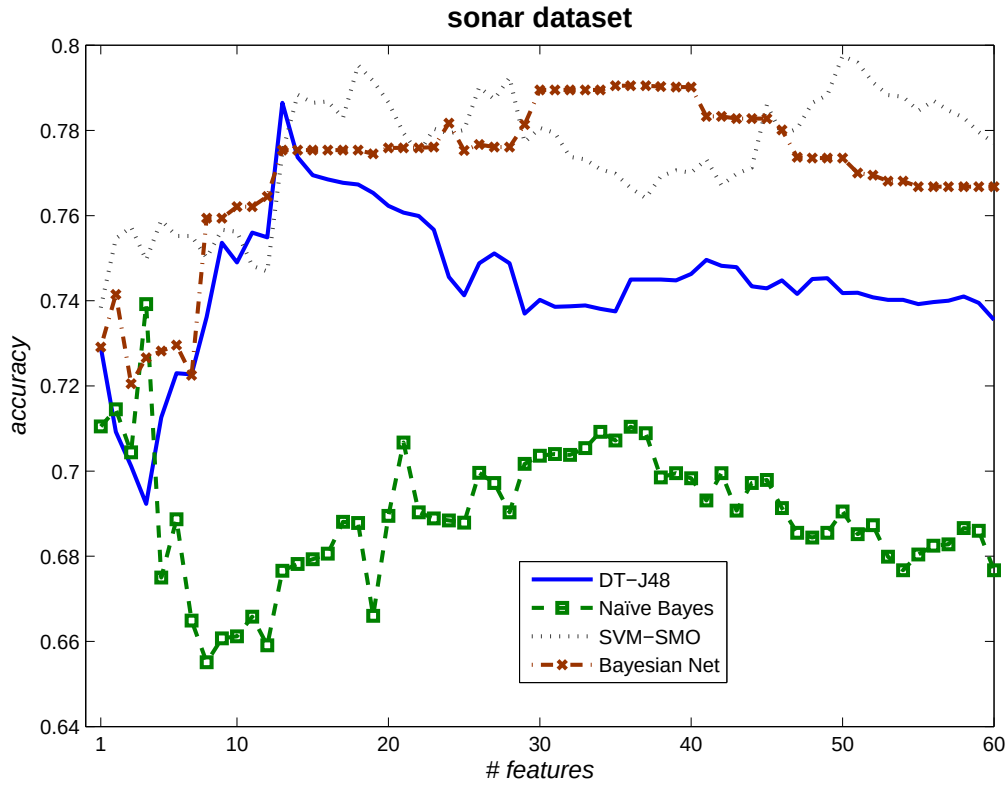


Figure 5.7: Accuracy of different classifiers in the Sonar classification task by using different numbers of ranked features.

performance, but actually causes a considerable deterioration. This is particularly true for all the classifiers in the Ionosphere dataset and all the classifiers (except SVM) in the WBC-Diagnostic dataset.

5.6 Summary and Discussion

In this chapter, GP was used to find the importance of the input features in a classification task and then use this information to rank the features. GP is used to evolve weak classifiers. During the course of evolution, GP implicitly found a group of features that are required to build a good weak classifier. Then features were credited and ranked based on their ap-

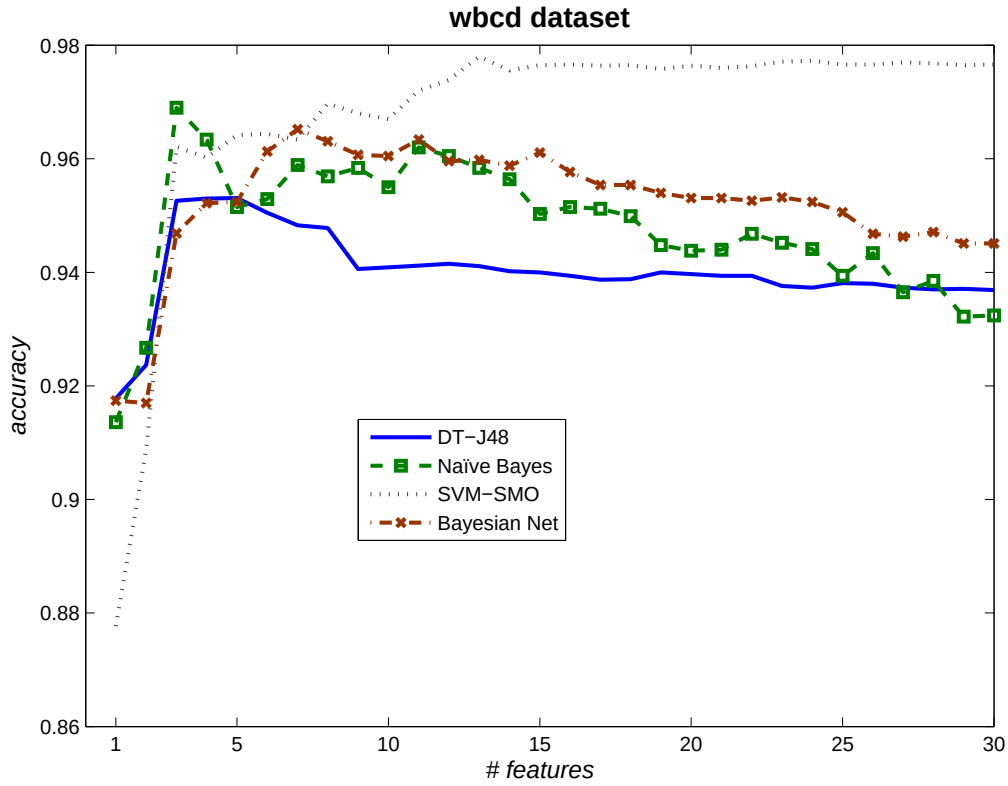


Figure 5.8: Accuracy of different classifiers in the WBC-Diagnostic classification task by using different numbers of ranked features.

pearance in good weak classifiers. Most feature ranking methods measure the importance of features individually. However, although our algorithm ranks single features, it considers the importance of a feature in the context of other required features.

Our results show that the output of the proposed algorithm can reflect the true importance of the input features of a classification problem. We found that, a variety of different classifiers restricted to just a few highest-ranked features work well. In most cases, by using less than 10% of the highly-ranked features, we gained the same classification performance as that gained when all the available features are used. In fact, in most cases, there was a set of highest-ranked features which led to a better classifica-

tion performance than that obtained by using all the features.

Using all the features does not achieve the best result in any of these three datasets. In fact, it is observed that having too many features causes the classification performance to considerably deteriorate in most problems and for most classifiers. This may seem counter-intuitive to our expectation that ideally, by being fed more information, a good learning algorithm should perform monotonically better. However, in practice many well-known machine learning techniques are severely sensitive to curse of dimensionality and noisy information. This suggests that dimensionality reduction is essential for classification tasks.

Commonly, the size of the search space of a feature selection task is 2^m which is the number of possible subsets of the input features. The proposed GP-based feature ranking, however, provides heuristics for having a smaller search space for feature selection. One can think of the cardinality of a set of highest-ranked features as a search space that contains one point for each possible set of highest-ranked features. There are m points (subsets) in the search space: one subset with cardinality 1 (containing the highest-ranked feature), one subset with cardinality 2 (containing the two highest-ranked features) and so on until a subset of cardinality m containing all the input features. By looking at the performance curves of all the datasets and all the classifiers, it is observed that there is at least one point in this search space where the classification performance is better than the performance obtained when all the features are used. The new search space is so small that it can even be searched exhaustively in $O(n)$. Using this smaller search space can be very beneficial in a feature selection algorithm.

All single-feature ranking methods suffer from two serious deficiencies. The first deficiency is evident when a group of features carry some important information about target classes, but their importance is not individually detectable. We did partly address this issue by taking a context-sensitive approach, but the context information was lost during scoring

and single-feature ranking. The next chapter will thoroughly address this issue, by introducing subset feature ranking and selection. The second deficiency is evident when two or more features that are individually ranked as high carry very similar information. This phenomenon is called redundancy. Selecting features in the way it is done in this chapter—that is, by making a set of highest-ranked features—is not effective when the high-rank features are redundant. We address the issue of redundancy in Chapter 7.

Chapter 6

Ranking and Selection of Subsets of Features

6.1 Introduction

Most feature ranking methods fall into the *filter approach* category, and almost all the filter-based ranking methods can only measure the goodness of a single feature. This includes all feature ranking measures from the information theoretic domain, such as information gain (IG), gain ratio, mutual information and the like [see Section 2.2.4, page 20]. Even though in majority of real world classification problems, one feature might not show any sign of being useful in the absence of other features, the majority of existing ranking methods cannot provide any explicit way of measuring the goodness of a group (subset) of features.

Another limitation of the majority of the existing methods is that they can only consider simple types of relationships between a feature and the target class. For example in the logistic regression model [18], the relationship is assumed to be linear; in most of the information theoretic measures, it is assumed that instances can be classified by setting a split point along the feature axis. As a consequence if a feature or a group of features is relevant to the target concepts in a way that cannot be handled by one of the

predefined templates used in these methods, then their importance cannot be measured.

6.1.1 Chapter Goals

The main goal of this chapter is to develop a GP-based system for ranking and selection of subsets of features. In particular, the following criteria should be met:

1. for efficiency reasons and for the sake of generality, the proposed algorithm should not wrap any particular classifier to explore the search space of features. That is, a filter approach is adopted;

2. the system must evaluate the goodness of a subset of features as opposed to single features only;

3. the proposed system must be able to detect those good features that are not normally detected by existing methods;

4. the algorithm must be able to explore the space of subsets of features properly via considering the topological characteristics of the search space;

5. the provided ranking scheme for the subsets of features should give a good insight into the actual importance of the subsets and the classification performance that can be obtained by using them;

6. for feature selection, the system must take a multi-objective approach where the objectives are maximising the relevance of subsets and minimising their sizes.

6.2 GP for Ranking Subsets of Features

In this section, we propose some conceptual elements required to build a GP-based system for ranking and selection of subsets of features.

6.2.1 Overview

We refer to the variable terminals used in a GP program tree as a subset of features. A GP program defines a function over its variable terminals. We evolve GP programs in a way that the function defined by a GP program helps evaluate the goodness of the subset of features used in the program. We define a virtual program-tree structure and a fitness function in a way that the fitness of a GP program shows its relevance to the classification task. Through some case studies, we describe how the proposed system can handle multiple features and how it can find those good features that are usually missed by other relevance measures.

We use GP to explore different subsets of features. Since the fitness of a program shows the relevance of its features to the target attribute, during the course of evolution, GP goes towards finding more promising subsets of features. We propose a mechanism to improve the exploration performance of GP. By conducting several GP runs, the relevance of a number of subsets of features is revealed. Among these subsets, are a group of high-performing subsets. We will then describe how this information is used to form a Pareto-front on which feature selection can be performed.

6.2.2 Program Trees: A Virtual Structure

We extend the concept of relevance measure by proposing a virtual structure for GP program trees. Figure 6.1 shows such a structure for a GP program; it measures the relevance of a subset of features to a classification task. At the top (root) of the tree, there is a relevance measure function denoted by RM. This function measures the relevance of its right subtree to

the class label variable denoted by C. The node RM can be a very simple function in terms of the types of relationships it can detect. However, by providing a good subprogram as its right subtree, we can discover more complex relationships between the features used in the subtree and the class variable. For example, RM could be a linear correlation function for one single feature, but with a sophisticated subtree underneath, we would be able to detect nonlinear relationships between a subset of features and the class variable. We use the power of GP to evolve a rich subtree which leads to a high relevance at the root node. We then regard the contributing variable terminals in that subprogram as a subset of features and the output of the program as the goodness of that subset. This structure is virtual in the sense that the top node of the tree does not take part in any genetic operations and so, in practice, it can be implemented as part of the GP tree evaluation process rather than the GP tree representation.

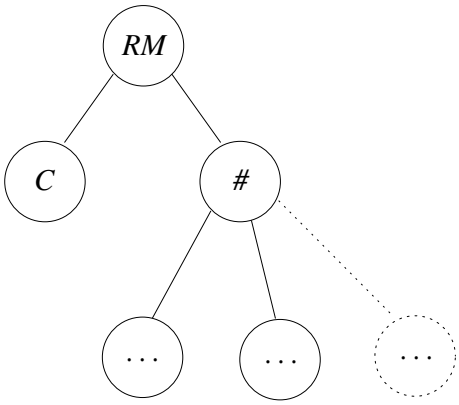


Figure 6.1: A virtual structure for a GP program for measuring the usefulness of a subset of features.

6.2.3 Relevance Measure

In the filter approach to feature selection for classification tasks, as a widely-used hypothesis, a good feature is considered to be highly related to the class variable [47]. That is, knowing the value of a related feature should

change the probability distribution of the class variable. We are looking for a relevance measure (function) that can be used in the node RM. Since the class label is a nominal (categorical) variable, the function used to measure this relevance should be capable of handling this type of data. Information theoretic measures like information gain (IG) and gain ratio can be used only if the feature being measured is nominal itself or has already been discretised. These methods are also limited to measuring the correlation between the class labels and a single feature rather than a group (subset) of features.

To measure the correlation between a continuous feature and a binary class variable, one could use the logistic regression (LR) model

$$\log\left(\frac{\pi(x)}{1-\pi(x)}\right) = \alpha + \beta x \tag{6.1}$$

where $\pi(x)$ is the probability of an instance belonging to a particular class given the value of feature x, and the right hand side of the equation is a linear approximation of the logit function. The magnitude of the coefficient β is used as an indicator of linear correlation, where a value of zero shows no linear correlation between the continuous feature x and the class label [2]. The parameter β and constant coefficient α can be estimated by maximizing the likelihood function through the Newton-Raphson method, but it is too expensive a procedure to be considered as a candidate function for the node RM.

Here, we define a binary relevance function (BR) that measures the linear relationship between a nominal and a numeric variable but is computationally cheaper. We define BR to be

$$BR(x,c) = \left(\frac{Cov(x,\omega(c))}{\sigma(x)\sigma(\omega(c))}\right)^2 \tag{6.2}$$

which is actually the square of Pearson's correlation between a numeric random variable x and a function ω, with numerical range, of a nominal random variable c. $Cov(\cdot,\cdot)$ and $\sigma(\cdot)$ denote, the covariance function and

the standard deviation, respectively. The squared form is used because we only consider the magnitude of relevance and not the direction (sign). For a binary classification task with $c \in \{C_A, C_B\}$, we define $\omega(c)$ to be

$$\omega(c) = \begin{cases} +\sqrt{\frac{n_B}{n_A}}, & c = C_A \\ -\sqrt{\frac{n_A}{n_B}}, & c = C_B \end{cases} \tag{6.3}$$

where n_A and n_B are the numbers of instances belonging to class C_A and class C_B, respectively, and $n_A + n_B = n$, which is the total number of samples in the training set. The function $\omega(c)$ is a standardized variable with the following expected value and variance:

$$E(\omega(c)) = p(C_A)\sqrt{\frac{n_B}{n_A}} - p(C_B)\sqrt{\frac{n_A}{n_B}} = \frac{1}{n}(n_A\sqrt{\frac{n_B}{n_A}} - n_B\sqrt{\frac{n_A}{n_B}}) = 0 \tag{6.4}$$

where $p(.)$ denotes the class distribution. Consequently,

$$\sigma^2(\omega(c)) = E(\omega^2(c)) = \frac{n_A}{n}\frac{n_B}{n_A} + \frac{n_B}{n}\frac{n_A}{n_B} = 1 \tag{6.5}$$

Therefore, the empirical BR can, given that the total number of instances n is large enough, be simplified to

$$BR(x, c) = \frac{(\sum_{i=1}^{n}(x_i - \bar{x})\omega(c_i))^2}{n\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{6.6}$$

where x_i is the i-th observation (the output of the subprogram in Figure 6.1 for the i-th instance), c_i is the class label of the i-th observation, and $\bar{x}$ is the sample mean. The complexity of computing BR is $O(n)$, which makes it a good candidate for the node RM.

To show why BR is a good alternative to LR for feature ranking, we generate a set of artificial data. We consider one feature, x, and a binary class variable $c \in \{C_A, C_B\}$. Instances of classes C_A and C_B are distributed based on the same distribution but different parameters. We chose normal distributions with means μ_A and μ_B each having 21 values, namely, $\{0, \pm1, \ldots, \pm10\}$, and variances $\sigma_A^2 = \sigma_B^2 = 1$. In total we have 441 (21×21)

artificial datasets, each containing one feature with 200 instances (100 for each class).

Figure 6.2 shows the BR function and the magnitude of the parameter β from the logistic regression with respect to μ_A and μ_B. In both graphs, there is a valley-like area along the diagonal where the means of the two classes are close to each other. In this area, the instances of the two classes are almost mixed together and the knowledge of the value of x would not be helpful in discriminating the instances.

In the figure, as we go towards the areas off the diagonal, where the distance between the means becomes larger, the magnitude starts increasing in both figures. However, there is a difference between these two functions. When the means of the classes get far away from each other, $|\beta|$ in LR starts decreasing, indicating a lower relevance. However, from a classification point of view, as the margin between the instances of two the classes provided by a feature increases, the feature is considered to be better. In contrast to LR, the BR function returns larger values as the distance between the two classes increases. So BR is a better measure than LR for feature ranking.

6.2.4 Fitness Function

Algorithm 9 shows how the fitness of a GP individual is calculated using the BR function. The class label vector $\mathbf{c}[i]$ can take only two values in $\{A, B\}$, each representing one of the classes in the task. The values of n_A and n_B (the number of instances in the two classes) are calculated once at the beginning of the GP runs. With only one *for* loop, the algorithm can calculate the fitness in a single pass. The value obtained at line 12 is effectively equal to equation (6.6).

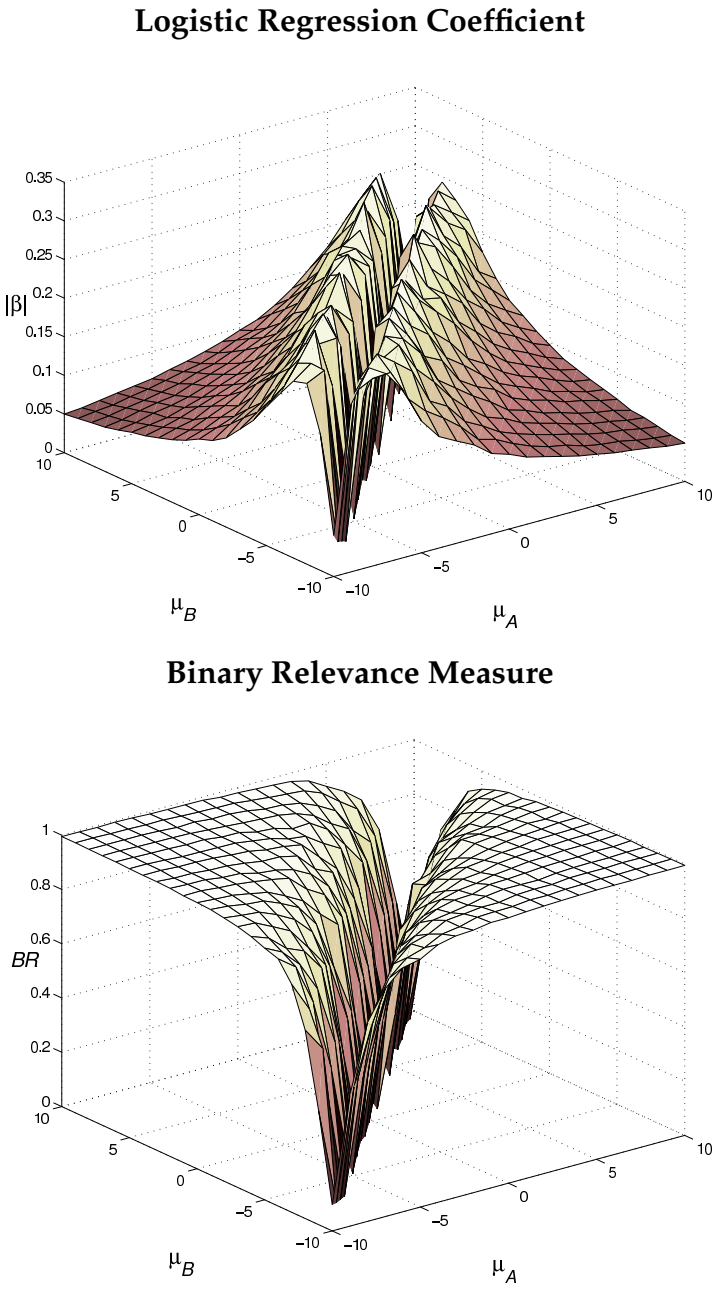


Figure 6.2: Magnitude of parameter β in LR (top) and BR function (bottom) with respect to the mean of classes A and B.

