Genetic Programming for Binary Classification with High-dimensional Unbalanced Data

by

Wenbin Pei

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
2021

Abstract

Class imbalance and *high dimensionality* have been acknowledged as two tough issues in classification. Learning from unbalanced data, the constructed classifiers are often biased towards the majority class, and thereby perform poorly on the minority class. Unfortunately, the minority class is often the class of interest in many real-world applications, such as medical diagnosis and fault detection. High dimensionality often makes it more difficult to handle the class imbalance issue. To date, most existing works attempt to address one single issue, without consideration of solving the other. These works could not be effectively applied to some challenging classification tasks that suffer from both of the two issues.

Genetic programming (GP) is one of the most popular techniques from evolutionary computation, which has been widely applied to classification tasks. The built-in feature selection ability of GP makes it very powerful for use in classification with high-dimensional data. However, if the class imbalance issue is not well addressed, the constructed GP classifiers are often biased towards the majority class. Accordingly, this thesis aims to address the joint effects of class imbalance and high dimensionality by developing new GP based classification approaches, with the goal of improving classification performance.

To effectively and efficiently address the performance bias issue of GP, this thesis develops a fitness function that considers two criteria, namely the approximation of area under the curve (AUC) and classification clarity (i.e. how well a program can separate the two classes). To further improve the efficiency, a new program reuse mechanism is designed to reuse previous effective GP individuals. According to experimental results, GP with the new fitness function and the program reuse mechanism achieves good performance and significantly saves training time. However, this method treats the two criteria equally, which is not always reasonable.

To avoid manually weighing the two criteria in the fitness evaluation process, we propose a novel two-criterion fitness evaluation method, where the obtained values on the two criteria are combined in pairs, instead of summing them together. Then, a three-criterion tournament selection is designed to effectively identify and select good programs to be used by genetic operators for generating better offspring during the evolutionary learning process. Experimental results show that the proposed GP method achieves better classification performance than compared methods.

Cost-sensitive learning is a popular approach to addressing the problem of class imbalance for many classification algorithms in machine learning. However, cost-sensitive algorithms are dependent on cost matrices that are usually designed manually. Unfortunately, it is often not easy for humans, even experts, to accurately specify misclassification costs for different mistakes due to the lack or incompleteness of domain knowledge related to actual situations in many complex tasks. As a result, these cost-sensitive algorithms cannot be directly applied. This thesis develops new GP based approaches to developing cost-sensitive classifiers without requiring cost matrices from humans. The newly developed cost-sensitive GP methods are able to construct classifiers and learn cost values or intervals automatically and simultaneously. The experimental results show that the new cost-sensitive GP methods outperform compared methods for high-dimensional unbalanced classification in almost all comparisons.

Cost-sensitive GP classifiers treat the minority class as being more important than the majority class, but this may cause an accuracy decrease in the overlapping areas where the prior probabilities of the two classes are about the same. In the thesis, we propose a neighborhood method to detect overlapping areas, and then use GP to develop cost-sensitive classifiers that employ different classification strategies to classify instances from the overlapping areas or the non-overlapping areas.

Acknowledgments

I wish to express my sincere gratitude to those who helped me during my PhD study.

First and foremost, my most sincere appreciation goes to my supervisors, Prof. Bing Xue and Prof. Mengjie Zhang, who spend dedicated time in training my research skills. Undoubtedly, this thesis would not have been possible without their guidance, encouragement, and support. Bing and Meng are always very nice to talk and discuss, and provide constructive comments as detailed as possible to improve the quality of my research work. Bing is very patient, and her views are always inspiring me to think deeper and further. Meng is very open-minded, and has a down-to-earth attitude towards work and research. I am also very grateful to A/Prof Lin Shang from Nanjing University. Dr. Lin Shang is also very nice and thoughtful. My special thanks go to Mr. Gerard Coyle and Dr. Diana Siwiak, who are of great help in improving my writing and oral presentation skills.

I greatly appreciate the China Scholarship Council / Victoria University of Wellington Scholarship for providing me financial support to pursue my PhD degree in the past four years. I am very grateful to the Embassy of the People's Republic of China in New Zealand for delivering love and care from our motherland to us. I am very grateful for financial support from the Faculty Strategic Research Grants and PGSA Academic Conference Travel Grants to help me attend international conferences.

Last but not least, I wish to thank my family members (particularly my parents) for their unconditional love, support and encouragement. I wish to thank my dearest friends in China and New Zealand (particularly Limin). I wish to thank all

of the members (particularly Truong) in the Evolutionary Computation Research Group (ECRG) and Feature Analysis, Selection, and Learning in Image and Pattern Recognition (FASLIP) group for nice discussions and free comments. My special thanks go to Yuxin for helping me rent a house on the first day I came to New Zealand and my landlords for a warm room, nice food and generous help.

List of Publications

1. Wenbin Pei, Bing Xue, Lin Shang, Mengjie Zhang. High-dimensional Unbalanced Classification by Genetic Programming with Multi-criterion Fitness Evaluation and Selection. (Accepted by Evolutionary Computation, MIT press)

2. Wenbin Pei, Bing Xue, Lin Shang, Mengjie Zhang. Developing Interval-based Cost-sensitive Classifiers by Genetic Programming for Binary High-dimensional Unbalanced Classification [Research Frontier]. IEEE Computational Intelligence Magazine, 2021, 16(1): 84-98.

3. Wenbin Pei, Bing Xue, Lin Shang, Mengjie Zhang. Genetic Programming for Development of Cost-sensitive Classifiers for Binary High-dimensional Unbalanced Classification. Applied Soft Computing, 2021, vol. 101. doi: 10.1016/j.asoc.2020.106989.

4. Wenbin Pei, Bing Xue, Lin Shang, Mengjie Zhang. Genetic Programming for High-Dimensional Imbalanced Classification with A New Fitness Function and Program Reuse Mechanism. Soft Computing, 2020, 24(23): 18021-18038.

5. Wenbin Pei, Bing Xue, Lin Shang, Mengjie Zhang. Genetic Programming for Borderline Instance Detection in High-dimensional Unbalanced Classification. Proceedings of the 2021 Genetic and Evolutionary Computation Conference, https://doi.org/10.1145/3449639.3459284. ACM, 2021: 349–357.

6. Wenbin Pei, Bing Xue, Lin Shang and Mengjie Zhang. A Threshold-free Classification Mechanism in Genetic Programming for Unbalanced High-dimensional Classification. Proceedings of the 2020 IEEE Congress on Evolutionary Computation, IEEE, 2020: 1-8.

7. Wenbin Pei, Bing Xue, Lin Shang, Mengjie Zhang. A Genetic Programming Method for Classifier Construction and Cost Learning in High-dimensional Unbalanced Classification. Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion. 2020: 149-150.

8. Wenbin Pei, Bing Xue, Lin Shang and Mengjie Zhang. A Cost-sensitive Genetic Programming Approach for High-dimensional Unbalanced Classification. Proceedings of the 2019 IEEE Symposium Series on Computational Intelligence. IEEE, 2019: 1770-1777.

9. Wenbin Pei, Bing Xue, Lin Shang and Mengjie Zhang. New Fitness Functions in Genetic Programming for Classification with High-dimensional Unbalanced Data. Proceedings of the 2019 IEEE Congress on Evolutionary Computation. IEEE, 2019: 2779-2786.

10. Wenbin Pei, Bing Xue, Mengjie Zhang. Reuse of Program Trees in Genetic Programming with a New Fitness Function for High-dimensional Unbalanced Classification. Proceedings of the 2019 Genetic and Evolutionary Computation Conference Companion. 2019: 187-188.

11. Wenbin Pei, Bing Xue, Lin Shang and Mengjie Zhang. Genetic Programming based on Granular Computing for High-dimensional Data in Classification. Proceedings of the 2018 Australasian Joint Conference on Artificial Intelligence. Springer, Cham, 2018: 643-655.

Submitted or completed drafts:

12. Wenbin Pei, Bing Xue, Lin Shang, Mengjie Zhang. Detecting Overlapping Areas in Unbalanced High-dimensional Data Using Neighborhood Rough Set and Genetic Programming. (This paper was submitted to IEEE Transactions on Cybernetics, February 2021)

13. Wenbin Pei, Bing Xue, Mengjie Zhang, Lin Shang, Xin Yao. A Survey on Unbalanced Classification: What Can Evolutionary Computation Help? (A survey paper, prepared for submission to IEEE Transactions on Evolutionary Computation, June 2021)

List of Abbreviations

ML	Machine Learning
KNN	K-nearest Neighbors
SVM	Support Vector Machines
NB	Naive Bayes
DT	Decision Trees
EC	Evolutionary Computation
GP	Genetic Programming
IR	Imbalance Ratio
AUC	Area Under a Curve
AI	Artificial Intelligence
NN	Neural Networks
ROC	Receiver Operating Characteristic
TPR	True Positive Rate
TNR	True Negative Rate
RUS	Random Undersampling
ROS	Random Oversampling
SMOTE	Synthetic Minority Oversampling Technique
ADASYN	Adaptive Synthetic Sampling
OCC	One-class Classification

EA	Evolutionary Algorithms
SI	Swarm Intelligence
GA	Genetic Algorithms
ES	Evolution Strategy
EP	Evolutionary Programming
PSO	Particle Swarm Optimization
ACO	Ant Colony Optimization
EMO	Evolutionary Multi-objective Optimization
DE	Differential Evolution
EDAs	Estimation of Distribution Algorithms
AIS	Artificial Immune Systems
NPGA	Niched Pareto Genetic Algorithms
NSGA-II	Nondominated Sorting Genetic Algorithm II
PAES	Pareto-archived Evolution Strategy
MOEA/D	Multiobjective Evolutionary Algorithm based on Decomposition
SPEA	Strength Pareto Evolutionary Algorithm
STGP	Strongly-Typed Genetic Programming
G3P	Grammar-Guided Genetic Programming
CFG	Context-free Grammar
BNF	Backus-Naur Form
SGP	Stack-based Genetic Programming

LGP Linear Genetic Programming

CGP Cartesian Genetic Programming

PMB-GP Probabilistic Model Building in Genetic Programming

PIPE Probabilistic Incremental Program Evolution

EDP Estimation of Distribution Programming

ECGP Extended Compact Genetic Programming

RSS Random Subset Selection

DSS Dynamic Subset Selection

HSS Historical Subset Selection

MOGP Multi-objective GP

SVDD Support Vector Data dDescription

GPFRM Genetic Programming with a New Fitness Function and Reuse Mechanism

GPMFS Genetic Programming with Multi-criterion Fitness Evaluation and Selection

CS-GP Cost-sensitive Genetic Programming

ICS-GP Interval-based Cost-sensitive Genetic Programming

CSGPNOD Cost-sensitive Genetic Programming with Neighborhood-based Overlapping Detection

List of Tables

List of Figures

Contents

Chapter 1

Introduction

This chapter introduces the problem statement, then outlines the motivations, research objectives and the organization of this thesis.

1.1 Problem Statement

Machine learning (ML) algorithms aim to discover useful information, knowledge expression, and hidden patterns from data. Classification is one of the most important tasks in ML, which refers to an algorithmic procedure to assign a piece of input data into its corresponding class or category [84]. Many classification algorithms have been proposed and widely used, e.g. K-nearest neighbors (KNN) [26], support vector machines (SVMs) [54], Naive Bayes (NB) [121], and decision tree (DTs) [71].

Classification has a wide range of applications, but many of them encounter a problem of *class imbalance*, i.e. the number of instances per class is disproportionate or uneven. In binary classification with unbalanced data, one class includes only a few instances (called the minority class), and the rest of instances belong to the other class (called the majority class) [11]. Class imbalance is a common issue in many real-world applications, such as medical diagnosis, bioinformatics and fault detection. In these applications, normal instances are usually abundant, while abnormal instances are often rare and of the most interest.

Standard classification algorithms often assume that all the instances in a dataset are of the same importance, and thereby treat them equally. This assumption does not always hold in many real-world applications. Due to the assumption, learning from unbalanced data, the constructed classifiers are biased towards the majority class because its number is typically larger than that of the minority class. As a consequence, these biased classifiers perform well on the majority class but poorly on the minority class. This is known as the issue of performance bias.

To resolve the issue of class imbalance, in general, existing methods can be grouped into two categories, i.e. data-level methods and algorithm-level methods. At the data level, sampling methods [16, 29, 62, 106, 110, 151] are very popular, which aim to re-balance unbalanced datasets to ensure the same or a similar number of instances per class. The main drawback of sampling methods is that the original data distribution is changed. At the algorithm level, cost-sensitive learning [35,92,184] is popularly used, which considers costs (mostly misclassification costs) to treat different kinds of mistakes differently. Usually, the misclassification costs are incorporated into classification paradigms or objective functions to construct cost-sensitive classifiers [5, 69, 91, 217]. However, most existing cost-sensitive algorithms require cost matrices that are often provided by domain experts. Unfortunately, it is often difficult for humans to assign accurate misclassification cost values to different kinds of mistakes [102].

In many real-world applications, a growing number of unbalanced datasets are high-dimensional, e.g. gene expression data [186]. However, very few existing methods for unbalanced classification have specifically considered high-dimensional data. High dimensionality often makes it more difficult to address the issue of class imbalance [63]. For classification with high-dimensional data, usually, neither the majority class nor the minority class has a sufficient number of instances. Data becomes sparse if the number of features is typically much larger than the number of instances. This may increase the difficulty in constructing a classifier with a good generalization ability. In fact, many features in high-dimensional datasets are irrelevant or redundant [134]. The presence of these features may increase the model complexity and even degrade the classification

performance. Therefore, feature selection is often used to reduce dimensionality of the data. However, feature selection is an NP-hard problem and has a large search space [204]. More importantly, in unbalanced classification, it is very likely to select features that are biased towards the majority class if the class imbalance issue is not well addressed.

Evolutionary computation (EC) [4] is inspired by biological evolution and natural selection, which is a group of heuristic techniques to discover optimal solutions to a problem. Genetic programming (GP) [143] is an EC technique that aims at automatically generating computer programs as solutions. GP has been proven to work effectively for a variety of classification tasks. In a population, each GP individual (also called a program or tree) is often used as a classifier, which can simultaneously and automatically select good-quality features for use. However, if the issue of class imbalance is not well addressed, similar to other classification methods, the GP classifiers are often biased towards the majority class [11].

This thesis aims to address the joint effects of high dimensionality and class imbalance by designing new GP based classification methods that are expected to achieve better performance. The thesis focuses on binary classification because of the following reasons. Firstly, many real-world applications related to unbalanced data are often binary classification. Secondly, binary classification is still very challenging if data is high-dimensional as well as unbalanced. Thirdly, a multi-class classification task could be conducted by decomposing it into multiple binary classification tasks. Hence, we believe high-dimensional unbalanced binary classification is still worth investigating.

1.2 Motivations

1.2.1 Difficulties of Classification with High-dimensional Unbalanced Data

Learning from unbalanced data, the main difficulty is from the following factors:

1) Skewed data distribution.

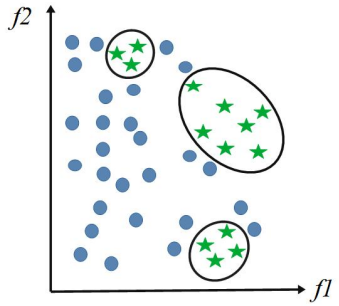


Figure 1.1: Within-class imbalance.

Data exhibits the skewed data distribution. This causes the performance bias issue (i.e. a big gap between the performances on different classes) when classifiers treat all the instances equally [14]. In many extremely unbalanced cases, biased classifiers may ignore instances from the minority class.

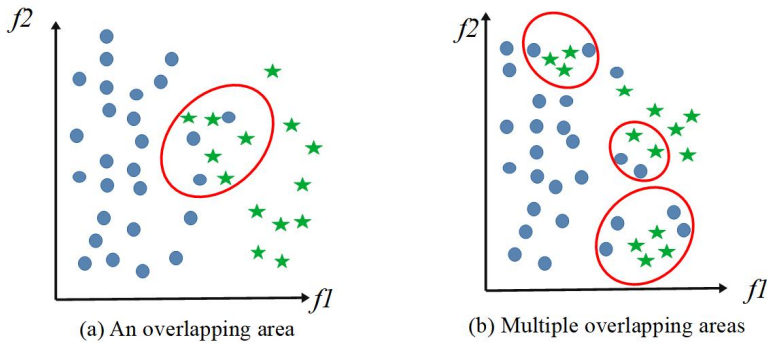
The imbalance ratio (IR) is used to measure the imbalance degree of a dataset. IR is the number of instances in the majority class divided by that in the minority class, defined as [209]:

$$IR = \frac{|Maj|}{|Min|} \tag{1.1}$$

where $|Maj|$ and $|Min|$ indicate the numbers of instances in the majority class (Maj) and the minority class (Min), respectively.

2) Within-class imbalance [176].

Within-class imbalance refers to a disproportionate data distribution within a class [63]. It occurs when a class is composed of several sub-clusters, at least one of which significantly outnumbers others. Within-class imbalance increases the data complexity and is often closely intertwined with small disjuncts [63, 176]. The presence of within-class imbalance is often implicit rather than clearly stated [63].



(a) An overlapping area (b) Multiple overlapping areas

Figure 1.2: Class overlap.

Figure 1.1 shows an example of within-class imbalance in a 2-dimensional space. The minority class has three sub-clusters (indicated by black circles), and a sub-cluster outnumbers the other two.

3) Class overlap [45].
For complicated unbalanced classification tasks, the minority class may overlap with the majority class, which is known as the issue of class overlap. In the overlapping areas, instances from different classes have some similar characteristics, often making it more difficult for classifiers to correctly discriminate boundaries between the majority class and the minority class. The prior probabilities of both classes in an overlapping area are almost the same [57].

In binary classification, there is usually one overlapping area between the minority class and the majority class. However, if a class has multiple sub-clusters, each sub-cluster may overlap with the other class and produce multiple overlapping areas in a binary classification task. In Figure 1.2 (a), an overlapping area is indicated, while three overlapping areas are indicated in Figure 1.2 (b).

High dimensionality often makes it more difficult to address the issue of class

imbalance [63]. Data becomes sparse if features typically outnumber instances, which increases the difficulty in constructing a classifier with a good generalization ability. Moreover, there might be some irrelevant features in high-dimensional data. This may aggravate the class overlap issue because some instances from different classes become similar when they have identical values on the irrelevant features.

For classification with high-dimensional data, feature selection is widely used to reduce dimensions of the data by selecting the smallest number of informative features [193]. The selected features should be necessary and sufficient to describe the target labels. However, feature selection is also a challenging task due to the large search space (the total number of possible solutions is 2^n, where n is the number of original features). Moreover, a feature may interact with other features. Accordingly, it is possible for a feature that is weakly relevant to the target labels to become important when using it together with other features [204].

1.2.2 Why GP?

GP is used to develop classifiers because of its advantages as follows:

1) **The built-in feature selection ability.**
   All the features are often fed into GP. However, a GP individual does not need to use all of the features, but automatically selects informative ones that can benefit the individual to achieve a better fitness value. In classification, the fitness function is usually designed as an accuracy measure. Similar to GP, DT also has a built-in feature selection ability, but DT usually employs a greedy search that may get stuck in local optima.

2) **The flexibility of the GP representation.**
   In GP, the individuals are often structured in terms of trees, where internal nodes of a tree are taken from a function set and leaf nodes are taken from a terminal set. The flexibility of the GP representation enables it to evolve various kinds of models, such as discriminant functions, rules, or decision

trees, etc [37]. Moreover, due to its flexible tree representation, GP is able to perform multiple learning tasks simultaneously.

1.2.3 Main Limitations of Existing Methods

1) Sampling Methods

Sampling techniques are the most popular data-level methods for resolving the issue of class imbalance because they are not limited to a specific classification algorithm. In general, sampling methods fall into three groups, i.e. undersampling [100, 106], oversampling [16, 62], and hybrid sampling [151]. For a hybrid sampling method, it takes advantage of oversampling and undersampling, e.g. new instances are generated for the minority class by oversampling and then some of the less useful ones are removed by undersampling.

However, sampling methods have the following disadvantages. For undersampling methods, it is often challenging to avoid losing useful information when determining which instances are to be excluded from the majority class, particularly when a small number of instances are available. For oversampling methods, some instances are repeatedly learned or synthetically generated. Therefore, classification algorithms need to take a longer training time to develop classifiers, and may have a risk of being over-fitting [19]. Therefore, this thesis focuses mainly on the algorithm-level methods, i.e. to improve GP for use in high-dimensional unbalanced classification without changing data.

2) Cost-sensitive Learning

Cost-sensitive learning has been successfully applied to unbalanced classification, and many standard classification algorithms have been extended to cost-sensitive versions. However, most existing cost-sensitive algorithms are dependent on manually-designed cost matrices. Unfortunately, in many cases, it is often not easy for humans, even experts, to accurately specify misclassification costs for different mistakes due to the lack of domain knowledge. Moreover, different experts may have different opinions when evaluating the same type of mistake.

When possible cost values for each class are specified and considered as a group, classification algorithms need to be well tuned based on these cost values [102]. This may increase the computational cost of applying a cost-sensitive algorithm to a classification task. To date, the use of cost-sensitive learning with GP has not been heavily investigated. It is important to explore how cost-sensitive learning can be used to improve the classification performance of GP in unbalanced classification when cost matrices are not available.

1.3 Goals

The overall goal of the thesis is to develop effective GP-based methods for binary classification with high-dimensional unbalanced data, with the expectation of achieving better classification performance than existing methods. To achieve the overall goal, the specific objectives are introduced as follows.

(1) Developing a novel GP method by designing a new fitness function and a program reuse mechanism for classification with high-dimensional unbalanced data. The proposed method is expected to improve the classification performance and save training time.

In GP, a fitness function can be used to effectively and directly solve the performance bias issue in unbalanced classification. Area under a curve (AUC) is an important measure in unbalanced classification because it evaluates the performance of a classifier across varying thresholds. GP using AUC as the fitness function often achieves a promising classification performance, but it is very time-consuming [11]. To save training time, AUC approximation measures have been developed and used as a fitness function in GP [11]. However, the improvement in efficiency is often at the expense of the decreased classification performance.

In this objective, a new AUC approximation measure will be designed. Based on that, a new fitness function will be proposed by considering the AUC approximation measure and a classification clarity measure, in order to

effectively evaluate the goodness of individuals. To improve the efficiency, a program reuse mechanism will be designed to reuse previous good GP individuals.

(2) Developing a new GP method by designing new multi-criterion fitness evaluation and selection methods for classification with high-dimensional unbalanced data. The proposed GP method is expected to be able to consider multiple criteria without using pre-designed weights to combine them in the evaluation process and effectively identify good individuals in the selection process, in order to improve the classification performance of GP.

In a single-objective GP method, when n criteria are considered in the fitness function, the most popular method is to aggregate the n criteria by the weighted sum approach. However, the weights are usually not easy to determine without domain knowledge. In objective (1), the AUC approximation and classification clarity measures (i.e. two criteria) are equally weighted and summed together to be a fitness function. However, the two criteria are not always equally important. To avoid weighting them, a new two-criterion fitness evaluation method will be designed to enable individuals to be independently evaluated by the two criteria. Then, a new three-criterion tournament selection will be designed, which allows a set of solutions to be filtered according to a cascading set of priorities in the selection process.

(3) Designing a new cost-sensitive GP method for classification with high-dimensional unbalanced data. The proposed method is expected to achieve good performance when manually-designed cost matrices are not available.

Cost-sensitive learning is a popular method to resolve the problem of class imbalance in machine learning, which has been successfully used by many classification algorithms. However, it is less investigated how GP is used with cost-sensitive learning to avoid the performance bias issue in unbalanced classification. In this objective, the use of cost-sensitive learning with GP will be systematically investigated.

For many existing cost-sensitive methods, cost matrices are often manually-designed and problem-specific. The misclassification costs in a cost matrix are often given as precise values. Unfortunately, it is usually not easy for domain experts to accurately specify or assign the precise cost values to different kinds of mistakes. If no cost information is available, the easiest method is to use the class imbalance ratio of a dataset to construct a cost matrix for an unbalanced classification task [38]. However, this method is often criticized because it is over-simplified without considering data characteristics [38]. In this objective, we aim to propose a new cost-sensitive GP method, which is expected to achieve promising performance for binary classification with high-dimensional unbalanced data when cost matrices are not available. To achieve this goal, a tree representation, a terminal set and a function set are designed to construct classifiers and optimize cost values automatically and simultaneously.

(4) Developing a new interval-based cost-sensitive GP method for classification with high-dimensional unbalanced data. The proposed method is expected to automatically learn *cost intervals* and use them to effectively construct cost-sensitive classifiers.

Cost information has three main types, i.e, cost values, cost intervals, and cost distribution. In the literature published to date, most existing cost-sensitive methods are based on cost values. When using a real value, a decision result is usually sought to be determinate and rigid, while when using intervals, uncertainty is considered to tolerate possible mistakes in a decision-making process [96].

In this objective, we aim to provide a new cost-sensitive GP method, which is expected to construct interval-based cost-sensitive classifiers to achieve promising performance when cost matrices are not available. To achieve this, new function and terminal sets will be designed to learn reasonable cost intervals and construct classifiers automatically and simultaneously. In order to effectively use the learned cost interval, a new classification strategy

will be designed for GP.

(5) Developing a new GP method with the detection of overlapping instances for classification with high-dimensional unbalanced data. The proposed method is expected to detect overlapping instances in order to effectively classify them by constructed GP classifiers.