Algorithm 9: $BR\text{-}Fitness(\mathbf{D}, \phi)$

```
/* Given the dataset (training or validation) of a
   binary classification task and a GP program,
   the relevance between the subset of features
   used in the program and the target class is
   calculated.                                    */
```

Input: $\mathbf{D}$, a dataset of the form $\mathbf{D} = (\mathbf{X}, \mathbf{c})$ where $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$ is a set of vectors of length n containing samples from the m original features in the problem and $\mathbf{c}$ is a vector of class labels for the corresponding observations in $\mathbf{X}$

Input: ϕ, a GP program which acts as a function $\mathbb{R}^m \mapsto \mathbb{R}$

Output: $fitness$, a real value in $[0, 1]$ showing the relevance between the features used in ϕ and the target class

$sum_y, sum_{y^2}, sum_\omega, sum_{y\omega} \leftarrow 0$; // initialising the sums

for $i \leftarrow 1$ **to** n **do**

  $y \leftarrow \phi(\mathbf{x_1}[i], \mathbf{x_2}[i], \ldots, \mathbf{x_m}[i])$; // transformation

  $sum_y \leftarrow sum_y + y$; // updating the sum

  $sum_{y^2} \leftarrow sum_{y^2} + y^2$; // updating the sum of squares

  if $\mathbf{c}[i] = A$ **then**

    $\omega \leftarrow +\sqrt{\frac{n_B}{n_A}}$;

  else

    $\omega \leftarrow -\sqrt{\frac{n_A}{n_B}}$;

  $sum_\omega \leftarrow sum_\omega + \omega$; // updating the sum

  $sum_{y\omega} \leftarrow sum_{y\omega} + y\omega$; // updating the sum of products

$fitness \leftarrow \frac{sum_{y\omega} - \frac{sum_y \; sum_\omega}{n}}{n \; sum_{y^2} - sum_y^2}$;

return $fitness$;

6.2.5 Case Studies

We set up a number of experiments on artificial data to investigate what type of relevance the proposed GP system is able to capture, but the other methods are not. In particular, we study a bimodal class distribution scenario in which a system seeks to capture non-linear changes in the class probability, and a binary classification problem with two correlated features in which the system is to measure the goodness of a subset of features rather than individuals. Single GP runs are conducted in each case and the results are compared to those of other methods. Values regarding this comparison are presented in Table 6.1, and the details of the two cases, bimodal class distribution and correlated features, follow.

Table 6.1: Three relevance measures on two case studies

Ranking Method	**Bimodal Distribution**	**Correlated Features**
Logistic Regression		
$\|\beta\|$ coefficient	x: 0.03	x: 0.15, y: 0.13
Information Gain (IG)		
Entropy	0.61	0.69
Split point	x: 2.8	x: 7.4, y: 4.6
Gain	x: 0.23	x: 0.29, y: 0.00
Gain rate	x: 38%	x: 42%, y: 0.0%
GP Relevance Measure		
GP-based relevance	x: 82%	$\{x, y\}$: 87%

6.2.5.1 Bimodal Class Distribution

Figure 6.3 shows a binary classification problem with a single feature x, where the distribution of one of the classes, A, is bimodal.

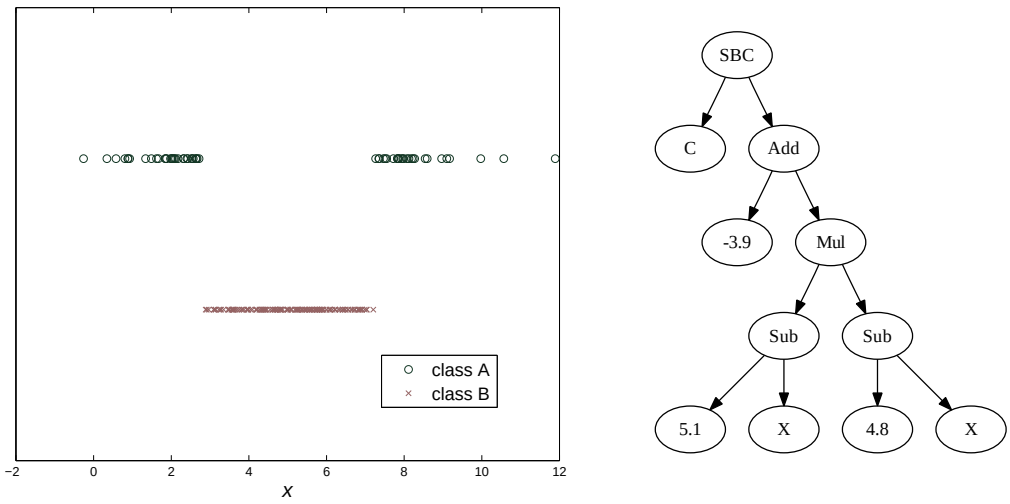


Figure 6.3: A bimodal class distribution along feature x in a binary classification problem (Left). The evolved GP program tree for measuring the goodness of x (Right).

This feature is observably good because by setting up an interval around instances of class B, the classification problem can be solved. However, since the class probability does not change linearly with respect to x, the LR method does not consider x to be a good feature, returning a β coefficient close to zero (0.03, Table 6.1). The IG method tries to find the best split point (2.8, Table 6.1) using which, instances from different classes can be separated. However, as all the instances cannot be separated around one split point, IG does not report this feature to be a very good one. In contrast, GP evolves a program tree like the one in Figure 6.3 (right) which transforms the relationship to a linear form that can be detected by the BR function. These results show how this feature is dismissed as irrelevant by methods like LR and IG, but it is successfully detected by the proposed GP-based measure. Notice that the gain rate and GP-based relevance are calculated differently, but they can be regarded as an indicator of how good a feature is for the problem.

6.2.5.2 Correlated Features

Figure 6.4 shows another binary classification problem with two features, x and y, that are correlated.

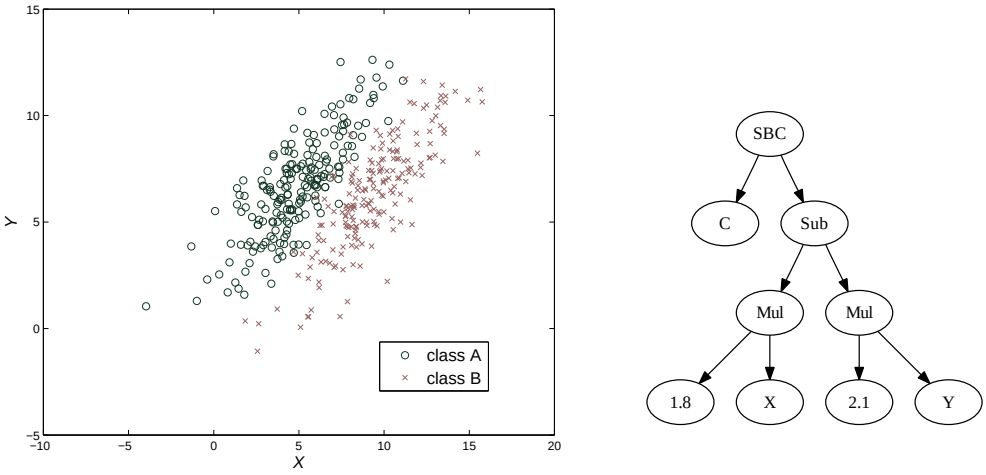


Figure 6.4: A binary classification task presented with respect to two of its features, x and y, which are correlated (Left). The evolved GP program tree for measuring the goodness of the subset $\{x, y\}$ (Right).

These two features can be very useful for this classification task as a (straight) line passing through the boundary of the two classes can identify the class of instances. However, to those relevance measures that consider each feature individually, neither feature is necessarily good. LR returns a $|\beta|$ of less than 0.2 for each of these features and the gain rate of IG for x is pretty low and for y is zero (Table 6.1, column 3). On the other hand, GP maximises the BR function by finding an appropriate program tree. This program tree is shown in Figure 6.4 (right). The subprogram, as the right child of the root node, actually constructs (by combining x and y) a meta-feature along which the instances are easily separable. The relevance of $\{x, y\}$ is measured to be 87% by GP.

6.3 Exploring the Search Space of Subsets of Features

6.3.1 Search Space Topology

The size of the search space of a feature selection problem grows exponentially with respect to the number of features in any given classification task. With m features in a classification task, the search space includes 2^m points, one for each candidate subset (solution). A common representation of these solutions is to use a string of zeros and ones with length m, where a zero or one at the i-th position specifies the absence or presence of the i-th feature in the solution. This representation is common in genetic algorithms for feature selection [137, 117]. This representation can be improved by mapping the string to a **lattice** of subsets of features in which adjacent nodes are obtained by inclusion or exclusion of a feature from the existing node (subset). One such lattice is depicted in Figure 6.5. The lattice can give more topological information to a feature manipulation algorithm [155].

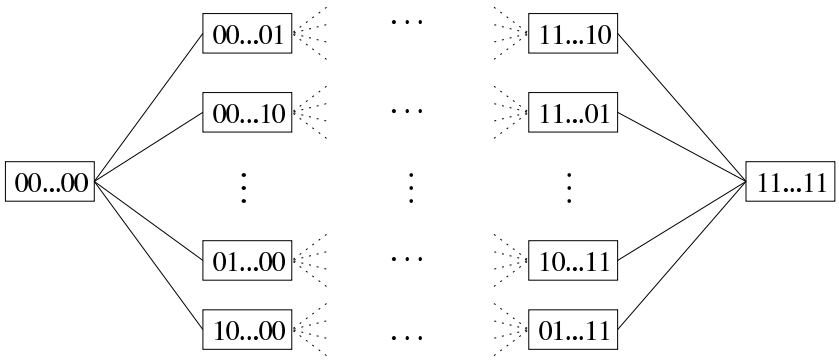


Figure 6.5: Search space of the feature subsets in the form of a lattice where each node represents a feature subset with a string of zeros and ones showing the absence and presence of the corresponding original features.

6.3.2 Difficulties in Exploring the Search Space

As a common practice in using GP for feature ranking/selection, the presence or absence of a variable terminal in a program tree is used as an indication of the selection of the corresponding feature [87, 105]. This practice, although effective, in certain situations cannot explore some points on the search lattice. Theoretically, a binary GP tree (i.e., a GP tree including only binary primitive functions) of depth d is capable of hosting up to 2^{d-1} features. That is, to explore subsets of features of size up to $max\text{-}cardinality$, we need GP trees of a minimum depth of $\lceil \log_2 max\text{-}cardinality + 1 \rceil$. In practice, however, the GP behaviour can be quite different.

Some preliminary experiments are conducted to see how GP explores the search space of feature subsets. We use the proposed fitness function and three datasets, namely Ionosphere, Sonar and WBC-Diagnostic. The details of these datasets which are later used in our main experiments will be described in Section 6.6. We use a standard GP system with maximum tree depth of 6, a population of size 2048 and a maximum number of generations of 50.

Figure 6.6 gives some statistics on the results. The horizontal axes show the cardinality of a feature subset and the vertical axes show the frequency (number of occurrences) of such subsets in logarithmic scale. The solid line with '$\circ$' marks shows the size of the lattice (in terms of number of points) for the given subset cardinalities. This curve is obtained by calculating the binomial coefficient $\binom{m}{s}$ where s is the cardinality of the subset. The other two curves show the number of subsets of features explored by GP: one for the number of all unique feature subsets that appeared in the structure of one or more program trees and the other is a subset of the GP-explored subsets where the relevance (fitness) of the corresponding program is higher than a **minimum acceptable relevance**, τ. Feature subsets appearing in program trees with a fitness of less than τ could not be considered as truly explored subsets as the low fitness might be due to poor program structure rather than the quality of the features. We set

$\tau = 0.3$, however, any value in $[0.3, 0.5]$ seems reasonable.

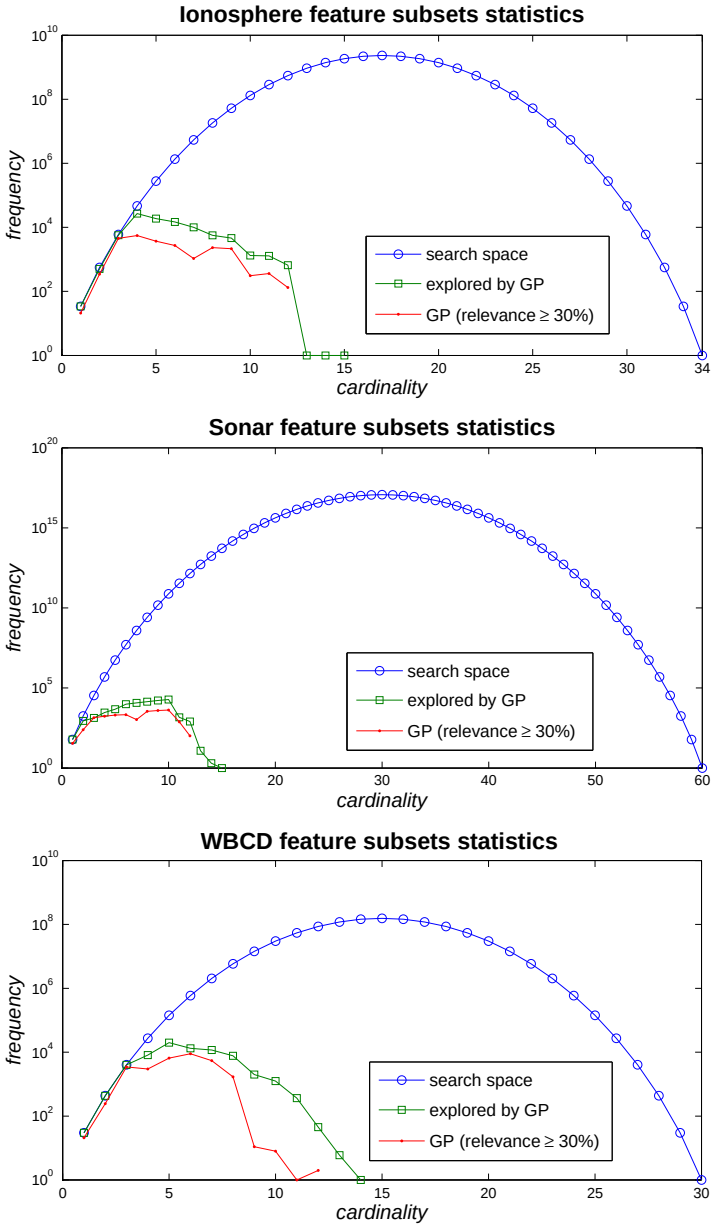


Figure 6.6: Frequency of subsets of features in the three datasets with respect to the cardinality of subsets. The three curves are for: all possible combinations in the search space, subsets explored by standard GP, and subsets explored by standard GP having a relevance greater than 0.3.

The most noticeable observation we can make based on these figures is that although a complete binary tree of depth 6 is theoretically capable of using 32 features, in practice, in none of these problems the cardinality of the explored feature subsets is greater than 15. This could be due to a variety of reasons:

- the fitness of GP trees with large numbers of features is so low that they are not selected to enter the next generations or contribute in making larger trees;
- high fitness programs happen to have non-full binary tree structures which consequently reduce their capacity for hosting larger numbers of features;
- the complexity of the problem requires a lot of other operations (e.g. functions, constant values, etc), occupying a lot of nodes and leaving very little room for variable terminals (features).

The observations in this experiment may suggest that the GP search is generally biased towards exploring subsets with relatively low cardinality. This is potentially a good property for a feature selection algorithm where smaller subsets are more desirable (solution points which are as close as possible to the left side of the search lattice depicted in Figure 6.5). However, one should make sure that the cardinality of the optimum subset is not beyond the exploration power of GP (the maximum number of features that can be practically reached in each GP program). On the other hand, there is no explicit way of finding the optimal cardinality for a subset of features without conducting a search.

6.3.3 Improving Search Space Exploration

One solution to the problem mentioned previously would be to lift the depth control or set it to a large number. This remedy, however, may cause

bloating in GP programs [145]. So we start with a basic idea which is similar to the concept of dynamic bloat control by changing the depth limit during runtime [132]. We make some alterations to the standard GP algorithm in a way that depth limit is set to a low number at the start of a GP run and it can, when genetic operators are applied, increase only if the resulting program tree has a higher relevance (fitness) than the best relevance so far.

Although the optimistic intention of having deeper program trees with higher fitness is to explore towards the right side of the search lattice and consider larger subsets with higher relevance, it comes with two potential side effects:

1. a deeper program tree with higher fitness is not necessarily incorporating a larger set of features with higher relevance; it might be an overfitted model using the same features as explored previously. Figure 6.7 illustrates a situation in which a 4th degree GP program transforms a bimodally distributed dataset to a unimodal one where the true relevance can be calculated using equation 6.6. The demonstrated 4th degree model is general enough to exclude the noisy observations. However, a higher degree model using the same features can produce slightly higher relevance by overfitting the data and including the noisy observations as well;

2. in a similar scenario to that of Figure 6.7, a deeper GP program might overfit the data by incorporating a redundant feature x', which is highly correlated to an existing feature x. In this case, although the program is using a larger subset of features, the resulting relevance is not completely true due to the overfitted model.

To address these issues, in our modified version of the GP algorithm, we adopt the notion of using validation data to avoid overfitting [130]. The training data is virtually partitioned into a training set and a validation set. Two fitness values are kept for each individual in the population:

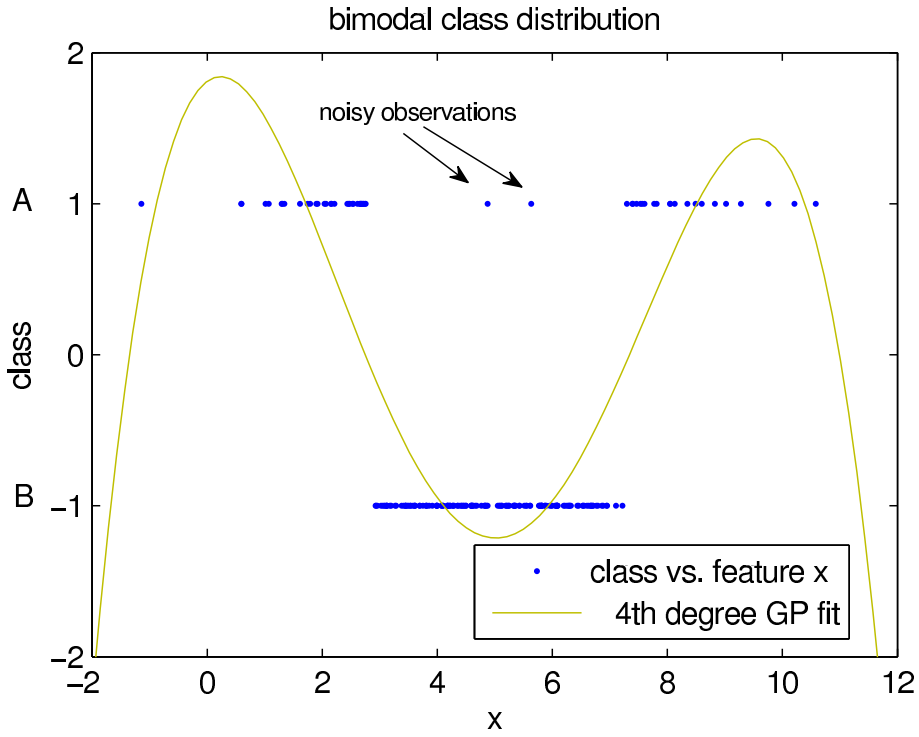


Figure 6.7: An artificial binary classification problem with classes A and B (visualised at 1 and -1) along a numerical feature x. The feature is observably good, as one can find certain boundaries along it to separate class A from class B. However, since the class distribution is bimodal, the feature does not seem important to many feature selection methods like Pearson's correlation (showing only 5% relevance) and Information Gain (showing only 43% relevance). A 4th degree GP program $0.01x^4 + 0.12x^3 - 0.6x^2 + 0.24x + 1.8$, however, maps the data to a new space where the fitness function shows 85% relevance. The GP model is simple enough not to overfit the problem (considering the noisy observations).