In classification, when class overlap is intertwined with the issue of class imbalance, it is often challenging to discover useful patterns because of an ambiguous boundary between the majority class and the minority class. Although standard classification algorithms attempt to correctly separate different classes, they usually learn patterns from the whole training set. If a task suffers from a serious class overlap issue, these algorithms do not specifically detect overlapping areas, and thereby often treat overlapping instances and non-overlapping instances equally. This makes it hard to effectively classify instances from the overlapping areas [90].

In this objective, a new GP method will be presented, which aims to address the issues caused by class overlap and class imbalance in classification with high-dimensional data. In the proposed method, a novel neighborhood based method will be designed to detect instances in overlapping areas before the evolutionary learning process of GP. Note that the designed overlapping detection method is expected to not only detect the only overlapping area in a dataset, as described in Figure 1.2 (a), but also to detect multiple overlapping areas, as described in Figure 1.2 (b). Moreover, different classification rules will be designed to classify instances from the non-overlapping area or from the overlapping area in order to enhance the classification performance.

In summary, we will investigate how a fitness function is used in GP to solve the class imbalance issue in research objectives 1 and 2. We will investigate how GP is used with cost-sensitive learning in research objectives 3 and 4. We will investigate how the class overlap issue intertwined with the class imbalance issue can be effectively addressed in research objective 5.

1.4 Major Contributions

The major contributions of this thesis are summarized as follows:

(1) The thesis shows how GP using a fitness function can solve the problem of class imbalance and how effective GP individuals can be reused for improving the efficiency. A new GP method is proposed by designing a fitness function and a program reuse mechanism. Based on the experimental results, the proposed method is able to achieve good classification performance and significantly reduce training time.

Part of this contribution has been published in:

Wenbin Pei, Bing Xue, Lin Shang, Mengjie Zhang. Genetic Programming for High-Dimensional Imbalanced Classification with A New Fitness Function and Program Reuse Mechanism. Soft Computing, 2020, 24(23): 18021-18038.

Wenbin Pei, Bing Xue, Mengjie Zhang. New Fitness Functions in Genetic Programming for Classification with High-dimensional Unbalanced Data. Proceedings of the 2019 IEEE Congress on Evolutionary Computation. IEEE, 2019: 2779-2786.

Wenbin Pei, Bing Xue, Mengjie Zhang. Reuse of Program Trees in Genetic Programming with a New Fitness Function for High-Dimensional Unbalanced Classification. Proceedings of the 2019 Genetic and Evolutionary Computation Conference Companion. 2019: 187-188.

Wenbin Pei, Bing Xue, Lin Shang and Mengjie Zhang. Genetic Programming based on granular Computing for High-dimensional data in Classification. Proceedings of the 2018 Australasian Joint Conference on Artificial Intelligence. Springer, Cham, 2018: 643-655.

(2) The thesis shows how an individual in the population can be effectively evaluated by multiple criteria without using pre-designed weights, to improve the performance of GP for high-dimensional unbalanced classification. A new GP method is proposed, which designs new multi-criterion

evaluation and selection methods. The experimental results show that the proposed method outperforms the compared methods in almost all comparisons. Further analysis reveals that the proposed method is able to select a small number of features for classification with high-dimensional data.

Part of this contribution has been accepted for publication in:

Wenbin Pei, Bing Xue, Lin Shang, Mengjie Zhang. High-dimensional Unbalanced Classification by Genetic Programming with Multi-criterion Fitness Evaluation and Selection. Evolutionary Computation journal, MIT press, 2021.

(3) This thesis investigates how cost-sensitive learning can be used with GP, and proposes a new cost-sensitive GP method that is able to construct classifiers and learn the cost values simultaneously and automatically. The results show that the proposed method achieves better (or similar) performance than the compared methods in almost all comparisons. Further in-depth analysis shows that the proposed method is able to learn cost values and effectively construct cost-sensitive classifiers.

Part of this contribution has been published in:

Wenbin Pei, Bing Xue, Lin Shang, Mengjie Zhang. Genetic Programming for Development of Cost-sensitive Classifiers for Binary High-dimensional Unbalanced Classification. Applied Soft Computing, 2021, vol. 101. doi: 10.1016/j.asoc.2020.106989.

Wenbin Pei, Bing Xue, Lin Shang and Mengjie Zhang. A Cost-sensitive Genetic Programming Approach for High-dimensional Unbalanced Classification. Proceedings of the 2019 IEEE Symposium Series on Computational Intelligence. IEEE, 2019: 1770-1777.

(4) The thesis investigates how cost intervals can be automatically learned by GP to effectively construct interval-based cost-sensitive classifiers. Based on that, the thesis proposes a new interval-based cost-sensitive GP method

for classification with high-dimensional unbalanced classification. The experimental results show that the proposed method is able to learn cost intervals and outperforms the compared methods (including GP using sampling methods, GP using different fitness functions, and cost-sensitive GP based on cost values) in almost all comparisons.

Part of this contribution has been published in:

Wenbin Pei, Bing Xue, Lin Shang, Mengjie Zhang. Developing Interval-based Cost-sensitive Classifiers by Genetic Programming for Binary High-dimensional Unbalanced Classification [Research Frontier]. IEEE Computational Intelligence Magazine, 2021, 16 (1): 84-98.

(5) The thesis investigates how the issue of class overlap can be addressed in classification with high-dimensional unbalanced data by using GP. Based on that, the thesis proposes a new method, which is able to detect overlapping instances. The detected overlapping instances and non-overlapping instances are classified by using different classification strategies designed for GP. The experimental results show that the classification performance of GP is enhanced after the class overlap issue is well-addressed.

Part of this contribution has been accepted for publication or submitted to:

Wenbin Pei, Bing Xue, Lin Shang, Mengjie Zhang. Genetic Programming for Borderline Instance Detection in High-dimensional Unbalanced Classification. Proceedings of the 2021 Genetic and Evolutionary Computation Conference, https://doi.org/10.1145/3449639.3459284. ACM, 2021: 349–357.

Wenbin Pei, Bing Xue, Lin Shang, Mengjie Zhang. Detecting Overlapping Areas in Unbalanced High-dimensional Data Using Neighborhood Rough Set and Genetic Programming. (This paper was submitted to IEEE Transactions on Cybernetics, 2021.)

1.5 Organization of the Thesis

The thesis is outlined in Figure 1.3, which also shows the connections among the research objectives introduced in Section 1.3. Chapter 2 introduces the background knowledge and reviews the related work. Chapters 3-7 are the contribution chapters, each of which presents a new GP method to achieve one of the research objectives listed in Section 1.3. Chapter 8 concludes the whole thesis and points out future research directions.

Chapter 2 is devoted to introducing the related background knowledge, including basic concepts and techniques in machine learning and evolutionary computation (particularly GP). The related existing works are introduced and summarized in this chapter.

Chapter 3 presents a new GP method to address the performance bias issue of GP in classification with high-dimensional unbalanced data. In this chapter, a new fitness function is developed to address the issue of class imbalance. Furthermore, to improve the efficiency, a program reuse mechanism is designed to reuse effective GP programs that are previously evolved. The new method is compared with popular classification methods, and the experimental results are analyzed and discussed.

Chapter 4 presents a new GP method with multi-criterion evaluation and selection for classification with high-dimensional unbalanced data. The proposed method does not need a weight to combine two criteria (i.e. AUC approximation and the classification clarity) in the evaluation process. A new three-criterion tournament selection operator is designed to effectively identify and select good programs to be used by genetic operators for generating better offspring during the evolutionary learning process. The proposed method is compared with the popular classification methods, and the experimental results are analyzed and discussed in depth.

Chapters 5 and 6 introduce new cost-sensitive GP methods, which are able to develop classifiers and learn cost information automatically and simultaneously. Chapter 5 introduces how GP is used to learn cost values and develop

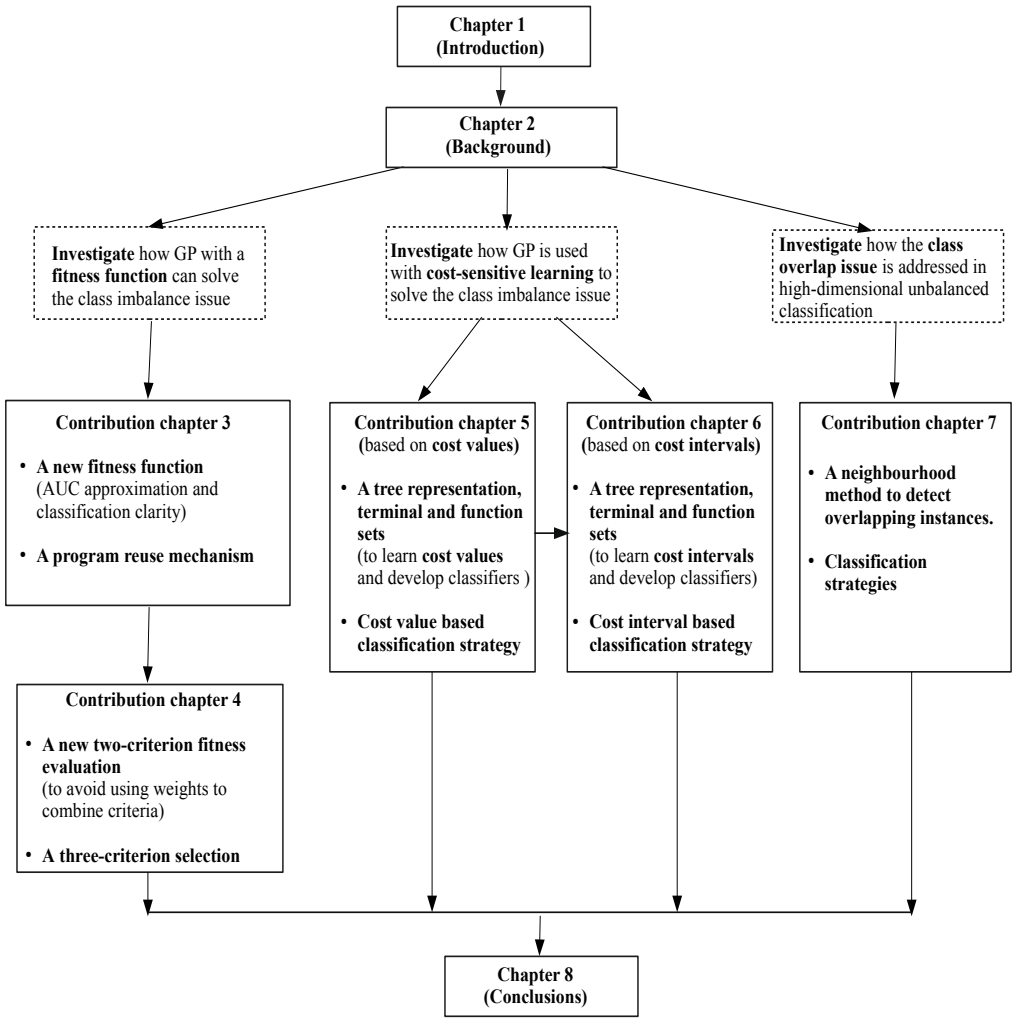


Figure 1.3: The outline of the thesis.

cost-sensitive classifiers simultaneously. Chapter 6 introduces how GP is used to learn cost intervals to construct cost-sensitive classifiers. The experimental results are analyzed and discussed in detail.

Chapter 7 introduces a new GP method that targets to address the issue of class overlap in classification with high-dimensional unbalanced data. The proposed method detects overlapping instances before the evolutionary learning process of GP. The experimental results are also analyzed and discussed.

Chapter 8 summarizes the major contributions of this thesis and concludes the whole thesis. Future work and possible research directions are also discussed.

1.6 Benchmark Datasets

In the thesis, the effectiveness of the proposed methods is examined on gene expression datasets [221][1]. Gene expression data is from bioinformatics and is about different diseases, such as lung and colon. Because of privacy protection policies, the number of collected patient samples (i.e. instances) is usually small, but each sample is involved with a large number of genes (i.e. features). Generally, throughout gene expression datasets, the number of features varies from hundreds to tens of thousands. Hence, gene expression datasets are high-dimensional and many of them are unbalanced datasets.

[1]These datasets are available at:
http://csse.szu.edu.cn/staff/zhuzx/Datasets.html;
https://schlieplab.org/Static/Supplements/CompCancer/datasets.htm;

Chapter 2

Literature Survey

This chapter covers essential background, concepts of machine learning and evolutionary computation techniques. We also review the related literature in this chapter.

2.1 Machine Learning

Machine learning (ML) is a very important branch of artificial intelligence (AI). ML was first proposed in 1959 by Arthur Samuel [160]. ML aims at using techniques to make computer systems with the ability to learn, without being explicitly programmed [3]. In 1998, Tom Mitchell pointed out a learning problem: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E " [115].

Based on the type of feedback, machine learning algorithms are usually categorized into three groups, namely, supervised learning, unsupervised learning and reinforcement learning [156].

- Supervised learning: Supervised learning maps an input to an output when learning from labeled instances. Supervised learning algorithms need to learn from the training data to discover inherent patterns or functions to

predict labels of unseen instances [179]. Classification is one of the most important tasks in supervised learning.

- Unsupervised learning: Unsupervised learning [179] is a group of learning techniques that attempt to learn inherent patterns from non-labeled data. Clustering is one of the representative tasks in unsupervised learning.

- Reinforcement learning: Reinforcement learning [180] is different from supervised learning that requires a direct relationship between the input and output, such as classification. For reinforcement learning, desired outputs are not directly provided. Every action of a learner or an agent has an impact on an environment, and correspondingly, the environment provides some feedback in terms of rewards and punishments as consequences of actions of learners.

This thesis focuses on one of the most common tasks, i.e. classification, in supervised learning.

2.1.1 Training and Testing Processes in Classification

Classification refers to an algorithmic procedure to assign a piece of input data into its corresponding class. Classification tasks often have two processes, namely, training and testing. The training process refers to a learning process of inducing a new model or pattern from the given observations, and in the testing process, the performance of this induced model is examined on unseen observations in the same problem domain [197]. The sets of instances used in the training and testing processes are called the training set and the test set, respectively. Note that instances in the test set are not used and remain unseen in the training process.

To split a dataset into the training set and the test set, a straightforward method is holdout, which divides a dataset into two disjoint sets for the training and testing processes according to a predefined proportion [79]. However, this method cannot ensure the class imbalance ratio in the training and test sets to be the same as the original dataset. Accordingly, stratified sampling [128] has been proposed

and widely used in unbalanced classification, which can ensure the same class imbalance ratio in the training/test split and the original dataset.

2.1.2 Classification Algorithms

This section introduces several popular classification algorithms, including K-nearest neighbors (KNN), Naive bayes (NB), decision trees (DTs), support vector machines (SVMs) and neural networks (NNs).

K-Nearest Neighbor (KNN). KNN [185, 215] is one of the most well-known ML algorithms. KNN is an instance-based learning (also called lazy learning) method, where the training instances are directly used as prototypes of classifiers. To classify an unseen instance, KNN calculates the distance (e.g. Euclidean or Manhattan distances) between the unseen instance to the training instances, to identify the nearest k neighbors in the training set [81]. Then, the unseen instance is classified to a class to which the majority of its neighbors belong.

KNN does not require the training phase [30]. This would make KNN faster than other classification algorithms that require the training phase. Besides, new instances can be seamlessly added. However, KNN is not efficient if data is high-dimensional because of the high computational cost to calculate distances [119]. In addition, KNN may not work effectively if there are categorical or symbolic features in data [17]. This is mainly because distance measures may not accurately embody semantic information of the data. Furthermore, the k value is usually hard to determine.

Naive Bayes (NB). Bayesian classifiers are a kind of probabilistic-based classification algorithms, in which NB is the most common one [121]. NB is robust to classification with missing or noisy data [80]. However, NB is based on a strict assumption that features are statistically independent. The assumption does not always hold in many real-world applications, which may restrict the use of NB in many applications.

Decision Trees (DTs). DTs [149, 158, 170] are based on a tree-like representation. Most DT algorithms employ an entropy function and a greedy search strategy to choose the best node at each stage, e.g. ID3 [149] and C4.5 [150, 202]. DTs are easy to interpret, which is, however, at the expense of a relatively low classification accuracy [170]. Another disadvantage of DT is instability, i.e. small changes in the training set may cause different classification results for the same validation instances because different trees are constructed [89].

Support Vector Machines (SVMs). SVMs have been acknowledged as one of the most powerful ML algorithms. SVMs map input data into a high dimensional space, and construct an optimal hyperplane that is expected to maximize the margin between two classes in the space [54, 94]. However, SVMs often require extensive memory.

Neural Networks (NNs). NNs are data-driven self-adaptive methods, i.e. they can adjust themselves to the data without any explicit specification of functional or distributional form for the underlying model [211]. NNs are non-linear models and are able to approximate any function with arbitrary accuracy [66, 211]. However, NNs are computationally expensive, and are prone to be overfitting when a small number of instances are available for training [194].

2.1.3 Classification Measures

This section introduces classification measures. A confusion matrix is illustrated in Table 2.1.

The Overall Classification Accuracy and Error Rate

The overall classification accuracy and error rate are defined as follows:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{N} \tag{2.1}$$

$$Err = \frac{FP + FN}{TP + TN + FP + FN} = \frac{FP + FN}{N} \tag{2.2}$$

Table 2.1: Confusion matrix.

		Actual: Positive	Actual: Negative	Total
Predicted	Positive	TP (True Positive)	FP (False Positive)	$TP + FP$
	Negative	FN (False Negative)	TN (True Negative)	$FN + TN$
	Total	$TP + FN$	$FP + TN$	N

Precision and Recall

Precision is used to measure exactness (i.e. what percentage of instances classified as positive are truly positive instances); recall is to measure completeness (i.e. what percentage of instances in the positive class are classified correctly as positive instances). The two measures show an inverse relationship between each other, so a combination of precision and recall has been considered to better evaluate the classification performance of learning algorithms in unbalanced scenarios [63]. Precision and recall are defined as [130]:

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

F-measure (or called F-score)

$F - measure$ [130, 145] is a combination of $Precision$ and $Recall$, defined as:

$$F - measure = \frac{(1 + \beta^2) * Precision * Recall}{\beta^2 * Precision + Recall} \tag{2.5}$$

where β is a coefficient to adjust the relative importance of precision versus recall.

Geometric Mean

Geometric mean (G_Mean) [63] is the square root of the product of the true positive rate (TPR, also called sensitivity) and the true negative rate (TNR, also

called specificity). It is defined as:

$$G\_Mean = \sqrt{\frac{TP}{TP+FN} * \frac{TN}{TN+FP}} \tag{2.6}$$

Weighted Average Classification Accuracy

It is the weighted average of TPR and TNR, defined as:

$$Ave = W * \frac{TP}{TP+FN} + (1-W) * \frac{TN}{TN+FP} \tag{2.7}$$

where W is a weight. Usually, W is set to 0.5, i.e. assuming that the two classes are equally important [11].

Area Under Curve (AUC)

Receiver operating characteristic (ROC) curve can be used to evaluate the performance of a classifier in unbalanced classification [145]. Both TPR and FP rate (FPR) are estimated at multiple thresholds. The area under a ROC curve (i.e. AUC) shows the classification ability of a classifier across varying thresholds. Full AUC (Auc_F) is defined as follows:

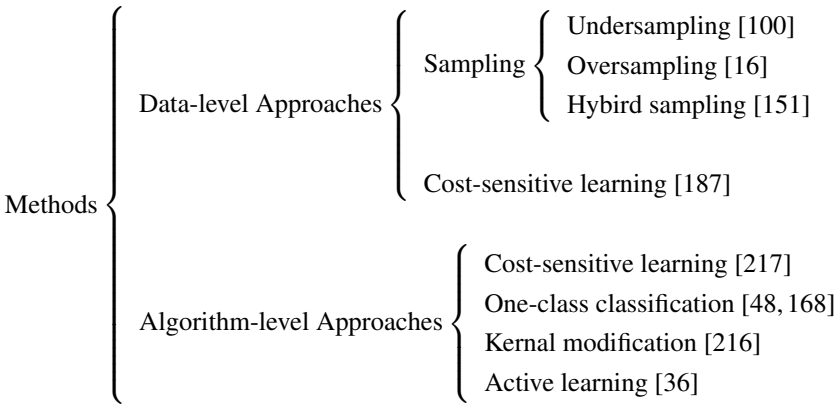
$$Auc_F = \sum_{i=1}^{N-1} \frac{1}{2} * (FPR_{i+1} - FPR_i)(TPR_{i+1} + TPR_i) \tag{2.8}$$

where N is the number of classification thresholds, and TPR_i and FPR_i are TPR and FPR at the ith threshold.

2.2 Unbalanced Classification

Traditional classification algorithms usually expect the same or a similar number of instances per class. Therefore, they usually treat every instance equally. As a consequence, when data is unbalanced, trained classifiers are often biased towards the majority class, and thereby achieve a low accuracy on the minority class that is often the class of great interest.

In order to address the class imbalance issue, the methods are generally grouped into two categories, i.e. data-level approaches and algorithm-level approaches, shown as follows [57, 63]:



Note that cost-sensitive learning can be used at both of the data and algorithmic levels [57]. These methods will be introduced in more detail in the following subsections.

2.2.1 Sampling Methods

Sampling methods have been widely used to re-balance unbalanced datasets. In general, sampling methods fall into three groups, i.e. undersampling, oversampling and hybrid sampling methods [57].

Undersampling

For undersampling, it removes some instances from the majority class to ensure its number to be the same or roughly the same as that of the minority class. The easiest undersamping method is random undersampling (RUS), which randomly discards some instances of the majority class. However, RUS may lose important instances, and thereby cause unavoidable information loss. To overcome this

limitation, some other techniques, such as ensemble learning [100, 106], clustering [95,129] and evolutionary computation [34,46,49], have been used to improve RUS.

Oversampling

Different from undersampling, oversampling replicates or creates synthetic instances for the minority class, to make sure its number to be the same or roughly the same as that of the majority class. Random oversampling (ROS) is the easiest oversampling method, where some instances are selected at random and reused multiple times. However, this method may increase the risk of a classifier overfitting to training data because some instances are learned repeatedly. Therefore, a synthetic minority oversampling technique (SMOTE) [16] has been proposed, which is a representative oversampling technique to create synthetic instances for the minority class.

An instance (from the minority class) is expressed as $x = (v_1, v_2, \cdots, v_d)$, where v_f means the value of x on a feature f. To generate a synthetic instance based on x, SMOTE has following steps:

- The k-nearest neighbors around instance x are identified, based on KNN.
- One of the k nearest neighbors is randomly selected, denoted as x^n.
- A synthetic instance $x_{synthetic}$ is created by the following equation:

$$x_{synthetic} = x + (x^n - x) * rand(0, 1) \tag{2.9}$$

  where $rand(0, 1)$ is a random number that is uniformly generated in the range of 0 and 1.

The main idea of SMOTE is described and explained in Figure 2.1. In this example, for instance x, its 5 nearest neighbors are identified by KNN. Afterwards, x^n is randomly selected from the 5 neighbors and used with x to generate $x_{synthetic}$ based on Eq. (2.9).

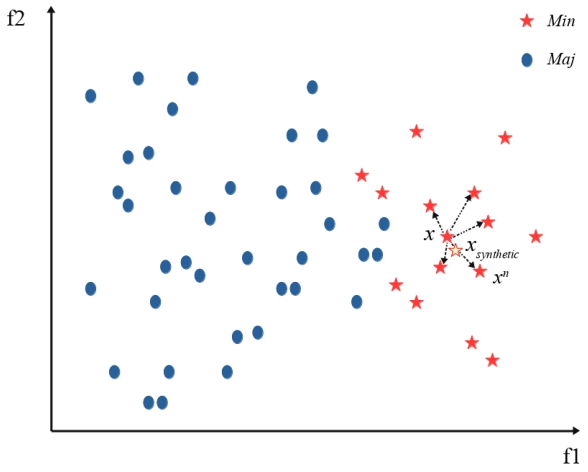


Figure 2.1: Idea of SMOTE.

For each instance from the minority class, after identifying the k nearest neighbors, how many neighbors are chosen (with replacement) is determined by the number of synthetic instances that are required to generate. For example, the minority class has 20 original instances, and 40 synthetic instances are required to generate. Hence, for every instance in the minority class, 2 out of the k nearest neighbors are randomly selected to generate 2 new instances, i.e. 1 neighbor x^n is used with x to generate 1 synthetic instance according to Eq. (2.9). In [109, 110], the properties of SMOTE have been investigated for high-dimensional unbalanced classification.

However, SMOTE assumes that all the instances from the minority class are equally important, so it uniformly assigns a number of synthetic instances to be generated by each instance in the minority class. As a result, SMOTE generates the same number of synthetic instances for every instance in the minority class, and thereby increases the occurrence of overlapping among instances [63]. This problem is known as *over-generalization*, which may increase the risk of overfitting [63]. To address or alleviate the over-generalization problem, new synthetic oversampling methods have been developed, e.g. borderline-SMOTE [60] and an

adaptive synthetic sampling algorithm (ADASYN) [62].

Borderline-SMOTE has two versions, i.e. borderline-SMOTE1 and borderline-SMOTE2 [60]. For instance x from the minority class, after identifying the k nearest neighbors, if the number of its neighbors from the majority class is more than that of its neighbors from the minority class, x is put into a DANGER set. In borderline-SMOTE1, the instances in the DANGER set are used for generating synthetic instances (the process is the same as SMOTE). The way of obtaining the DANGER set in borderline-SMOTE2 is the same as that in borderline-SMOTE1. The main difference between the two methods is the way of how an instance in the DANDER set is used to generate a new instance. In borderline-SMOTE2, a synthetic instance is generated by using x and one of its nearest neighbors that can be either from the minority class or the majority class. Note that the difference between instance x and its neighbor from the majority class is multiplied by a random number between 0 and 0.5 (instead of a random number between 0 to 1). However, borderline-SMOTE may generate noisy instances if the minority class has noise. Both borderline-SMOTE1 and borderline-SMOTE2 use the DANGER set that may include noisy instances because a noisy instance may also be surrounded by many neighbors from the majority class.

ADASYN uses a density distribution (to measure the classification difficulty of instances) to determine how many synthetic instances are generated by each instance in the minority class. Besides, different from SMOTE, ADASYN does not generate synthetic instances for an instance (from the minority class) whose k-nearest neighbors do not contain any instance from the majority class. However, for ADASYN, noisy instances that exist in the minority class may have the highest priority because they are usually surrounded by instances from the majority class. Therefore, these noisy instances are selected to generate many synthetic instances (which are noisy and may mislead classifiers).

Oversampling methods can also be used with ensemble learning [19, 154], clustering [18, 161] and evolutionary computation [25, 31, 70, 178], etc.

Hybrid Sampling

Hybrid sampling is a combination of oversampling and undersampling, i.e. new instances are synthetically generated for the minority class by oversampling and then some of the less useful ones are removed by undersampling [151].

In summary, sampling methods have to change data distributions. Moreover, it is often hard for undersampling methods to avoid losing useful information when determining which instances are to be excluded from the majority class, particularly when a small number of instances are available. For oversampling methods, some instances are repeatedly learned or synthetically generated. Therefore, classification algorithms need to take a longer training time to develop classifiers, and may have a risk of generating noisy instances and over-fitting. In the thesis, we focus mainly on algorithm-level approaches.

2.2.2 Cost-sensitive Learning

Most classification algorithms assume that all the instances are equally important. However, in unbalanced classification, this assumption may result in a performance bias of the constructed classifiers [63]. Moreover, in many real-world applications, different mistakes usually lead to different losses. For example, in medical diagnosis, the mistake that a cancer patient is mistakenly diagnosed as a healthy person is much more troublesome than that of misdiagnosing a healthy person as a cancer patient.

Cost-sensitive learning [35] has emerged as an effective method for unbalanced classification, which can be used at both of the data and algorithmic levels. In cost-sensitive learning, the costs (mainly misclassification costs, i.e. the penalty of classifying an instance from one class to another) are considered in order to treat different mistakes differently. The misclassification costs are often provided by the domain experts, and later used by an algorithm to develop cost-sensitive classifiers. The goal of a cost-sensitive learning algorithm is to minimize the *total cost*, instead of only simply minimizing the number of mistakes [218]. Note that the total cost is calculated after predictions of a cost-sensitive classifier on a given

classification task.

Usually, the misclassification costs have two types [219]:

- *Class-dependent cost* (i.e. a class is associated with a cost);
- *Instance-dependent cost* (i.e. every instance is associated with a cost);

In the thesis, we mainly consider the class-dependent costs, and the minority class $Class\ 0$ and the majority class $Class\ 1$ are seen as the positive set and the negative set, respectively. A *cost matrix* is used to indicate the misclassification costs. A class-dependent cost matrix is shown as follows [35]:

$$C\_M = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix}$$

where C_{10} is a cost of a false negative, C_{01} is a cost of a false positive, C_{00} and C_{11} are the costs of a true positive and a true negative, respectively. In C_M, $C_{10} \geq C_{01}$, $C_{10} > C_{00}$ and $C_{01} > C_{11}$ [35].

Main Techniques in Cost-sensitive Learning

In general, cost-sensitive techniques are divided into two categories [57]:

- Changing the data distribution of the training data (at the data level).

  Data distribution is adjusted in proportion to corresponding costs [218]. Rescaling (also called rebalancing) is one of the most popular cost-sensitive approaches proposed to rebalance data [218]. Rescaling aims to adjust the influences of different classes to be proportional to their costs. This could be achieved via resampling or reweighting the data. For resampling, it removes some instances associated with lower misclassification costs to ensure that the number of instances from different classes is in proportion to their costs. For reweighting, it assigns a weight to each of the training instances based on its cost [187]. The value of a weight indicates the influence of a misclassification. For example, a larger weight is usually assigned to an instance that has a higher misclassification cost.

- Modifying the classification algorithms (at the algorithmic level).

  Classification algorithms are improved for use in unbalanced classification. In this category, cost-minimizing techniques (also called loss-minimizing) are used, or costs are directly incorporated into the classification mechanisms or objective functions to develop cost-sensitive classifiers [69, 140, 209, 217].

Usually, cost-sensitive learning has been more frequently used at the algorithmic level than the data level, and has been acknowledged as the most popular method at the algorithmic level.

Making Optimal Classification Predictions based on a Class-dependent Cost Matrix

In cost-sensitive learning, an optimal classification decision should cause the lowest expected cost [35]. For instance x, the expected cost of predicting x into $Class\ i$ can be calculated by the following equation [35]:

$$R(x,i) = \sum_j P(j|x)C_{ij} \tag{2.10}$$

where $P(j|x)$ is the probability of x belonging to class j; C_{ij} is a misclassification cost of predicting instance x into class i when its true class label is j. If $i = j$, then the prediction is correct; if $i \neq j$, then the prediction is incorrect.

Based on Eq. (2.10), the expected costs of classifying instance x to $Class\ 0$ or $Class\ 1$ are [35]:

$$R(x,0) = P(0|x)C_{00} + P(1|x)C_{01}$$
$$R(x,1) = P(0|x)C_{10} + P(1|x)C_{11}$$

Instance x is predicted to $Class\ 1$ if and only if $R(x,1) \leq R(x,0)$.

Given $P(1|x) = p$, $P(0|x) = 1-p$, $R(x,1) \leq R(x,0) \Rightarrow (1-p)C_{10} + pC_{11} \leq (1-p)C_{00} + pC_{01} \Rightarrow p \geq \frac{C_{10}-C_{00}}{C_{10}-C_{00}+C_{01}-C_{11}}$. Therefore, if $P(1|x) \geq \frac{C_{10}-C_{00}}{C_{10}-C_{00}+C_{01}-C_{11}}$, then x is classified into $Class\ 1$ [35].

2.2.3 Kernel Modification

Kernel modification methods [64] focus mainly on choosing a suitable kernel function for SVMs or other kernel algorithms to address the class imbalance issue.

2.2.4 One-class Classification

One-class classification (OCC) [39, 74] learns from a training set that includes only a specific class, in order to distinguish the instances of the specific class from others. OCC can be easily applied to binary unbalanced classification tasks, where the majority class is seen as the "normal" class and the minority class is seen as the "abnormal" class.

2.2.5 Active Learning

Active learning [164] (also called query learning) allows a learning algorithm to interactively query a user to label unlabeled instances with the desired outputs during the training process. This contributes an active learning algorithm to achieving better accuracy when learning from a few labeled training instances [164]. Active learning is also used to solve the class imbalance issue, which is often incorporated with kernel modification methods or with sampling methods [36, 220, 222].

2.3 Feature Selection and Feature Construction

Feature selection [55] is an important data preprocessing step to enhance the performance of algorithms by eliminating irrelevant or redundant features. It targets to select the smallest subset of features, and the selected features should be necessary and sufficient to describe the target labels. The classical definition of feature selection is to select m features from the n original features ($m < n$), in order to optimize the value of a criterion over all the subsets of the size m [123]. *Feature construction* [188] is to convert original feature space into new high-level features. Constructed features are mathematical expressions of the original features, which

are used to replace the original features. Feature construction is similar to feature selection, but the output is the new high-level features, rather than the selected features.

Based on the way to evaluate a feature subset, feature selection approaches are grouped into three categories, i.e. filter [97], wrapper [172, 173] and embedded approaches [181]. For filter approaches [97], the goodness of features is evaluated by measures or metrics, such as correlation [58, 208], distance [15], information gain [42], consistency [23, 98], and dependency [117]. Filter approaches are often cheaper and easier than wrapper and embedded approaches because they are independent of a classification algorithm and do not need a cross-validation process [181]. Different from filter approaches, wrapper approaches [172, 173] employ a classification algorithm, e.g. KNN, and NB, to evaluate the goodness of the selected features. The evaluation process is considered as a black box, and any classification algorithm can be employed. Compared with filter approaches, wrapper approaches usually achieve better performance, while they are more expensive. For embedded approaches [181], feature selection is built-in with the process of classifier construction, and they take advantage of both the wrapper and filter approaches.

2.4 Evolutionary Computation

2.4.1 Overview

Evolutionary Computation (EC) [4] is a class of heuristic techniques, inspired by biological evolution and natural selection. Generally speaking, EC techniques are grouped into three categories, i.e. evolutionary algorithms (EAs), swarm intelligence (SI) and other techniques. This section introduces the main concepts of EC, particularly GP that is the main focus of the thesis.

Evolutionary Algorithms (EAs)

EAs are a crucial branch of EC. The representatives of EAs include genetic algorithms (GAs) [65, 200], genetic programming (GP) [82], evolution strategy (ES) [7, 114, 153], and evolutionary programming (EP) [43, 44].

GA [65] is a population-based evolutionary search algorithm. In a population, a chromosome (i.e. a candidate solution) consists of genes that are the basic building blocks. In standard GAs, a fixed-size array is used to represent a chromosome. GAs adopt genetic operators, i.e. crossover and mutation, to create new offspring. Generally, there are two versions of GAs, i.e. binary GAs and continuous GAs.

GP targets to evolve computer programs as candidate solutions to a problem. Similar to GAs, it also adopts genetic operators. However, for standard GP, an individual is typically represented as a tree. The tree structure enables GP to be flexible for use in various tasks or applications. Apart from the tree structure, other representations, e.g. graph and linear representations, can also be used to represent a GP individual. GP is the main focus of the thesis, and it will be introduced in Section 2.5 (page 36) in detail.

ES has been applied to various optimization tasks. In ES, a pair of real vectors is used to represent an individual in a population [113]. In ES, the search process is mainly driven by a high mutation rate. The simplest ES is (1+1)-ES, which operates on a current individual (parent) and its mutated offspring [114]. If the fitness value of the mutated offspring is at least as good as its parent, then it becomes a new parent of the next generation. Otherwise, the new offspring is discarded. In $(1 + \lambda)$-ES , λ new offspring are generated to compete with their parent. The best individual becomes the parent of the next generation, and the current parent is always discarded.

EP is based on a finite state machine model in its early stage [199]. EP typically does not use crossover, and emphasizes on mutation. In general, EPs employ the Gaussian distributed mutation, i.e. a weight vector is perturbed with a zero mean multivariate Gaussian distribution. EP emphasizes on the behavioral linkage between parents and their offspring [199].

Swarm Intelligence (SI)

SI [51] algorithms are inspired by the collective intelligence of social insects or units. Two representative algorithms of SI are particle swarm optimization (PSO) and ant colony optimization (ACO).

PSO [72] is an effective search technique for continuous optimization problems, taken inspiration from social behaviors of birds flocking or fish schooling. In PSO, in a swarm, a candidate solution is encoded as a particle that moves to search for optimal solutions based on communications with other particles. The movements of the particles are guided by their own best known position ($pbest$) as well as the best known position in the entire swarm ($gbest$).

ACO [32] is another SI algorithm that mimics behaviors of ants seeking the shortest path between their colony and a destination. Every individual (i.e. an ant in a swarm) deposits pheromone on the ground to show their preferred path so that other ants could follow. The best path has the most amount of pheromone.

Other EC Techniques

Apart from EAs and SI, other EC techniques mainly include evolutionary multi-objective optimization (EMO), differential evolution (DE) [177], estimation of distribution algorithms (EDAs) [88], and artificial immune systems (AIS) [73], etc.

EMO refers to the use of EAs to search for solutions to a problem that is involved with two or more (usually conflicting) objective functions [21]. Representative EMO algorithms include Niched Pareto genetic algorithms (NPGA) [155], nondominated sorting genetic algorithm II (NSGA-II) [24], Pareto-archived evolution strategy (PAES) [78], a multiobjective evolutionary algorithm based on decomposition (MOEA/D) [214], the strength Pareto evolutionary algorithm (SPEA) [224] and SPEA 2 [223].

NPGA [155] incorporates the Pareto domination to its selection operator, and applies a niching pressure to spread its population along the Pareto front. In NSGA-II [24], a fast non-dominated sorting method and a selection operator are

presented to create a mating pool by combining the parent and child populations, in which top N ranked solutions are selected according to fitness values (where N is the population size). SPEA uses a regular population and an archive that is variable in size, while in SPEA2, the size of the archive population is constant [223]. Furthermore, compared to SPEA, SPEA2 uses a more specific fitness assignment scheme, protects the boundary solution from being removed, and incorporates density information into raw fitness values for diversity protection [223].

DE [147, 177] is a population-based algorithm to optimize real parameters or real-valued functions, based on a parallel search strategy. DE does not require that the objective function is differentiable, continuous, or linear. Different from GP, mutation is more important than crossover in DEs. The trial vector generation scheme is crucial, which could significantly influence DEs' performance [198].

EDAs [87, 88] are a group of probabilistic-based evolutionary algorithms. EDAs estimate the probability distribution of selected individuals at each generation, and then sample the probabilistic model to create a new population. EDAs regard individuals as a set of random variables [166]. Univariate EDAs assume that variables are marginally independent [47, 165, 166]. Bivariate EDAs consider dependencies between a pair of random variables, while multivariate EDAs consider the conditional independence among variables [87].

2.5 Genetic Programming

GP is a population-based approach, inspired by biological evolution. In GP, individuals are often structured in terms of trees, where nodes of a tree are chosen from a function set and a terminal set. A function set is constituted by all the possible internal nodes, which could be operators or functions, e.g. arithmetic operators or mathematics functions. A terminal set is constituted by terminals, i.e. inputs for the internal nodes. Note that terminals do not have arguments. For a classification task, a terminal set usually includes features of the task and random numbers. The size of a GP individual is usually limited by a maximum depth which is the longest path from the root to a leaf node.

GP [143] has the following steps:

1) Initialization: an initial population of individuals (or called trees or programs) is randomly generated.

2) The iteration of following steps is performed when the stopping criterion is satisfied:

2.1) Evaluation: the goodness of each individual is evaluated by a predefined fitness function.

2.2) Selection: individuals are selected from the current population based on their fitness values.

2.3) Evolution: new individuals are created by genetic operators (e.g. reproduction, mutation and crossover) with pre-defined probabilities.

3) Return the best individuals.

2.5.1 Initialization

In GP, three popular initialization approaches are full, grow and ramped half-and-half [143]. The full approach generates full trees where all the leaves are at the same depth. Internal nodes are randomly taken from a function set until the maximum tree depth is reached, and then leaf nodes are taken at random from a terminal set. The generated full trees have the same tree depth, but it does not mean that the generated trees definitely have the same tree size or shape [143]. However, for the generated trees by using the full approach, the range of tree sizes or shapes is still relatively limited. Different from the full approach, the grow approach is able to generate trees where nodes are taken randomly from both the function set and the terminal set until the maximum tree depth is reached. Once the maximum tree depth is reached, nodes can only be selected from the terminal set [143]. Since terminal nodes might appear at any depth level, the grow approach could generate initial trees with various sizes and shapes [143].

However, both the full and grow approaches are hard to create trees with a wide range of size or shapes [143]. Ramped half-and-half is a combination of the

full and grow approaches, which employs the full approach to initialize a half of the population and employs the grow approach to initialize the other half [143]. It has been acknowledged that Ramped half-and-half is able to create diverse trees with various sizes and shapes for the initial population.

2.5.2 Selection

In GP, two popular selection methods are proportional selection (also called roulette wheel selection) and tournament selection. Proportional selection probabilistically selects individuals based on their fitness values [143]. As a result, good individuals are more likely to produce more offspring than inferior individuals. However, in some cases, the population may suffer from an issue of being stuck in a local optima. In order to overcome this drawback, tournament selection randomly chooses a number of individuals in a tournament and then selects the best from the tournament [143]. Moreover, tournament selection provides consistent and parameterized selection pressure by allowing designers to choose a tournament size.

2.5.3 Genetic Operators

The selected individuals are used as parents to create offspring by applying genetic operators, such as crossover, mutation and reproduction, based on different probabilities.

The most commonly used crossover operator is subtree crossover, which randomly chooses a crossover point in each parent tree (two parents given) and generates an offspring by exchanging the subtrees rooted at each crossover point in the corresponding parent tree. It is also possible to design a crossover operator to produce two offspring [143]. There are several types of crossover, e.g. uniform crossover [144], context-preserving crossover [56] and size-fair crossover [86].

The most popular mutation operator is subtree mutation, which uniformly chooses a mutation point in a parent tree and then replaces the subtree rooted at the mutation point with a randomly generated subtree. There are many types of mu-

tation methods in use, e.g. size-fair subtree mutation [85], shrink mutation [163] and hoist mutation [6].

For the reproduction operator, good individuals are selected based on its fitness values and directly inserted into the next generation. Note that the sum of the crossover rate, mutation rate and the reproduction rate is equal to 1, to maintain the same number of individuals in a population across different generations.

2.5.4 Variants of GP

Apart from standard tree-based GP, variants of GP have been introduced and proposed, including strongly-typed genetic programming [118], grammar-guided genetic programming [201], etc.

Strongly-Typed Genetic Programming (STGP): STGP enforces data type constraints on functions and terminals when used to generate strongly-typed expressions [118]. In STGP, every terminal has a data type, and correspondingly, every function has types for its arguments and the returned value [118]. STGP could reduce the search space because the generated trees cannot violate the type constraints.

Grammar-Guided Genetic Programming (G3P): G3P can evolve programs in any language described by a context-free grammar (CFG). The commonly-used grammar is the Backus-Naur form (BNF) grammar. The BNF grammar describes language constructs through a 4-tuple $G = \{S, N, T, P\}$, where S denotes a start symbol, N denotes a non-terminal set, T is a terminal set, and P is a set of production rules [152]. A derivation syntax tree starts with the start symbol S, and then is created according to production rules in the set P, where internal nodes are taken from N and leaf nodes are taken from T [108].

Stack-based Genetic Programming (SGP): Different from tree-based GP, in SGP, programs are represented by a stack-based programming language [141]. In standard SGP, the programs are LISP s-expressions, each of which is constituted

by a set of functions and terminals. All the functions receive arguments from a numerical stack and the returned result is pushed to the stack [141]. Terminals are a set of functions that push preset variables into the stack when they are executed.

Linear Genetic Programming (LGP): Most computer programs are represented in terms of linear sequences, and computers do not run tree-based programs, so they require the use of interpreters or compilers [143]. Accordingly, LGP has been proposed, and individuals are represented by linear sequences (or strings) of instructions [143]. In LGP, instructions typically involve 3 registers, where the register 0 shows the program output at termination [20].

Cartesian Genetic Programming (CGP): CGP [116] is an efficient and flexible version of GP that uses a graph to represent a program. Graphs are often easily applied to many fields, e.g. electronic circuits and neural networks. Standard CGP uses an integer-based genetic representation in the form of a directed graph, which has been extended, e.g. modular CGP [196] and self-modifying CGP [61].

Probabilistic Model Building in Genetic Programming (PMB-GP): The first PMB-GP system is probabilistic incremental program evolution (PIPE), which introduces a probabilistic prototype tree as an individual [159]. PIPE iteratively generates successive populations of functional programs according to an adaptive probability distribution over all the possible programs. During every iteration, the best program is used to refine the distribution, and thereby better programs are generated randomly and incrementally. PIPE saves lots of time in fitness evaluation since the distribution refinements depend only on the best individual of the current population [159].

Estimation of distribution programming (EDP) employs bivariate conditionally dependent variables [77]. It is based on a probability distribution expression using a Bayesian network, and individuals are generated based on the estimated distribution [206]. Extended compact genetic programming (ECGP) [77, 162] uses multivariate conditionally dependent variables, which adaptively identifies

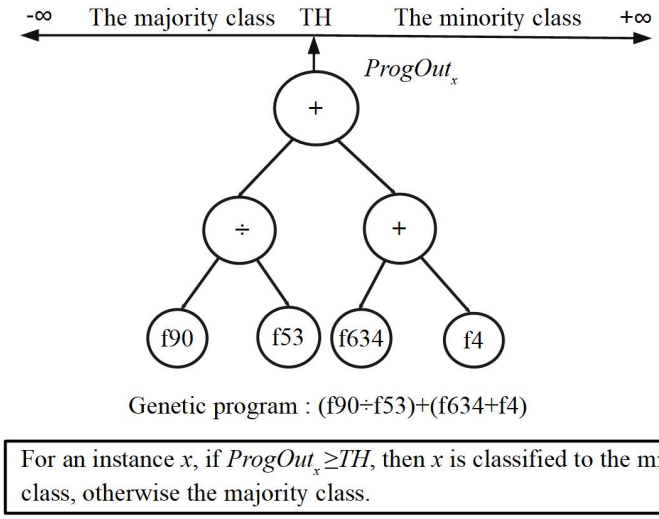


Figure 2.2: An example of a GP classifier.

and exchanges non-overlapping building blocks.

2.5.5 GP for Binary Classification

Because of the flexible tree representation, various classification tasks can be performed by using the different kinds of models, such as rules, decision trees, and discriminant functions [37]. Generally, a GP tree can be translated into an arithmetic expression, and the output of the expression is used to classify instances. In binary classification, a classification threshold is set to separate two classes (usually, the threshold is set to 0). An instance is classified into $Class\ 0$ if the program output is greater than 0; otherwise, the input instance is classified into $Class\ 1$.

In Figure 2.2, we show a GP classifier, where $\div$ and $+$ are taken from a function set and $f90$, $f53$, $f634$ and $f4$ (i.e. features) are taken from a terminal set. This GP tree can be translated into an arithmetic expression $(f90 \div f53)+(f643+f4)$. When instance x is input to this arithmetic expression, an output value (i.e. $ProgOut_x$) is obtained. If $ProgOut_x$ is greater than or equal to a threshold TH, this instance is classified into the minority class; otherwise it is classified into the

majority class.

In classification, the accuracy measures are often chosen as the fitness function to evaluate the performance of an individual. A widely-used fitness function for classification is the overall classification accuracy Acc (or the error rate), which measures a proportion of how many instances are correctly (or incorrectly) classified among the total number of the training instances. However, using Acc as the fitness function may cause GP to be biased towards the majority class when a dataset is unbalanced.

2.6 Rough Sets and Three Schemes for Class Overlapping Problems

This selection is devoted to introducing the techniques that will be used in Chapter 7 to detect overlapping areas.

2.6.1 Rough Sets

Rough set theory (RST) is a method to analyze incertitude information, based on the idea of granulation [136]. In standard RST, the equivalence relation is employed to divide an universe of discourse (denoted as U) to obtain granules of knowledge, and then use them to calculate two crisp sets (i.e. the upper approximation set and lower approximation set). A boundary set is the difference set of the two crisp sets.

In an information system $S = (U, A)$, A is the set of attributes (note that a feature can be seen as an attribute). Given $R \subseteq A$, if $\forall f \in R$, $f(x) = f(y)$ (note that x and y are from U). It means that x is indiscernible from y, denoted as xRy. The equivalence class of x is $[x]_R = \{y \in U | xRy\}$ [136, 137].

Definition 2.1 [136] Given $X \subseteq U$, the R-lower approximation of X is defined as:

$$R_{-}(X) = \{x \in U | [x]_R \subseteq X\} \tag{2.11}$$

the R-upper approximation of X is defined as:

$$R^{-}(X) = \{x \in U | [x]_R \cap X \neq \varnothing\} \tag{2.12}$$

and the R-boundary of X is defined as:

$$Bn_R(X) = R^{-}(X) - R_{-}(X) \tag{2.13}$$

Neighborhood Rough Set

$\forall x \in U$, the neighborhood of x is denoted as $\sigma(x) = \{y | y \in U, \nabla(x, y) \leq \sigma\}$, where ∇ is a metric (usually, ∇ is a distance measure, such as Euclidean distance or Minkowski distance) and σ is a threshold [212]. Note that the equivalence class $[x]_R$ could be seen as the neighborhood of x [207].

Definition 2.2 [212] Given $X \subseteq U$, the N-lower approximation of X is defined as:

$$N_{-}(X) = \{x \in U | \sigma(x) \subseteq X\} \tag{2.14}$$

the N-upper approximation of X is defined as:

$$N^{-}(X) = \{x \in U | \sigma(x) \cap X \neq \varnothing\} \tag{2.15}$$

and the N-boundary of X is defined as:

$$Bn_n(X) = N^{-}(X) - N_{-}(X) \tag{2.16}$$

2.6.2 Three Schemes for Class Overlapping Problems

There are usually three schemes to deal with the class overlap issue, including the discarding scheme, the merging scheme and the separating scheme [203].

The discarding scheme: After discovering overlapping areas, this scheme discards instances in the overlapping areas, and the rest of the instances are used to train a classifier.

The merging scheme: Two classifiers are trained. The first is trained to distinguish between overlapping and non-overlapping areas, and it directly classifies the instances in the latter. The second classifier is trained to classify the instances in the overlapping areas identified by the first classifier.

The separating scheme: After distinguishing between non-overlapping and overlapping areas, one classifier is trained for classifying instances in the overlapping areas, while the other is trained for classifying instances in the non-overlapping areas.

Note that, in the separating scheme, overlapping areas are distinguished from a training set before two classifiers are trained. This is different from that in the merging scheme.

2.7 Related Work

2.7.1 GP for Classification with Unbalanced Data

In GP, the methods to solve the problem of class imbalance can be categorized into two groups, i.e. at the data level and at the algorithmic level.