$fitness_t$, which is obtained by applying the program to the training samples in order to transform them into a new space and then calculate the fitness (relevance measure) by using the BR function, and $fitness_v$, which is obtained by applying the same procedure to the validation data. The

training fitness of the fittest individual, $fitness_t^\star$, and the corresponding fitness using the validation data, $fitness_v^\star$, are kept globally during the GP run. Selection operators use only $fitness_t$ to select individuals to apply other genetic operators to. If a new individual, resulting from applying GP operators, with fitness values $fitness_t'$ and $fitness_v'$, is deeper than the current depth limit and

$$\begin{aligned} fitness_t' > & fitness_t^\star \\ \text{and } \ fitness_v' \geq & fitness_v^\star \end{aligned} \tag{6.7}$$

then the individual can enter the next generation and the depth limits are updated. Otherwise, the new program will be discarded and there will be no changes in the depth limits.

6.4 Creating a Pareto Front

6.4.1 Feature Selection Objectives

In feature selection we are interested in finding a subset of a minimal number of features that satisfies a learning objective (e.g. improving the classification performance). By concentrating on the smallest possible solutions we implicitly address the need to eliminate irrelevant and redundant features from the solution. Therefore, there are two aspects in determining the *best solution*: relevance and cardinality. During the search process, as long as a new candidate solution (subset) is smaller than the previously discovered subsets and results in higher relevance, making choices is easy; that is, the new candidate subset can replace the previous best solution. However, the situation is not trivial when the feature selection algorithm, for example, finds a considerably smaller feature subset by compromising only a little relevance. This is because in some scenarios having a simpler efficient model is better than having a very complex model, which is just slightly fitter than the simple one.

In some algorithms, there are *a priori* assumptions based on which a solution (candidate subset) should be chosen when the above-mentioned situation arises. In [87, 83, 42, 22] the winner is simply the solution that produces higher relevance. In more formal terms, in single objective feature selection algorithms, given two solutions (subsets of features) $\mathcal{S}$ and $\mathcal{S}'$, if $relevance(\mathcal{S}') > relevance(\mathcal{S})$, then $\mathcal{S}'$ dominates $\mathcal{S}$. In [105], a more sophisticated system has been proposed, where a composite fitness function determines, through some parameters, the importance of relevance and smallness. That is, $\mathcal{S}'$ dominates $\mathcal{S}$ if and only if

$$composite(relevance(\mathcal{S}'), |\mathcal{S}'|) > composite(relevance(\mathcal{S}), |\mathcal{S}|)$$

where $composite$ is an objective function that returns a single scalar value as the goodness/fitness of a subset with respect to its size and relevance. Designing such an objective function requires a set of assumptions about the relative importance of the objectives through some parameters. These parameters need to be set before starting the search process on the basis of the designer's (user's) experience. However, since a feature selection search is computationally expensive, finding the optimal value of these parameters by trial and error could be an issue.

6.4.2 Pareto Archive

Instead of having one single best solution, a group of solutions that are the best at least in one of the objectives is kept. From this standpoint, given two solutions $\mathcal{S}$ and $\mathcal{S}'$, $\mathcal{S}'$ dominates $\mathcal{S}$ if and only if $relevance(\mathcal{S}') > relevance(\mathcal{S})$ and $|\mathcal{S}'| \leq |\mathcal{S}|$. However, if $relevance(\mathcal{S}') > relevance(\mathcal{S})$ but $|\mathcal{S}'| > |\mathcal{S}|$, neither solution can dominate the other. The collection of all non-dominating solutions constitutes a surface called the **Pareto front**. The Pareto front consists of those solutions for which there exists no better solution in both criteria [140]. Having a Pareto front in feature selection, there is no need for any *a priori* assumptions about the importance of objectives [138]. The pareto front can also serve as a trade-off matrix, showing

what relevance can be gained in return for increased complexity due to using larger subsets of features.

The measurements in the second objective of our algorithm, cardinality, are discrete. That is, the cardinality of a subset of features is a discrete variable that can take values from $\{1, 2, ..., m\}$. This means that the whole Pareto front can be stored in a vector of size m. In other words, having a Pareto front archive for all the individuals of a GP population is, in terms of memory usage, $O(m)$, which is quite efficient. The Pareto front vector, $\mathbf{p}$, is formally defined as

$$\mathbf{p} = (\mathcal{S}_1^\star, \mathcal{S}_2^\star, \ldots, \mathcal{S}_m^\star : \mathcal{S}_i^\star \subseteq \mathbb{F} \text{ and } |\mathcal{S}_i^\star| = i, \forall i \in \{1, 2, \ldots, m\}) \tag{6.8}$$

where

$$\forall \mathcal{S} \subseteq \mathbb{F}, |\mathcal{S}'| = i \Rightarrow relevance(\mathcal{S}') \leq relevance(\mathcal{S}_i^\star) \ .$$

During a GP run, after each fitness calculation, the Pareto front must be updated to meet the above-mentioned criteria.

6.5 The Main System

Figure 6.8 depicts the overall architecture of the system. The dataset of a binary classification task including a training set and a test set is given as the input. A number of GP runs are conducted; each of them maximise the relevance function over the data. The BR function in equation (6.6) is used as the fitness (relevance) function. We regard the relevance obtained from each individual as the quality of the features being used in that GP program. Over the course of evolution the search moves towards finding more promising subsets of features. A small proportion of the training data is put aside to be used as validation data. There are two fitness values for each GP individual: one calculated over the training data and the other calculated over the validation data. The latter is used to update the depth limits according to equation (6.7).

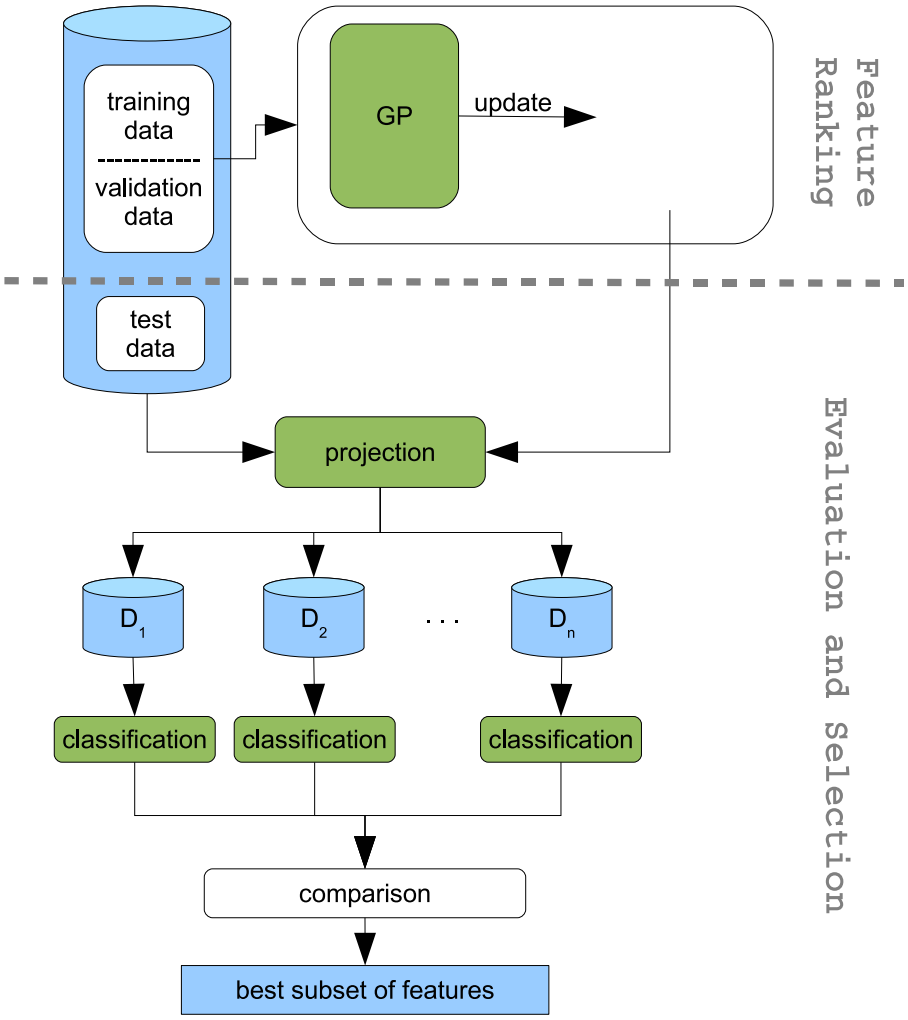


Figure 6.8: The diagram of the proposed GP-based subset ranking/selection system.

Whenever the algorithm visits a new subset of features, the subset, the corresponding GP program and the corresponding fitness (relevance) is stored in a hash table called the **ranking table**. At the end of the GP runs the ranking table is used to analyse the way GP explores the space of the subsets of features. We also keep a Pareto front that contains a subset of

solutions in the ranking table that meet the criteria in equation (6.8). The ranking table and the Pareto front are persistent through the GP runs and the elements are accumulated gradually. After each fitness calculation, the cardinality of the subset of features being used in the program and its relevance are compared to the ranking table and the Pareto front, and the two are updated if necessary.

Once the GP runs are finished, the ranking table and the corresponding Pareto front of size up to m are available as outputs. The dataset is then projected through the subsets in the Pareto front, generating up to m new partial datasets each including only certain selected features of the original dataset D. These new datasets, denoted by D_1, D_2, ..., D_m, are fed to a classification algorithm. The users can then compare the results and select their most desirable subset of features. The objective of this comparison could be maximising the classification performance or minimising the model complexity while retaining an acceptable performance.

6.6 Empirical Results

6.6.1 Design of Experiments

Generally, there is no explicit way to evaluate a subset ranking/selection system for two reasons: (A) there is no global specification for the best subset of features. Even if the objective is to maximise the classification performance, the best subset for one classification algorithm is not necessarily the best for others [113]; (B) as the search space grows exponentially, and there are vast numbers of different feature combinations to examine; therefore for a large n, one cannot make sure that a particular solution is a global optimum. Therefore, the proposed system is implicitly evaluated by measuring different properties of the system via answering the following questions:

1. to what extent the ranking provided by the proposed system reflects

the actual importance (usefulness) of the subsets of features;

2. how the provided ranking can be used to find the best subset of features;

3. how well the proposed system can actually explore the search space and create a Pareto front; and

4. how the classification performance and complexity obtained by using selected features compare to the initial performance (without selection).

6.6.1.1 Datasets

Three datasets with a relatively large number of features from the UCI machine learning repository [6] are used in the experiments. Table 6.2 summarises the main characteristics of these datasets.

Table 6.2: Specifications of datasets used in experiments

Problem	# Features	# Instances	# Classes
JH Ionosphere	34	351	2
Sonar	60	208	2
WBC-Diagnostic (WBCD)	30	569	2

6.6.1.2 GP Settings and Implementation Details

We use the standard tree-based GP model [71]. In this model, each program produces a single floating-point number at its root as the result of its evaluation (output). Table 6.3 shows various settings of the GP system we developed for the experiments. There is one variable terminal for each feature in the problem. A number of randomly generated constants are

also used as terminals. The four standard arithmetic operators were used to form the function set. The division operator is *protected*—that is, it returns zero for division by zero. All the members of the function set are binary—they take two parameters.

The ramped half-and-half method [71] is used for generating programs in the initial population and for the mutation operator. The initial maximum program tree depth is set to 5, but it can increase using the proposed mechanism in Section 6.3. The probability of the crossover and mutation operators are adapted automatically at runtime [21]. An elitist approach has been taken to ensure that the performance of the fittest individual in the population never deteriorates. The evolution is terminated, at the latest, after the 50th generation or when a solution of fitness (relevance) 1.0 is found. The platform is implemented in Java and we use grid computing to have parallel GP runs.

Table 6.3: GP Settings

Function Set:	$+, -, \times, \div$ (protected division)
Variable Terminals:	The original features ($\{x_1, x_2, \ldots, x_m\}$)
Constant Terminals:	Randomly Generated
Population Size:	2048
Number of Generations:	50
Initialisation:	Ramped half and half
Mutation:	Subtree creation
Selection:	Tournament (size=5)
Initial Tree Depth:	5
Maximum Tree Depth:	Based on the proposed mechanism in Section 6.3
Mutation Probability:	Adaptive [21]
Cross-over Probability:	Adaptive [21]
Elitism:	Yes

6.6.1.3 Evaluation

Table 6.4 shows the settings involved in the evaluation process. Two types of classifiers are used in our experiments, namely the J48 implementation of the C4.5 decision tree [127, 152], and the SMO version of the SVM classifier [61]. Since none of the datasets that are used in our experiments come with a specific test set, we adopt a 10-fold cross-validation approach. The dataset is shuffled and stratified to 10 folds. Stratified folds have the same proportion of instances form different classes. Each time one of the folds is taken as the test set and the remaining as the training set. We use 10% of the instances of the training data as the validation set. We conduct 50 GP jobs which combined with 10-fold cross-validation means there will be 500 GP runs. This number of runs allows us to explore the feature space properly and accumulates a rich Pareto front archive. To compensate for the effect of defective GP individuals which can misrepresent a subset of features, we only consider GP programs (subsets) whose relevance is higher than 0.3. We use the Weka [152] library for the classification and evaluation processes.

Table 6.4: Evaluation Settings

Validation:	10-fold cross-validation with stratified folds
Classifiers:	Decision Tree (J48 version of C4.5), SVM (SMO)
GP jobs:	50
GP runs:	$50 \times 10 = 500$
τ:	0.3 (Minimum Acceptable Relevance)

6.6.2 Results

6.6.2.1 Subset Ranking

In Figure 6.9, there is a plot for each of the three datasets that shows the relevance values calculated by GP versus classification performance (ac-

curacy on test fold) for all the subsets of features explored during 500 GP runs. An explored subset is represented by two points, one dark for classification performance using SVM, and one light for classification performance using the J48 decision tree. The dark and light points are along the same vertical line, showing the relevance of the subset. There are more than 20,000 feature subsets processed for each dataset.

The plots indicate that there is roughly a linear relationship between the GP-calculated relevance and the classification performance, where the higher the relevance value, the better the classification performance. This linear relationship is particularly obvious in the Ionosphere and Sonar datasets. In the breast cancer dataset, SVM exhibits a strong linear relationship while J48 exhibits a weaker one which is due to the fact that the decision tree classifier does not generally perform as well as SVM on this dataset [113].

Table 6.5 illustrates quantitative measurements of the linear relationship between the GP-calculated relevance and the classification performances of SVM and J48. The second column, $\#subsets$ represents the total number of explored subsets of features by the end of the GP runs. The next two columns show the coefficient of linear correlation between the relevance and the classification performance using SVM and J48 respectively. Almost all the cases show a strong correlation between the GP-calculated relevance and the classification performance. We conduct a test to determine the statistical significance of our results. The test statistic for testing the significance of the correlation coefficient ρ is $T = \frac{\rho\sqrt{s-2}}{\sqrt{1-\rho^2}}$, where T has a t-distribution with $s-2$ degrees of freedom [90]. For the given values of s and ρ in Table 6.5, p-values corresponding to the above test statistic are all smaller than 0.01, which implies that all the estimated correlation coefficient values in the table are statistically significant at a 99% confidence level.

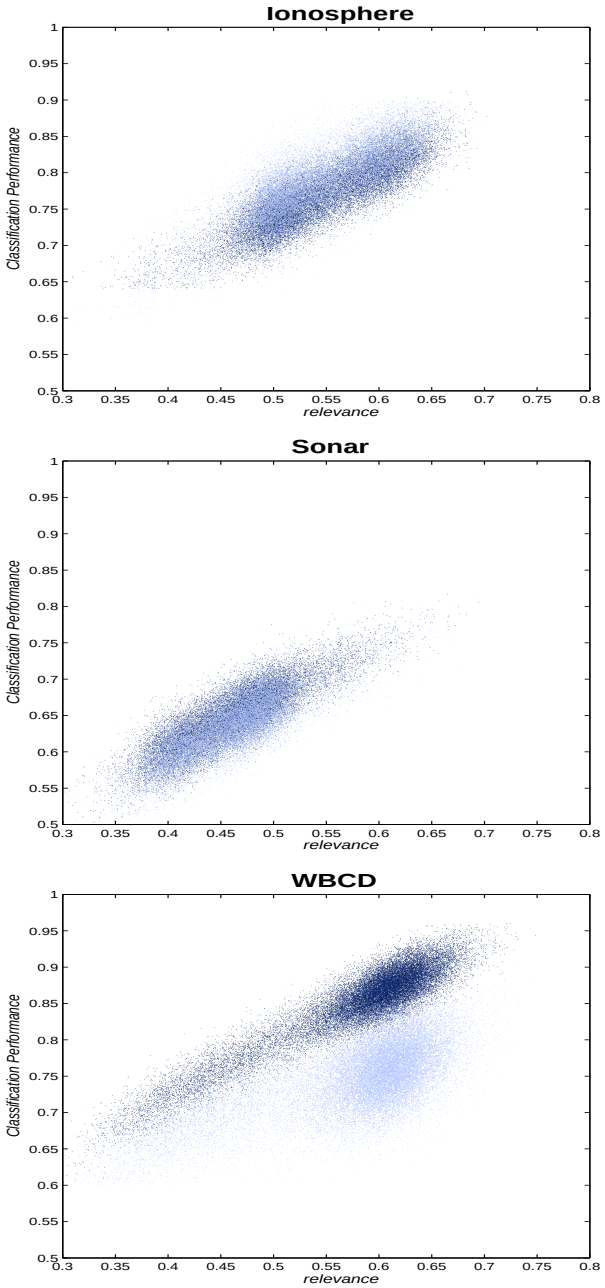


Figure 6.9: Relevance of subsets of features vs. classification performance obtained by using the subsets and SVM (the dark cloud) and J48 decision tree (light cloud on top of the dark one) in the three datasets.

Table 6.5: Correlation between GP-calculated relevance and classification performance

Dataset	$\#subsets$	$\rho_{\text{relevance,SVM}}$	$\rho_{\text{relevance,J48}}$
JH Ionosphere	22592	0.86	0.78
Sonar	20972	0.84	0.81
WBC-Diagnostic	29266	0.93	0.64

6.6.2.2 The Relation between the Highest-Rank and the Best Subset of Features

Among the subsets of features explored by GP, we call the one that leads to the best classification performance the **best subset of features**[1]. The GP-calculated relevance for subsets of features imposes a ranking over the subset of features. An ideal feature ranking mechanism would be the one that would rank such a subset as the highest (rank 1). However, in practice, since the best performing subset of features might be different from classifier to classifier, it is hard to provide such a ranking mechanism. We want to know how the GP-provided ranking can actually help finding the best subset of features.

Suppose that the best subset of features is ranked at the r_0-th position rather than the first (the highest). We need to know, given a ranking for the subsets of features, how many high-rank subsets should be evaluated to find the best subset of features. Thus, we define the **search effort** to be $search\text{-}effort = \frac{r_0-1}{s}$, where s is the total number of ranked subsets of features. The search effort is a number between 0 and 1 which tells us what proportion of the high-rank features should be searched (i.e. evaluated by measuring the classification performance) to find the best subset of features. Since we are interested in having the minimum search effort, we

[1]Like solutions obtained by other evolutionary algorithms, the best subset of features might be a sub-optimal and not necessarily the best subset of features globally.