Data-Level Methods

At the data level, traditional sampling methods (e.g. undersampling and oversampling) can also be used to resolve the issue of class imbalance. In [104], a GP method was proposed, where SMOTE was used to address the class imbalance issue. Apart from the traditional sampling methods, subset selection methods can also be used to solve the issue of class imbalance. Subset selection methods mainly include random subset selection (RSS), dynamic subset selection (DSS) and historical subset selection (HSS) [52]. These methods can select a number of instances from the majority class, so that the number of the majority class is the same or similar to that of the minority class. The main difference of the three methods lies in the way to select a subset of instances from the training set. For

RSS, during each generation, instances are randomly selected, while DSS tends to choose instances which are easily misclassified or have not been selected in several previous generations [22, 52]. HSS selects instances based on the classification difficulty of each instance, determined by how many times an instance is incorrectly classified by the best GP programs from each of the generations in previous GP runs [52]. Song et al. [171] designed a two-layer subset selection sampling approach to linear GP, where the first layer was based on RSS and then instances in the second layer were sampled by DSS. Curry et al. [22] proposed a family of hierarchical DSS algorithms, such as cascaded RSS-DSS and balanced-block DSS, for large datasets.

Hunt et al. [67] investigated and proposed static and dynamic sampling methods with GP for classification with unbalanced data. For the proposed static sampling method, the training set is uniformly sampled to obtain balanced data before the evolutionary learning process, and then the balanced training set is used to train classifiers. For the proposed dynamic sampling methods, the original training set is sampled to obtain balanced data at each generation during the evolutionary learning process. In [33], a subset of instances were uniformly selected and used in the fitness evaluation process, and an AUC approximation metric was proposed as the fitness function of GP.

Khanchi et al. [75] investigated the use of GP with active learning and designed a framework for streaming unbalanced data in botnet detection. Khanchi et al. [76] further investigated how the issue of class imbalance can be addressed for streaming data classification in botnet detection by using GP and active learning. Hamida et al. [59] introduced active learning to GP, and proposed a new adaptive sampling strategy to obtain a subset of training instances in the fitness evaluation process.

Algorithm-Level Methods

At the algorithmic level, there are three main methods to solve the problem of class imbalance in GP, including cost-sensitive learning, development of fitness functions, and EMO.

Li et al. [93] showed how cost-sensitive learning can be employed by grammar-guided GP in unbalanced classification. Cost values in a cost matrix were directly embedded in the fitness function. In [2], a cost-sensitive GP method was proposed, where the cost values of two classes were also incorporated into the fitness function for churn prediction and identification of the influencing factors in the telecommunication market. However, for most existing cost-sensitive methods, the cost matrix is manually-designed for a specific problem. In many real-world applications, the cost information is unknown or unavailable.

Using a fitness function for resolving the issue of class imbalance is another popular and straightforward method in GP. In [1, 11, 14, 27, 135], the effectiveness of using fitness functions to avoid the performance bias issue was investigated and new fitness functions were designed and proposed for GP in classification with unbalanced data. A commonly used fitness function is Ave (Eq. (2.7), Page 24), which is the weighted sum of the true positive rate (sensitivity) and the true negative rate (specificity). Ave needs a weighting coefficient W, to specify the relative importance of the majority class to the minority class. Usually, W is often not easy to determine without domain knowledge [11]. F-measure (Eq. (2.5), Page 23) [145] is a combination of $Precision$ and $Recall$, which also needs a coefficient to adjust the relative importance of $Precision$ versus $Recall$. G_Mean (Eq. (2.6), Page 24) is the square root of the product of sensitivity and specificity.

Bhowan et al. [11] proposed two fitness functions to improve Ave for GP. One new fitness function in [11], called the average mean squared error ($Amse$), utilizes the magnitude of a program output to calibrate the output to the defined target. Another fitness function in [11], named $Incr$, assigns incremental rewards to programs whose predictions fall further away from the defined classification threshold. GP using AUC as the fitness function achieves promising performance for unbalanced classification, while it needs to consume very long training time. In [11], in order to reduce the training time, two AUC approximation measures were proposed to measure how far program outputs for two classes are separated with each other.

By using EMO to develop multi-objective GP (MOGP) for unbalanced classi-

fication, the true positive rate and the true negative rate are often seen as two potentially conflicting objectives. Bhowan et al. [10] proposed a MOGP method based on NSGA-II, employing the negative correlation learning-based (NCL) measure as a diversity measure. However, NCL caused a substantial computational cost. Bhowan et al. [9] developed an evolutionary-based pruning method to find groups of highly cooperative individuals that improve the accuracy of the minority class. Bhowan et al. [12] examined the effectiveness of two EMO algorithms, i.e. SPEA2 and NSGA-II, to develop MOGP methods, and also investigated how the diversity of solutions can be encouraged. Bhowan et al. [13] designed a two-step MOGP approach. In the first step, a MOGP method based on SPEA2 was developed to form ensembles, and in the second step, an ensemble selection approach was proposed to reuse GP trees to automatically choose the best classifiers or a combination of classifiers in the ensemble. Mauvsa et al. [112] presented a new multi-subpopulation MOGP, where the subpopulations were trained individually and forward migrations performed after a number of generations, and introduced a co-evolutionary MOGP method to co-evolve two MOGP methods. In [174], semantics were incorporated into MOGP based on MOEA/D in classification with unbalanced data.

2.7.2 GP for Feature Selection and Feature Construction in Classification

Tran et al. [188] introduced GP to feature construction and feature selection in classification with high-dimensional data. To significantly reduce the search space and computational costs of GP, in [190], features were grouped into multiple clusters, and the best features from each of the clusters were fed into GP for feature construction. In [188–192], to avoid the performance bias issue when data is unbalanced, Ave (Eq. (2.7), $W = 0.5$, Page 24) was employed as the fitness function of GP.

Muni et al. [120] proposed an online feature selection approach based on multi-tree GP, which could simultaneously select informative features and develop

a classifier using the selected features. In the proposed method in [120], two new crossover operators were designed to enhance the performance of GP for feature selection. Purohit et al. [148] developed a modified crossover operator for GP, and the developed GP approach could also simultaneously select informative features and construct a classifier that consists of c trees for c-class classification tasks.

Neshatian et al. [125] proposed a filter approach to feature selection, which aimed to increase the depth limit of individuals during the run time to explore larger subsets of features and decrease risks of bloating and overfitting. Neshatian et al. [124] proposed a wrapper approach to feature selection, where GP was employed to search for optimal feature subsets and an extended version of NB was used to examine the quality of the selected features. Hunt et al. [68] proposed a hyper-heuristic approach to feature selection based on GP, which evolved new heuristics using some building blocks for searching for optimal feature subsets. Nag et al. [122] proposed a MOGP-based integrated method that simultaneously performed feature selection and classifier construction. The developed method in [122] decomposed a c-class classification task into c binary classification tasks, and it calculated the fitness as well as unfitness of features during the mutation operation.

Krawiec [83] introduced a general framework of GP-based feature construction, and proposed a method to preserve informative constructed features during the evolutionary learning process. Otero et al. [132] proposed a GP-based filter approach to feature construction, which could construct continuous (or real-valued) or Boolean features, and the fitness function was designed as an information gain ratio. Neshatian et al. [127] proposed a filter approach to multi-feature construction, where a fitness function was designed based on the class dispersion and entropy. Neshatian et al. [126] proposed a GP-based filter approach to feature construction, where an entropy-based fitness function was designed to maximize the purity of class intervals. Smith et al. [169] proposed an integrated approach based on GP and GA, where GP was used to construct new high-level features and GA was used for feature selection. Tariq et al. [183] proposed an efficient feature construction method based on random projections and GP, where a Mathew's cor-

relation coefficient was employed as the fitness function. Mahanipour et al. [111] proposed a novel method, where a fuzzy rough set was used for feature selection and then the selected features were fed to GP for feature construction.

Recently, most GP-based feature selection and feature construction approaches did not typically consider addressing the class imbalance issue. Because of uneven data distributions and high dimensionality issues, feature selection becomes more challenging.

2.7.3 Other Related Work

Cost-sensitive Learning

Zhang and Zhou [217] developed cost-sensitive kernel logistic regression and cost-sensitive k-nearest neighbor, and applied them to face recognition. In [5], a cost-sensitive DT algorithm was proposed, which incorporated instance-dependent misclassification costs into an impurity measure and pruning criteria. Iranmehr et al. [69] developed a new cost-sensitive SVMs with an extension of a standard loss function to optimize a classifier with respect to class-dependent costs and instance-dependent costs. Cost-sensitive learning is frequently applied to binary classification, but it is less investigated for multi-class classification. Zhou et al. [219] investigated the use of cost-sensitive learning for multi-class classification, and suggested examining the consistency of costs before using rescaling to improve the classification performance. However, for many existing cost-sensitive methods, cost matrices are often manually-designed for a specific problem. The misclassification costs in a cost matrix are often given as precise values. Unfortunately, it is usually not easy for domain experts to accurately specify or assign the precise cost values to different kinds of mistakes. Liu et al. [102] investigated how cost intervals and cost distributions can be used to develop cost-sensitive SVMs. The proposed methods still required domain experts to specify cost intervals or cost distributions in advance.

When no cost information is available, the easiest method is to use the class imbalance ratio of a dataset to construct a cost matrix [38]. However, this method

is often criticized because it is over-simplified without considering the data characteristics [38]. Gu et al. [53] proposed a bi-parameter space partition algorithm to fit all solutions for each cost parameter pair, and then k invariant regions were superposed to one for calculating the global optimal solutions for cost-sensitive SVMs.

In [142], GA was used to optimize a class-dependent cost matrix. In this study, each gene in a chromosome indicates an element in a cost matrix. For example, for a binary classification task, a chromosome has four genes, i.e. C_{10}, C_{01}, C_{00} and C_{11} in C_M (on Page 30). Initial values of C_{00} and C_{11} are 0, and initial values of C_{10} and C_{01} are randomly taken from the range of (0, 100). Zhang et al. [209, 210] investigated the use of an adaptive DE to optimize a class-dependent cost matrix for cost-sensitive deep belief networks. In each candidate solution vector, an element represents a cost value for a corresponding class. The learned cost values for different classes are used by cost-sensitive deep belief networks, and its classification performance is evaluated by G_Mean. Shen et al. [167] proposed dynamic cost-sensitive deep belief networks, where DE was used to optimize an instance-independent cost matrix. Zhang et al. [213] designed a cost-sensitive method that integrated an instance-dependent cost matrix into extreme learning machines, where the backtracking search optimization algorithm was employed to optimize the needed cost matrix. Note that the search space of optimizing an instance-dependent cost matrix is significantly larger than that of optimizing a class-dependent cost matrix. This potentially requires a larger population size and more number of generations. Padurariu et al. [133] proposed a two-step approach, which used DE to optimize a class-independent cost matrix in the first step and then to optimize an instance-independent cost matrix in the next step.

To date, there is no existing work that uses GP for optimizing cost matrices to automatically develop cost-sensitive GP classifiers.

Methods for Addressing the Issue of Class Overlap

In [50, 146], the issue of class overlap and its influences in unbalanced classification were investigated. In [203], support vector data description (SVDD) was

used to detect overlapping areas in the training set, based on the idea that the data dropped into two spheres is considered as the overlapping data (a sphere is trying to contain all or most of the instances from a class). Unfortunately, when the within-class imbalance exists, a class has multiple sub-clusters, each of which may overlap with the other class. In this case, it is not easy for SVDD to accurately detect overlapping areas by finding two spheres.

In [28], overlapping areas were detected by one-class SVM and treated as outliers. Tang et al. [182] adopted a probabilistic neural network, where overlapping areas were detected by a margin at each side of a decision boundary. Lee et al. [90] designed a method to train overlap-sensitive margin classifiers based on a modified fuzzy SVM and KNN. In [90], KNN was used to measure the overlap degree for identifying the overlapping and non-overlapping areas. In [195], for each instance, the membership of its neighbors was used to judge whether this instance is located in the overlapping areas or not.

2.8 Chapter Summary

This chapter introduces the essential background and basic concepts, including classification algorithms in ML, performance measures, main techniques in unbalanced classification, and EC techniques (particularly GP). This chapter also reviews the related work. The main limitations of the existing methods are summarized as follows:

- For GP-based feature selection approaches, most of them have not typically considered addressing the class imbalance issue. Because of uneven data distributions and high dimensionality issues, feature selection becomes more challenging.

- Using a suitable fitness function can directly resolve the class imbalance issue in GP. The use of AUC as the fitness function has been proven to be effective but very time-consuming.

- The use of cost-sensitive learning with GP has seldom been investigated. Moreover, many cost-sensitive learning methods depend on cost matrices that are usually manually-designed. It is of importance to investigate how GP can be used to construct cost-sensitive classifiers when the manually-designed cost matrices are not available.

- When class overlap is intertwined with the issues of class imbalance and high dimensionality, it is often more challenging to discover useful patterns because of an ambiguous boundary between the majority class and the minority class. Particularly for cost-sensitive learning methods, they treat the minority class as being more important than the majority class, which may cause an accuracy decrease in the overlapping areas where the prior probabilities of the two classes are about the same. Therefore, it is necessary to address the class overlap issue in high-dimensional unbalanced classification.

In the following chapters, we will investigate and introduce how the above-mentioned limitations can be addressed. More specifically, Chapters 3 and 4 investigate the use of fitness functions, Chapters 5 and 6 investigate the use of cost-sensitive learning with GP, and Chapter 7 investigates how the class overlap issue can be addressed in high-dimensional unbalanced data classification.

Chapter 3

GP with a New Fitness Function and Program Reuse Mechanism

3.1 Introduction

High-dimensional unbalanced classification is challenging due mainly to the joint effects of high dimensionality and class imbalance. When GP is used to construct classifiers, it has a built-in capability to automatically select informative features that can improve the classification performance. Therefore, GP has potential benefits for use in high-dimensional classification (the reasons for using GP were discussed in Section 1.2.2 of Chapter 1, on Page 6).

Learning from unbalanced data, GP tends to develop biased classifiers which achieve a high accuracy on the majority class but a low accuracy on the minority class. Unfortunately, the minority class is often the class of interest in many real-world applications. The biased classifiers are built because GP uses the overall classification accuracy Acc as the fitness function when data is unbalanced. As a result, these biased classifiers are overestimated in the fitness evaluation process because the majority class has a significantly larger number of instances than the minority class to increase the overall classification accuracy, and then they are likely to be mistakenly selected by the selection operator to create offspring.

AUC has been acknowledged as a reliable measure to examine the classifica-

tion capability of classifiers in unbalanced classification [11]. AUC can be used as a fitness function to replace Acc to achieve better performance in unbalanced classification, but it is computationally expensive [11]. To reduce the computational costs of the fitness evaluation process, AUC is approximated. However, for an approximation measure, the improvement in efficiency is often at the expense of the decreased classification performance.

In this chapter, we investigate how GP can be effectively and efficiently utilized for high-dimensional unbalanced classification. To improve the classification performance, we propose to evaluate each GP program using two criteria, i.e. AUC approximation and classification clarity (to measure how well a program can separate the two classes). To improve the efficiency, we design a program reuse mechanism to reuse previous good GP individuals.

3.1.1 Chapter Goals

The overall goal of this chapter is to develop a new GP method for classification with high-dimensional unbalanced data, to increase the classification performance as well as save the training time. The overall goal is composed of the following three sub-goals:

1) Develop a fitness function to address the problem of class imbalance,

2) Develop a program reuse mechanism to reuse previous good GP trees, and

3) Investigate whether GP with the proposed fitness function and reuse mechanism can achieve significantly better or similar performance in classification with high-dimensional unbalanced data, compared with other existing classification algorithms.

3.1.2 Chapter Organization

The rest of this chapter is organized as follows. Section 3.2 introduces the proposed GP method. Section 3.3 presents the experimental design. The results are

discussed in Section 3.4 and further analyzed in Section 3.5. Section 3.6 draws the conclusions and summarizes this chapter.

3.2 The Proposed Method

In this section, we introduce a new GP method, called **G**enetic **P**rogramming with a New **F**itness Function and **R**euse **M**echanism (**GPFRM**). The majority class and the minority class are denoted as Maj and Min, respectively.

3.2.1 Two-criterion Fitness Function

In the proposed fitness function, it considers two criteria, i.e. AUC approximation and classification clarity (to measure how well a program can separate the two classes).

(1) Criterion 1: AUC Approximation

Wilcoxon-Mann-Whitney (denoted as Auc_w) provides a direct estimator for AUC metric [11, 205], which is defined as:

$$Auc_w = \frac{\sum_{i \in Min} \sum_{j \in Maj} I_{wmw}(P_i, P_j)}{|Min| * |Maj|} \tag{3.1}$$

where $I_{wmw}(P_i, P_j) = \begin{cases} 1, & P_i > P_j \; and \; P_i \geq 0 \\ 0, & otherwise \end{cases}$

P_i and P_j are two output values of a program P when instance $i \in Min$ and $j \in Maj$ are input to P. $|Min|$ and $|Maj|$ stand for the number of instances in Min and Maj, respectively.

Using Auc_w to evaluate a program P, when an instance $i \in Min$ is input to P, its output value (i.e. P_i) needs to be compared with all possible P_j ($\forall j \in Maj$). Therefore, Auc_w is computationally intensive due to $|Min|*|Maj|$ times pairwise comparisons to evaluate a program. To approximate Auc_w, $\forall i \in Min$, the output value P_i is expected to be greater than t (where t is the maximum output value of

P taking Maj as the inputs) [139]. Therefore, after determining t, $|Min|$ times pairwise comparisons are required to evaluate a program. This criterion (denoted as $C1$) is defined as:

$$C1 = \frac{\sum_{i \in Min} I(P_i, t)}{|Min|} \tag{3.2}$$

where $I(P_i, t) = \begin{cases} 1, & P_i > t \; and \; P_i \geq 0 \\ 0, & otherwise \end{cases}$, $t = max\{\cup P_j, \forall j \in Maj\}$.
$C1$ has a linear time complexity in an evaluation, compared with Auc_w that has a quadratic time complexity $O(|Maj| * |Min|)$.

The discrimination ability of $C1$ has a close relationship with the number of instances in the minority class. This thesis is targeted at high-dimensional unbalanced classification, particularly for tasks where features typically outnumber instances. Therefore, the minority class usually does not contain many instances. This may cause multiple programs in a tournament to have the same $C1$ value.

(2) Criterion 2: Classification clarity

It is of importance to measure how well a program can separate the majority class and the minority class, which is defined as classification clarity here. This could further distinguish between two programs that achieve the same fitness value on $C1$ to identify a better one. Figure 3.1 explains the importance of considering the classification clarity. In Figure 3.1, if $C1$ is considered as a fitness function, program 1 and program 2 have the same fitness value, but if the classification clarity is also considered, program 2 is preferred to program 1 because it has better clarity.

In this chapter, we use the correlation ratio [41] to evaluate the classification clarity of a program. The output values of the correlation ratio are in the range of [0, 1], where 0 indicates the worst clarity and 1 indicates the best clarity. Correlation ratio (denoted as $C2$) is defined as:

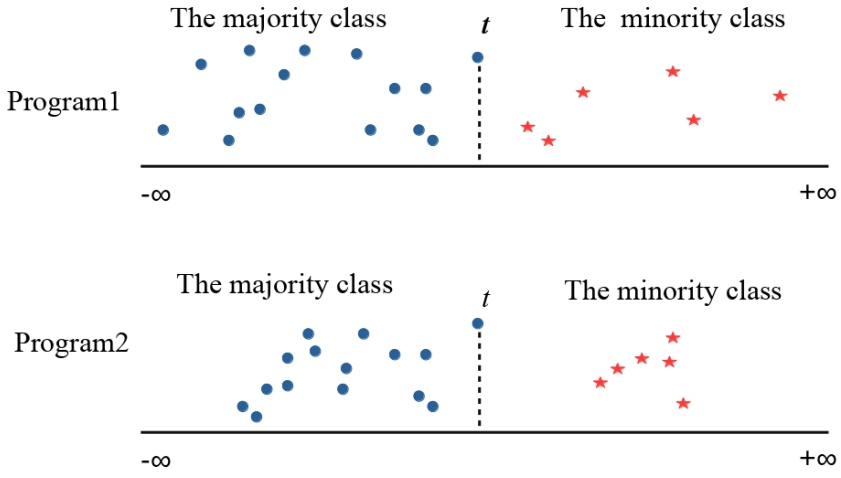


Figure 3.1: Importance of the classification clarity when comparing two programs.

$$C2 = \sqrt{\frac{\sum_{c=1}^{K} N_c(\mu_c - \overline{\mu})^2}{\sum_{c=1}^{K}\sum_{i=1}^{N_c}(P_{ci} - \overline{\mu})^2}} \tag{3.3}$$

where K is the number of classes ($K = 2$ in binary classification), N_c is the number of instances in class c, P_{ci} stands for an output value of a genetic program taking instance i from a class c as an input, $\mu_c = \frac{\sum_{i=1}^{N_c} P_{ci}}{N_c}$ and $\overline{\mu} = \frac{\sum_{c=1}^{K} N_c\mu_c}{\sum_{c=1}^{K} N_c}$.

(3) Overall Fitness Function

Based on Eqs. (3.2) and (3.3), a new two-criterion fitness function is designed as:

$$Min\_Corr = C1 + C2 = \frac{\sum_{i\in Min} I(P_i, t)}{|Min|} + \sqrt{\frac{\sum_{c=1}^{K} N_c(\mu_c - \overline{\mu})^2}{\sum_{c=1}^{K}\sum_{i=1}^{N_c}(P_{ci} - \overline{\mu})^2}} \tag{3.4}$$

3.2.2 Program Reuse Mechanism

When GP is used for classification, all the features in a dataset are usually fed into GP as terminals to build classifiers. However, for high-dimensional data,

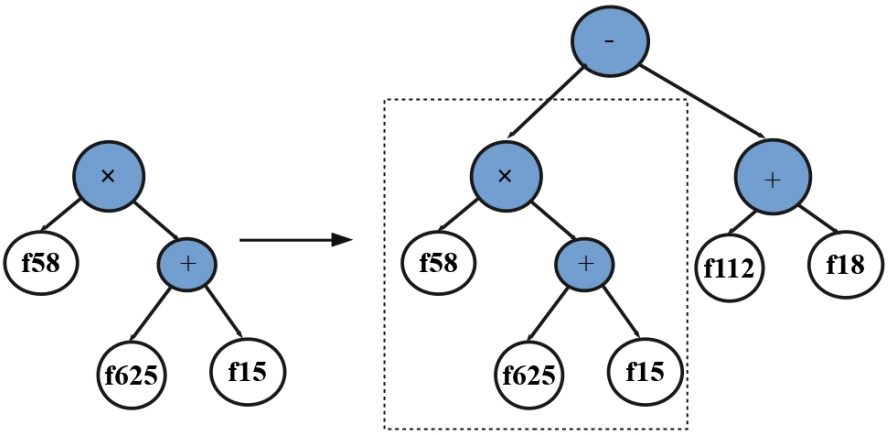


Figure 3.2: Reuse of programs in initialization.

the search space is huge. In this chapter, all features are randomly divided into five feature sets (denoted as $F1$, $F2$, $F3$, $F4$ and $F5$) with roughly the same size. The whole evolutionary learning process has five sub-processes (denoted as GP1, GP2, GP3, GP4 and GP5). For the first sub-process, the ramped half-and-half initialization method is employed to create an initial population, and the first feature set $F1$ is fed into GP as terminals to evolve classifiers. After GP finishes its search in the first sub-process, good programs are obtained and the features selected by the good programs are saved as good features (denoted as $F1^*$). Note that $F1^*$ is expected to have the similar discrimination ability as its original set $F1$.

Apart from the good features in $F1^*$, good programs could carry useful information, so they are worth being reused by the following sub-processes to further enhance the effectiveness and efficiency. Figure 3.2 explains how a good program is reused. In Figure 3.2, the program on the left is a good GP program that was evolved previously, which is reused as a terminal of the program on the right in the next sub-process.

After each evolutionary sub-process, the top 1% programs are reused by the

next sub-process. The reuse mechanism starts from the second sub-process, and ends with the fifth sub-process. Note that, each sub-process (after the first sub-process) only reuses good programs from an earlier sub-process. For example, for the third sub-process (i.e. GP3), it only reuses good programs from the second GP sub-process (i.e. GP2).

3.2.3 Overall Design of GPFRM

The overall design of GPFRM in the training process is shown in Figure 3.3.

$F1$ is fed to GP1 as terminals, using ramped half-and-half for initialization. The proposed fitness function is used to evaluate the goodness of each individual in a population. After GP1 finishes its search, top 1% good GP programs and features selected by these programs (i.e. $F1^*$) are saved. These good programs are reused as a part of the terminal set in the initialization of GP2, and $F1^*$ combined with $F2$ is fed to GP2. The similar processes continue until GP5 finishes to evolve classifiers. The best individual from each sub-process is chosen to obtain five trained classifiers, and the majority voting is used to make a final decision on each unseen instance in the test process.

3.3 Experiment Design

3.3.1 Datasets

Table 3.1 describes the key information of the used high-dimensional unbalanced datasets (gene expression data) in the experiments. Clearly, these unbalanced datasets have a large number of features. However, many gene expression datasets have a relatively low imbalance ratio (IR). The main reason is that neither the majority class nor the minority class has a sufficient number of instances in a high-dimensional dataset. In order to examine the classification performance of GPFRM on datasets with higher class imbalance ratios, Tomlins-2006-v1 (5 classes) and Lung (5 classes) are changed into binary datasets. For Tomlins-2006-v1, class 5 (“STROM”, 12 instances) is used as the minority class, while the rest

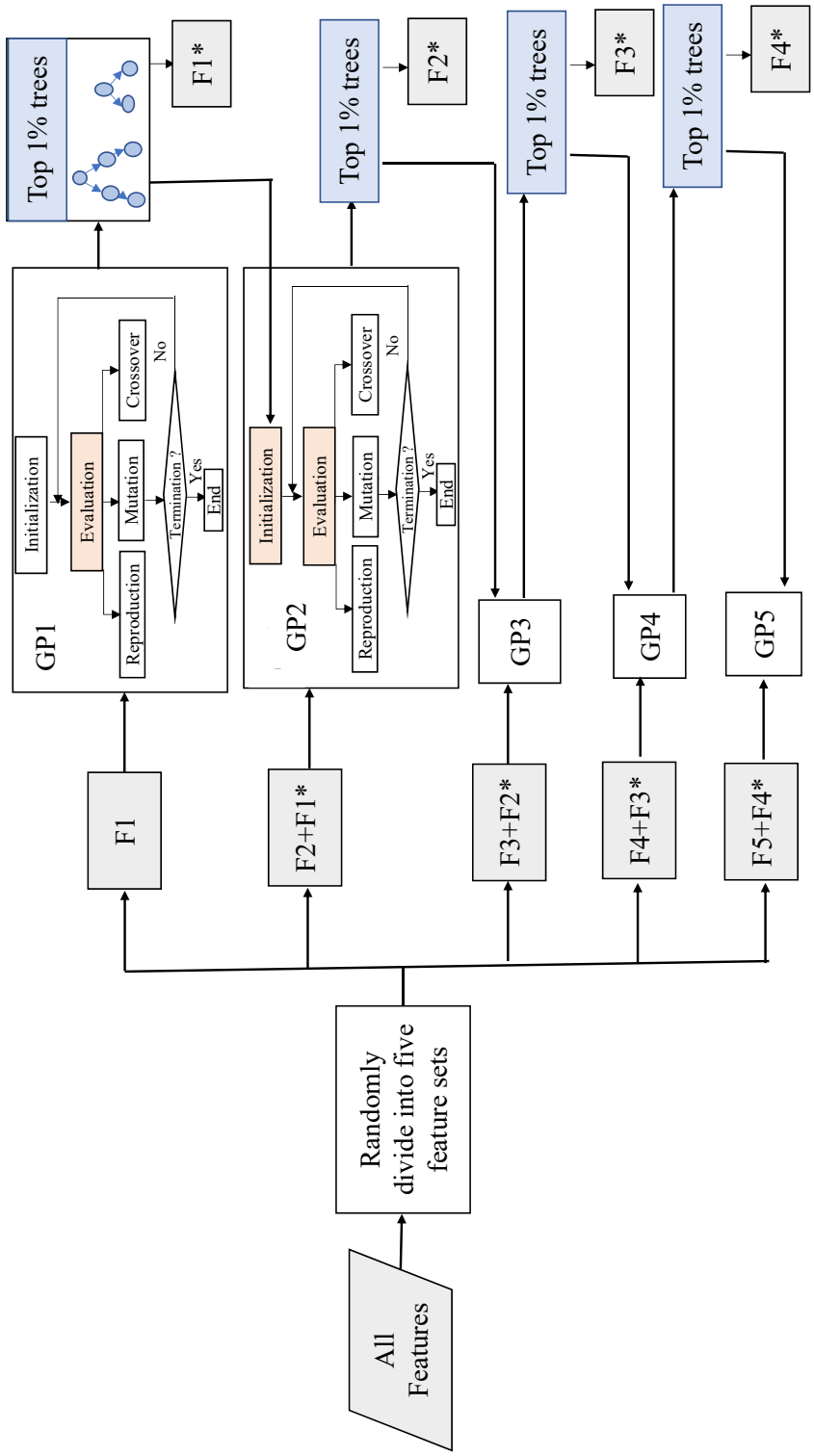


Figure 3.3: Overall design of GPFRM.

of classes are combined together and used as the majority class (92 instances). For Lung, class 1 ("AD", 139 instances) is used as the majority class and class 2 ("NL", 17 instances) is used as the minority class. A dataset is split into the training set (70%) and the test set (30%) by stratified sampling [128] that ensures the same class imbalance ratio in the training set, the test set and the whole original dataset.

3.3.2 Baseline Methods

The baseline methods include GP based methods and non-GP methods, shown in Table 3.2.

GP baseline methods

Sampling methods are very popular because they are usually not limited to a specific classification algorithm. Since the used datasets have a small number of instances, oversampling methods are more suitable than undersamping methods. SMOTE [16] is the most popular oversampling method, and its improved versions mainly include borderline-SMOTE1 [60], borderline-SMOTE2 [60], and adaptive synthetic sampling approach (ADASYN) [62]). The compared oversampling-based GP methods are:

- GP$_{SMOTE}$: An unbalanced training set is rebalanced by SMOTE before GP using Acc (Eq. (2.1)) as the fitness function (denoted as GP$_{Acc}$) is used to train classifiers.

- GP$_{BSMOTE1}$: An unbalanced training set is rebalanced by Borderline-SMOTE1 before classifiers are trained by GP$_{Acc}$.

- GP$_{BSMOTE2}$: An unbalanced training set is rebalanced by Borderline-SMOTE2 before classifiers are trained by GP$_{Acc}$.

- GP$_{ADASYN}$: An unbalanced training set is rebalanced by ADASYN before classifiers are trained by GP$_{Acc}$.

Table 3.1: Dataset description.

Dataset	#Features	#Instances	Majority Class (Proportion %)	Minority Class (Proportion %)	IR (Rounding)
Armstrong-2002-v1	1081	72	66.67	33.33	2
Golub-1999-v1	1868	72	65.28	34.72	2
Colon	2000	62	64.52	35.48	2
Leukemia	7129	72	65.28	34.72	2
Shipp-2002-v1	798	77	75.32	24.68	3
DLBCL	5469	77	75.32	24.68	3
Gordon-2002	1626	181	82.87	17.13	5
Yeoh-2002-v1	2526	248	82.66	17.34	5
Tomlins-2006-v1	2315	104	88.46	11.54	8
Lung	12600	156	89.10	10.90	8

1: The proportions of the majority class and the minority class are rounded to two decimal places.

2: # stands for the cardinality.

Table 3.2: Baseline methods.

GP Methods	Oversampling based	GP_{SMOTE} [16]		
		$GP_{BSMOTE1}$ [60]		
		$GP_{BSMOTE2}$ [60]		
		GP_{ADASYN} [62]		
	Fitness function based	GP_{Ave}		
		GP_{G_Mean}		
		GP_{Amse} [11]		
		GP_{Dist} [11]		
		GP_{Corr} [11]		
		GP_{Auc_w} [11, 205]		
Non-GP Methods	SMOTE-1NN	B-SMOTE1-1NN	B-SMOTE2-1NN	ADASYN-1NN
	SMOTE-DT	B-SMOTE1-DT	B-SMOTE2-DT	ADASYN-DT
	SMOTE-RF	B-SMOTE1-RF	B-SMOTE2-RF	ADASYN-RF
	SMOTE-GBDT	B-SMOTE1-GBDT	B-SMOTE2-GBDT	ADASYN-GBDT
	SMOTE-NB	B-SMOTE1-NB	B-SMOTE2-NB	ADASYN-NB
	SMOTE-MLP	B-SMOTE1-MLP	B-SMOTE2-MLP	ADASYN-MLP

1: B-SMOTE stands for borderline-SMOTE.

Apart from the sampling methods, a fitness function can be directly used to avoid the performance bias issue. In this category, the compared methods are introduced as follows:

- GP$_{Ave}$: GP uses the balanced classification accuracy Ave ($Ave = 0.5 \times \frac{TP}{TP+FN} + 0.5 \times \frac{TN}{TN+FP}$) as the fitness function.
- GP$_{G_Mean}$: GP uses G_Mean (Eq. (2.6)) as the fitness function.
- GP$_{Amse}$: GP uses $Amse$ [11] as the fitness function.
- GP$_{Corr}$: GP uses $Corr$ [11] as the fitness function.
- GP$_{Dist}$: GP uses $Dist$ [11] as the fitness function.
- GP$_{Auc_w}$: GP uses Auc_w (Eq. (3.1)) as the fitness function.

Ave and G_Mean are the most popular fitness functions in GP when data distribution is skewed. GP$_{Ave}$ and GP$_{G_Mean}$ use the standard classification strategy (i.e. the classification threshold $TH = 0$ is employed to separate the original output values of a GP program). Usually, GP$_{Auc_w}$ achieves better AUC results than other GP methods in unbalanced classification, while it consumes longer training time. To save the training time, $Corr$ and $Dist$ [11] have been designed as AUC approximation measures to be used by GP.

Non-GP baseline methods

For the non-GP methods, there are six classification algorithms, each of which is used with four oversampling methods (including SMOTE, borderline-SMOTE1, borderline-SMOTE2 and ADASYN). The six different classification algorithms include 1-nearest neighbours (1NN), decision tree (DT), random forests (RF), gradient boosting decision tree (GBDT), naive bayes (NB) and multilayer perceptron (MLP). KNN is a lazy learning-based classification method, where the training instances are used as prototypes of the classifiers. DT uses a tree-like representation, which is easier to interpret. RF and GBDT are variants of DT, based on

Table 3.3: Parameter settings.

Parameters	Baseline GP methods	Each GP sub-process of GPFRM
Population size	1024	256
Generations	50	40
Initial population	Ramped half-and-half	Ramped half-and-half
Maximum tree depth	10	10
Mutation rate	0.2	0.2
Crossover rate	0.8	0.8
Elitism	1	1
Selection method	Tournament (size=6)	Tournament (size=6)

ensemble learning. Bayesian classifiers are a kind of probabilistic-based classification algorithm, in which NB classifiers are the most common one. MLP is a feed-forward artificial neural network.

3.3.3 Parameter Settings

Table 3.3 shows the parameter settings. Because GPFRM has five GP sub-processes, for each of them, the population size is 256 for 40 generations. This ensures that the total number of evaluations (i.e. 256*40*5) in GPFRM is roughly the same as that of the baseline GP methods (i.e. 1024*50) for a relatively fair comparison. The function set includes four basic arithmetic functions (+, $-$, $\times$, and protected division $\div$), and a conditional operator If function. Note that protected division returns zero when dividing by zero. For the If function, it has three arguments. If the first argument is negative, the second argument is returned, otherwise it returns the third argument. For the baseline GP methods, the terminal set includes features of a dataset and a random constant.

3.4 Results and Discussions

AUC is the most widely used metric in unbalanced classification because it is invariant to skewed data distributions [11]. The AUC results of the proposed method (i.e. GPFRM) and the baseline GP methods are reported in Table 3.4. The Wilcoxon rank-sum test is a popular statistical hypothesis test used to compare two paired groups. In the experiments, each GP method was independently conducted 30 runs with 30 different random seeds (note that all the GP methods use the same set of random seeds for fair comparisons). Wilcoxon rank-sum test was used to compare GPFRM with a baseline GP method to examine whether the two methods have a significant difference or not, after receiving their AUC results of the 30 runs. The significance level is set to 0.05. In Table 3.4, indicators "+", "=" and "−" are used to show that GPFRM is significantly better than, no significant difference, and significantly worse than a compared method, respectively. In Table 3.4, bold values are the highest AUC result achieved and the shortest training time consumed by methods on a dataset.

Table 3.4: GPFRM versus the baseline GP methods on the test sets.

		AUC ($\times$ 100)			Training time (Seconds)
Datasets	Methods	Best	Mean±Std		Mean
Armstrong-2002-v1	GP_{SMOTE}	100	91.3 ±9.83	+	144.32
	$GP_{BSMOTE1}$	100	94.16 ± 7.43	+	141.78
	$GP_{BSMOTE2}$	100	91.22 ± 10.27	+	153.6
	GP_{ADASYN}	100	92.21 ± 9.62	+	143.04
	GP_{Ave}	100	94.48 ± 8.4	+	114.86
	GP_{G_Mean}	100	92.13 ± 8.01	+	114.88
	GP_{Amse}	100	90.17 ± 7.65	+	146.24
	GP_{Corr}	100	94.67 ± 7.56	+	142.55
	GP_{Dist}	100	95.84 ± 3.93	=	141.33
	GP_{Auc_w}	100	94.46 ± 4.93	+	1917.99

Continued on next page

Table 3.4 – Continued from previous page

	GPFRM	100	**98.6 ± 3.56**		**42.13**
Golub-1999-v1	GP_{SMOTE}	100	92.38 ± 10.31	+	195.79
	$GP_{BSMOTE1}$	100	89.11 ± 11.08	+	187.11
	$GP_{BSMOTE2}$	100	85.8± 14.56	+	205.14
	GP_{ADASYN}	100	91.8 ±9.72	+	198.39
	GP_{Ave}	100	91.93 ± 10.09	+	158.77
	GP_{G_Mean}	100	88.99 ± 11.89	+	158.31
	GP_{Amse}	100	82.78 ± 11.62	+	226.35
	GP_{Corr}	100	96.06 ± 6.32	=	225.06
	GP_{Dist}	100	96.9 ± 5.23	=	229.09
	GP_{Auc_w}	100	**98.42 ± 3.38**	=	3089.78
	GPFRM	100	98.04±3.93		**58.64**
Colon	GP_{SMOTE}	92.86	75.99 ± 10.55	+	222.64
	$GP_{BSMOTE1}$	88.1	71.51 ± 12.64	+	206.97
	$GP_{BSMOTE2}$	94.05	73.69 ± 15.28	+	228.62
	GP_{ADASYN}	88.1	74.6 ± 10.25	+	227.64
	GP_{Ave}	91.67	75.52 ± 10.11	+	177.08
	GP_{G_Mean}	92.86	71.51 ± 12.95	+	174.21
	GP_{Amse}	95.24	74.8 ± 10.76	+	203.33
	GP_{Corr}	96.43	75.28 ± 10.1	+	201.08
	GP_{Dist}	92.86	76.59 ± 9.63	+	203.64
	GP_{Auc_w}	91.67	78.97 ± 7.3	=	2348.72
	GPFRM	88.69	**81.9 ± 6.26**		**50.80**
	GP_{SMOTE}	100	87.56 ± 10.01	=	919.55
	$GP_{BSMOTE1}$	98.21	85.6 ± 13.38	+	948.52
	$GP_{BSMOTE2}$	99.11	82.47± 12.94	+	1065.41
	GP_{ADASYN}	100	**89.73± 8.56**	=	951.82

Continued on next page

Table 3.4 – Continued from previous page

Leukemia	GP_{Ave}	98.21	88.79 ± 7.74	=	975.7
	GP_{G_Mean}	100	81.79 ± 15.38	+	979.29
	GP_{Amse}	100	81.73 ± 11.84	+	793.34
	GP_{Corr}	100	86.16 ± 10.84	=	785.98
	GP_{Dist}	97.32	86.32 ± 8.95	=	788.82
	GP_{Auc_w}	100	86.28 ± 9.68	=	10396.7
	GPFRM	90.18	88.01 ± 4.47		**157.34**
Shipp-2002-v1	GP_{SMOTE}	98.15	82.15 ± 11.77	=	132.95
	$GP_{BSMOTE1}$	95.37	77.93 ± 12.35	=	126.82
	$GP_{BSMOTE2}$	100	79.46 ± 14.87	=	146.3
	GP_{ADASYN}	96.3	79.88 ±12.18	=	137.39
	GP_{Ave}	99.07	82.85 ±9.81	=	95.15
	GP_{G_Mean}	96.3	83.09 ± 9.21	=	97.85
	GP_{Amse}	96.3	75.26 ±13.2	+	225.24
	GP_{Corr}	99.07	83.02 ±13.33	=	216.72
	GP_{Dist}	99.07	**84.81 ±8.96**	−	214.44
	GP_{Auc_w}	100	82.62 ±9.45	=	3192.03
	GPFRM	100	79.81 ± 9.35		**80.29**
DLBCL	GP_{SMOTE}	98.15	83.21 ±9.56	=	816.12
	$GP_{BSMOTE1}$	99.07	75.77 ± 18.56	+	795.67
	$GP_{BSMOTE2}$	98.15	79.41 ± 11.68	+	846.71
	GP_{ADASYN}	100	79.48± 10.92	+	831.63
	GP_{Ave}	98.15	75.4 ± 15.67	+	740.03
	GP_{G_Mean}	100	77.01 ± 15.75	+	731.45
	GP_{Amse}	100	77.19 ± 13.18	+	638.3
	GP_{Corr}	98.15	81.02± 11.42	+	643.18
	GP_{Dist}	99.07	84.35 ± 9.96	=	633.55

Continued on next page

Table 3.4 – Continued from previous page

	GP_{Auc_w}	100	85.54 ± 10.83	=	7845.03
	GPFRM	100	**86.2 ± 7.84**		**153.92**
Gordon-2002	GP_{SMOTE}	100	97.49 ± 2.92	+	630.05
	$GP_{BSMOTE1}$	100	98.71 ± 2.67	=	584.44
	$GP_{BSMOTE2}$	100	96.35 ± 4.92	+	669.96
	GP_{ADASYN}	100	97.81 ± 2.9	+	598.04
	GP_{Ave}	100	98.26±2.81	=	321.69
	GP_{G_Mean}	100	98.38 ±3.01	=	323.48
	GP_{Amse}	100	96.49 ±4.46	+	734.51
	GP_{Corr}	100	96.95 ±6.41	+	718.51
	GP_{Dist}	100	98.9 ±3.62	=	720.18
	GP_{Auc_w}	100	99.23 ± 2.06	=	21978.57
	GPFRM	100	**99.37 ± 1.89**		**186.7**
Yeoh-2002-v1	GP_{SMOTE}	100	87.44 ± 9.24	+	1321.18
	$GP_{BSMOTE1}$	99.75	86.23 ± 10.64	+	1290.39
	$GP_{BSMOTE2}$	98.39	83.27 ± 10.15	+	1388.96
	GP_{ADASYN}	100	84.28± 10.24	+	1452.27
	GP_{Ave}	100	83.97 ± 11.91	+	773.26
	GP_{G_Mean}	95.78	66.33 ± 16.09	+	764.4
	GP_{Amse}	92.06	63.79 ± 12.06	+	717.74
	GP_{Corr}	100	93.29 ± 7.77	+	685.35
	GP_{Dist}	100	91.1 ± 8.22	+	694.73
	GP_{Auc_w}	100	**98.95 ± 2.32**	=	24174.23
	GPFRM	100	97.44 ± 2.77		**152.78**
	GP_{SMOTE}	100	83.27± 13.43	+	499.69
	$GP_{BSMOTE1}$	100	87.37 ± 10.14	+	521.56
	$GP_{BSMOTE2}$	100	90.18 ± 10.78	+	500.18

Continued on next page

Table 3.4 – Continued from previous page

Tomlins-2006-v1	GP_{ADASYN}	100	84.67± 13.16	+	478.33
	GP_{Ave}	100	88.87±13.47	+	293.08
	GP_{G_Mean}	100	84.54 ±14.55	+	296.92
	GP_{Amse}	100	92.96 ± 8.04	+	655.62
	GP_{Corr}	100	90.42 ± 14.1	+	648.78
	GP_{Dist}	100	95.83 ± 6.73	=	646.03
	GP_{Auc_w}	100	91.10 ±9.75	+	7865.15
	GPFRM	100	**97.62 ± 5.32**		**126.56**
Lung	GP_{SMOTE}	100	81.7 ± 15.9	+	4298.87
	$GP_{BSMOTE1}$	100	88.89 ±10.76	+	3631.29
	$GP_{BSMOTE2}$	100	84.16± 15.78	+	3901.94
	GP_{ADASYN}	100	82.97 ± 14.98	+	4137.5
	GP_{Ave}	100	83.46 ± 14.73	+	3048.62
	GP_{G_Mean}	99.05	80.89 ± 18.41	+	3038.89
	GP_{Amse}	100	81.78 ± 16.55	+	2503.15
	GP_{Corr}	100	80.71 ±17.21	+	2490.26
	GP_{Dist}	100	84.27 ± 14.8	+	2493.37
	GP_{Auc_w}	100	92.35 ± 13.23	+	45375.33
	GPFRM	100	**95.52 ± 6.2**		**413.20**
Total	70 +, 29 =, 1 −				

3.4.1 Results Analysis

As can be seen from Table 3.4, based on statistical significance tests, in all the 100 cases, GPFRM achieves significantly better than or similar performance to the GP baseline methods in 99 cases, 70 of which are significantly better performances and 29 of which are similar performances. Based on the mean results of the 30 runs, GPFRM performs best on six datasets (i.e. Armstrong-2002-v1, Colon, DLBCL, Gordon-2002, Tomlins-2006-v1 and Lung) than other baseline

methods. Besides, by comparing the best AUC results (from the 30 runs) on each dataset, GPFRM achieves better or the same performance in 82 out of the 100 comparisons. Furthermore, the standard deviation (Std) of AUC results from the 30 runs is often smaller than that of other GP methods.

Comparing with the four oversampling-based methods (including GP_{SMOTE}, $GP_{BSMOTE1}$, $GP_{BSMOTE2}$ and GP_{ADASYN}), GPFRM performs significantly better in 32 out of the 40 cases (similar performance in the other 8 cases). GP_{Ave} and GP_{G_Mean} are the most commonly used GP methods in unbalanced classification. Compared with GP_{Ave} and GP_{G_Mean}, in all the 20 cases, GPFRM achieves significantly better performance in 15 cases and similar performance in the other 5 cases. Auc_w is an AUC measure, but GP using it as the fitness function needs to consume very long training time. Compared to GP_{Auc_w}, GPFRM achieves significantly better or similar performance on all the experimental datasets (significantly better performances in 2 cases and similar performances in 8 cases). Besides, GPFRM consumes the shortest training time than other baseline GP methods on all the datasets.

3.4.2 Comparison with Non-GP Classification Methods Using Sampling Methods

The results of non-GP baseline methods on the test sets are reported in Tables 3.5, 3.6, 3.7 and 3.8.

In Table 3.5, when compared with the traditional classification methods using SMOTE, based on the mean AUC result of GPFRM, in total of the 60 comparisons, GPFRM achieves better classification performance in 56 comparisons. As can be seen from Table 3.6, by comparing the mean AUC result of GPFRM with that of the traditional classification methods using Borderline-SMOTE1 on a dataset, GPFRM outperforms in 53 out of the 60 comparisons. As can be seen from Table 3.7, by comparing the mean AUC result of GPFRM with that of the traditional classification methods that use Borderline-SMOTE2 on each dataset, GPFRM outperforms in 46 out of the 60 comparisons. As can be seen from Table

Table 3.5: GPFRM versus the non-GP classification methods using SMOTE (AUC$\times$100).

Dataset	SMOTE-1NN	SMOTE-DT	SMOTE-RF	SMOTE-GBDT	SMOTE-NB	SMOTE-MLP	**GPFRM**	
							Best	Mean
Armstrong-2002-v1	96.67	88.46	90.97	89.52	85.71	92.85	100	**98.6**
Golub-1999-v1	93.75	89.91	89.80	92.86	68.75	96.43	100	**98.04**
Colon	74.40	64.04	64.46	65.83	47.62	62.60	88.69	**81.9**
Leukemia	90.18	86.61	81.52	86.61	**100**	69.61	90.18	88.01
Shipp-2002-v1	72.22	68.24	72.22	69.44	58.33	76.67	100	**79.81**
DLBCL	69.44	67.31	77.13	72.22	80.86	74.63	100	**86.2**
Gordon-2002	98.91	86.09	92.94	90.93	88.89	85.74	100	**99.37**
Yeoh-2002-v1	86.29	96.03	72.66	96.15	80.58	84.10	100	**97.44**
Tomlins-2006-v1	**98.21**	80.36	88.51	83.75	75	75.95	100	97.62
Lung	85.24	95.21	82.23	**100**	80.00	82.52	100	95.52

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 3.6: GPFRM versus the non-GP classification methods using Borderline-SMOTE1 (AUC$\times$100).

Dataset	B-SMOTE1-1NN	B-SMOTE1-DT	B-SMOTE1-RF	B-SMOTE1-GBDT	B-SMOTE1-NB	B-SMOTE1-MLP	**GPFRM**	
							Best	Mean
Armstrong-2002-v1	96.67	89.16	85.29	89.52	85.71	92.86	100	**98.6**
Golub-1999-v1	93.75	90.15	83.96	92.86	68.75	93.75	100	**98.04**
Colon	74.40	63.12	65.04	62.26	47.62	74.40	88.69	**81.9**
Leukemia	**96.43**	86.61	76.16	86.61	93.75	75	90.18	88.01
Shipp-2002-v1	75.00	68.42	68.15	69.44	66.67	75	100	**79.81**
DLBCL	80.56	70.37	70.93	73.43	80.56	69.44	100	**86.2**
Gordon-2002	97.83	93.94	91.96	93.36	88.89	**100**	100	99.37
Yeoh-2002-v1	96.15	96.15	66.90	96.15	82.20	87.28	100	**97.44**
Tomlins-2006-v1	**100**	84.29	80.36	81.67	62.50	**100**	100	97.62
Lung	95.24	95.75	78.55	**99.92**	70.00	90.00	100	95.52

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 3.7: GPFRM versus the non-GP classification methods using Borderline-SMOTE2 (AUC$\times$100).