Table 6.6: Selection Gain

Dataset	Selection Gain	
	SVM	Decision Tree
JH Ionosphere	99%	82%
Sonar	99%	100%
WBC-Diagnostic	99%	99%

define the **selection gain** to be $1 - search\text{-}effort$.

Table 6.6 shows the selection gain obtained using the proposed GP system on the test datasets. It shows that in almost all cases, the provided ranking is highly useful in finding the best subset of features. The results show that for both the classifiers and all the datasets, the selection gain is quite high. The table also suggests that even for datasets and classifiers for which GP-calculated relevance and the classification performance are not highly correlated (like the WBC-Diagnostic dataset and the decision tree classifier), the highest classification performance is obtained from a subset of features that is ranked very close to the first.

6.6.2.3 Search Space Exploration

Another aspect of the proposed system to consider is the way GP explores the space of subsets of features and forms the Pareto front. Figure 6.10 shows the explored subsets in relation to their cardinality and relevance. In each figure, the horizontal axis shows the cardinality of the subsets of features. The left and right sides of this axis correspond to the left and right side of the search lattice. The vertical axis shows the relevance value obtained by applying the BR function on the output of the GP program trees. Each small point in these figures represents an explored subset of features. The left most column in each figure is relatively sparse as there is only n subsets with cardinality 1 to explore. Towards the right the vast num-

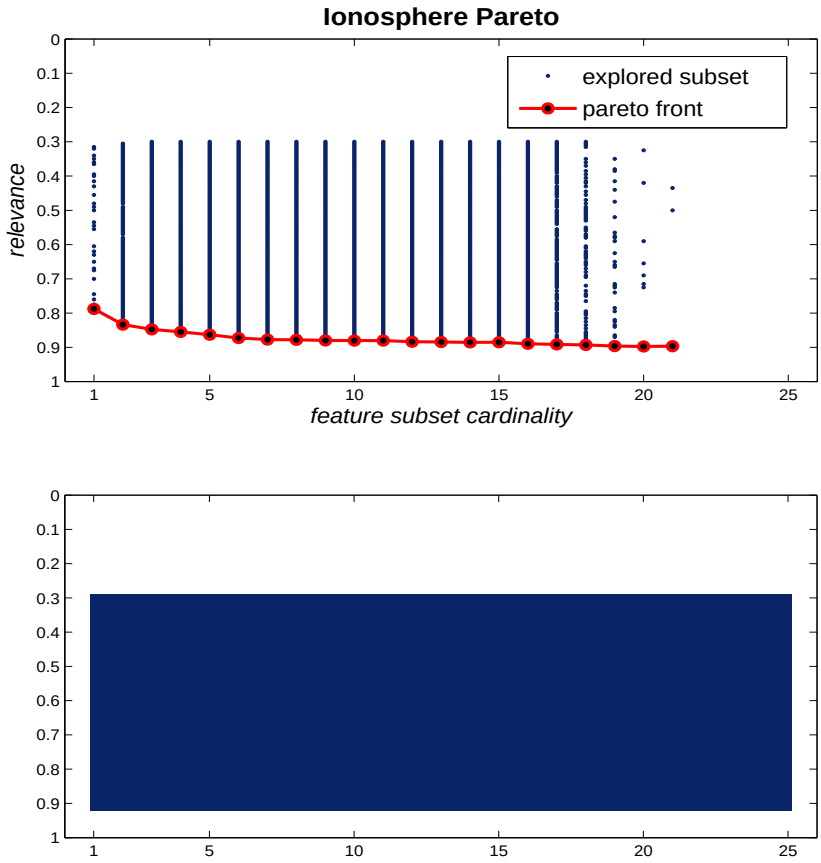
ber of explored subsets cause the columns to look like continuous lines. In the Ionosphere, Sonar and WBC-Diagnostic problems, GP has explored subsets of sizes up to 21, 25, and 19 respectively. Compared to our preliminary experiments presented in Figure 6.6, the proposed system has been able to explore larger subsets while producing high relevance values and avoiding overfitting.

There is a Pareto front line at the bottom of each figure. The bold black point at the bottom of each column represents the subset with the highest relevance for that given cardinality. The goal of the search is to find solutions which are as close as possible to the bottom left corner, that is, solutions with minimum cardinality and maximum relevance. We can see that in all three examples, the slope of the Pareto front line is very low particularly towards the right—that is, as the cardinality of subsets of features increase, for each unit increase in cardinality, only a little more relevance can be gained.

There is an abnormality in these figures that should be explained. In a single GP run, based on the rules defined in Section 6.3.3, for any two columns, all the subsets on the right column should not have lower relevance than the subsets on the left column. However, this is not the case in these figures (for example Pareto front points at 15 and 16 in Sonar). This is primarily because, these figures are based on data accumulated over 50 GP runs. Another possible explanation is that although the depth limit can be increased if a good program tree using a larger number of features can produce an outperforming relevance, there is no guarantee that after this point all the new programs with that number of features will result in a relevance as high as the first one (e.g. the last column of all three figures).

6.6.2.4 Subset Selection

Basically, a Pareto front represents the trade-off between the relevance and cardinality and it is up to the users to choose the desired subset of features according to their needs. In certain classification tasks, however,


Ionosphere Pareto
explored subset
pareto front
relevance
feature subset cardinality
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1
5
10
15
20
25

one might be interested in finding the subset of features which maximises the classification performance, i.e. the best subset of features. Given a Pareto front, to find the best subset of features, one must project the original dataset through the subsets in the Pareto front. In the case of our experiments, after projection, there are 21, 25 and 19 projected datasets, one for each point in the Pareto front of the three datasets. We examine two classifiers, namely decision tree (J48) and SVM, on the projected datasets and find the subset which maximises the performance of these classifiers.

Table 6.7: Classification Results

Dataset	Decision Tree (J48)		SVM	
	Accuracy	Complexity	Accuracy	Complexity
JH Ionosphere				
(before selection)	89.7	34 (35)	88.0	34
(after selection)	**93.0**	**3 (5)**	**89.3**	**13**
Sonar				
(before selection)	73.6	60 (35)	77.8	60
(after selection)	**78.8**	**12 (23)**	**79.6**	**12**
WBC-Diagnostic				
(before selection)	93.7	30 (35)	97.7	30
(after selection)	**95.8**	**4 (19)**	**97.8**	**6**

Table 6.7 shows the results. The table has three sections, one for each dataset. For each dataset, the first line represents the classification performance (accuracy) and classification model complexity before the selection (using all the features) and the second line shows this information after selecting the best subset of features. We define the **complexity** to be the number of features being used by the classifier. For the complexity of C4.5, we also consider the size of the constructed decision tree (number of nodes) after pruning, which is shown in brackets. The table shows that in all datasets the performance has been improved by finding the best

subset of features. In some cases (like J48) the improvement is quite considerable. More importantly, we have been able to significantly reduce the model complexity while improving the performance. The results of J48 are interesting from a different point of view as well. J48 performs its own feature selection using the *Information Gain* (IG) algorithm. Therefore, the J48 results are implicitly a comparison between the proposed algorithm and IG. We see that in all three cases, the new method has achieved better results than those of J48 and its internal IG feature selection.

6.7 Summary and Discussion

This chapter proposed a filter-based genetic programming system for measuring the relevance and then selection of subsets of features in binary classification tasks. Unlike most filter methods that usually deal with single features, our proposed algorithm explores subsets of features. A virtual program structure and an evaluation function are defined in a way that constructed GP programs can measure the goodness of subsets of features. Mathematical expressions built by GP transform the feature space in a way that the relevance of subsets of features can be measured using a simple relevance function such as BR.

Our empirical results indicate that the proposed system is good at ranking subsets and giving insight into the actual classification performance. Although the proposed system does not wrap any classifier for measuring the relevance of subsets of features, the resulting ranking is highly correlated to the actual classification performance. In addition, the best subset of features can be found in the first percentile of high-rank features, in most cases. It is found that GP can recognise relevant features in situations where many other measures cannot. For example, the proposed system can detect relevant subsets of features when the class distribution is multimodal or when the features are correlated.

Using an inexpensive fitness function makes it feasible to explore a

large number of different feature combinations. A standard GP search, however, tends to explore feature subsets of low cardinality. We make some modifications to the standard GP to make it explore large subsets of features when necessary. This is done by increasing the depth limit gradually at runtime. We also try to minimise the effect of bloating and overfitting by using a validation mechanism.

We considered two objectives in the search for optimal subsets: maximising the relevance of the subsets and minimising their cardinality. The output of the algorithm is a collection of solutions (subsets of features) in the form of a Pareto front that have the highest relevance for each given cardinality. The Pareto front vector can serve as a trade-off matrix for the user. It is observed how an inexpensive search over the Pareto front vector can improve the classification performance and reduce the model complexity of all the classifiers in all the benchmark problems investigated in this chapter.

The proposed system might not be quite appropriate for the cases where the best subset of features is expected to have a very large number of features. With the current model, in order to process such feature sets, very deep program trees are required. Deep program trees are computationally more expensive and more vulnerable to genetic operators. Therefore, one should modify the current model in a way that it can manipulate more features in program trees without increasing their depths. One may also consider using some new genetic operators specific to exploring the feature subset space. Another possible future task is to find a workaround to extend the current model to multiple-class classification tasks.

Chapter 7

Feature Selection via Redundancy Elimination

7.1 Introduction

The goal of feature selection is to find a minimal subset of features that is sufficient to describe target concepts. Feature ranking is an avenue to feature selection in which features are ranked based on their relative importance (relevance) with respect to target concepts [60]. Most feature ranking algorithms fall into the *filter approach* category [129, 60], and can only measure the goodness of a single feature [see Chapter 2]. This includes information-theoretic algorithms like information gain and gain ratio, and statistical algorithms like χ^2 (Chi-square) [160].

The assumption in feature selection using single-feature ranking is that high-ranked features are more important (have higher prediction power) and low-ranked features are less relevant to the classification task. Given a good single-feature ranking method—an algorithm that can measure the importance of individual features correctly—feature selection can be achieved by selecting a number of high-ranked features and discarding the rest. We discussed this type of selection in Chapter 5. Although this type of selection is generally quite effective, it has a serious drawback. When

there are a lot of redundant features —features that provide very similar information—feature selection by simply choosing high-ranked features might not achieve good results. The reason is that the information provided by chosen high-ranked features may be very similar. Therefore, when selecting features in descending order of their relevance, it is important not to select those features that are redundant to already-chosen features.

Addressing redundancy is important because when a classification algorithm is limited to use a certain number of features, a feature subset with no redundancy is more efficient and could yield better learning performance. By not having redundant features among a set of selected high-ranked features, the user can get the most information for a given number of features. Detecting and removing redundant features is also useful in scenarios (like medical domains) where extracting/measuring features is a costly task.

If redundancy happens between two single features and follows a certain type of correlation, then there are a number of *univariate* statistical methods which can be used for redundancy detection. However, in practice, there might be some types of dependency between a group of features which do not necessarily follow any particular functional template. Genetic programming (GP) has proved to be a powerful search technique for discovering sophisticated relationships between groups of features; in Chapter 6 we saw that the proposed GP-based system could detect complicated relationships between features and target concepts. For the same reason, GP seems very promising in detecting inter-feature relationships and redundancies.

7.1.1 Chapter Goals

The goal of this chapter is to devise a GP-based system to detect and remove redundant features and meet the following objectives:

- the system must be *filter-based* (as opposed to wrapping another classification algorithm for making decisions);
- the system must be able to measure the degree of redundancy of a feature with respect to a group of features;
- using the redundancy measures, the system must be able to improve the feature selection performance.

To achieve these objectives, we propose an unsupervised GP-based method which extends the *univariate* linear definition of correlation to a *non-linear multivariate* correlation to measure the degree of redundancy of a feature with respect to a group of features. We then introduce a forward selection algorithm which can be used along with the proposed measure to perform feature selection based on the output of a feature ranking algorithm.

7.2 Primary Concepts

We start with some underlying concepts of redundancy. We give the definition of redundancy and its measure and discuss how a GP search can be beneficial in this context. We also prove how our proposed measure of redundancy can reduce the size of the GP search space.

7.2.1 Redundant Features

The output of a single-feature ranking algorithm is presented as a vector of positive integers called a **ranking vector** whose elements are indexes to the features of a dataset. The elements are sorted in descending order of the importance of their corresponding features. Usually a small number of features indexed at the beginning of this vector are used to build learning models. However, as features are examined individually during the ranking process, it happens quite often that high-ranked features exhibit

redundancy. For example, although a feature at the i-th rank is more relevant than a feature at the $(i+1)$-th rank, the former would not be as useful as the latter if the former is redundant with respect to features at ranks 1 to $i-1$ while the latter is not. Redundancy can be defined based on general consensus:

Definition A feature X is **redundant** with respect to a subset $\mathcal{A}$ of features if and only if it can be approximated (reconstructed) by a function of $\mathcal{A}$.

That is, we consider a feature X to be redundant if its information content can be provided by a function of $\mathcal{A}$. **Redundancy removal** is the process of correcting a ranking vector by replacing redundant features with non-redundant features at lower ranks.

7.2.2 Degrees of Redundancy

In practical situations, the redundancy of a feature with respect to a subset of other features has different degrees (partial redundancy). That is, given a feature X, and a subset of features, $\mathcal{A}$, only part of the information content of X can be expressed by a function of $\mathcal{A}$. Therefore, to decide whether X is redundant with respect to $\mathcal{A}$, one requires a measure of the degree of redundancy to determine what proportion of information provided by X is already contained in $\mathcal{A}$.

One way of detecting/measuring redundancy could be through symbolic regression. In symbolic regression, the solutions are functions that approximate the target variable as closely as possible. Therefore, if a symbolic regression algorithm (like GP) can find a function of a group of features that can approximate another feature, there is a redundancy. In the same manner, measuring redundancy can be translated to measuring the performance of a symbolic regression where a close approximation of X by a function of $\mathcal{A}$ suggests that X is redundant. Although it is possible to use symbolic regression for redundancy detection, it is not very feasible. In redundancy detection, we are only interested in the existence of good

approximating functions and not their exact formulations. Searching for the exact formulation of an approximating function, the way symbolic regression does, increases the required computational effort. Given a fixed amount of computational effort, the probablity of verifying the existance of an approximating function decreases.

Another way of measuring redundancy could be through dependency checking. Presence of dependency between two features is an indication of redundancy between the two. A common measure of dependency between two random variables is the quotient of their linear relationship. Linear correlation between two features is a sufficient condition, but it is not necessary—there might be a non-linear relationship between two features. This is in fact a disadvantage of using linear correlation as a measure for redundancy. Another disadvantage of this method is that it can be applied to only two features at a time.

7.3 Using Genetic Programming to Measure Redundancy

We use GP to overcome the limitations of symbolic regression and linear correlation and making use of their advantages at the same time. GP can construct a function with any number of arguments and any degree of freedom to optimise a fitness function; it can be used to measure a wide variety of linear and non-linear correlations between a single feature and a subset of features by using GP-constructed functions (programs) to transform the subset to one scalar feature [see Chapter 6]. Our goal here is to use GP to measure functional dependencies between features without putting too much computational effort into finding the exact formulation of the functions.

7.3.1 A GP-based Redundancy Measure

Irrespective of the type of relationship between a given feature and a group of other features, it is possible to find a *function* of the other features that is linearly related to the given feature. It can be proved that as the linear correlation between a feature and a function of some other features increases, the error of approximating the feature by the function decreases.

Proposition 7.3.1. *The error of approximating a feature X by a linear function of $\phi(\mathcal{A})$ approaches zero as $\rho^2 = \mathrm{Cor}^2(X, \phi(\mathcal{A})) \in [0, 1]$, the square of Pearson's product-moment correlation coefficient, approaches one.*

Proof. Let $\hat{X} = \alpha + \beta\phi(\mathcal{A})$, $\alpha, \beta \in \mathbb{R}$, be a linear approximation of X with the error of approximation defined as $\varepsilon = X - \hat{X}$. If α and β are determined by the least squares method, the following properties hold [58]:

(i) $E(\varepsilon) = E(X - \hat{X}) = 0$

(ii) $\mathrm{Cov}(\varepsilon, \hat{X}) = \mathrm{Cov}(X - \hat{X}, \hat{X}) = 0$ (since $\hat{X}$ and ε are orthogonal)

Because of property (ii), the covariance between X and $\hat{X}$ reduces to the variance of $\hat{X}$;

$$\mathrm{Cov}(X, \hat{X}) = \mathrm{Cov}(X - \hat{X}, \hat{X}) + \mathrm{Cov}(\hat{X}, \hat{X}) = \mathrm{Var}(\hat{X}),$$

and hence, the correlation between X and $\hat{X}$ becomes

$$\mathrm{Cor}(X, \hat{X}) = \frac{\mathrm{Cov}(X, \hat{X})}{\sqrt{\mathrm{Var}(X)\mathrm{Var}(\hat{X})}} = \frac{\mathrm{Var}(X)}{\sqrt{\mathrm{Var}(X)\mathrm{Var}(\hat{X})}} = \frac{\sqrt{\mathrm{Var}(\hat{X})}}{\sqrt{\mathrm{Var}(X)}} \tag{7.1}$$

Alternatively, the correlation between X and $\hat{X}$ can be derived as follows

$$\begin{aligned} \mathrm{Cor}(X, \hat{X}) =& \mathrm{Cor}(X, \alpha + \beta\phi(\mathcal{A})) \\ =& \frac{\beta\ \mathrm{Cov}(X, \phi(\mathcal{A}))}{\sqrt{\mathrm{Var}(X)\beta^2\mathrm{Var}(\phi(\mathcal{A}))}} = \mathrm{Cor}(X, \phi(\mathcal{A})) = \rho \end{aligned} \tag{7.2}$$

and therefore, from equations (7.1) and (7.2), we get

$$\rho^2 = \frac{\text{Var}(\hat{X})}{\text{Var}(X)} \tag{7.3}$$

The error of approximation, more specifically, the mean squared error (MSE) of the approximation is a function of this squared correlation coefficient. The MSE of the approximation, using property (i), is

$$MSE(\hat{X}) = E(\varepsilon^2) = E[(X - \hat{X})^2] = \text{Var}(\varepsilon) + E^2(X - \hat{X}) = \text{Var}(\varepsilon)$$

where the variance of ε is

$$\text{Var}(\varepsilon) = \text{Var}(X - \hat{X}) = \text{Var}(X) + \text{Var}(\hat{X}) - 2\text{cov}(X, \hat{X}) = \text{Var}(X) - \text{Var}(\hat{X})$$

Therefore, from (7.3), it follows that

$$MSE(\hat{X}) = \text{Var}(X) - \rho^2 \text{ Var}(X) = (1 - \rho^2) \text{ Var}(X)$$

Hence, as ρ^2 increases, $MSE(\hat{X})$ decreases. More precisely

$$\lim_{\rho^2 \to 1} MSE(\hat{X}) = \lim_{\rho^2 \to 1} (1 - \rho^2)\text{Var}(X) = 0 \ .$$

□

Therefore, the linear relationship (correlation) between the feature X and the function of features $\phi(\mathcal{A})$ is a good measure of the redundancy between the two. More specifically, if a GP search succeeds in maximising the squared correlation between X and $\phi(\mathcal{A})$, $\text{Cor}^2\,(X, \phi(\mathcal{A}))$, the error of approximating X by some function of $\mathcal{A}$ can be minimised.

Definition The degree of redundancy of a feature X with respect to a subset $\mathcal{A}$ of features is defined as

$$\rho^2_{max} = \max_{\phi \in \Phi} \{\text{Cor}^2\,(X, \phi(\mathcal{A}))\}$$

where Φ is a finite set of GP-constructable functions of $\mathcal{A}$ and $\text{Cor}^2\,(\cdot, \cdot)$ is the square of Pearson's product-moment correlation coefficient between two random variables.

The range of ρ^2_{max} is $[0, 1]$ where zero and one correspond to the minimum and maximum possible degrees of redundancy. The functions in Φ are formed by GP using a set of primitive operators and a set of *variable terminals* which correspond to the features in $\mathcal{A}$. GP tries to maximise $\mathrm{Cor}^2(\cdot, \cdot)$ as its fitness function and depending on how successful it is, the degree of redundancy can be determined.