Dataset	B-SMOTE2-1NN	B-SMOTE2-DT	B-SMOTE2-RF	B-SMOTE2-GBDT	B-SMOTE2-NB	B-SMOTE2-MLP	**GPFRM**	
							Best	Mean
Armstrong-2002-v1	93.33	89.79	88.86	89.52	**100**	92.86	100	98.6
Golub-1999-v1	93.75	90.03	79.11	92.86	77.68	**100**	100	98.04
Colon	70.24	76.69	62.74	77.38	43.45	60.12	88.69	**81.9**
Leukemia	96.43	86.61	78.01	86.61	**100**	81.25	90.18	88.01
Shipp-2002-v1	77.78	73.61	72.50	72.22	75	**91.67**	100	79.81
DLBCL	75	76.85	75.37	83.33	**88.89**	69.44	100	86.2
Gordon-2002	97.83	93.68	89.41	93.36	**100**	**100**	100	99.37
Yeoh-2002-v1	**100**	99.49	68.23	**100**	90.69	87.28	100	97.44
Tomlins-2006-v1	89.29	87.50	92.38	87.50	62.50	**98.21**	100	97.62
Lung	91.67	99	80.31	**100**	80.00	87.62	100	95.52

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 3.8: GPFRM versus the non-GP classification methods using ADASYN (AUC$\times$100).

Dataset	ADASYN-1NN	ADASYN-DT	ADASYN-RF	ADASYN-GBDT	ADASYN-NB	ADASYN-MLP	**GPFRM**	
							Best	Mean
Armstrong-2002-v1	93.33	89.59	91.97	89.52	92.86	92.85	100	**98.6**
Golub-1999-v1	93.75	90.51	89.35	92.86	68.75	92.86	100	**98.04**
Colon	74.40	66.69	66.19	62.20	47.62	67.26	88.69	**81.9**
Leukemia	83.04	86.61	80.47	86.61	**100**	65.15	90.18	88.01
Shipp-2002-v1	**83.33**	80.46	75.00	83.33	58.33	80.92	100	79.81
DLBCL	77.78	67.31	74.35	72.22	**88.8**	79.44	100	86.2
Gordon-2002	98.91	83.24	95.01	83.33	88.89	83.15	100	**99.37**
Yeoh-2002-v1	91.13	95.64	72.66	96.15	86.04	82.07	100	**97.44**
Tomlins-2006-v1	**98.21**	80.36	86.90	82.5	62.5	85.53	100	97.62
Lung	76.90	97.83	79.74	**99.67**	80.00	74.60	100	95.52

1: Bold values are the highest AUC result achieved by methods on a dataset.

3.8, by comparing the mean AUC result of GPFRM with that of the traditional classification methods using ADASYN on a dataset, GPFRM performs better in 51 out of the 60 comparisons.

Overall, by comparing with other GP methods and non-GP classification methods, the proposed GPFRM method achieves at least similar performance.

3.5 Further analysis

In the experiments, we test the classification performance of the variants of GPFRM to reveal contributions of different components to improving the effectiveness and efficiency. The variants of GPFRM are listed as follows:

- GP_{C1}: GP uses $C1$ as the fitness function.
- GP_{Min_Corr}: GP uses Min_Corr as the fitness function.

Note that GP_{C1} and GP_{Min_Corr} do not use the program reuse mechanism. Comparisons between GP_{Min_Corr} and GPFRM could reveal whether the program reuse mechanism is useful. The results of GP_{C1} and GP_{Min_Corr} are reported in Table 3.9, where bold values are the highest AUC result achieved by a GP method or the shortest training time consumed on a dataset.

Table 3.9: GPFRM versus the variants of GPFRM on the test sets.

		AUC ($\times$ 100)		Training time (Seconds)
Datasets	Methods	Best	Mean$\pm$Std	Mean
Armstrong-2002-v1	GP_{C1}	100	96.03 $\pm$ 4.44	138.36
	GP_{Min_Corr}	100	96.63 $\pm$ 4.55	143.6
	GPFRM	100	**98.6 $\pm$ 3.56**	**42.13**
Golub-1999-v1	GP_{C1}	100	97.44 $\pm$ 5.59	231.29
	GP_{Min_Corr}	100	97.23 $\pm$7.42	252.73
	GPFRM	100	**98.04$\pm$3.93**	**58.64**

Continued on next page

Table 3.9 – Continued from previous page

Colon	GP_{C1}	96.43	78.53 ± 9.06	206.74
	GP_{Min_Corr}	95.24	74.76 ± 12.4	225.65
	GPFRM	88.69	**81.9 ± 6.26**	**50.80**
Leukemia	GP_{C1}	100	**90.74 ±5.75**	776.28
	GP_{Min_Corr}	100	90.09 ± 6.81	793.88
	GPFRM	90.18	88.01 ± 4.47	**157.34**
Shipp-2002-v1	GP_{C1}	92.59	78.12 ±13.49	224.18
	GP_{Min_Corr}	98.15	**82.65 ± 9.87**	215.61
	GPFRM	100	79.81 ± 9.35	**80.29**
DLBCL	GP_{C1}	96.3	75.34 ± 13.71	565.51
	GP_{Min_Corr}	100	81.14 ± 15.28	670.87
	GPFRM	100	**86.2 ± 7.84**	**153.92**
Gordon-2002	GP_{C1}	100	98.65±1.94	721.9
	GP_{Min_Corr}	100	98.48 ± 2.35	724.74
	GPFRM	100	**99.37 ± 1.89**	**186.7**
Yeoh-2002-v1	GP_{C1}	100	98.06 ± 4.02	675.85
	GP_{Min_Corr}	100	**98.8 ± 3.35**	703.08
	GPFRM	100	97.44 ± 2.77	**152.78**
Tomlins-2006-v1	GP_{C1}	100	89.02 ±11.05	650.09
	GP_{Min_Corr}	100	93.65 ±9.97	750.1
	GPFRM	100	**97.62 ± 5.32**	**126.56**
Lung	GP_{C1}	100	**95.97 ± 5.77**	2230.73
	GP_{Min_Corr}	100	90.1 ± 11.29	2639.07
	GPFRM	100	95.52 ± 6.2	**413.20**

3.5.1 Investigation into $C1$

$C1$ is used to approximate Auc_w to evaluate the classification capability of a program, with the expectation of saving training time. By comparing GP_{C1} and GP_{Auc_w} (the results of GP_{Auc_w} have been reported in Table 3.4, Page 66) on the ten datasets, GP_{C1} consumes much less training time than GP_{Auc_w}. However, for an AUC approximation measure, it is nearly impossible to fully correlate with a full AUC measure. $C1$ is not as imprecise as Auc_w, so it may make some mistakes when selecting between two individuals.

By comparing $C1$ and Auc_w, if the majority class and the minority class are correctly separated, $C1$ is able to effectively approximate Auc_w (i.e. both $C1$ value and Auc_w value are equal to 1). During the training process, the best programs from the final population usually are able to correctly classify all of the training instances. Besides, GP using $C1$ as the fitness function becomes sensitive when an instance from the minority class is mistakenly classified since the $C1$ value will drop significantly.

3.5.2 Investigation into the Program Reuse Mechanism

As can be seen from Table 3.9, based on the mean AUC results, GPFRM performs better than GP_{Min_Corr} on seven datasets (out of the ten datasets). On Leukemia, Shipp-2002-v1, and Yeoh-2002-v1, the mean AUC results of GPFRM are slightly decreased, compared with GP_{Min_Corr}. This is possibly because some GP classifiers are under-fitting due mainly to the smaller population size (i.e. 256) and the number of generations (i.e. 40) used to train classifiers during each sub-process. Therefore, it is likely for several weak GP classifiers against a good GP classifier in the test process to make a wrong decision.

The standard deviation of the AUC results achieved by GPFRM on each dataset is smaller than that of GP_{Min_Corr}, which may show that the stability of GPFRM is better than GP_{Min_Corr}. More importantly, the average training time of GPFRM is much faster than GP_{Min_Corr} on all the datasets, only 15.57%-37.24% of training time of GP_{Min_Corr}.

3.6 Chapter Summary

The goal of this chapter was to improve the effectiveness and efficiency of GP on high-dimensional unbalanced classification. In order to achieve this goal, we investigated the influence of using different fitness functions, based on which we designed a new fitness function. The newly-designed fitness function considers two criteria, i.e. AUC approximation and classification clarity. To further improve the efficiency, a program reuse mechanism was proposed, which suggests not only reusing good features previously selected but also reusing good trees in the initialization of the later GP sub-process.

On the basis of the proposed fitness function and the program reuse mechanism, we proposed a new GP based classification method (called GPFRM) to evolve classifiers for classification with high-dimensional unbalanced data. In the experiments, high-dimensional unbalanced datasets were used to examine the classification performance of GPFRM. The experimental results show that GPFRM significantly reduced training time, and more importantly, the classification performance was increased in almost all cases. Therefore, the goal of this chapter has been successfully achieved.

The classification performance is improved by GPFRM due mainly to the new fitness function that approximates AUC effectively. The efficiency of GPFRM is improved owing mainly to the program reuse mechanism. The program reuse mechanism does not require feeding all the features into GP at the beginning of the evolutionary learning process, but the features are fed into GPFRM sequentially. More importantly, the good features and programs are effectively reused to further improve the effectiveness.

However, the new fitness function equally treats the AUC approximation and classification clarity criteria. This is the main drawback of GPFRM since the two criteria are not always equally important. To distinguish and select between two programs, the classification clarity becomes important when the two programs achieve the same AUC performance. We will investigate how the drawback can be overcome in Chapter 4.

Chapter 4

GP with Multi-criterion Evaluation and Selection

4.1 Introduction

In single-objective GP methods, when two criteria are considered in the fitness evaluation process, the most popular method is to use the weighted sum method, i.e. the two criteria are weighted and combined into a single fitness function. However, it is usually hard to determine an optimal weight value without domain knowledge, although humans can roughly confirm one criterion being more important than the other.

For the GPFRM method in Chapter 3, the main drawback is that the AUC approximation and classification clarity criteria are weighted as equal and summed together in the fitness function (i.e. Eq. (3.4), Page 57). However, the two criteria are not always equally important. To distinguish between two programs, the AUC performance is usually more important than the classification clarity, and the classification clarity becomes important only when the two programs achieve the same AUC performance.

Therefore, in this chapter, we aim to overcome the drawback by developing a new GP method. In the new method, an individual is evaluated by the two criteria, and the obtained values on the two criteria are combined in pairs (called fitness

tuple) in the evaluation process. The fitness tuple is used to show the goodness of programs on the two criteria, and also used as an input for the selection process. A three-criterion tournament selection (apart from the above-mentioned two criteria, the program size is used as the third criterion) is also designed, which filters out solutions according to a cascading set of priorities.

4.1.1 Chapter Goals

The overall goal of this chapter is to develop a GP based classification method, which avoids weighing criteria (i.e. AUC approximation and classification clarity) in the fitness evaluation process and can better identify good solutions based on multiple criteria in the selection process, to enhance the performance for classification with high-dimensional unbalanced data. The overall goal is composed of the following sub-goals:

(1) Investigate how the two criteria can be combined (avoiding using a weight) in the evaluation process,

(2) Develop a new two-criterion fitness function,

(3) Develop a new three-criterion selection operator for identifying and selecting good programs, and

(4) Investigate whether the proposed method can effectively and efficiently improve the performance of GP in classification with high-dimensional unbalanced data.

4.1.2 Chapter Organization

The remainder of this chapter is as follows. Section 4.2 introduces the proposed GP method. Section 4.3 introduces the experimental design, and the results are discussed and analyzed in Section 4.4. Further investigations are shown in Section 4.5. Section 4.6 draws the conclusions of this chapter.

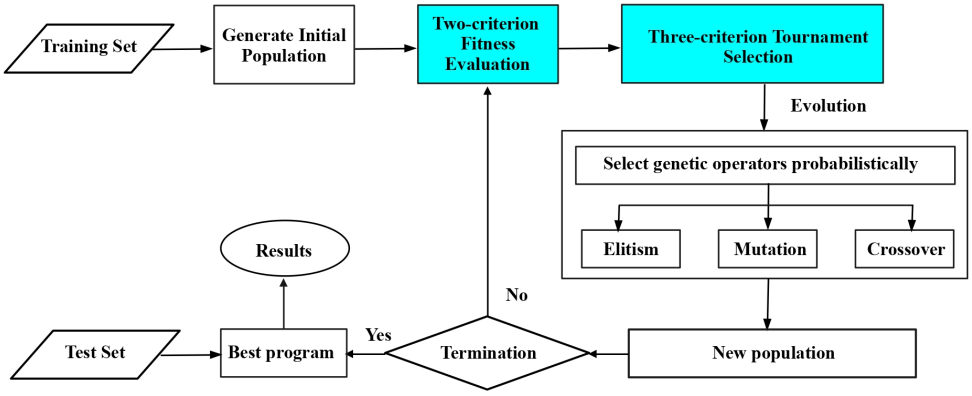


Figure 4.1: The overall design of GPMFS.

4.2 The Proposed Method

The proposed method is called **G**enetic **P**rogramming with **M**ulti-criterion **F**itness Evaluation and **S**election (**GPMFS**). In this section, we introduce the overall design of GPMFS and its components.

4.2.1 The Overall Design of GPMFS

In GPMFS, a novel two-criterion fitness function and three-criterion tournament selection are designed for improving the performance of GP in high-dimensional unbalanced classification. The overall design of GPMFS is shown in Figure 4.1. A dataset is split into a training set and a test set by stratified sampling. In the training process, individuals in a population are evaluated by the new two-criterion fitness function, i.e. an individual is evaluated by two criteria and then the obtained values on the two criteria are combined in pairs. The obtained fitness tuples are then input to the three-criterion tournament selection for selecting better individuals. The selected individuals are used by genetic operators (i.e. crossover, mutation and elitism) to generate new offspring for a new population. After finishing the training process (i.e. GPMFS finishes its search in the final generation), the best

program of the final generation is used as a classifier to classify unseen instances.

4.2.2 The Two-criterion Fitness Evaluation Method

For the GPFRM method in Chapter 3, the AUC approximation criterion (i.e. $C1$) and classification clarity criterion (i.e. $C2$) were weighted as equal and summed together to be a fitness function. However, the two criteria are not always equally important. To avoid using a weight to combine them, an individual is evaluated by the two criteria, and a pair of obtained values on $C1$ and $C2$ (called a fitness tuple) is used during the evaluation process. A new two-criterion fitness function is defined as:

$$A\_S = (C1, C2) = \left(\frac{\sum_{i \in Min} I(P_i, t)}{|Min|}, \sqrt{\frac{\sum_{c=1}^{K} N_c(\mu_c - \overline{\mu})^2}{\sum_{c=1}^{K} \sum_{i=1}^{N_c} (P_{ci} - \overline{\mu})^2}} \right) \quad (4.1)$$

where $C1$ and $C2$ are defined by Eqs. (3.2) and (3.3), respectively. The details about $C1$ and $C2$ were introduced in Section 3.2.1 (Page 55). The fitness tuple is used to show the goodness of a program on $C1$ and $C2$, which is also used as the input for the following selection process.

4.2.3 The Three-criterion Tournament Selection

The GPFRM method in Chapter 3 used tournament selection. There are two problems for using standard tournament selection with the proposed fitness function A_S in this chapter. Firstly, A_S produces fitness tuples, rather than fitness values. Secondly, the standard tournament selection chooses the best individual in a tournament, which does not consider possible estimation errors in the fitness evaluation process and thereby ignores some potentially good individuals.

A new selection operator is proposed, where three criteria are considered sequentially: $C1$, $C2$ and $C3$ ($C3$ is the program size). The main reason for using $C3$ as the final criterion is because for high-dimensional data, GP tends to generate large programs that usually use many features as terminals. When the program size is considered in selecting between two programs, the small program is preferred, which often uses a small number of features. Moreover, large programs

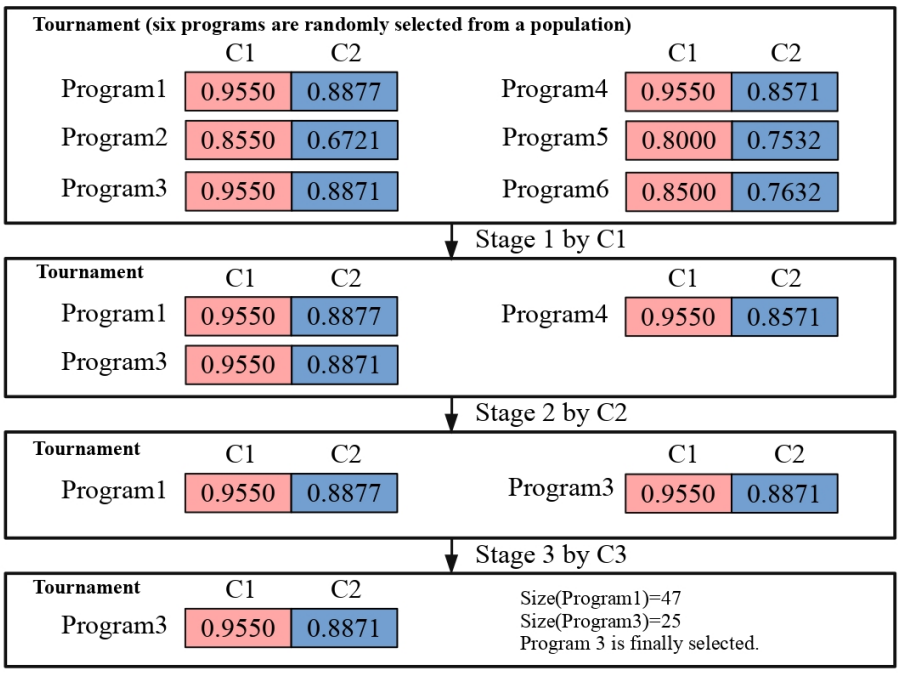


Figure 4.2: Selection process in a tournament.

usually consume more training time. In the three-criterion tournament selection, the main steps of selecting the best individual from a tournament are listed as follows:

Step1: Programs are selected if they achieve the highest $C1$ value;

Step2: The programs selected by $C1$ undergo further selection according to $C2$ to identify programs that achieve good $C2$ values (if the number of programs selected by $C1$ is more than 1);

Step3: If two or more programs are selected by $C1$ and $C2$, the program size $C3$ is used to decide which program is finally selected (a small program is selected with a very high chance).

Figure 4.2 shows an example of the selection process in a tournament by the three-criterion tournament selection. After selecting six programs from a population at random, Programs 1, 3 and 4 are selected because they achieve the best fitness value on $C1$ (i.e. 0.9550). Afterwards, the selected programs are further

distinguished according to $C2$. At this stage, Programs 1 and 3 are selected but Program 4 is discarded (we will explain why Program 3 is selected). Finally, $C3$ is used to determine which program is selected. Because the size of Program 3 is smaller than that of Program 1, Program 3 is finally selected as a winner in the tournament.

(1) Why possible estimation errors of $C2$ values need to be considered?

The dominance relation is a generalized partial order relation, where only reflexivity and transitivity are required [40]. To rank two values by the standard dominance relation, there are three dominant results, including $\geq$, $=$, and $\leq$. However, the standard dominance relation is relatively strict and finds it hard to tolerate potential mistakes to rank numerical values. For example, based on the dominance relation, 96.5556 is greater than 96.5555, which is an absolute answer in some cases because it does not consider some factors, like noise. We now start to explain why Program 3 in Figure 4.2 is selected at Step 2 and why it is necessary to consider possible estimation errors of $C2$ values.

In the proposed fitness function A_S, the two criteria (i.e. $C1$ and $C2$) have the same value domain (i.e. [0, 1]). However, the gap between two different adjacent $C2$ values is smaller than that of two different adjacent $C1$ values. This is because the variation of $C1$ values is mainly determined by the number of instances in the minority class Min. For example, if Min has 10 instances in a training set, the $C1$ value of a program increases by 10% when $I(q,t)$ increases by 1 according to Eq. (3.2) (Page 56). However, the variation of $C2$ values is determined by all the output values of a program by taking all of the training instances as the inputs. Therefore, to compare two programs based on their $C2$ values, the gap between two different values may be narrow. In that case, it is better to consider possible estimation errors of values when ranking programs based on $C2$. For example, in Figure 4.2, when Program 1 is compared with Program 3, they achieve the same fitness value on $C1$, but Program 1 is just slightly better than Program 3 on $C2$ (i.e. 0.0006 higher). Based on the dominance relation, Program 1 is better than Program 3 ($0.8877 \geq 0.8871$), so Program 1 is selected and Program 3 is

discarded. In fact, it is hard to ensure the absolute accurateness of values due to some factors (like noise). The goodness of Program 3 is likely to be similar to that of Program 1, so it should have a chance to further compete with Program 1.

Therefore, in this chapter, a new three-criterion tournament selection is proposed, which considers possible estimation errors when selecting a program from a tournament by using a new generalized dominance relation, called the quasi-dominance relation [138]. Based on the quasi-dominance relation, Program 3 is quasi-dominated by Program 1, instead of being absolutely dominated by Program 1.

(2) How a quasi-dominating set of a program is calculated?

The quasi-dominance relation is a generalized partial order relation. Slightly different from the dominance relation, the quasi-dominance relation has reflexivity and quasi-transitivity [138]. Compared with the dominance relation, the quasi-dominance relation allows the use of a dominant threshold [138].

Definition 4.1 Given Z is a function ($C2$ is seen as Z), d is a dominant threshold, the quasi-dominating set of a program p_b is defined as:

$$D_q(p_b) = \cup\{p_j | Z(p_j) \geq (1-d) * Z(p_b)\} \tag{4.2}$$

Note that, in the new selection operator, the quasi-dominance relation is not used to rank $C1$ values. The main reason is that it is less likely for two different $C1$ values to be very similar because the minority class usually has a small number of instances in an unbalanced high-dimensional dataset. In addition, the use of the quasi-dominance relation actually causes additional computational costs when calculating quasi-dominating solutions. Hence, it is not necessary to use the quasi-dominance relation on $C1$.

The selection process in a tournament is detailed as follows. The programs that achieve the highest $C1$ value are chosen (denoted as Pro_{c1}). Afterwards, a program in Pro_{c1} that achieves the highest $C2$ value is selected and denoted as p_b. Based on $C2$, the quasi-dominating set of p_b is calculated and denoted as Pro_{c2}, which includes the programs selected by $C2$ (note that these programs have the

same $C1$ value). There are three cases in total. First, Pro_{c2} has only one program, then this program is selected as a winner. Second, Pro_{c2} has two programs, so their tree sizes are used to determine which one is a winner. Usually, a small program is preferred. However, it is not always the case that a small program performs better than a large program. Therefore, a large program needs to be given a small chance to be selected. Third, there are more than two programs in Pro_{c2}. Because programs in Pro_{c2} achieve the same $C1$ value and very similar $C2$ values, two programs are selected at random to make sure every program in Pro_{c2} having an equal chance to be selected, and then the third case turns to be the second case for the purpose of simplification. In the third step of selection, the program sizes of $p1$ and $p2$ (i.e. $Size1$ and $Size2$) are calculated and compared. If $Size1 < Size2$ and $Rad < 1$ (note that Rad has 99% probability to be a number that is less than 1), then the small program $p1$ is selected; If $Size1 = Size2$, then $p1$ or $p2$ is randomly selected; otherwise, the large program $p2$ is selected.

In the three-criterion selection operator, we do not prescribe the frequency of using each criterion because the three criteria are considered sequentially for choosing a good individual in a tournament. $C1$ is considered at first and in all cases. This is because $C1$ is more important (with the highest priority) than the other two criteria. Therefore, a program is selected from a tournament if it achieves the highest $C1$ value. The frequency of using $C2$ or $C3$ differs in different tournaments. If two or more programs achieve the highest $C1$ value, these programs will undergo a further selection according to $C2$. Similarly, if two or more programs are selected by $C1$ and $C2$, the program size $C3$ will be used to decide which program is finally selected. It is noteworthy that individuals in a tournament are selected according to $C1$, $C2$ and $C3$ in a lexicographical order, which is a known concept in computer science.

In this chapter, we have introduced the novel three-criterion selection operator. The proposed selection operator shares a number of similarities with lexicographic tournament selection that is able to select an individual from a tournament by considering both the fitness value and tree size in a lexicographical order [107]. The main difference between the two selection methods is the use

of the quasi-dominance relation when ranking $C2$ values to identify good solutions in a tournament. In addition, the proposed method is a single-objective GP algorithm, where multiple criteria are considered in evaluation and selection processes. We did not apply evolutionary multi-objective optimization (EMO) since these criteria are not always conflicting.

4.3 Experiment Design

4.3.1 Datasets

In the experiments, we used ten gene expression datasets to examine and investigate the effectiveness of the proposed GPMFS method. The details of the ten datasets are shown in Table 4.1 (the same as those used in Chapter 3). These datasets are unbalanced, and contain a small number of instances described by a large number of features. A dataset is split into the training set (70%) and the test set (30%) by stratified sampling.

4.3.2 Baseline Methods

The baseline methods are the same as that used in Chapter 3, listed in Table 3.2 (the details of these baseline methods were introduced in Chapter 3, Page 61). The proposed GPMFS is also compared with GPFRM proposed in Chapter 3.

4.3.3 Parameter Settings

For GP methods, the population size is 1024 and the number of generations is 50. Ramped half-and-half is used to initialize a population. The mutation rate and crossover rate are 0.2 and 0.8, respectively. Elitism is used to copy the top program into the next generation. The maximum tree depth is set to 10. The function set includes four arithmetic functions (i.e. +, $-$, $\times$, and protected division $\div$) and a conditional operator If function. The terminal set includes features of a dataset and a random constant. For the baseline GP methods, tournament selection is

Table 4.1: Dataset description.

Dataset	#Features	#Instances	Proportion % (Majority Class)	Proportion % (Minority Class)	IR (Rounding)
Armstrong-2002-v1	1081	72	66.67	33.33	2
Golub-1999-v1	1868	72	65.28	34.72	2
Colon	2000	62	64.52	35.48	2
Leukemia	7129	72	65.28	34.72	2
Shipp-2002-v1	798	77	75.32	24.68	3
DLBCL	5469	77	75.32	24.68	3
Gordon-2002	1626	181	82.87	17.13	5
Yeoh-2002-v1	2526	248	82.66	17.34	5
Tomlins-2006-v1	2315	104	88.46	11.54	8
Lung	12600	156	89.10	10.90	8

1: The proportions of the majority class and the minority class are rounded to two decimal places.

used, where the tournament size is 6. For the three-criterion tournament selection in GPMFS, the tournament size is 6 and the dominant threshold d is 0.1.

4.4 Results and Discussions

Table 4.2 reports the AUC results of the GP methods on the test sets. Bold values in Table 4.2 are the highest AUC achieved or the shortest training time consumed by these GP methods on a dataset. On each dataset, the AUC results of the proposed method from the 30 runs are compared with a baseline GP method by using the Wilcoxon rank-sum test, with the significance level of 0.05. The results of the significance test are reported in Table 4.2, where "+", "=" and "$-$" are used to indicate that the proposed GPMFS method is significantly better than, similar to, and significantly worse than a compared method.

4.4.1 GPMFS Versus the Baseline GP Methods

Table 4.2: Results on the test sets (GPMFS versus the baseline GP methods).

		AUC $\times 100$			Training time (Seconds)
Datasets	Methods	Best	Mean±Std		Mean
Armstrong-2002-v1	GP_{SMOTE}	100	91.3 ±9.83	+	144.32
	$GP_{BSMOTE1}$	100	94.16 ± 7.43	+	141.78
	$GP_{BSMOTE2}$	100	91.22 ± 10.27	+	153.6
	GP_{ADASYN}	100	92.21 ± 9.62	+	143.04
	GP_{Ave}	100	94.48 ± 8.4	+	114.86
	GP_{G_Mean}	100	92.13 ± 8.01	+	114.88
	GP_{Amse}	100	90.17 ± 7.65	+	146.24
	GP_{Corr}	100	94.67 ± 7.56	+	142.55
	GP_{Dist}	100	95.84 ± 3.93	+	141.33
	GP_{Auc_w}	100	94.46 ± 4.93	+	1917.99
	GPFRM	100	98.6 ± 3.56	=	**42.13**
	GPMFS	100	**98.76 ± 2.33**		105.99
Golub-1999-v1	GP_{SMOTE}	100	92.38 ± 10.31	+	195.79
	$GP_{BSMOTE1}$	100	89.11 ± 11.08	+	187.11
	$GP_{BSMOTE2}$	100	85.8± 14.56	+	205.14
	GP_{ADASYN}	100	91.8 ±9.72	+	198.39
	GP_{Ave}	100	91.93 ± 10.09	+	158.77
	GP_{G_Mean}	100	88.99 ± 11.89	+	158.31
	GP_{Amse}	100	82.78 ± 11.62	+	226.35
	GP_{Corr}	100	96.06 ± 6.32	+	225.06
	GP_{Dist}	100	96.9 ± 5.23	+	229.09
	GP_{Auc_w}	100	98.42 ± 3.38	=	3089.78

Continued on next page

Table 4.2 – Continued from previous page

	GPFRM	100	98.04±3.93	=	**58.64**
	GPMFS	100	**99.14 ± 2.64**		169.33
Colon	GP_{SMOTE}	92.86	75.99 ± 10.55	=	222.64
	$GP_{BSMOTE1}$	88.1	71.51 ± 12.64	+	206.97
	$GP_{BSMOTE2}$	94.05	73.69 ± 15.28	+	228.62
	GP_{ADASYN}	88.1	74.6 ± 10.25	+	227.64
	GP_{Ave}	91.67	75.52 ± 10.11	=	177.08
	GP_{G_Mean}	92.86	71.51 ± 12.95	+	174.21
	GP_{Amse}	95.24	74.8 ± 10.76	+	203.33
	GP_{Corr}	96.43	75.28 ± 10.1	=	201.08
	GP_{Dist}	92.86	76.59 ± 9.63	=	203.64
	GP_{Auc_w}	91.67	78.97 ± 7.3	=	2348.72
	GPFRM	88.69	**81.9 ± 6.26**	=	**50.80**
	GPMFS	89.29	78.02 ± 6.06		168.05
Leukemia	GP_{SMOTE}	100	87.56 ± 10.01	=	919.55
	$GP_{BSMOTE1}$	98.21	85.6 ± 13.38	+	948.52
	$GP_{BSMOTE2}$	99.11	82.47± 12.94	+	1065.41
	GP_{ADASYN}	100	89.73± 8.56	=	951.82
	GP_{Ave}	98.21	88.79 ± 7.74	=	975.7
	GP_{G_Mean}	100	81.79 ± 15.38	+	979.29
	GP_{Amse}	100	81.73 ± 11.84	+	793.34
	GP_{Corr}	100	86.16 ± 10.84	+	785.98
	GP_{Dist}	97.32	86.32 ± 8.95	+	788.82
	GP_{Auc_w}	100	86.28 ± 9.68	+	10396.7
	GPFRM	90.18	88.01 ± 4.47	=	**157.34**

Continued on next page

Table 4.2 – Continued from previous page

	GPMFS	98.21	**90.71 ± 6.74**		641.39
Shipp-2002-v1	GP_{SMOTE}	98.15	82.15 ± 11.77	=	132.95
	$GP_{BSMOTE1}$	95.37	77.93 ± 12.35	+	126.82
	$GP_{BSMOTE2}$	100	79.46 ± 14.87	+	146.3
	GP_{ADASYN}	96.3	79.88 ±12.18	+	137.39
	GP_{Ave}	99.07	82.85 ±9.81	=	**95.15**
	GP_{G_Mean}	96.3	83.09 ± 9.21	=	97.85
	GP_{Amse}	96.3	75.26 ±13.2	+	225.24
	GP_{Corr}	99.07	83.02 ±13.33	=	216.72
	GP_{Dist}	99.07	**84.81 ±8.96**	=	214.44
	GP_{Auc_w}	100	82.62 ±9.45	=	3192.03
	GPFRM	100	79.81 ± 9.35	+	**80.29**
	GPMFS	96.3	83.75±7.55		193.76
DLBCL	GP_{SMOTE}	98.15	83.21 ±9.56	+	816.12
	$GP_{BSMOTE1}$	99.07	75.77 ± 18.56	+	795.67
	$GP_{BSMOTE2}$	98.15	79.41 ± 11.68	+	846.71
	GP_{ADASYN}	100	79.48± 10.92	+	831.63
	GP_{Ave}	98.15	75.4 ± 15.67	+	740.03
	GP_{G_Mean}	100	77.01 ± 15.75	+	731.45
	GP_{Amse}	100	77.19 ± 13.18	+	638.3
	GP_{Corr}	98.15	81.02± 11.42	+	643.18
	GP_{Dist}	99.07	84.35 ± 9.96	=	633.55
	GP_{Auc_w}	100	85.54 ± 10.83	=	7845.03
	GPFRM	100	86.2 ± 7.84	=	**153.92**
	GPMFS	100	**88.58 ±8.67**		504.44

Continued on next page

Table 4.2 – Continued from previous page

Gordon-2002	GP_{SMOTE}	100	97.49 ± 2.92	+	630.05
	$GP_{BSMOTE1}$	100	98.71 ± 2.67	=	584.44
	$GP_{BSMOTE2}$	100	96.35 ± 4.92	+	669.96
	GP_{ADASYN}	100	97.81 ± 2.9	+	598.04
	GP_{Ave}	100	98.26±2.81	=	321.69
	GP_{G_Mean}	100	98.38 ±3.01	=	323.48
	GP_{Amse}	100	96.49 ±4.46	+	734.51
	GP_{Corr}	100	96.95 ±6.41	+	718.51
	GP_{Dist}	100	98.9 ±3.62	=	720.18
	GP_{Auc_w}	100	99.23 ± 2.06	=	21978.57
	GPFRM	100	**99.37 ± 1.89**	=	**186.7**
	GPMFS	100	99.21 ± 1.03		701.67
Yeoh-2002-v1	GP_{SMOTE}	100	87.44 ± 9.24	+	1321.18
	$GP_{BSMOTE1}$	99.75	86.23 ± 10.64	+	1290.39
	$GP_{BSMOTE2}$	98.39	83.27 ± 10.15	+	1388.96
	GP_{ADASYN}	100	84.28± 10.24	+	1452.27
	GP_{Ave}	100	83.97 ± 11.91	+	773.26
	GP_{G_Mean}	95.78	66.33 ± 16.09	+	764.4
	GP_{Amse}	92.06	63.79 ± 12.06	+	717.74
	GP_{Corr}	100	93.29 ± 7.77	+	685.35
	GP_{Dist}	100	91.1 ± 8.22	+	694.73
	GP_{Auc_w}	100	98.95 ± 2.32	=	24174.23
	GPFRM	100	97.44 ± 2.77	=	**152.78**
	GPMFS	100	**99.26 ± 2.12**		575.4
	GP_{SMOTE}	100	83.27± 13.43	+	499.69

Continued on next page

Table 4.2 – Continued from previous page

Tomlins-2006-v1	$GP_{BSMOTE1}$	100	87.37 ± 10.14	+	521.56
	$GP_{BSMOTE2}$	100	90.18 ± 10.78	=	500.18
	GP_{ADASYN}	100	84.67± 13.16	+	478.33
	GP_{Ave}	100	88.87±13.47	+	293.08
	GP_{G_Mean}	100	84.54 ±14.55	+	296.92
	GP_{Amse}	100	92.96 ± 8.04	=	655.62
	GP_{Corr}	100	90.42 ± 14.1	+	648.78
	GP_{Dist}	100	95.83 ± 6.73	=	646.03
	GP_{Auc_w}	100	91.10 ±9.75	=	7865.15
	GPFRM	100	**97.62 ± 5.32**	=	**126.56**
	GPMFS	100	94.48± 9.2		636.76
Lung	GP_{SMOTE}	100	81.7 ± 15.9	+	4298.87
	$GP_{BSMOTE1}$	100	88.89 ±10.76	+	3631.29
	$GP_{BSMOTE2}$	100	84.16± 15.78	+	3901.94
	GP_{ADASYN}	100	82.97 ± 14.98	+	4137.5
	GP_{Ave}	100	83.46 ± 14.73	+	3048.62
	GP_{G_Mean}	99.05	80.89 ± 18.41	+	3038.89
	GP_{Amse}	100	81.78 ± 16.55	+	2503.15
	GP_{Corr}	100	80.71 ±17.21	+	2490.26
	GP_{Dist}	100	84.27 ± 14.8	+	2493.37
	GP_{Auc_w}	100	92.35 ± 13.23	+	45375.33
	GPFRM	100	95.52 ± 6.2	=	**413.20**
	GPMFS	100	**97.74 ± 3.83**		2019.01
Total	74 +, 36 =, 0 −				

Discussions on AUC Results on the Test Sets

By comparing the mean AUC results from the 30 runs shown in Table 4.2, GPMFS performs best on six datasets (out of the ten datasests). Moreover, based on the statistical significance tests, GPMFS achieves significantly better than or similar performance to the baseline GP methods in all the 110 cases.

GPMFS achieves better performance in both slightly and highly unbalanced datasets. Besides, for GPMFS, the standard deviation of the AUC results from the 30 runs is often smaller than that of the baseline GP methods. This may show that GPMFS is relatively more robust than the baseline GP methods. A possible reason is that the three-criterion tournament selection in GPMFS could alleviate the risk of overfitting by using program size $C3$ as the final criterion to select a good program from the programs selected by $C1$ and $C2$, to reduce the chance of choosing over-complicated programs.

Finally, the best AUC achieved by GPMFS is the same or better than other GP methods in almost all comparisons. Moreover, in GPMFS, the gap between the best AUC and the mean AUC is often narrower than that of the baseline GP methods on each dataset. This is because the mean AUC from the 30 runs is improved, without the decrease in the best AUC in almost all cases.

Discussions on Training Time

According to Table 4.2, GP$_{Auc_w}$ often performs very well, but it consumes much longer training time than the other GP methods, particularly for datasets which have a relatively large number of instances in a training set. GPMFS has an efficiency advantage (except for GPFRM) when compared with the other baseline GP methods. As can be seen from Table 4.2, GPMFS consumes less training time in 70 out of the 110 comparisons, while it achieves significantly better or similar AUC performance than other GP baseline methods in all the 110 comparisons (significantly better performance in 74 cases and similar performance in 36 cases). This shows that the improved efficiency of GPMFS is not at the expense of the classification performance.

The efficiency of GPMFS is improved due mainly to two reasons. The first reason is that GPMFS just requires a smaller number of pairwise comparisons than GP_{Auc_w} in an evaluation. Another reason is due to the sizes of the programs. In the three-criterion tournament selection, a small program is preferred after selecting top programs by $C1$ and $C2$. Therefore, even though the three-criterion tournament selection needs to do more calculations to compare and select good trees, GPMFS is relatively efficient in most cases.

Note that the GPFRM method in chapter 3 has a very good efficiency, which is due mainly to the program reuse mechanism. GPMFS does not use the program reuse mechanism, and the efficiency of GPMFS will be further compared with GPFRM without the program reuse mechanism in Section 4.5.

4.4.2 GPMFS Versus the Non-GP Baseline Methods

In Tables 4.3, 4.4, 4.5 and 4.6, we report the results of the non-GP baseline methods on the test sets.

Based on Table 4.3, by comparing the mean AUC result of GPMFS with that of the traditional classification methods using SMOTE on a dataset, GPMFS achieves better classification performance in 57 out of the 60 comparisons. As can be seen from Table 4.4, by comparing the mean AUC result of GPMFS with that of the traditional classification methods using Borderline-SMOTE1 on a dataset, GPMFS outperforms in 54 out of the 60 comparisons. As can be seen from Table 4.5, by comparing the mean AUC result of GPMFS with that of the traditional classification methods that use Borderline-SMOTE2 on each dataset, GPMFS outperforms in 46 out of the 60 comparisons. As can be seen from Table 4.6, by comparing the mean AUC result of GPMFS with that of the traditional classification methods using ADASYN on a dataset, GPMFS performs better in 55 out of the 60 comparisons.

In summary, GPMFS effectively handles the performance bias issue of GP for unbalanced classification. GPMFS often achieves better or similar performance when compared with the baseline methods in high-dimensional unbalanced classification.

Table 4.3: GPMFS versus the non-GP classification methods using SMOTE (AUC×100).

Dataset	SMOTE-1NN	SMOTE-DT	SMOTE-RF	SMOTE-GBDT	SMOTE-NB	SMOTE-MLP	**GPMFS**	
							Best	Mean
Armstrong-2002-v1	96.67	88.46	90.97	89.52	85.71	92.85	100	**98.76**
Golub-1999-v1	93.75	89.91	89.80	92.86	68.75	96.43	100	**99.14**
Colon	74.40	64.04	64.46	65.83	47.62	62.60	89.29	**78.02**
Leukemia	90.18	86.61	81.52	86.61	**100**	69.61	98.21	90.71
Shipp-2002-v1	72.22	68.24	72.22	69.44	58.33	76.67	96.3	**83.75**
DLBCL	69.44	67.31	77.13	72.22	80.86	74.63	100	**88.58**
Gordon-2002	98.91	86.09	92.94	90.93	88.89	85.74	100	**99.21**
Yeoh-2002-v1	86.29	96.03	72.66	96.15	80.58	84.10	100	**99.26**
Tomlins-2006-v1	**98.21**	80.36	88.51	83.75	75	75.95	100	94.48
Lung	85.24	95.21	82.23	**100**	80.00	82.52	100	97.74

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 4.4: GPMFS versus the non-GP classification methods using Borderline-SMOTE1 (AUC×100).

Dataset	B-SMOTE1-1NN	B-SMOTE1-DT	B-SMOTE1-RF	B-SMOTE1-GBDT	B-SMOTE1-NB	B-SMOTE1-MLP	**GPMFS**	
							Best	Mean
Armstrong-2002-v1	96.67	89.16	85.29	89.52	85.71	92.86	100	**98.76**
Golub-1999-v1	93.75	90.15	83.96	92.86	68.75	93.75	100	**99.14**
Colon	74.40	63.12	65.04	62.26	47.62	74.40	89.29	**78.02**
Leukemia	**96.43**	86.61	76.16	86.61	93.75	75	98.21	90.71
Shipp-2002-v1	75.00	68.42	68.15	69.44	66.67	75	96.3	**83.75**
DLBCL	80.56	70.37	70.93	73.43	80.56	69.44	100	**88.58**
Gordon-2002	97.83	93.94	91.96	93.36	88.89	**100**	100	99.21
Yeoh-2002-v1	96.15	96.15	66.90	96.15	82.20	87.28	100	**99.26**
Tomlins-2006-v1	**100**	84.29	80.36	81.67	62.50	**100**	100	94.48
Lung	95.24	95.75	78.55	**99.92**	70.00	90.00	100	97.74

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 4.5: GPMFS versus the non-GP classification methods using Borderline-SMOTE2 (AUC$\times$100).

Dataset	B-SMOTE2-1NN	B-SMOTE2-DT	B-SMOTE2-RF	B-SMOTE2-GBDT	B-SMOTE2-NB	B-SMOTE2-MLP	**GPMFS**	
							Best	Mean
Armstrong-2002-v1	93.33	89.79	88.86	89.52	**100**	92.86	100	98.76
Golub-1999-v1	93.75	90.03	79.11	92.86	77.68	**100**	100	99.14
Colon	70.24	76.69	62.74	77.38	43.45	60.12	89.29	**78.02**
Leukemia	96.43	86.61	78.01	86.61	**100**	81.25	98.21	90.71
Shipp-2002-v1	77.78	73.61	72.50	72.22	75	**91.67**	96.3	83.75
DLBCL	75	76.85	75.37	83.33	**88.89**	69.44	100	88.58
Gordon-2002	97.83	93.68	89.41	93.36	**100**	**100**	100	99.21
Yeoh-2002-v1	**100**	99.49	68.23	**100**	90.69	87.28	100	99.26
Tomlins-2006-v1	89.29	87.50	92.38	87.50	62.50	**98.21**	100	94.48
Lung	91.67	99	80.31	**100**	80.00	87.62	100	97.74

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 4.6: GPMFS versus the non-GP classification methods using ADASYN (AUC×100).

Dataset	ADASYN-1NN	ADASYN-DT	ADASYN-RF	ADASYN-GBDT	ADASYN-NB	ADASYN-MLP	**GPMFS**	
							Best	Mean
Armstrong-2002-v1	93.33	89.59	91.97	89.52	92.86	92.85	100	**98.76**
Golub-1999-v1	93.75	90.51	89.35	92.86	68.75	92.86	100	**99.14**
Colon	74.40	66.69	66.19	62.20	47.62	67.26	89.29	**78.02**
Leukemia	83.04	86.61	80.47	86.61	**100**	65.15	98.21	90.71
Shipp-2002-v1	83.33	80.46	75.00	83.33	58.33	80.92	96.3	**83.75**
DLBCL	77.78	67.31	74.35	72.22	**88.8**	79.44	100	88.58
Gordon-2002	98.91	83.24	95.01	83.33	88.89	83.15	100	**99.21**
Yeoh-2002-v1	91.13	95.64	72.66	96.15	86.04	82.07	100	**99.26**
Tomlins-2006-v1	**98.21**	80.36	86.90	82.5	62.5	85.53	100	94.48
Lung	76.90	97.83	79.74	**99.67**	80.00	74.60	100	97.74

1: Bold values are the highest AUC result achieved by methods on a dataset.

4.5 Further Analysis

In this section, we further analyze the results of GPMFS, with a goal to reveal contributions of different components to improving the effectiveness and efficiency of GPMFS. In the experiments, we examine and investigate the influence of using the quasi-dominance relation on $C2$, the influence of using the program size $C3$, and the influence of setting the dominant threshold d.

The GPMFS variants are listed as follows:

- GP$_{C1}$: GP uses $C1$ as the fitness function and tournament selection.
- GP$_{No}$: GP uses the proposed fitness function and the selection method, where the quasi-dominance relation is not considered on $C2$.
- GP$_{NoSize}$: GP uses the proposed fitness function and the selection method, where the program size $C3$ is not considered.
- GP$_{Prop}$: GP uses proportional selection, and $C1$ and $C2$ are equally weighted and summed together as the fitness function.
- GP$_{Tourna}$: GP uses tournament selection, and $C1$ and $C2$ are equally weighted and summed together as the fitness function. Note that GP$_{Tourna}$ was called GP_{Min_Corr} in Chapter 3.
- GPMFS(dR): GP uses the proposed fitness function and the three-criterion tournament selection that a dominant threshold d is an uniformly distributed random number in the range of (0, 0.2]. Note that, the main difference between GPMFS and GPMFS(dR) is the setting of the dominant threshold d.

The results of these methods are reported in Table 4.7, including AUC results, training time, and the information related to program sizes (in the final generation).

Table 4.7: AUC, training time and program sizes of the GPMFS variants for further analysis.

		AUC ×100		Training Time (Seconds)	Program Size (Best Programs)		Averaged Program Size (The whole population)	
Dataset	Method	Best	Mean±Std	Mean	Smallest	Mean±Std	Smallest	Mean±Std
Armstrong-2002-v1	GP_{C1}	100	96.03 ± 4.44	138.36	5	30.77 ± 27.25	70.43	106.8 ± 20.62
	$GP_{Noquasi}$	100	97.68 ± 3.99	124.79	27	89.2 ± 29.04	7.06	83.4 ± 27.26
	GP_{NoSize}	100	98.95 ± 2.38	120.19	5	76.27 ± 43.43	63.81	97.55 ±18.21
	GP_{Prop}	100	96.35 ± 7.14	151.33	3	56 ± 44.99	78.27	96.27 ±16.3
	GP_{Tourna}	100	96.63 ± 4.55	143.6	54	127.27 ±51.03	53.38	124.81 ± 45.44
	GPMFS(dR)	100	**99.3 ± 1.51**	115.42	5	26.2 ± 22.84	6.47	10.19 ± 5.97
	GPMFS	100	98.76 ± 2.33	**105.99**	5	**17.57 ± 13.05**	6.61	**8.37 ±2.78**
Golub-1999-v1	GP_{C1}	100	97.44 ± 5.59	231.29	14	59.1 ± 38.24	70.79	119.94 ± 40.25
	$GP_{Noquasi}$	100	98.13 ± 4.5	203.69	58	111.47 ± 36.26	61	86.39 ± 35.7
	GP_{NoSize}	100	97.53 ± 4.44	206.47	17	75.6 ±42.1	48.06	106.69 ± 24.65
	GP_{Prop}	100	97.22 ± 5.69	222.38	7	65.97 ± 50.9	71.96	102.92 ± 39.04
	GP_{Tourna}	100	97.23 ± 7.42	252.73	70	136.8 ±43.34	71.03	132.62 ± 37.44
	GPMFS(dR)	100	98.96 ± 2.44	189.4	7	**33 ± 18.32**	6.88	15.85 ±7.62
	GPMFS	100	**99.14 ± 2.64**	**169.33**	9	33.27 ± 19.12	8.12	**14.09 ± 6.63**
Colon	GP_{C1}	96.43	**78.53 ± 9.06**	206.74	12	93.53 ± 68.61	82.45	143.4 ± 65.39
	$GP_{Noquasi}$	92.86	73.08 ± 9.48	187.25	31	96.6 ± 27.65	50.86	87.54 ± 23.8
	GP_{NoSize}	90.48	77.58 ± 8.25	192.66	7	81.5 ± 37.09	70.52	105.96 ± 24.1
	GP_{Prop}	90.48	74.73 ± 7.76	217.68	3	81.23 ± 64.15	70.24	105.18 ± 29.95
	GP_{Tourna}	91.67	74.76 ± 12.4	225.65	35	109.2 ± 48.32	38.74	106.17 ± 38.77

Continued on next page

Table 4.7 – Continued from previous page

	GPMFS(dR)	91.67	77.69 ± 7.75	**162.88**	5	**33.1 ± 21.04**	6.55	**19.1 ± 12.81**
	GPMFS	89.29	78.02 ± 6.06	168.15	5	35.53 ± 21.82	6.55	20.03 ± 12.72
Leukemia	GP_{C1}	100	**90.74 ±5.75**	776.28	5	34.97 ± 25.86	71.78	107.54 ±31.56
	$GP_{Noquasi}$	100	89.88 ± 6.12	666.95	50	115.67 ± 41.38	42.22	104.89 ± 36.44
	GP_{NoSize}	99.11	88.84 ± 7.23	694.68	9	72.53 ± 50.12	72.28	98.96 ± 19.98
	GP_{Prop}	100	88.63 ±6.69	715.23	3	50.8 ± 46.73	60.77	95.92 ± 27.75
	GP_{Tourna}	100	90.09 ± 6.81	793.88	18	118.53 ± 50.39	47.25	115.99 ± 40.08
	GPMFS(dR)	100	90.45 ± 5.59	668.65	3	**21.37 ± 18.78**	6.66	**10.26 ± 4.17**
	GPMFS	98.21	90.71 ± 6.74	**641.39**	5	23.47 ± 17.32	6.81	11.05 ± 5.61
Shipp-2002-v1	GP_{C1}	92.59	78.12 ±13.49	224.18	36	91.87 ± 43.51	76.36	135.03±30.18
	$GP_{Noquasi}$	100	**84.81±8.07**	210.87	9	104.0 ±58.21	7.18	92.25±48.05
	GP_{NoSize}	100	82.16 ± 9.7	216.89	20	90.9 ±43.99	53.49	99.3±26.95
	GP_{Prop}	99.07	77.13 ±12.69	237.54	28	78.3 ± 30.92	65.61	104.74±25.53
	GP_{Tourna}	98.15	82.65 ± 9.87	215.61	58	113.1±49.32	46.81	109.39±42.39
	GPMFS(dR)	96.3	84.17 ±8.85	197.29	13	47.67 ±19.84	20.31	32.42±7.76
	GPMFS	96.3	83.75±7.55	**193.76**	15	**40.03± 18.15**	12.03	**26.94±12.5**
DLBCL	GP_{C1}	96.3	75.34 ± 13.71	565.51	19	111.9 ± 74.86	46.84	142.18 ± 65.56
	$GP_{Noquasi}$	96.3	79.04 ± 12.52	538.77	23	100.03 ± 44.11	33.36	98.92 ± 39.82
	GP_{NoSize}	98.15	82.44 ± 11.82	585.1	14	109.33 ± 64.84	69.18	123.45 ± 49.69
	GP_{Prop}	99.07	77.7 ± 13.63	651.71	5	71.7 ± 61.91	64.84	103.19 ± 34.68
	GP_{Tourna}	100	81.14 ± 15.28	670.87	50	137.93 ± 71.87	55.27	140.67 ± 74.62
	GPMFS(dR)	100	86.64 ± 9.57	546.79	11	36.17 ± 16.03	11.22	23.7 ± 10.05
	GPMFS	100	**88.58 ±8.67**	**504.44**	5	**33.27 ± 24.98**	6.76	**23.09 ± 19.19**
	GP_{C1}	100	98.65±1.94	721.9	7	30 ±19.01	67.02	100.73±21.9