An advantage of this redundancy measure is that GP does not need to find any actual approximation for X directly, but once it finds any member of the family of functions that optimises $\mathrm{Cor}^2(\cdot, \cdot)$, we would know that X is redundant. In particular, (7.2) implies that if $\hat{X} = \alpha + \beta\phi(\mathcal{A})$ is the best linear approximation for X, where $\alpha, \beta \in \mathbb{R}$, any function of the form $\alpha' + \beta'\phi(\mathcal{A})$ where $\alpha', \beta' \in \mathbb{R}$ can maximise the fitness function. This is particularly important because GP is not normally equipped with any type of hybrid learning (like gradient descent or least square) to find right values for numeric constants efficiently. Therefore, a GP search with such a fitness function is much more relaxed compared to a scenario where MSE is used as the fitness function. This fact can significantly reduce the computational cost and accordingly improve the probability of success of the GP runs.

7.3.2 A Synthetic Example

We give an example to illustrate how the proposed measure can detect redundant features that are not normally detectable by ordinary methods and to show how it can reduce the size of the GP search space. Consider two random variables $X_1 \sim N(\mu = 0, \sigma^2 = 2)$ and $X_2 \sim N(\mu = 0, \sigma^2 = 1)$. We define a third random variable with high functional dependency (redundancy) as $X_3 = -3X_1X_2 + 2 + \xi$ where $\xi \sim U(-1, 1)$ is noise. We create sample vectors $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ of size 10,000 from the random variables X_1, X_2 and X_3 respectively. Figure 7.1 shows these three vectors plotted against each other. There is no mutual linear relationship visible between

these vectors as expected.

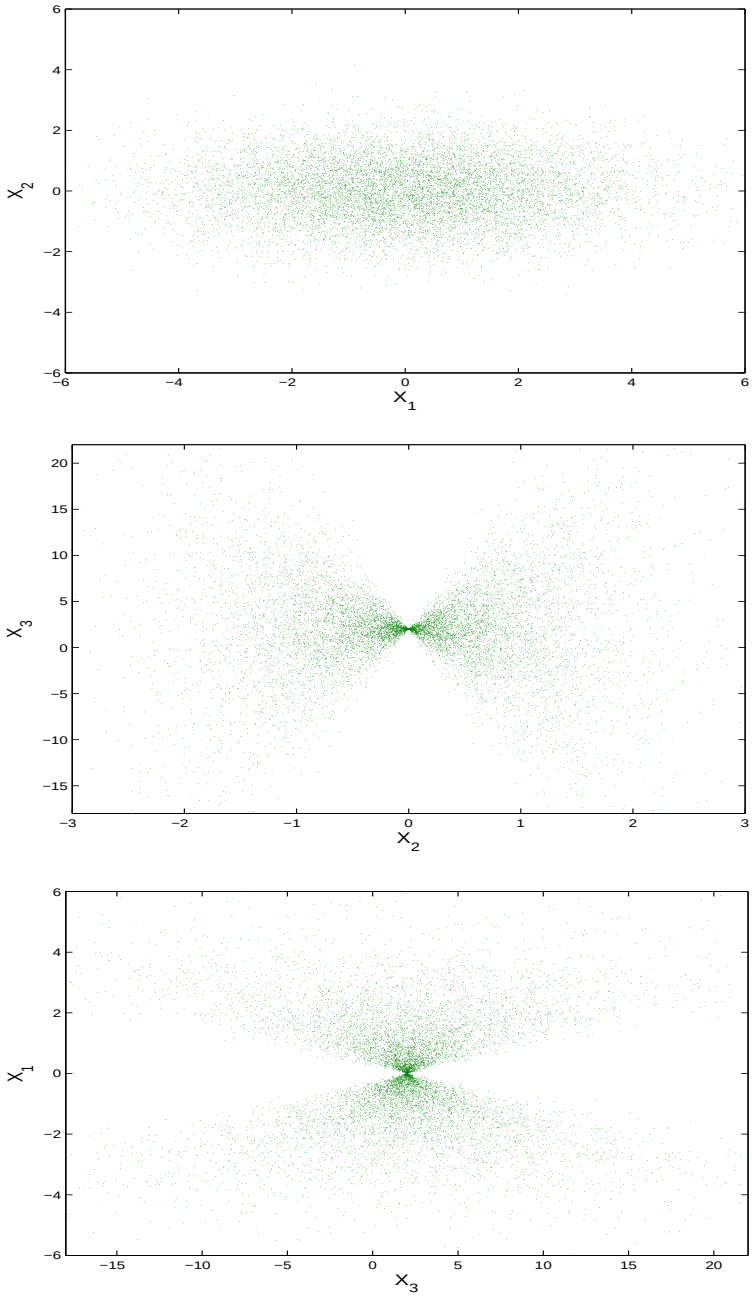


Figure 7.1: An artificial example where $\mathbf{x}_3$ is redundant in the context of $\mathbf{x}_1$ and $\mathbf{x}_2$. The scatter plots of the three features are visualised.

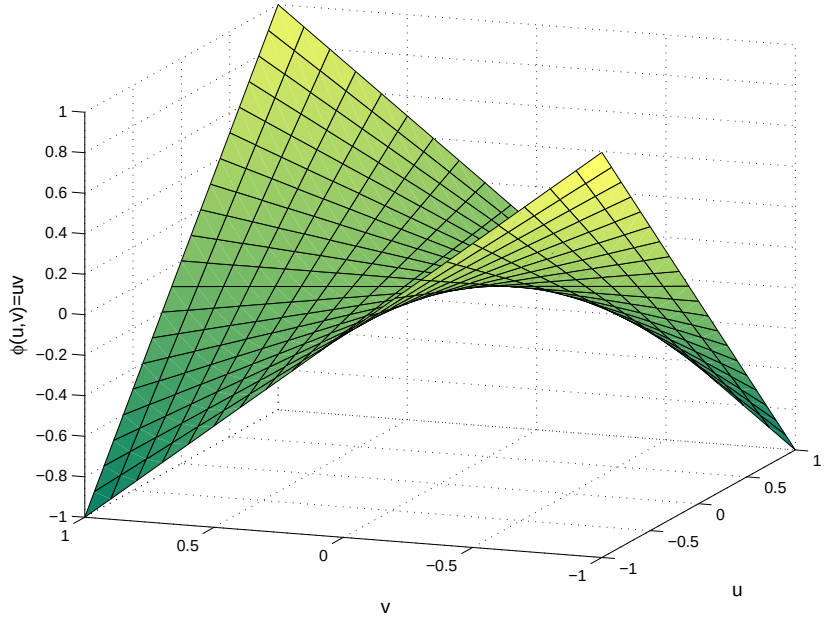


Figure 7.2: A non-linear transformation function (a GP program) that can detect the redundancy in Figure 7.1.

In terms of measurements, $\mathrm{Cor}^2(\mathbf{x}_3, \mathbf{x}_1) = 0.00$ and $\mathrm{Cor}^2(\mathbf{x}_3, \mathbf{x}_2) = 0.00$. That is, the redundancy of $\mathbf{x}_3$ cannot be detected by measuring its correlation against the other two random variables individually. Using a simple GP search to maximise $\mathrm{Cor}^2(\mathbf{x}_3, \phi(\mathbf{x}_1, \mathbf{x}_2))$, however, results in a variety of solutions like $\phi(u, v) = uv$ as illustrated in 7.2. Using this GP-constructed function, $\mathrm{Cor}^2(\mathbf{x}_3, \phi(\mathbf{x}_1\mathbf{x}_2)) = 0.99$, which indicates a high redundancy. We can see that GP does not need to find the exact formula of $\mathbf{x}_3$; actually any linear combination of uv would yield the same result.

7.3.3 Algorithm

The GP algorithm used to measure the degree of redundancy between a feature f and a subset of features $\mathcal{A}$ is presented in Algorithm 10. The values of the features (instances) are stored in a vector $\mathbf{x}$. The fitness function

of the GP search is $\mathrm{Cor}^2(\mathbf{x}, \phi(\mathcal{A}))$. The computational complexity of this measure is $O(n)$, which is quite good for a fitness function. The goal of the GP search is to maximise this measure. The threshold θ which determines the maximum acceptable redundancy is a parameter of the algorithm. The search would stop once the fitness of an individual reaches this threshold. The output of the algorithm is the highest achieved fitness which is the degree of redundancy ρ^2_{max} according to the definition.

The algorithm starts with adding all the features in $\mathcal{A}$ to the GP variable terminals set. Lines 2 and 3 initialise the population and add all the variable terminals as single-node trees to the population. This is to make sure that at all times, the redundancy will be measured against every single feature in the population. Each GP program in the population defines a function $\phi_{program} : \mathbb{R}^{|A|} \mapsto \mathbb{R}$ which transforms an input vector $(\mathbf{x_1}[i], \mathbf{x_2}[i], \ldots, \mathbf{x}_{|\mathcal{A}|}[i])$ into a scalar value $\mathbf{y}[i]$. Lines 9 to 15 calculate the fitness. Line 16 updates the measured degree of redundancy ρ^2_{max} if the fitness of the current program exceeds the current value of ρ^2_{max}.

7.4 Feature Selection

7.4.1 System Diagram

Figure 7.3 shows the diagram of a feature selection system that uses GP to measure the redundancy between features and then removes the unwanted features. Given the dataset of classification task, the system first uses a single-feature ranking algorithm to rank the features. The system then uses a forward selection algorithm to form a selected set of high-rank features that are not redundant with respect to each other. The selection algorithm uses GP to measure the redundancy between features. The selected subset of features is then used with the classification algorithm. We used the classification performance to validate the proposed system.

Algorithm 10: $Measure\text{-}Redundancy(\mathbf{x}, \mathcal{A}, \theta)$

```
/* Use GP to measure the redundancy between a
   feature and a set of features.                    */
```

Input: $\mathbf{x}$, the values (observations) of a feature f, where $f \in \mathbb{F}$

Input: $\mathcal{A}$, a subset of features, where $\mathcal{A} \subseteq \mathbb{F} \backslash \{f\}$

Input: θ, maximum acceptable redundancy quotient

Output: ρ^2_{max}, the redundancy between $\mathbf{x}$ and $\mathcal{A}$ where $\rho^2_{max} \in [0, 1]$

$\mathcal{T} \leftarrow \mathcal{A}$; // variable terminals include all the features in $\mathcal{A}$

$\mathcal{P} \leftarrow$ *a population of randomly-generated GP programs*;

$\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{T}$; // include all single node (terminal) programs

$\rho^2_{max} \leftarrow 0$; // initialise the measure of redundancy

while $\neg max\text{-}generations \wedge (\rho^2_{max} < \theta)$ **do**

  foreach $\phi \in \mathcal{P}$ **do**

    /* calculate the fitness for each program */

    $s_x, s_y, s_{x^2}, s_{y^2}, s_{xy} \leftarrow 0$; // initialising the sums

    for $i \in \{1, 2, \ldots, n\}$ **do**

      $\mathbf{y}[i] \leftarrow \phi(\mathbf{x}_1[i], \mathbf{x}_2[i], \ldots, \mathbf{x}_{|\mathcal{A}|}[i])$; // transformation

      $s_x \leftarrow s_x + \mathbf{x}[i]$;

      $s_y \leftarrow s_y + \mathbf{y}[i]$;

      $s_{x^2} \leftarrow s_{x^2} + (\mathbf{x}[i])^2$;

      $s_{y^2} \leftarrow s_{y^2} + (\mathbf{y}[i])^2$;

      $s_{xy} \leftarrow s_{xy} + \mathbf{x}[i]\mathbf{y}[i]$;

    $fitness_\phi \leftarrow \left(\frac{n s_{xy} - s_x s_y}{\sqrt{n s_{x^2} - s_x^2} \sqrt{n s_{y^2} - s_y^2}} \right)^2$; // calculating $\mathrm{Cor}^2(\mathbf{x}, \mathbf{y})$

    $\rho^2_{max} \leftarrow \mathtt{max}(\rho^2_{max}, fitness_\phi)$;

  $\mathcal{P} \leftarrow$ *new population using genetic operators, keeping the best*

return ρ^2_{max};

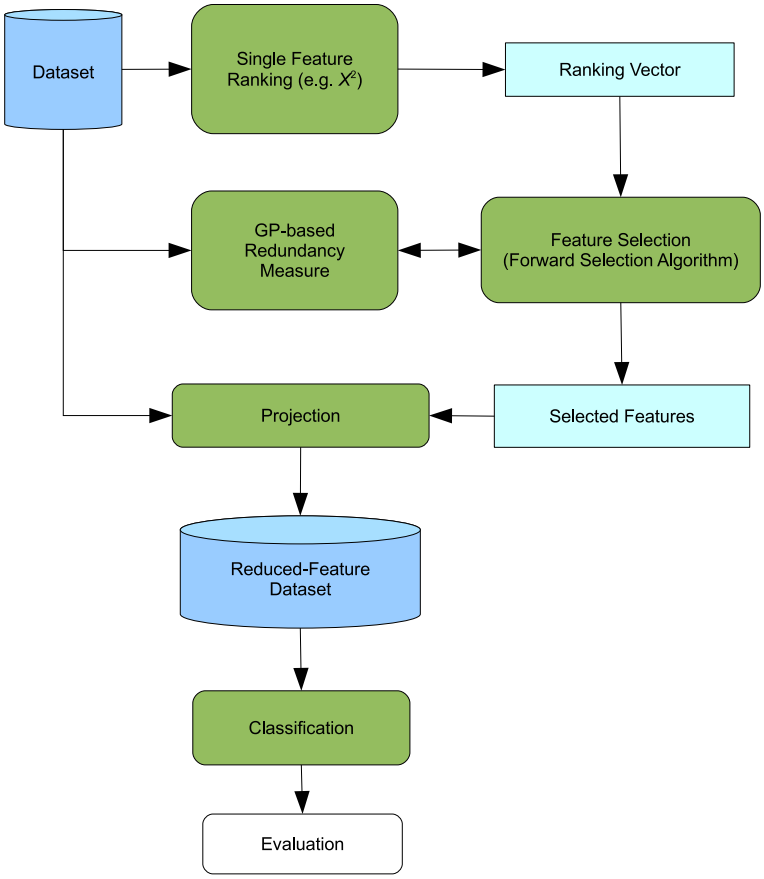


Figure 7.3: Diagram of a feature selection system using GP to evaluate redundancy.

7.4.2 Forward Selection Algorithm

Given a *preliminary* ranking vector (the output of a single-feature ranking algorithm) and a threshold for the maximum redundancy, we introduce a *forward selection* algorithm that selects non-redundant features by performing redundancy removal. The steps are presented in Algorithm 11. The algorithm takes as inputs the set of all m features in the dataset $\mathbb{F}$, a ranking vector r, which is the output of a single-feature ranking algorithm, the desired number $m^\star$ of features to be selected and a threshold θ which determines the maximum acceptable redundancy. At the first line,

the highest-ranked feature is added to the set $\mathcal{F}^\star$ of selected features. In the loop starting on line 3, the degree of redundancy of the next highest-ranked feature will be measured (on line 4) and the feature will be added to $\mathcal{F}^\star$ only if its redundancy is less than the threshold θ. The algorithm stops when $m^\star$ features are selected or all the features in the dataset have been processed.

Algorithm 11: $Forward\text{-}Selection(\mathbf{X}, \mathbf{r}, m^\star, \theta)$

```
/* The algorithm finds a (sub)optimal subset of
   features by adding high ranked features
   incrementally while eliminating features
   reported redundant by GP.                    */
```

Input: $\mathbf{X}$, a matrix of the form $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$ where $\mathbf{x_i}$ $(i = 1, \ldots, m)$ is a vector of length n containing samples from the i-th original feature in the problem, and m is the total number of input features

Input: $\mathbf{r} = (r_1, r_2, \ldots, r_m)$, a ranking vector

Input: $m^\star$, the desired number of selected features

Input: θ, the maximum acceptable redundancy where $\theta \in [0, 1]$

Output: $\mathcal{F}^\star$, the set of selected features

1 $\mathcal{F}^\star \leftarrow \{\mathbf{x}_{r_1}\}$; // adding the feature at the highest rank

2 $i \leftarrow 2$; // the next rank to be processed

3 **while** $(|\mathcal{F}^\star| < m^\star \wedge i \leq m)$ **do**

4 $\quad \rho^2 = Measure\text{-}Redundancy(\mathbf{x}_{r_i}, \mathcal{F}^\star, \theta)$; // measure the redundancy

5 $\quad$ **if** $\rho^2 \leq \theta$ **then**

6 $\quad\quad \mathcal{F}^\star \leftarrow \mathcal{F}^\star \cup \{\mathbf{x}_{r_i}\}$;

7 $\quad i \leftarrow i + 1$;

8 **return** $\mathcal{F}^\star$;

7.5 Empirical Results

7.5.1 Design of Experiments

The procedure we adopt for validating the system is based on the flow of data and processes depicted in Figure 7.3. We choose a dataset whose features are likely to exhibit some degree of redundancy. We use Chi-square (χ^2) for single-feature ranking and then the resulting ranking vector is used by the forward selection algorithm. We then compare the classification performance by using high-rank features (single-feature ranking, without redundancy removal) with the classification performance obtained by using the selected set of features (the output of the forward selection algorithm).

7.5.1.1 Datasets

We use the Isolet5 dataset from the UCI machine learning repository [6]. The dataset has been created by recording the voice of 30 people pronouncing the names of the 26 English alphabets twice. There are 52 samples per person and 1559 samples in total (one sample is missing). The task is to classify the alphabets. This dataset was chosen since it contains a large number of features. There are 617 features available in total including spectral coefficients, contour features, sonorant features, pre-sonorant and post-sonorant features. All the features are real-valued, continuous and scaled into the range $[-1, 1]$.

7.5.1.2 GP Settings and Implementation Details

We use the standard tree-based GP model [71]. In this model, each program produces a single floating-point number at its root as the result of its evaluation (output). Table 7.1 shows various settings of the GP system we developed for the experiments. There is one variable terminal for each feature in the problem. A number of randomly generated constants are

also used as terminals. The four standard arithmetic operators were used to form the function set. The division operator is *protected*—that is, it returns zero for division by zero. All the members of the function set are binary—they take two parameters.

Table 7.1: GP Settings

Function Set:	$+, -, \times, \div$ (protected division)
Variable Terminals:	The original features ($\{x_1, x_2, \ldots, x_m\}$)
Constant Terminals:	Randomly Generated
Population Size:	1024
Number of Generations:	50
Initialisation:	Ramped half and half
Mutation:	Subtree creation
Selection:	Tournament (size=5)
Initial Tree Depth:	4
Maximum Tree Depth:	6
Mutation Probability:	Adaptive [21]
Cross-over Probability:	Adaptive [21]
Elitism:	Yes

The ramped half-and-half method [71] is used for generating programs in the initial population and for the mutation operator. The initial maximum program tree depth is set to 4, but it can increase to 6 during evolution. We use a population size of 1024, however, if the cardinality of $\mathcal{A}$ in Algorithm 10 is very high, using a bigger population is recommended. The probability of the crossover and mutation operators are adapted automatically at runtime [21]. An elitist approach has been taken to ensure that the performance of the fittest individual in the population never deteriorates. The evolution is terminated, at the latest, after the 50th generation or when a solution of fitness (relevance) 1.0 is found. The platform is implemented in Java and we use grid computing to have parallel GP runs.

7.5.1.3 Evaluation

The evaluation settings are available in Table 7.2. We use Chi-square (χ^2) for feature ranking, and J48 decision tree [152] and SVM (SMO) [61] for classification. Since no separate test data is available, we will use 10-fold cross-validation in our experiments. We use the Weka [152] library for the preliminary χ^2 ranking, classification and evaluation processes.

Table 7.2: Evaluation Settings

Validation:	10-fold cross-validation with stratified folds
Classifiers:	Decision Tree (J48 version of C4.5), SVM (SMO)
θ:	$\{0.0, 0.1, \ldots, 1.0\}$ (maximum tolerance for redundancy)
$m^\star$:	$\{1, 2, 3, \ldots, 30\}$ (number of selected features)

7.6 Results and Analysis

Table 7.3 presents the result of applying the proposed forward selection algorithm on the Isolet5 dataset for different values of θ and $m^\star$. The numbers in each column are indexes to the features in the dataset. By increasing $m^\star$ from 1 to 30, one step at a time, new ranking vectors are created for the given values of θ. Each column presents the *corrected ranking* generated by removing redundant features from the preliminary ranking vector for the given value of θ. In the first column $\theta = 1$ which means any level of redundancy is accepted. Therefore the content of this column is actually the preliminary ranking vector, the output of the χ^2 ranking algorithm, without any changes.

Table 7.3: Corrected ranking based on different redundancy thresholds (θ)

$m^\star$ / Rank	$\theta =$ 1.0	$\theta =$ 0.9	$\theta =$ 0.8	$\theta =$ 0.7	$\theta =$ 0.6	$\theta =$ 0.5	$\theta =$ 0.4	$\theta =$ 0.3	$\theta =$ 0.2	$\theta =$ 0.1	$\theta =$ 0.0
1	584	584	584	584	584	584	584	584	584	584	584
2	390	390	390	389	548	548	548	548	548	548	548
3	392	392	548	548	419	419	419	419	413	474	474
4	391	395	419	419	107	358	325	325	325	410	528
5	395	548	73	73	412	411	474	474	474	448	378
6	389	419	413	413	358	171	410	472	448	577	-
7	548	73	517	358	11	474	78	427	214	48	-
8	419	462	10	139	546	387	472	20	480	582	-
9	73	107	358	546	388	425	352	130	48	292	-
10	549	75	458	386	474	12	590	480	437	433	-
11	394	9	325	266	425	522	427	181	435	599	-
12	462	413	139	362	522	545	523	351	164	530	-
13	393	517	515	323	5	472	20	481	528	368	-
14	74	358	546	485	203	352	130	112	463	-	-
15	107	42	386	327	472	69	480	437	347	-	-
16	75	458	134	474	363	589	181	333	428	-	-
17	9	325	362	397	322	214	332	435	334	-	-
18	413	415	173	425	382	427	259	364	331	-	-
19	106	11	360	198	352	446	372	164	370	-	-
20	412	139	474	12	448	451	481	525	600	-	-
21	517	411	397	522	486	20	398	595	473	-	-
22	418	359	387	174	589	130	437	4	-	-	-
23	10	547	110	78	143	321	435	532	-	-	-
24	461	515	233	5	214	480	364	433	-	-	-
25	358	546	425	203	576	577	445	377	-	-	-
26	417	386	198	424	427	493	406	224	-	-	-
27	416	265	76	472	452	16	164	336	-	-	-
28	42	550	426	101	446	541	528	403	-	-	-
29	458	134	298	352	14	24	28	341	-	-	-
30	457	518	12	448	20	332	595	252	-	-	-

The feature at the first rank is always the same; this is due to the first line of the forward selection algorithm. However, as the redundancy threshold decreases, features at the lower ranks might be removed due to being redundant with respect to the features at higher ranks. For example, by decreasing θ from 1 to 0.9, the feature at the fourth rank (391) is considered redundant with respect to the three feature at higher ranks (584, 390 and 392) and hence, is removed and replaced by the next feature (395), which in this case is not redundant with respect to those three features. As θ decreases, more features are removed due to redundancy. For very low values of θ, like 0.2 and lower, the number of selected features, i.e. the number of remaining features after redundancy removal, is quite low (less than 30).

To study the effect of elimination of redundant features, we use the selected features in groups of size 5, 10, ..., 30. Table 7.4 shows the number of redundant (and hence, eliminated) features for different thresholds. For a given θ, the number in each column represents the number of features that have been removed in order to select $m^\star$ features. In the first row, where $\theta = 1$, no feature is removed. The hyphens in the table indicate situations where the desired number of selected features cannot be obtained due to the large number of features that have been removed.

It should be noted that although in theory only features with some level of redundancy exhibit a ρ^2_{max} greater than zero, in practice even two independent features may have a ρ^2_{max} greater than zero. The major cause is that the true quotient of correlation can only be obtained by an unlimited number of observations. All the measurements obtained from real problems are actually estimations of the true values. For instance, although X_1 and X_2 in our synthetic example are completely independent, their correlation is not absolute zero due to the limited number of observations (in that case 10,000). A minor cause could be the existence of confounding factors or lurking variables [123] which happens when features show some correlation, but their contents are completely different. Therefore, large

Table 7.4: Number of eliminated redundant features

θ	$m^\star = 5$	$m^\star = 10$	$m^\star = 15$	$m^\star = 20$	$m^\star = 25$	$m^\star = 30$
1.0	0	0	0	0	0	0
0.9	2	6	13	24	34	36
0.8	4	19	45	74	82	86
0.7	4	50	78	96	104	118
0.6	15	84	120	128	141	151
0.5	47	106	138	160	171	199
0.4	89	159	179	241	284	330
0.3	89	184	281	317	494	514
0.2	89	286	468	555	-	-
0.1	143	487	-	-	-	-
0.0	340	-	-	-	-	-

numbers of eliminated features for low values of θ are not necessarily due to true redundancy.

Tables 7.5 and 7.6 show the results of applying two classification algorithms, J48 and SVM, on different numbers of selected features. The structure of the table is similar to that of Table 7.4. The numbers in the table are classification test performances which are calculated as the ratio of correctly classified instances to the total number of instances through a 10-fold cross-validation process. The first row, where $\theta = 1$ (with no redundant features being removed), is considered the baseline. Therefore, the first row shows the classification performance using the first 5, 10, ..., 30 top features obtained directly from the χ^2 ranking algorithm. The performance measures on the second and lower rows are obtained using features selected by removing redundant features. In each column, the performance results are obtained based on the same number of features and the highest performance is in bold.

Table 7.5: Performance of the J48 classifier using the selected features

θ	$m^\star = 5$	$m^\star = 10$	$m^\star = 15$	$m^\star = 20$	$m^\star = 25$	$m^\star = 30$
1.0	0.22	0.43	0.54	0.62	0.68	0.68
0.9	0.31	0.53	0.60	0.66	0.67	0.67
0.8	0.34	0.53	0.61	**0.67**	**0.69**	**0.70**
0.7	0.32	**0.54**	0.57	0.63	0.67	0.67
0.6	**0.38**	**0.54**	**0.64**	0.63	0.67	**0.70**
0.5	0.33	0.52	0.60	0.64	0.66	0.66
0.4	0.30	0.48	0.55	0.56	0.58	0.60
0.3	0.30	0.40	0.52	0.57	0.58	0.58
0.2	0.34	0.48	0.55	0.56	-	-
0.1	0.27	0.36	-	-	-	-
0.0	0.22	-	-	-	-	-

None of the performance results obtained by using the original ranking (the baseline) have achieved the best performance. In fact, compared to the lower rows, the performance of the original ranking is quite low in most cases. By decreasing the redundancy threshold on the lower rows, the performance starts rising. The best performances are spread among rows with θ between 0.5 and 0.8. In most cases, the difference between the baseline and the highest performance is quite significant. This indicates that replacing redundant features by non-redundant ones has a major effect on the performance. On the other hand, as θ decreases below 0.5, resulting in an aggressive removal of redundant and semi-redundant features, the performance reduces down to the baseline or even less. However, this is not unexpected, since as described earlier, part of the measured redundancy could be due to the limited number of observations or the presence of confounding features. It is also observed that although the two classifiers have different classification performances on the same subsets of

Table 7.6: Performance of the SVM classifier using the selected features

θ	$m^\star = 5$	$m^\star = 10$	$m^\star = 15$	$m^\star = 20$	$m^\star = 25$	$m^\star = 30$
1.0	0.23	0.46	0.60	0.69	0.76	0.77
0.9	0.31	0.58	0.70	0.74	0.76	0.78
0.8	0.36	0.57	0.66	**0.76**	0.78	0.82
0.7	0.35	0.57	0.65	**0.76**	**0.82**	0.84
0.6	**0.43**	**0.59**	**0.72**	0.73	0.81	**0.85**
0.5	0.29	**0.59**	0.68	0.74	0.81	0.83
0.4	0.27	0.45	0.64	0.71	0.73	0.75
0.3	0.27	0.45	0.59	0.69	0.76	0.76
0.2	0.33	0.52	0.64	0.69	-	-
0.1	0.29	0.42	-	-	-	-
0.0	0.25	-	-	-	-	-

features, their performance trends with respect to the changes in θ and $m^\star$ are similar and they seem to conform with each other on the best selected subset of features.

7.7 Summary and Discussion

Feature selection algorithms that merely rely on feature ranking methods can severely suffer from the redundancy issue. Features at high ranks, although highly related to target concepts, might be redundant with respect to each other. In this chapter, we devised a GP-based algorithm to measure the redundancy of a feature with respect to a group of features. The algorithm can measure non-linear redundancies that are not detectable by traditional dependency measures.

We used the GP-based measure with a forward selection algorithm for feature selection. The highest tolerable quotient of redundancy can be ad-

justed through the parameter θ in the algorithm. Our results show that removing redundant features using a certain range of θ can significantly boost the classification performance. We also observed that the impact of loss of information by removing features that exhibit only a very low level of redundancy might be higher than the benefit of discarding redundant features. We discovered that a moderate value for θ (e.g. $\theta \in [0.5, 0.8]$) can yield the optimal performance.

The proposed GP-based redundancy measure in this chapter is unsupervised; that is, it does not need the instances to be labelled. An unsupervised approach can be quite efficient in applications where labeling data is costly. Not having to label data also make it easier to use a large amount of data with the algorithm and thus, have better estimations for redundancy and then better feature selection. We tested the proposed algorithm on a classification task, but it can generally be applied to any problem with numeric features.

Chapter 8

Conclusions

The goal of this thesis was to use Genetic Programming (GP) for feature manipulation in classification problems through by discovering relationships between the variables of classification task. The focus was on using GP to discover complex relationships that could not have been discovered by commonly-used classification algorithms otherwise. The goal has been achieved by using GP in three aspects of feature manipulation, namely feature construction, feature ranking and feature selection. The thesis demonstrated a set of new ideas and methodologies that use GP to modify the input representation of classification tasks in order to improve the classification performance and reduce the complexity of classifiers.

8.1 Achieved Objectives

The thesis has achieved the following objectives:

- The thesis proposes a GP-based method for constructing multiple high-level features in classification tasks. Using the constructed features, the classification performance is considerably improved in several benchmark problems.
- The thesis proposes a GP-based transformational dimensionality re-

duction method for classification problems. Transforming the input space using the evolved GP programs reduces the dimensionality of the problems while improving the classification performance.

- The thesis proposes two GP-based feature ranking methods that reveal the importance of features either in the form of a single feature or a subset of features. The ranking provided by GP reflects the importance of the input features and their influence on the classification performance.

- The thesis proposes two GP-based feature selection methods: one based on searching through a collection of subsets of high-rank features and the other by removing redundant features from a collection of single high-rank features. Both methods reduce the dimensionality of the classification problems and improve the classification performance.

8.2 GP and Feature Manipulation

It is found that in all three aspects of feature manipulation, the proposed GP-based systems are remarkably effective in finding relationships between the variables of a classification task. In feature construction, the proposed systems are able to successfully construct functions of *conditional variables* (input features) that capture the behaviour of *decision variables* (target classes). In feature ranking and selection, the proposed systems are able to successfully find pre-existing relationships between conditional variables and decision variables to evaluate the influence of input features on the prediction of class labels. In feature selection and redundancy removal, the proposed GP-based system is also able to detect and measure functional inter-dependency (redundancy) between conditional variables.

Throughout the thesis it is observed that GP is very effective at searching for functions of features satisfying certain objectives. For the sake of

generality and to decrease the computational cost, the thesis takes a *filter approach*—that is, the GP searches do not wrap another classification algorithm as an objective function. Nonetheless, by designing appropriate fitness functions, GP solutions can satisfy objectives in different aspects of feature manipulation. In almost all the aspects of feature manipulation, two outstanding attributes of GP have made it superior to many traditional methods in the field: i) being able to handle multiple features at the same time; ii) not being bound to a certain form or template for building functions.

8.2.1 GP for Feature Construction

Chapter 3 and Chapter 4 propose GP-based algorithms to transform the input space of classification tasks. Based on the observations in these two chapters, the following conclusions are drawn.

8.2.1.1 Improvements to the Classification Performance

In classifiers that are inherently incapable of transforming the input space effectively (like decision trees), using GP to construct high-level features can be very useful for improving the classification performance. Our results on several benchmark datasets show that in terms of classification accuracy, using GP-constructed features with the decisions tree classifier outperforms the standard decision tree approach (in which only the original features are used).

8.2.1.2 The Richness of GP-Constructed Features and Transformational Dimensionality Reduction

Observations on several benchmark classification datasets show that, in most cases, using only features constructed by the proposed GP system is enough to improve the classification performance. Even compared to augmented datasets that contain both the original and constructed features,

the performance results obtained by using only constructed features are still quite good. This suggests that the GP-constructed features are very rich in terms of content and in practice can encapsulate the information of several original features. The thesis uses this characteristic to achieve significant dimensionality reduction in several benchmark problems.

8.2.1.3 Reduction in the Complexity of Classification Models

Due to the nature of the fitness function proposed for feature construction, programs with high fitness can transform the input space in a way that instances of a certain class gather together and form an easily distinguishable band (interval) on the axis of a constructed feature. Classification models using such constructed features can be much simpler than those using the original features. Our observations on several benchmark problems show that decision tree classifiers learned using only GP-constructed features are much smaller (in terms of number nodes) and yield better performance compared to decision trees using the original features. Classifiers with lower complexity are better at generalisation, faster in execution, and easier to interpret. However, the constructed features themselves might be difficult to interpret meaningfully.

8.2.1.4 The Effect of Enrichment of Genetic Material on the Quality of Solutions

When the search space of possible transformations (constructed features) is huge—for example, when there are a large number of input features in the problem—enrichment of genetic material is very helpful in increasing the probability of finding good solutions. In GP, for dimensionality reduction, the genetic material was enriched by adding potentially useful linear transformations to the terminal set of GP. Using this technique, although the search space was huge, the system could successfully find the desired transformations. Compared to GP results without enrichment

or compared to similar linear transformations (like PCA), the proposed method produces better results.

8.2.2 GP for Feature Ranking

Chapter 5 and Chapter 6 propose GP-based algorithms to rank the input features of classification tasks. Based on the observations in these two chapters, the following conclusions are drawn.

8.2.2.1 Dimensionality Reduction via Using a few High-Rank Features

Using the output of our proposed GP-based single feature ranking system, we found out that a variety of classifiers learned using just a few high-rank features work well. In most cases, by using less than 10% of the high-rank features, we could gain the same classification performance as that gained when all the available original features are used. This suggests that the ranking provided by the proposed GP system can successfully rank the important features of a classification problem as high.

8.2.2.2 Deterioration in Classification Performance due to Excessive Number of Low-Ranked Features

If we feed the highest-ranked feature to a classifier and then continue adding features at the next ranks, the classification performance starts rising. In most problems and for most classifiers, the rise is very rapid at the start, then becomes gradual and stops, and then the performance starts deteriorating. This implies that using an excessive number of features might cause deterioration in classification performance. On the other hand, it confirms the proposed GP feature ranking system has ranked irrelevant and noisy features—those that can cause deterioration in classification performance—as low.

8.2.2.3 Finding the Complicated Relationships between Groups of Input Features and Target Concepts

The proposed GP-based subset ranking system has the capability of finding hidden relationships between a subset of features and target class variables. It is found that the system can recognise relevant features in situations where many other methods—like those coming from information theory and linear correlation—cannot. We particularly realised that the proposed system is good at handling situations where the class variable has a multimodal distribution or the features are mutually correlated.

8.2.2.4 High Positive Linear Correlation between Provided Ranking and Actual Classification Performance

The proposed GP-based subset ranking system can quantify the importance of subsets of features without wrapping any classification algorithm. The classification performance obtained by using subsets of features is highly correlated to the importance quotient measured by the system. This suggests that the proposed system can measure the relevance of subsets of features to the classification task.

8.2.3 GP for Feature Selection

Chapter 6 and Chapter 7 propose GP-based algorithms for feature selection including multi-objective subset ranking/selection and redundancy removal. Based on the observations in these two chapters the following conclusions are drawn.

8.2.3.1 Finding the Best Subset of Features through High-Rank Subsets

Assuming that the best subset of features is the one that produces the highest classification performance for a certain classifier, in most cases, it can be found within the first percentile of high-rank subsets of features ranked

by the proposed GP-based subset ranking system. In all the observations, the number of features in the best subsets of features has been far less than the total number of features in the problem and the classification performance has been higher. This suggests that the proposed GP system is very effective at finding optimal subsets of features.

8.2.3.2 Finding Optimal Subsets of Features on a Pareto Front

The proposed GP subset ranking and selection system takes two objectives into account: maximising the relevance of the subsets and minimising their cardinality. The system provides a trade-off curve in the form of a Pareto front that contains points (subsets) producing the highest relevance for each given cardinality. It is found that an inexpensive search over the Pareto front vector can improve the classification performance and reduce the complexity of classifiers.

8.2.3.3 Detecting Non-Linear and Multivariate Dependencies

The proposed GP-based redundancy measure can detect redundancy within a group of input features. The system is superior to traditional redundancy measures in two ways: i) the system can detect redundancy in situations where traditional statistical correlation checking methods cannot easily do due to complicated relationships between input features; ii) the system can measure the redundancy between a feature and a group of features.

8.2.3.4 Improving Classification Performance through Removing Redundant Features

Feature Selection algorithms that merely rely on finding features that are relevant to the classification task could severely suffer from the redundancy issue. Features at high ranks, although highly related to the target

concepts, might be redundant with respect to each other. Using our proposed unsupervised GP-based redundancy measure and a forward selection algorithm, one can remove redundant features from a classification problem. Our results show that removing redundant features can significantly improve classification performance.

8.3 Impact and Utilisation of Findings

This section discusses how the advances made by the thesis in the area of evolutionary feature manipulation contributes to related fields such as Knowledge Discovery in Databases (KDD), Data Mining (DM) and Machine Learning (ML), and how the results can be further utilised. It is discussed in what situations the proposed algorithms can be used and how they can improve the existing situations. The clear illustration of the algorithms and the related concepts proposed in the thesis makes it feasible to implement them in existing DM software packages such as Weka[1].

8.3.1 Improving the Performance of Symbolic Learners in Numeric Domains

Scenario and existing issues: A *symbolic learning* algorithm induces a set of rules that describes the relationship between input features and decision variables (class labels). The majority of symbolic learning algorithms work with categorical variables. When applied to a problem with numeric features, the numeric features must first be discretised. Applying the discretisation to numeric features generates a set of split points. The split points partition the numerical input space into several rectangular regions. The number of split points determines the granularity of the partitions. There is a trade-off between granularity on one hand and model simplicity and generalisation capability on the other hand. If the granularity increases,

[1] http://www.cs.waikato.ac.nz/ml/weka/

so does the model complexity and the chance of overfitting. Although, in theory, one could achieve 100% classification accuracy on training data by increasing the granularity, simpler (and thus more intelligible) models with high generalisation capability are more desirable.

Proposed Resolution: The proposed algorithm in Chapter 3 can construct multiple features in the form of functions of the original features. The algorithm searches for features (functions) that can capture the target concepts using only two split points (an interval). The constructed features often reduce the size of the induced decision trees and improve their generalisation capability and classification performance. Unlike existing GP-based algorithms for feature construction, the proposed algorithm is capable of constructing multiple features using a filter approach. The algorithm can be applied to numeric domains and can also be extended to use the information of categorical variables in the construction of features.

8.3.2 More Promising Transformational Dimensionality Reduction

Scenario and existing issues: *Transformational Dimensionality Reduction* (TDR) is usually used when a mere feature selection (choosing a small set of original features) does not provide enough information to a classification algorithm to perform at a desired performance level. The aim of using TDR is to construct a few high-level features which provide maximum information. The space of possible transformations (functions) grows exponentially with respect to the number of original features in a classification problem. This usually forces users to use simple (e.g. linear) and fixed function forms (e.g. PCA).

Proposed Resolution: A heuristic algorithm is introduced to enrich the *variable terminal set* of GP by performing class-wise orthogonal transforma-

tions. The transformed variable terminals are more likely to create a good separation between classes (see Chapter 4). The enriched variable terminal set helps GP search the immense space of possible transformations (functions) in high-dimensional classification problems. The proposed algorithm can only be applied to problems with numeric features. The results on benchmark problems show that it can be helpful in both reducing the dimensionality and reducing the average size of decision trees while preserving their performance.

8.3.3 Improved Feature Selection by Subset Ranking

Scenario and existing issues: A common approach to feature selection is through ranking the original features and then selecting a number of top-ranked features. Many of the commonly-used ranking algorithms like Information Gain, Logistic Regression and Pearson's Correlation examine the relationship between only one input feature and the class label, missing relationships (epistasis) between groups of features and a class label.

Proposed Resolution: A GP-based algorithm using a computationally cheap fitness function (acting as a surrogate classifier) is proposed that can find the relationship between a group of input features and the class label (See Chapter 5 and 6). The relationships discovered by this method are limited to linear (or any other predefined) relations. Theoretically, the proposed algorithm is capable of finding any type of relationship that can be expressed by the GP function set. In practice the proposed method has been tested and works well for problems up to 60 features. As the number of features increases, however, deeper program trees and larger population sizes are required which can drastically slow down the algorithm (See discussions in Chapter 6).

8.3.4 A New Way of Detecting and Removing Redundant Features

Scenario and existing issues: A consequence of using a ranking algorithm that considers the importance of features individually is that good but redundant features are ranked top and therefore redundant features are selected even though they do not provide additional information. It is desired to select a subset of features that does not contain any redundant features.

Proposed Resolution: An algorithm is proposed that can detect and measure the degrees of complex relationships between input features (See Chapter 7). Together with a forward selection algorithm the proposed solution can improve the selection process by removing the redundant features from a subset (ranking vector). An illustration of the application of the algorithm on a dataset with more than 600 features is given in Chapter 7. The proposed algorithm can only be applied to problems with numeric features.

8.4 Future Directions

This section provides some possible future directions in the three aspects of feature manipulation using GP.

8.4.1 Directions in Using GP for Feature Construction

8.4.1.1 Handling Nominal Features and Features with Missing Values

The GP-based systems proposed in this thesis are *mono-typed*—that is, all the nodes in a program tree share the same type which is floating-point numbers. The experiments are limited to datasets with only numeric features. As a future direction, one might consider extending these systems in

a way that they can handle categorical and logical data as well. This may require using *strongly-typed GP* which is computationally more expensive than mono-typed GP and requires the development of more efficient approaches. One may also consider using features with missing values. In this case, an appropriate action should be taken when the program encounters a missing value in one of its variable-terminals.

8.4.1.2 Cooperative Co-Evolutionary Multiple Feature Construction

The GP-based feature-construction system proposed in this thesis produces multiple high-level features by constructing one feature per class label in the problem. Another filter-based way of constructing multiple features might be through using concurrent populations and evolving features that cooperate with each other toward a shared objective.

8.4.1.3 Further Enrichment of Genetic Material

Enrichment of genetic material in a GP search seems to be a very promising way of increasing the chances of finding solutions and reducing the computational effort. In using GP for transformational dimensionality reduction, we successfully used some heuristics to enrich the genetic material. It also seems plausible to think that if some genetic materials have been useful in constructing a feature to separate instances of c_1 from c_2 and c_3, it might be helpful to separate instances of c_2 from c_1 and $c3$ as well. Therefore, another way of enriching genetic material in multiple feature construction might be through recycling genetic material from GP runs for constructing a feature for one class to other GP runs for constructing a feature for another class.

8.4.1.4 Testing the Proposed Algorithm on Other Classifiers

The proposed GP-based feature construction algorithm is ideally used with classification algorithms that are incapable of performing input space trans-

formations. In this thesis, the algorithm was tested with decision trees. As a future work, one might consider studying the effectiveness of the algorithm on other classifiers in this category like Bayesian classifiers.

8.4.2 Directions in Using GP for Feature Ranking

8.4.2.1 Making GP Capable of Using Very Large Numbers of Features in Program Trees

According to our experiments, there seem to be a serious limitation in the maximum number of features that can fit in a program tree that performs reasonably good. Chapter 6 proposed some solutions that could partially improve the condition. However, one might consider improving GP capability in using very large numbers of features by devising a new program representation.

8.4.3 Directions in Using GP for Feature Selection

8.4.3.1 Testing the Proposed GP-based Methods on Other Datasets

The experiments in Chapter 6 and Chapter 7 might be repeated on other classification problems, particularly those with a large number of features. Since the proposed GP-based redundancy measure is an unsupervised algorithm, one may also want to consider the algorithm to non-classification tasks (e.g. clustering problems).

8.4.3.2 A Complete GP Ranking and Redundancy Removal System

As future work, one might consider merging the proposed redundancy removal method into previous research on using GP for feature subset selection to build up a GP-based feature selection system which is capable of handling complicated relationships between features and target classes while preserving the minimality.

8.4.3.3 Using GP to Explore the Feature Lattice Directly

Another future direction might be using GP to directly explore the search space of subsets of features (the feature lattice). Such a GP system could be equipped with proper operators to move through the space of subsets of features. In this scenario, the output of a program tree would be a point a on the lattice.

8.4.3.4 Using the GP-Based Algorithms on Problems with Large Numbers of Features

Further developments in feature manipulation using GP could be achieved through adapting the proposed algorithms (and perhaps developing new ones) to work with dataset with very large numbers of features. This is particularly important when there is only a small number of instances in a dataset.

Appendix A

Benchmark Datasets

The following benchmark datasets have been used in the experiments throughout the thesis and are available at the UCI Machine Learning Repository [6].

Balance Scale This dataset is used to explain developmental differences in children's thinking focusing on their comprehension of balance scale problems [131]. In more advanced models, both the weights placed on each side of the falcrum and their distance from the falcrum are taken into account. The 4 features/attributes of the dataset are the right and left weights and the right and left distances. There are 625 instances which are classified as having the balance scale tip to the left (L), tip to the right (R) or stay balanced (B).

Glass Identification The glass identification dataset was developed to identify the glass left behind at a crime scene [29]. There are 9 features/attributes measured in terms of weight percent in the corresponding oxide, and a class label on each of the 214 instances. The glasses collected for this dataset are of 7 different types (class labels). Some of the attributes included are sodium, silicon, iron, aluminum, magnesium, and potassium.

JH Ionosphere The John Hopkins University Ionosphere dataset is used to distinguish between "good" and "bad" radar returns based on suitability for further analysis: good radar returns show evidence of some type of structure in the inosphere and bad radar returns do not [27]. The dataset contains 34 features. The number of instances (patterns) present in the dataset is 351. The instances are classified into one of two classes: "good" if suitable for further analysis and "bad" if not.

Iris Plant [33, 121] This is perhaps the best-known dataset in the pattern recognition literature. There are three classes in the problem where each class refers to a type of iris plant. The classes are Iris Setosa, Iris Versicolour, and Iris Virginica. There are 50 instances for each class. One class is linearly separable from the other 2; the latter are not linearly separable from each other. The features are Sepal length, Sepal width, Petal length, and Petal width all measured in centimetres. This is quite an easy problem.

Isolet5 This is a voice recognition dataset with 26 classes. The dataset has been created by recording the voice of 30 people pronouncing the names of the 26 English alphabets twice. There are 52 samples per person and 1559 samples in total (one sample is missing). The task is to classify the alphabets. There are 617 features available in total including spectral coefficients, contour features, sonorant features, pre-sonorant and post-sonorant features. All the features are real-valued, continuous and scaled into the range $[-1, 1]$ [6].

Liver Disorders This dataset prepared by BUPA Medical Research Ltd. includes results on 5 blood tests on male individuals to detect liver disorders that arise from excessive consumption of alcohol, and amount of alcoholic beverages drunk in a day. The blood tests include the mean corpuscular volume, alkaline phosphotase, alamine phosphotase, aspar-

tate aminotransferase, gamma-glutamyl transpeptidase. There are 345 instances and 2 class labels in the dataset [6].

Pima Diabetes This dataset includes diagnostic reports on 768 individuals (instances) from a population living near Phoenix, Arizona, and has been used to predict the onset of diabetes [135]. The 8 features in the dataset are number of times pregnant, plasma glucose concentration, blood pressure, triceps skin fold thickness, 2-Hour serum insulin, body mass index, diabetes pedigree function, and age. The instances are in one of two classes: diabetic and non-diabetic.

Sonar This dataset has been used to distinguish between sonar returns from an undersea metal cylinder (mine) or a similarly shaped rock on the ocean floor [40, 41]. The sonar returns, collected at a range of 10 meters, were obtained from various aspect angles. The dataset contains 111 patterns on the metal cylinder and 97 patterns on the rock, 208 instances in total, and 60 features (spectral samples) with normalized values in the interval [0.0,1.0]. The two class labels in the dataset are ”M” for mine and ”R” for rock.

Thyroid Disease The dataset has been used to build predictive models for thyroid disease diagnosis. The 215 instances in the dataset are classified into 3 classes: euthyroidism, hypothyroidism and hyperthyroidism. The 5 features in the dataset are total serum thyroxin, T3-resin uptake test, total serum triiodothyronine, basal thyroid-stimulating hormone (TSH), and maximal absolute difference between TSH after injecting thyrotropin-releasing hormone and basal TSH [6].

Waveform This is an artificial dataset, generated by the Waveform Dataset Generator [6], with 21 continuous attributes (features) with values in the interval [0.0,6.0], and 3 classes of waves, namely, waveform1, waveform2

and waveform3; each class is a wave generated from a combination of 2 of 3 base waves. The dataset is considered artificial as the data is deterministically generated based on some equation of time. A random sample of 500 instances is used in the experiments, where the instances are uniformly distributed over the three classes.

WBC-Original The dataset is used for distinguishing between benign and malignant breast tumors. In this dataset, there are 699 instances, 16 of which have missing values and are removed, leaving 683 instances. There are 9 attributes/features in the dataset which include clump thickness, uniformity of cell size, marginal adhesion, etc., which are transformed into categorical (ordinal, 1 - 10) features. Each instance is in one of the two classes in the dataset: benign (2) and malignant (4) [153].

WBC-Diagnostic The data is used to diagnose breast cancer based on images derived from a fine needle aspiration (FNA) biopsy of a breast mass [144]. A typical image derived from the FNA contains 10 - 40 cells. For each cell nucleus, 10 real-valued features are computed. The features describe characteristics of the nuclei of cells, such as radius, texture, symmetry, etc. Then, for each image, the mean, standard error, and maximum (worst) value of each feature over the cell nuclei in the image is computed resulting in 30 features. There are 569 instances (images) in the dataset. The two classes in the dataset are malignant (M) and benign (B).

Wine Recognition The dataset is the result of a chemical analysis of three types of wine in a region of Italy. The wines are derived from three different cultivars. The analysis determined the quantities of 13 constituents (features) found in each of the three types of wine. The features are the quantity of Alcohol, Malic acid, Ash, Alcalinity of Ash, Magnesium, Total phenols, Flavanoids, Nonflavanoid Phenols, Proanthocyanins, Color Intensity, Hue, OD280/OD315 of Diluted Wines, and Proline. This is a well-

posed problem with ”well-behaved” class structures, but perhaps not very challenging [6].

Bibliography

[1] AGNAR, A., AND ENRIC, P. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications 7*, 1 (1994), 39–59.

[2] AGRESTI, A., AND AGRESTI, A. *Categorical Data Analysis*. Wiley, 2003.

[3] ALMUALLIM, H., AND DIETTERICH, T. G. Efficient algorithms for identifying relevant features. In *PROCEEDINGS OF THE BIENNIAL CONFERENCE-CANADIAN SOCIETY FOR COMPUTATIONAL STUDIES OF INTELLIGENCE* (1992), p. 38.

[4] ALMUALLIM, H., AND DIETTERICH, T. G. Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence 69*, 1-2 (Sept. 1994), 279–305.

[5] ALPAYDIN, E. *Introduction to Machine Learning*. MIT Press, Oct. 2004.

[6] ASUNCION, A., AND NEWMAN, D. UCI machine learning repository. http://archive.ics.uci.edu/ml/index.html, 2007. last accessed 2010.

[7] BANZHAF, W., AND WOLFGANG. *Genetic Programming*. Morgan Kaufmann, 1998.

[8] BATESON, W. Mendel's principles of heredity. *Molecular and General Genetics MGG 3*, 1 (1910), 108–109.

[9] BELKIN, M., AND NIYOGI, P. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation 15*, 6 (2003), 1373–1396.

[10] BHANU, B., AND KRAWIEC, K. Coevolutionary construction of features for transformation of representation in machine learning. In *Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference (GECCO 2002), AM Barry (ed.), AAAI Press: New York* (2002), pp. 249–254.

[11] BHANU, B., LIN, Y., AND KRAWIEC, K. *Evolutionary Synthesis of Pattern Recognition Systems*, 1 ed. Monographs in Computer Science. Springer, 2005.

[12] BIESIADA, J., DUCH, W., KACHEL, A., MACZKA, K., AND PALUCHA, S. Feature ranking methods based on information entropy with parzen windows. In *Proceedings of the 9th International Conference on Research in Electrotechnology and Applied Informatics (REI'05)* (Katowice, Poland., 2005), pp. 109–119.

[13] BISHOP, C. *Neural networks for pattern recognition*. Oxford University Press, USA, 1995.

[14] BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, Aug. 2006. Published: Hardcover.

[15] BONABEAU, E., DORIGO, M., AND THERAULAZ, G. *From Natural to Artificial Swarm Intelligence*. Oxford University Press, 1999.

[16] CARDIE, C. Using decision trees to improve case-based learning. In *Proceedings of the Tenth International Conference on Machine Learning* (1993), vol. 25, p. 32.

[17] CARREIRA-PERPINAN, M. A. A review of dimension reduction techniques. *University of Sheffield, Sheffield, UK, Tech. Rep. CS-96-09* (1997).

[18] CHENG, Q., VARSHNEY, P., AND ARORA, M. Logistic regression for feature selection and soft classification of remote sensing data. *Geoscience and Remote Sensing Letters, IEEE 3*, 4 (2006), 491–494.

[19] CRAMER, N. L. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the First International Conference on Genetic Algorithms* (1985).

[20] DASH, M., AND LIU, H. Feature selection for classification. *Intelligent data analysis 1*, 3 (1997), 131—156.

[21] DAVIS, L. Adapting operator probabilities in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms* (George Mason University, United States, 1989), Morgan Kaufmann Publishers Inc., pp. 61–69.

[22] DAVIS, R. A., CHARLTON, A. J., OEHLSCHLAGER, S., AND WILSON, J. C. Novel feature selection method for genetic programming using metabolomic 1H NMR data. *Chemometrics and Intelligent Laboratory Systems 81*, 1 (Mar. 2006), 50–59.

[23] DEJONG, K. A., AND JONG, K. A. D. *Evolutionary Computation*. The MIT Press, Mar. 2002.

[24] DORIGO, M., AND GAMBARDELLA, L. M. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation 1*, 1 (1997), 53—66.

[25] DUDA, R. *Pattern classification*, 2nd ed. Wiley, New York, 2001.

[26] EKART, A., AND MARKUS, A. Using genetic programming and decision trees for generating structural descriptions of four bar mechanisms. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 17*, 3 (2003), 205220. UK Cambridge University Press.

[27] EMMANOUILIDIS, C., HUNTER, A., AND MACINTYRE, J. A multiobjective evolutionary setting for feature selection and a commonality-based crossover operator. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)* (La Jolla, CA, USA, 2000), pp. 309–316.

[28] ERDOGMUS, D. *Information theoretic learning: Renyis entropy and its applications to adaptive system training*. PhD thesis, University of Florida, 2002.

[29] EVETT, I. W., AND SPIEHLER, E. J. Rule induction in forensic science. In *Knowledge Based Systems*. Halsted Press, 1988, pp. 152–160.

[30] FIRPI, H., GOODMAN, E., AND ECHAUZ, J. Genetic programming artificial features with applications to epileptic seizure prediction. IEEE.

[31] FIRPI, H., GOODMAN, E., AND ECHAUZ, J. On prediction of epileptic seizures by computing multiple genetic programming artificial features. In *8th European Conference on Genetic Programming* (Lausanne, Switzerland, 2005), Spring-Verlag, pp. 321–330.

[32] FIRPI, H., GOODMAN, E., AND ECHAUZ, J. On prediction of epileptic seizures by means of genetic programming artificial features. *Annals of Biomedical Engineering 34*, 3 (2006), 515–529.

[33] FISHER, R. The use of multiple measurements in taxonomic problems. *Annals Eugen. 7* (1936), 179–188.

[34] FOGEL, L. J., OWENS, A. J., AND WALSH, M. J. Intelligent decision-making through a simulation of evolution. *IEEE Transactions on Human Factors Electronics, 6* (1965), 13—23.

[35] FROHLICH, H., CHAPELLE, O., AND SCHOLKOPF, B. Feature selection for support vector machines by means of genetic algorithm. In

15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03) (2003), pp. 142—148.

[36] FUJIKO, C., AND DICKINSON, J. Using the genetic algorithm to generate LISP source code to solve the prisoner's dilemma. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application table of contents* (1987), pp. 236–240.

[37] FUKUNAGA, K. *Introduction to statistical pattern recognition (2nd edition)*. Academic Press, 1990.

[38] GOLDBERG, D. E. *Genetic algorithms in search, optimization, and machine learning*. Addison Wesley, 1989.

[39] GOLDBERG, D. E., AND HOLLAND, J. H. Genetic algorithms and machine learning. *Mach. Learn. 3*, 2-3 (1988), 95–99.

[40] GORMAN, R. P., AND SEJNOWSKI, T. J. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks 1*, 1 (1988), 75—89.

[41] GORMAN, R. P., AND SEJNOWSKI, T. J. Learned classification of sonar targets using a massively parallel network. *IEEE Transactions on Acoustics, Speech and Signal Processing 36*, 7 (July 1988), 1135–1140.

[42] GRAY, H. F., MAXWELL, R. J., MARTNEZ-PREZ, I., ARS, C., AND CERDN, S. Genetic programming for classification and feature selection: analysis of 1H nuclear magnetic resonance spectra from human brain tumour biopsies. *NMR in Biomedicine 11*, 4-5 (1998), 217–224.

[43] GUO, H., JACK, L. B., AND NANDI, A. K. Feature generation using genetic programming with application to fault classification. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics) 35*, 1 (2005), 89–99.

[44] GUO, H., AND NANDI, A. K. Breast cancer diagnosis using genetic programming generated feature. *Pattern Recognition 39*, 5 (May 2006), 980–987.

[45] GUYON, I., AND ELISSEEFF, A. An introduction to variable and feature selection. *The Journal of Machine Learning Research 3* (2003), 1157–1182.

[46] GUYON, I., WESTON, J., BARNHILL, S., AND VAPNIK, V. Gene selection for cancer classification using support vector machines. *Machine learning 46*, 1 (2002), 389–422.

[47] HALL, M. A. Correlation-based feature selection for discrete and numeric class machine learning. In *Proc. 17th International Conf. on Machine Learning* (2000), Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, pp. 359–366.

[48] HECKERMAN, D., GEIGER, D., AND CHICKERING, D. M. Learning bayesian networks: The combination of knowledge and statistical data. *Mach. Learn. 20*, 3 (1995), 197–243.

[49] HOLLAND, J. H. *Adaptation in natural and artificial systems*. MIT Press, 1992.

[50] HONG, G., JACK, L. B., AND NANDI, A. K. Automated feature extraction using genetic programming for bearing condition monitoring. In *Proceedings of the 2004 14th IEEE Signal Processing Society Workshop* (Sao Luis, Brazil, 2004), A. Barros, J. Principe, J. Larsen, T. Adali, and S. Douglas, Eds., IEEE, pp. 519–528.

[51] HU, Y. J. Constructive induction: Covering attribute spectrum. *Kluwer International Series In Engineering And Computer Science* (1998), 257–272.

[52] HUELSBERGEN, L. Toward simulated evolution of machine-language iteration. In *Proceedings of the First Annual Conference on Genetic Programming* (1996), pp. 315–320.

[53] JASKOWSKI, W., KRAWIEC, K., AND WIELOCH, B. Learning and recognition of Hand-Drawn shapes using generative genetic programming. In *Applications of Evolutionary Computing*, vol. 4448 of *Lecture Notes in Computer Science*. Springer, 2007, pp. 281–290.

[54] JENSEN, F. V. *Introduction to Bayesian Networks*. Springer-Verlag New York, Inc., 1996.

[55] JI, C., AND MA, S. Combinations of weak classifiers. *IEEE Transactions on Neural Networks 8*, 1 (1997).

[56] JIANGANG, Y., AND BHANU, B. Evolutionary feature synthesis for facial expression recognition. *Pattern Recognition Letters 27*, 11 (2006), 1289–1298. 8980106 Netherlands Elsevier 21.

[57] JOHN, G., AND LANGLEY, P. Estimating continuous distributions in bayesian classifiers. *In Proceedings Of The Eleventh Conference On Uncertainty In Artificial Intelligence* (1995), 338—345.

[58] JOHNSON, R. A., AND WICHERN, D. W. *Applied Multivariate Statistical Analysis*, 5 ed. Prentice Hall, 2002.

[59] JOLLIFFE, I. T. *Principal Component Analysis*, 2 ed. Springer, 2002.

[60] JONG, K., MARY, J., CORNUJOLS, A., MARCHIORI, E., AND SEBAG, M. Ensemble feature ranking. In *Principles and Practice of Knowledge Discovery in Databases (PKDD)* (2004), vol. 3202 of *Lecture Notes In Computer Science*, pp. 267–278.

[61] KEERTHI, S. S., SHEVADE, S. K., BHATTACHARYYA, C., AND MURTHY, K. R. K. Improvements to platt's SMO algorithm for SVM classifier design. *Neural Comp. 13*, 3 (Mar. 2001), 637–649.

[62] KENNEDY, J., EBERHART, R. C., AND SHI, Y. *Swarm intelligence*. Morgan Kaufmann, Mar. 2001.

[63] KIRA, K., AND RENDELL, L. A. The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the National Conference on Artificial Intelligence* (1992), John Wiley and Sons, pp. 129–129.

[64] KIRA, K., AND RENDELL, L. A. A practical approach to feature selection. In *Proceedings of the ninth international workshop on Machine learning table of contents* (1992), pp. 249–256.

[65] KISHORE, J. K., PATNAIK, L. M., MANI, V., AND AGRAWAL, V. K. Application of genetic programming for multicategory pattern classification. *IEEE Transactions on Evolutionary Computation 4*, 3 (2000), 242–258.

[66] KOHAVI, R., AND JOHN, G. Wrappers for feature subset selection. *Artificial Intelligence 97* (1997), 273–324.

[67] KOLLER, D., AND SAHAMI, M. Toward optimal feature selection. In *Machine Learning-International Workshop And Conference* (1996), Morgan Kaufmann, pp. 284–292.

[68] KONONENKO, I. Estimating attributes: Analysis and extensions of RELIEF. *Lecture Notes in Computer Science* (1994), 171.

[69] KORNS, M. F. Large-Scale, Time-Constrained symbolic Regression-Classification. In *Genetic Programming Theory and Practice V*. Springer US, 2008, pp. 53–68.

[70] KOZA, J. R. Hierarchical genetic algorithms operating on populations of computer programs. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence* (1989), vol. 1, pp. 768–774.

[71] KOZA, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[72] KOZA, J. R. *Non-linear genetic algorithms for solving problems by finding a fit composition of functions*. Google Patents, Aug. 1992. US Patent 5,136,686.

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

[74] KOZA, J. R., KEANE, M. A., AND BENNETT, F. H. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, 1999.

[75] KOZA, M. A. K. J. R., AND LANZA, G. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publisher, 2003.

[76] KRAWIEC, K. Genetic programming-based construction of features for machine learning and knowledge discovery tasks. *Genetic Programming and Evolvable Machines 3*, 4 (2002), 329–343.

[77] KRAWIEC, K. *Evolutionary Feature Programming: Cooperative learning for knowledge discovery and computer vision*. Wydawnictwo Politechniki Poznanskiej, Poznan University of Technology, Poznan, Poland, 2004.

[78] KRAWIEC, K. Evolutionary learning of primitive-based visual concepts. In *IEEE Congress on Evolutionary Computation* (2007), pp. 1308–1315.

[79] KRAWIEC, K. Generative learning of visual concepts using multiobjective genetic programming. *Pattern Recognition Letters 28*, 16 (2007), 2385–2400. Netherlands.

[80] KRAWIEC, K., AND BHANU, B. Visual learning by coevolutionary feature synthesis. *IEEE Transactions on System, Man, and Cybernetics Part B 35*, 3 (June 2005), 409–425.

[81] KRAWIEC, K., AND BHANU, B. Visual learning by evolutionary and coevolutionary feature synthesis. *IEEE Transactions on Evolutionary Computation 11*, 5 (2007), 635–650. USA.

[82] KRAWIEC, K., AND LIJEWSKI, P. Genetic graph programming for object detection. In *Lecture Notes in Computer Science*, L. Rutkowski, R. Tadeusiewicz, L. A. Zadeh, and J. Zurada, Eds., vol. 4029 of *LNCS*. Springer, 2006, pp. 804–813.

[83] LANDRY, J. A., COSTA, L. D., AND BERNIER, T. Discriminant feature selection by genetic programming: Towards a domain independent multi-class object detection system. *Systemics, Cybernetics and Informatics 3*, 1 (2006), 76–81.

[84] LANGDON, W. B., AND POLI, R. *Foundations of Genetic Programming*. Springer, Mar. 2002.

[85] LANGDON, W. B., AND QURESHI, A. Genetic programming: computers using natural selection to generate programs.

[86] LAST, M., K, A., AND MAIMON, O. Information-theoretic algorithm for feature selection. *Pattern Recognition Letters 22* (2001), 799—811.

[87] LIN, J., KE, H., CHIEN, B., AND YANG, W. Classifier design with feature selection and feature extraction using layered genetic programming. *Expert Syst. Appl. 34*, 2 (2008), 1384–1393.

[88] LIN, T., CHIU, S., AND TSAI, K. Supervised feature ranking using a genetic algorithm optimized artificial neural network. *Journal of Chemical Information and Modeling 46*, 4 (July 2006), 1604–1614.

[89] LIU, H., AND MOTODA, H. *Feature Extraction, Construction and Selection*. Kluwer Academic Publishers, 1998.

[90] LOWRY, R. *Concepts and Applications of Inferential Statistics*. VassarStat, 2008.

[91] MAATEN, L. J. P. V. D., POSTMA, E. O., AND HERIK, H. J. V. D. Dimensionality reduction: A comparative review. *Preprint* (2007).

[92] MACKAY, D. J. C. *Information theory, inference, and learning algorithms*. Cambridge University Press, Oct. 2003.

[93] MCLACHLAN, G. *Discriminant analysis and statistical pattern recognition*. Wiley-Interscience, Hoboken N.J., 2004.

[94] MICHALSKI, R. S., AND TECUCI, G. *Machine learning: A Multistrategy Approach*. Morgan Kaufmann, 1994.

[95] MICHIE, D., AND SPIEGELHALTER, D. J. *Machine learning, neural and statistical classification*. Ellis Horwood, 1994.

[96] MILLER, A. J. *Subset selection in regression*. 2002.

[97] MILLER, J. F. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Orlando, Florida, USA, 1999), W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., vol. 2, Morgan Kaufmann, pp. 1135–1142.

[98] MILLER, J. F., AND HARDING, S. L. Cartesian genetic programming. In *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation* (2008), pp. 2701–2726.

[99] MILLER, J. F., AND THOMSON, P. Cartesian genetic programming. In *Cartesian Genetic Programming*, vol. 1802 of *Lecture Notes in Computer Science*. Springer, 2000, pp. 121–132.

[100] MITCHELL, T. *Machine Learning*. McGraw-Hill, 1997.

[101] MONTANA, D. J. Strongly typed genetic programming. *Evolutionary computation 3*, 2 (1995), 199230.

[102] MUHARRAM, M., AND SMITH, G. D. Evolutionary constructive induction. *IEEE Transactions on Knowledge and Data Engineering 17*, 11 (2005), 1518–1528.

[103] MUHARRAM, M., AND SMITH, G. D. Evolutionary constructive induction. *IEEE Transactions on Knowledge and Data Engineering 17*, 11 (2005), 1518–1528.

[104] MUHARRAM, M. A., AND SMITH, G. D. Evolutionary feature construction using information gain and gini index. In *Lecture Notes in Computer Science*, M. Keijzer, U. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, Eds., vol. 3003 of *LNCS*. Springer-Verlag, Coimbra, Portugal, 2004, pp. 379–388.

[105] MUNI, D. P., PAL, N. R., AND DAS, J. Genetic programming for simultaneous feature selection and classifier design. *IEEE Transactions on Systems, Man and Cybernetics, Part B 36*, 1 (Feb. 2006), 106–117.

[106] NANDI, R. J., NANDI, A. K., RANGAYYAN, R., AND SCUTT, D. Genetic programming and feature selection for classification of breast masses in mammograms. *Conference Proceedings: Annual International Conference of the IEEE Engineering in Medicine and Biology Society. 1* (2006), 3021–3024.

[107] NARENDRA, P. M., AND FUKUNAGA, K. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers 100*, 26 (1977), 917–922.

[108] NESHATIAN, K., AND ZHANG, M. Genetic programming and Class-Wise orthogonal transformation for dimension reduction in classifi-

cation problems. In *Genetic Programming*, vol. 4971 of *Lecture Notes in Computer Science*. Springer, Napoli, Italy, 2008, pp. 242—253.

[109] NESHATIAN, K., AND ZHANG, M. Genetic programming for performance improvement and dimensionality reduction of classification problems. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on* (2008), IEEE, pp. 2811–2818.

[110] NESHATIAN, K., AND ZHANG, M. Genetic programming for feature subset ranking in binary classification problems. In *Genetic Programming*, vol. 5481 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, Tbingen, Germany, 2009, pp. 121–132.

[111] NESHATIAN, K., AND ZHANG, M. Pareto front feature selection: using genetic programming to explore feature space. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation* (Montreal, Qubec, Canada, 2009), ACM, pp. 1027–1034.

[112] NESHATIAN, K., AND ZHANG, M. Unsupervised elimination of redundant features using genetic programming. In *Proceedings of the 22nd Australasian Joint Conference on Advances in Artificial Intelligence*. Springer-Verlag, Melbourne, Australia, 2009, pp. 432–442.

[113] NESHATIAN, K., ZHANG, M., AND ANDREAE, P. Genetic programming for feature ranking in classification problems. In *Simulated Evolution and Learning*, vol. 5361 of *Lecture Notes in Computer Science*. Springer, Melbourne, Australia, 2008, pp. 544—554.

[114] NESHATIAN, K., ZHANG, M., AND JOHNSTON, M. Feature construction and dimension reduction using genetic programming. In *AI 2007: Advances in Artificial Intelligence*, vol. 4830 of *Lecture Notes in Computer Science*. Springer, Gold Coast, Australia, 2007, pp. 160—170.

[115] NORDIN, P. A compiling genetic programming system that directly manipulates the machine code. *Advances in genetic programming* (1994), 311–331.

[116] NORDIN, P., AND BANZHAF, W. Evolving turing-complete programs for a register machine with self-modifying code. In *Genetic algorithms: proceedings of the sixth international conference (ICGA95)* (1995), pp. 318–325.

[117] OH, I. S., LEE, J. S., AND MOON, B. R. Hybrid genetic algorithms for feature selection. *IEEE Transactions on Pattern Analysis and Machine Intellignece* (2004), 1424–1437.

[118] OK, S., MIYASHITA, K., AND NISHIHARA, S. Improving performance of GP by adaptive terminal selection. In *PRICAI 2000 Topics in Artificial Intelligence*, vol. 1886/2000 of *Lecture Notes in Computer Science*. Springer, 2000, pp. 435–445.

[119] O'NEILL, M., AND RYAN, C. *Grammatical evolution*. Springer, May 2003.

[120] OTERO, F., SILVA, M., FREITAS, A., AND NIEVOLA, J. Genetic programming for attribute construction in data mining. In *Genetic Programming*, vol. 2610 of *Lecture Notes in Computer Science*. Springer, 2003, pp. 101–121.

[121] PARROTT, D., LI, X., AND CIESIELSKI, V. Multi-objective techniques in genetic programming for evolving classifiers. In *The 2005 IEEE Congress on Evolutionary Computation* (2005), vol. 2, pp. 1141–1148.

[122] PATTERSON, G., AND ZHANG, M. Fitness functions in genetic programming for classification with unbalanced data. In *AI 2007: Advances in Artificial Intelligence*. Springer, 2007, pp. 769–775.

[123] PEARL, J. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, Mar. 2000.

[124] POLI, R., LANGDON, W. B., AND MCPHEE, N. F. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, Mar. 2008.

[125] PRINCIPE, J. C., XU, D., AND FISHER, J. Information theoretic learning. *Unsupervised adaptive filtering* (2000), 265–319.

[126] QUINLAN, J. R. Induction of decision trees. *Machine Learning 1* (1986), 81—106.

[127] QUINLAN, J. R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[128] RITCHIE, M. D., HAHN, L. W., ROODI, N., BAILEY, L. R., DUPONT, W. D., PARL, F. F., AND MOORE, J. H. Multifactor-dimensionality reduction reveals high-order interactions among estrogen-metabolism genes in sporadic breast cancer. *The American Journal of Human Genetics 69*, 1 (2001), 138–147.

[129] RUIZ, R., RIQUELME, J. C., AND AGUILAR-RUIZ, J. S. Fast feature ranking algorithm. *Knowledge-Based Intelligent Information and Engineering Systems* (2003), 325–331.

[130] RUSSELL, S. J., AND NORVIG, P. *Artificial Intelligence: A Modern Approach (Second Edition)*. Pearson Education, 2003.

[131] SIEGLER, R. S. Three aspects of cognitive development. *Cognitive Psychology 8*, 4 (Oct. 1976), 481–520.

[132] SILVA, S., AND COSTA, E. Dynamic limits for bloat control: Variations on size and depth. In *Genetic and Evolutionary Computation – GECCO-2004, Part II*, K. Deb, R. Poli, W. Banzhaf, H. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. Tyrrell, Eds., vol. 3103 of *Lecture Notes in Computer Science*. Springer-Verlag, Seattle, WA, USA, 2004, pp. 666–677.

[133] SMART, O., FIRPI, H., AND VACHTSEVANOS, G. Genetic programming of conventional features to detect seizure precursors. *Engineering Applications of Artificial Intelligence 20*, 8 (2007), 1070–1085.

[134] SMART, W., AND ZHANG, M. Using genetic programming for multiclass classification by simultaneously solving component binary classification problems. *Lecture Notes in Computer Science, 3447* (2005), 227—239.

[135] SMITH, J. W., EVERHART, J., DICKSON, W., KNOWLER, W., AND JOHANNES, R. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. *Proceedings of the Annual Symposium on Computer Application in Medical Care* (Nov. 1988), 261–265.

[136] SMITH, M., AND BULL, L. Feature construction and selection using genetic programming and a genetic algorithm. In *Genetic Programming*, vol. 2610 of *Lecture Notes in Computer Science*. Springer, 2003, pp. 93–100.

[137] SMITH, M. G., AND BULL, L. Genetic programming with a genetic algorithm for feature construction and selection. *Genetic Programming and Evolvable Machines 6*, 3 (2005), 265–281.

[138] SMITS, G., KORDON, A., VLADISLAVLEVA, K., JORDAAN, E., AND KOTANCHEK, M. Variable selection in industrial datasets using pareto genetic programming. In *Genetic Programming Theory and Practice III*, T. Yu, R. L. Riolo, and B. Worzel, Eds., vol. 9 of *Genetic Programming*. Springer, Ann Arbor, 2005, pp. 79–92.

[139] SMITS, G., KORDON, A., VLADISLAVLEVA, K., JORDAAN, E., AND KOTANCHEK, M. Variable selection in industrial datasets using pareto genetic programming. *Genetic Programming Theory and Practice III 9* (2006), 79–92.

[140] SMITS, G., AND KOTANCHEK, M. Pareto-Front exploitation in symbolic regression. In *Genetic Programming Theory and Practice II*, U. O'Reilly, T. Yu, R. L. Riolo, and B. Worzel, Eds. Springer, Ann Arbor, 2004, pp. 283–299.

[141] SPEARS, W., JONG, K. D., BCK, T., FOGEL, D., AND DE GARIS, H. An overview of evolutionary computation. In *Proceedings of the European Conference on Machine Learning* (1993), Springer-Verlag, pp. 442–459.

[142] SRINIVASAN, A., AND KING, R. D. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. *Data Mining and Knowledge Discovery 3*, 1 (1999), 37–57.

[143] STOPPIGLIA, H., DREYFUS, G., DUBOIS, R., AND OUSSAR, Y. Ranking a random feature for variable and feature selection. *The Journal of Machine Learning Research 3* (2003), 1399–1414.

[144] STREET, N., WOLBERG, W. H., AND MANGASARIAN, O. L. Nuclear feature extraction for breast tumor diagnosis. *Electronic Imaging Sience and Technology 1905* (1993), 861—870.

[145] TACKETT, W. A. *Recombination, selection, and the genetic construction of computer programs*. PhD thesis, University of Southern California, 1994.

[146] TALAVERA, L. An evaluation of filter and wrapper methods for feature selection in categorical clustering. In *Advances in Intelligent Data Analysis VI*. Springer, 2005, pp. 440–451.

[147] TENENBAUM, J. B., DE SILVA, V., AND LANGFORD, J. A global geometric framework for nonlinear dimensionality reduction. *Science 290*, 5500 (2000), 2319 – 2323.

[148] VAPNIK, V. N. *The nature of statistical learning theory*. Springer, 2000.

[149] VENKATRAMAN, V., DALBY, A. R., AND YANG, Z. R. Evaluation of mutual information and genetic programming for feature selection in QSAR. *Journal of Chemical Information and Computer Sciences 44*, 5 (2004), 1686–1692.

[150] WEISE, T., NIEMCZYK, S., SKUBCH, H., REICHLE, R., AND GEIHS, K. A tunable model for multi-objective, epistatic, rugged, and neutral fitness landscapes. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation* (2008), ACM, pp. 795–802.

[151] WHIGHAM, P. A. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications* (1995), vol. 16, pp. 33—41.

[152] WITTEN, I. H., AND FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition)*. Morgan Kaufmann, 2005.

[153] WOLBERG, W. H., AND MANGASARIAN, O. L. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences of the United States of America 87*, 23 (Dec. 1990), 9193–9196.

[154] YANG, J., AND HONAVAR, V. Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems 13* (1998), 380—385.

[155] YU, L., AND LIU, H. Efficient feature selection via analysis of relevance and redundancy. *Journal of Machine Learning Research 5* (2004), 1205–1224.

[156] ZHANG, M., AND SMART, W. Multiclass object classification using genetic programming. *Lecture notes in computer science 3005* (2004), 369–378.

[157] ZHANG, M., AND WONG, P. Genetic programming for medical classification: a program simplification approach. *Genetic Programming and Evolvable Machines 9*, 3 (2008), 229–255.

[158] ZHENG, Z. A comparison of constructive induction with different types of new attribute. Technical Report TR C96/8, Deakin University, Australia, 1996.

[159] ZHENG, Z. *Constructing new attributes for decision tree learning*. PhD thesis, Basser Department of Computer Science, The University of Sydney, March 1996.

[160] ZHENG, Z., SRIHARI, R., AND SRIHARI, S. A feature selection framework for text filtering. In *Proceedings of the Third IEEE International Conference on Data Mining* (2003), IEEE Computer Society Washington, DC, USA.