Continued on next page

Table 4.7 – Continued from previous page

Gordon-2002	$GP_{Noquasi}$	100	97.71±3.23	727.74	31	101.0 ±44.15	33.46	89.78±34.44
	GP_{NoSize}	100	98.52±1.84	737.03	3	52.23 ± 44.62	70.38	99.92 ± 17.94
	GP_{Prop}	100	97.09 ±3.2	747.43	3	61.67 ± 45.82	68.93	93.07±9.93
	GP_{Tourna}	100	98.48 ±2.35	724.74	39	111.17 ±37.64	40.15	111.0±35.75
	GPMFS(dR)	100	98.53±3.11	701.67	3	25.97 ±19.12	6.44	12.39±5.45
	GPMFS	100	**99.21 ±1.03**	**697.66**	3	**13.1±8.72**	6.65	**7.7±1.04**
Yeoh-2002-v1	GP_{C1}	100	98.06 ± 4.02	675.85	6	47.1 ± 38.08	43.43	98.02 ± 20.37
	$GP_{Noquasi}$	100	96.84 ± 6.18	596.43	34	99.43 ± 47.91	28.42	82.5 ±40.14
	GP_{NoSize}	100	98.31 ± 4.2	616.15	3	62.4 ± 54.66	63.43	90.63 ±13.8
	GP_{Prop}	100	97.79 ± 4.2	665.56	3	56.23 ± 51.67	70.66	93.62 ± 15.44
	GP_{Tourna}	100	98.8 ± 3.35	703.08	46	100.93 ± 27.83	46.82	99.77 ± 27.18
	GPMFS(dR)	100	99.18 ± 3	621.36	5	**18.13 ± 15.45**	6.76	11.19 ± 6.76
	GPMFS	100	**99.26 ± 2.12**	**575.4**	5	19.1 ± 9.52	6.83	**10.98 ± 4.85**
Tomlins-2006-v1	GP_{C1}	100	89.02 ±11.05	650.09	3	15.07 ±11.29	64.33	87.23 ±9.31
	$GP_{Noquasi}$	100	94.18± 9.29	652.31	36	92.93 ±35.97	35.15	84.17±28.79
	GP_{NoSize}	100	92.01 ±12.21	655.1	13	74.6 ± 47.68	58.36	81.22 ±12.73
	GP_{Prop}	100	92.53 ± 9.52	669.88	5	41.07 ± 33.27	60.3	82.14±111.12
	GP_{Tourna}	100	93.65 ± 9.97	650.1	47	112.17 ± 32.73	63.14	107.07±27.49
	GPMFS(dR)	100	**95.77± 8.48**	638.79	6	21.87± 14.5	8.14	11.07±3.09
	GPMFS	100	94.48± 9.2	**636.76**	5	**15.47±20.83**	6.69	**8.39 ± 1.92**
Lung	GP_{C1}	100	95.97 ± 5.77	2230.73	7	51.23 ± 32.04	66.58	112.08 ± 23.77
	$GP_{Noquasi}$	100	91.38 ± 15.71	2093.6	19	94 ± 32.01	11	85.47 ± 27.2
	GP_{NoSize}	100	94.49 ± 10.68	2193.57	11	79.43 ± 47.04	60.72	109.81 ± 26.26
	GP_{Prop}	100	95.25 ± 6.49	2189.17	5	66.03 ± 40.49	70.18	96.85 ± 18

Continued on next page

Table 4.7 – Continued from previous page

	GP_{Tourna}	100	90.1 ± 11.29	2639.07	46	114.63 ± 40.83	61.96	118.64 ± 34.36
	GPMFS(dR)	100	97.56 ± 7.72	2126.55	7	22.1 ± 20.11	6.62	**9.56 ± 3.72**
	GPMFS	100	**97.74 ± 3.83**	**2019.01**	5	**20.1 ± 16.79**	6.93	9.91 ± 3.67

4.5.1 Investigation into the Influence of the Quasi-dominance Relation Defined on $C2$

By comparing the mean AUC results, GPMFS often achieves better performance than $GP_{Noquasi}$ that does not use the quasi-dominance relation on $C2$ in 9 out of the 10 cases. This is mainly because the quasi-dominance relation considers the possible estimation errors of $C2$ values. Therefore, the programs that achieve slightly worse performance than the best program chosen by $C2$ will have a chance to compete with the best program, instead of simply discarding them. It is also noticed that the quasi-dominance relation on $C2$ is another factor contributing to the smaller sizes of programs in the whole population. Based on Table 4.7, for GPMFS, the average size of programs in the whole population is obviously smaller than $GP_{Noquasi}$ on all the datasets. This is mainly because some programs that are slightly worse than the best program (in a tournament) may have a smaller program size. Unfortunately, these programs are discarded by $GP_{Noquasi}$, but in GPMFS, these programs have a chance to compete with the best program based on the program size $C3$.

4.5.2 Investigation into the Influence of Considering Program Sizes ($C3$)

The mean AUC results achieved by GPMFS are better than GP_{NoSize} in 9 out of the 10 cases. In GPMFS, the average size of programs in the whole population is significantly smaller than GP_{NoSize} that does not utilize the program size $C3$ as the final criterion to determine which program is finally selected in a tournament. The

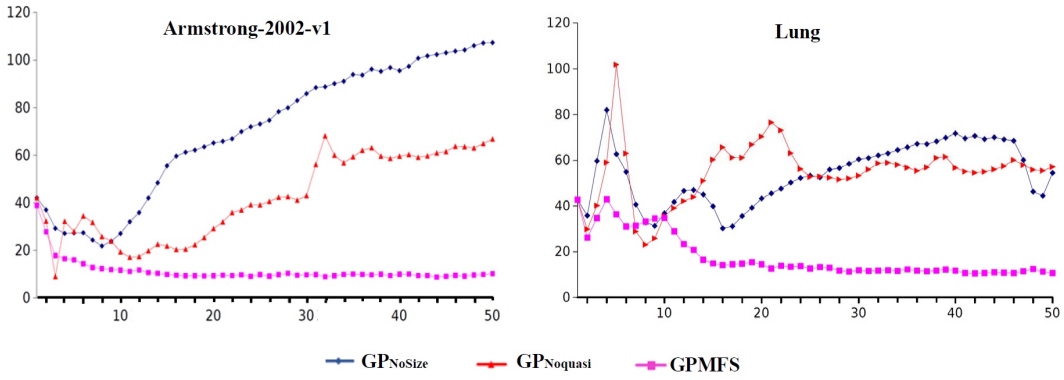


Figure 4.3: Changes of the average size of programs in a population during the training process (On Armstrong and Lung).

small program sizes in the whole population may contribute to the less training time consumed by GPMFS.

Figure 4.3 is used to show the changes of the average size of programs in a population during 50 generations in a GP run on Armstrong (a slightly unbalanced dataset) and Lung (a highly unbalanced dataset). The blue, red and pink lines describe changes of the average size of programs in a population in GP$_{NoSize}$, GP$_{Noquasi}$ and GPMFS, respectively. The two pictures in Figure 4.3 show that the pink line is clearly under the blue and red lines after around 15 generations. It concludes that the quasi-dominance relation on $C2$ and program size $C3$ should be considered together to control the growth of program size.

4.5.3 Investigation into Selection Operators

For GP$_{Prop}$ and GP$_{Tourna}$, $C1$ and $C2$ are equally weighted and summed together to be a fitness function. GP$_{Prop}$ uses proportional selection, while GP$_{Tourna}$ uses tournament selection. Based on the mean AUC results in Table 4.7, GPMFS achieves better performance than GP$_{Prop}$ and GP$_{Tourna}$ in all the 20 cases. In addition, GPMFS consumes less training time than GP$_{Prop}$ and GP$_{Tourna}$, even though the additional calculations are required to identify and select programs

in the three-criterion tournament selection. This is mainly because the average size of programs in a population is significantly smaller than that of GP_{Prop} and GP_{Tourna}.

For further analysis, GP_{Prop} and GP_{Tourna} are compared with GP_{NoSize} because the three methods use $C1$ and $C2$ and do not use $C3$. In most cases, GP_{NoSize} achieves at least similar performance compared with GP_{Prop} and GP_{Tourna}. This is because GP_{Prop} and GP_{Tourna} equally weigh $C1$ and $C2$ and sum them together, but $C1$ and $C2$ are not always equally important. $C2$ should be used to distinguish between two programs that achieve the same $C1$ value.

4.5.4 GPMFS(dR) Versus GPMFS

Experiments using GPMFS(dR) are conducted to further investigate the dominant threshold d (where d is used to calculate the quasi-dominating set of the best program based on $C2$) in the three-criterion tournament selection. By comparing the results of GPMFS(dR) and GPMFS, they achieve similar classification performance in almost all cases. This could indicate that the proposed selection method is not sensitive to the dominant threshold d.

4.5.5 Evolved Programs by GPMFS

Figure 4.4 shows two examples of the evolved programs by GPMFS. Note that in Figure 4.4, fnum stands for the (num+1)st feature in a dataset. For example, f690 is the 691st feature.

The program in Figure 4.4 (a) is the best program from the final generation of a GP run on Armstrong-2002-v1. The AUC result of this program is 1 on the test set. This program has a small size, only having 7 nodes in total. Armstrong-2002-v1 has 1081 features, but only 4 features are used by this program. Furthermore, for this GP run, the average size of programs in the whole population is 7.171. It reveals that programs evolved by GPMFS often use a very small number of features. The program in Figure 4.4 (b) is an evolved program on Lung (Lung has 12600 features, 156 instances, and $IR = 8$). Lung has 12600 features, but only 3

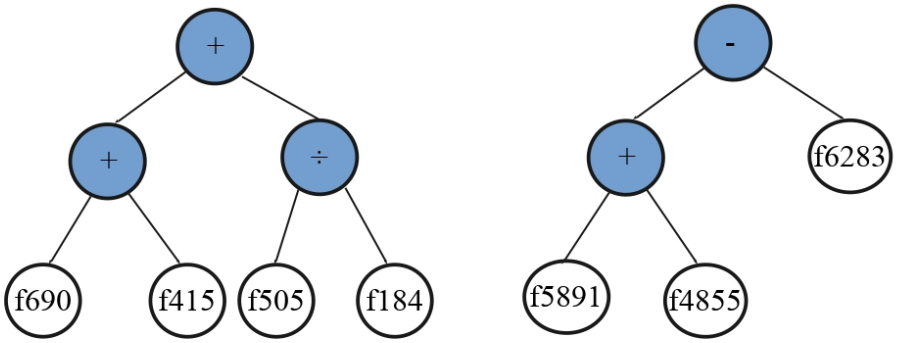


Figure 4.4: The examples of the evolved programs (On Armstrong and Lung).

features are used by this program to achieve a perfect AUC score on the test set (i.e. 1).

For gene expression datasets, many of them have thousands of features. As can be seen from Figure 4.4, the GP classifiers constructed by GPMFS only use a handful of features on Armstrong-2002-v1 and Lung to achieve good performance on the test sets. A possible reason is explained as follows. At the beginning of the evolutionary learning process, it is possible that some GP trees in a population select informative features to achieve good classification performance. Those trees are very likely to be selected as parents to generate offspring for the next generation. Due to the consideration of the program size in the selection process, those trees become smaller and smaller (i.e. irrelevant or redundant features are gradually removed, and good features are kept).

4.6 Chapter Summary

The goal of this chapter was to investigate how two criteria can be combined without requiring a pre-designed weight in the fitness evaluation process for improving

the classification performance of GP for high-dimensional unbalanced classification. To achieve the goal, in this chapter, a new GP method (named GPMFS) was proposed by developing a new two-criterion fitness function and three-criterion tournament selection. In the proposed fitness function, the two criteria, i.e. $C1$ and $C2$, are calculated independently in the evaluation process to return fitness tuples for the following selection process. The three-criterion tournament selection is designed to identify and select good programs in a tournament.

To examine the performance of GPMFS, high-dimensional unbalanced datasets were used. The experimental results show that GPMFS achieved promising performance on high-dimensional unbalanced datasets. Based on the AUC results on the test sets, GPMFS often achieved significantly better than or similar performance to other methods in both slightly and highly unbalanced cases. Further investigations on the contributions of different components in GPMFS show that the classification performance is improved by the cooperation of $C1$, $C2$ and program size $C3$ in the three-criterion tournament selection. Moreover, the high efficiency of GPMFS is gained due mainly to the small size of programs in a population.

In Chapters 3 and 4, we have investigated how the class imbalance issue can be resolved by means of fitness function in GP. In the next Chapter, we start to investigate how cost-sensitive learning is used with GP for high-dimensional unbalanced classification.

Chapter 5

Value-based Cost-sensitive GP

5.1 Introduction

Cost-sensitive learning [35] treats different errors differently, which can be utilized with classification algorithms to make them sensitive to different types of misclassification. Cost-sensitive learning has shown to be effective in addressing the problem of class imbalance in machine learning [38]. However, the use of cost-sensitive learning with GP has seldom been investigated, particularly for high-dimensional unbalanced data.

To date, most existing cost-sensitive algorithms work with a cost matrix that is often required from domain experts. Unfortunately, in many cases, experts feel difficult to provide exact cost values due to the lack of domain or specialized knowledge [102]. Besides, it is likely for different experts to have different opinions on the same error. Therefore, in many real-world applications, the misclassification cost values are unknown [209].

Without cost information from domain experts, a straightforward way is to use the class imbalance ratio of a dataset as the cost information [38]. However, this method is criticized mainly because it is over-simplified and does not consider data characteristics [38]. Moreover, this method assumes a direct relationship between class imbalance and cost sensitivity, which does not always hold. Many existing cost-sensitive algorithms usually use trial and error to determine cost ma-

trices, which may cause additional computations but may not lead to an optimal solution [209]. The acquisition of a cost matrix is still an open issue in cost-sensitive learning [38]. Therefore, it is essential to investigate how a cost matrix could be automatically obtained to construct cost-sensitive classifiers.

5.1.1 Chapter Goals

The overall goal of this chapter is to investigate how GP can be used with cost-sensitive learning to achieve good classification performance for high-dimensional unbalanced classification. In order to achieve this goal, there are three sub-goals:

1) Investigate how cost values are incorporated into GP,

2) Develop a GP method to construct cost-sensitive GP classifiers, where a cost matrix is automatically learned, and

3) Investigate whether the proposed method can achieve significantly better or similar performance than other methods.

5.1.2 Chapter Organization

The remainder of this chapter is as follows. Section 5.2 introduces the proposed method. Section 5.3 introduces experimental design. The results are discussed and analyzed in Section 5.4, and further analysis is in Section 5.5. In Section 5.6, we conclude this chapter.

5.2 The Proposed Method

In this section, we introduce the proposed method, named **C**ost-**S**ensitive **G**enetic **P**rogramming (**CS-GP**).

5.2.1 The Overall Design

In CS-GP, the *class-dependent misclassification cost matrix* (it was introduced in Section 2.2.2 in Chapter 2, Page 29) is considered to develop cost-sensitive classifiers. CS-GP is based on strongly-typed GP (STGP) [143]. This is because STGP makes it possible to evolve trees that follow a pre-designed tree representation. Different from standard tree-based GP, in STGP, every terminal has a data type, and each function also has types for its arguments and its returned values [143]. Every individual cannot violate type constraints in STGP.

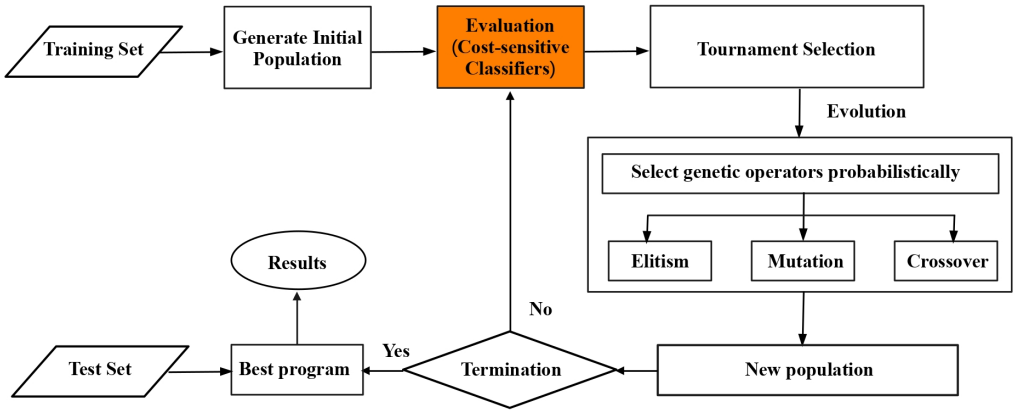


Figure 5.1: The overall design of CS-GP.

In CS-GP, for a tree, by using data type constraints, its left subtree is used to evolve classifiers, while its right subtree is used to learn cost values. The cost value evolved by the right subtree will be used to calculate a classification threshold for the evolved classifier (i.e. the left subtree) to make classification predictions (details will be introduced in the following subsections).

The overall design of CS-GP is shown in Figure 5.1. The initial population is generated by the ramped half-and-half initialization method. Then, the goodness of every tree in a population is evaluated by a fitness function, which is designed as the geometric mean G_Mean (Eq. (2.6), Page 24). Based on the fitness values, tournament selection is used to select better individuals. The new population is

generated by genetic operators, i.e. mutation, crossover and elitism. The evolutionary learning process stops after GP finishes its search in the final generation. The best tree is selected from the final generation to be used for classification on a test set.

5.2.2 How Cost-sensitive Classifiers can be Constructed by GP?

For binary classification, a classification threshold TH is often used to separate the output values of a GP tree to predict the majority class and the minority class. Usually, this classification threshold is set to 0 for separating the original output values, or 0.5 if the output values are converted into the range of [0,1] [11]. This threshold setting method works effectively in balanced classification, but may not be always effective in unbalanced classification. This is because the classification performance of an individual is only evaluated at this predefined threshold, but varying this threshold may result in different classification performance.

In cost-sensitive learning, the threshold-moving method is commonly used by algorithms for unbalanced classification [103]. The idea of the threshold-moving method is straightforward, i.e. the classification threshold moves towards the inexpensive instances with a lower misclassification cost, to enable expensive instances with a higher cost to be easily classified [103]. Figure 5.2 explains the threshold-moving idea. In Figure 5.2, in Case 1, when $TH = 0.5$ is used as the threshold, two instances from the majority class are mistakenly classified into the minority class, and two instances from the minority class are mistakenly classified into the majority class. Note that the two kinds of mistakes have different misclassification costs, and the cost of the minority class is greater than or equal to that of the majority class. After moving the classification threshold towards the majority class, like Case 2, two more instances from the minority class can be correctly classified, contributing to the decreased total cost.

In this chapter, we use the threshold-moving idea. To minimize the total cost of classification predictions, a classification threshold TH can be calculated based on the misclassification cost values to determine how much it moves towards the majority class [103]. In Chapter 2, we introduced the general process of making

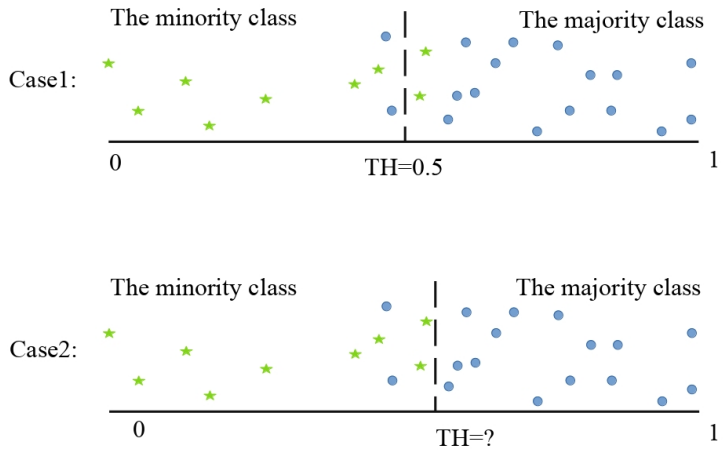


Figure 5.2: Threshold-moving idea.

optimal classification predictions by considering the misclassification cost values (Page 31). This is the principle behind the threshold-moving method. Based on this principle, a new classification threshold is defined as [35]:

$$TH = \frac{C_{10} - C_{00}}{C_{10} - C_{00} + C_{01} - C_{11}} \tag{5.1}$$

where C_{10} is a cost of a false negative, C_{01} is a cost of a false positive, C_{00} and C_{11} are the costs of a true positive and a true negative, respectively.

This threshold is used by a GP classifier to separate its output values for predicting the majority class and the minority class. It is assumed that C_{00} and C_{11} are equal to 0, i.e. correct predictions do not cause misclassification cost. The misclassification cost of a false positive C_{01} is set to 1, and the cost of a false negative C_{10} is set to C (C is a constant and $C \geq 1$) [101]. Therefore, Eq. (5.1) is simplified to:

$$TH = \frac{C_{10}}{C_{10} + C_{01}} = \frac{C}{C+1} \tag{5.2}$$

Hence, for instance x, if $p_x \geq TH$ (p_x is the probability of predicting x into the majority class), then x is classified into the majority class; otherwise x is classified into the minority class.

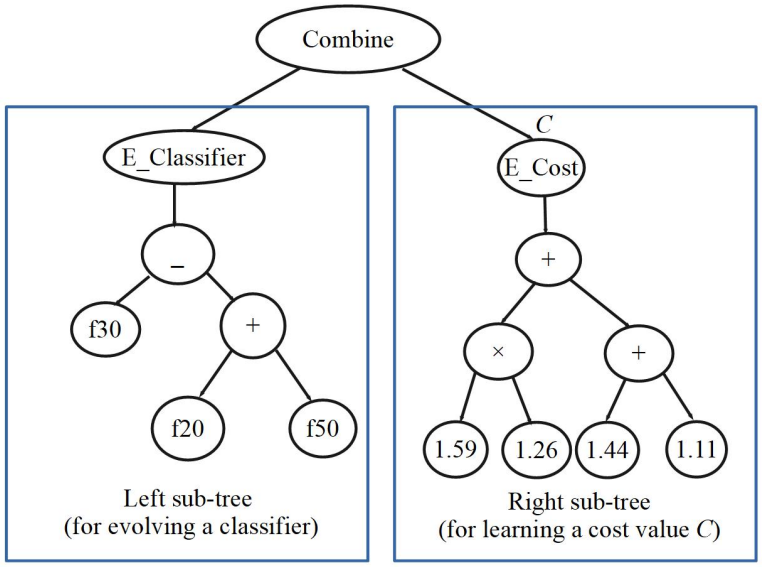


Figure 5.3: An example for tree representation.

We now start to introduce how a classifier is constructed and how a cost value C is learned automatically and simultaneously in CS-GP.

Tree representation. For an individual, its left subtree works as a classifier, meanwhile its right subtree is used to optimize a cost value C. Figure 5.3 intuitively explains how an individual works in CS-GP. Table 5.1 reports the terminal and function sets designed for CS-GP.

Classifier construction. In a tree, its *left subtree* is used as a *classifier*. The terminal set includes all the features and a random constant. The function set has six functions, including four arithmetic operators (i.e. +, $-$, $\times$ and protected division $\div$), If and $E\_Classifier$. The protected division $\div$ returns zero when dividing by zero. For function If, if the first argument is negative, the second argument is returned, otherwise the third argument is returned. For $E\_Classifier$, it takes one argument $argu$ (the value of $argu$ is the output of the evolved classifier, and its data type is $Iput$), and then it directly returns the value of $argu$ (output data

Table 5.1: Function and terminal sets in CS-GP.

A left subtree (for constructing a classifier)				
Terminal Sets		Function Sets		
Name	Type	Name	Type (Input)	Type (Output)
• Features of a dataset	$Iput$ (float)	• +	[$Iput$, $Iput$]	$Iput$
• A random constant	$Iput$ (float)	• $-$	[$Iput$, $Iput$]	$Iput$
		• $\times$	[$Iput$, $Iput$]	$Iput$
		• $\div$ (protected)	[$Iput$, $Iput$]	$Iput$
		• If	[$Iput$, $Iput$, $Iput$]	$Iput$
		• $E\_Classifier$	[$Iput$]	$Predi$ (float)
A right subtree (for learning a cost value)				
• Uniformly distributed random numbers in the range of [1, 2]	$Icost$ (float)	• +	[$Icost$, $Icost$]	$Icost$
		• $SubCost$	[$Icost$, $Icost$]	$Icost$
		• $\times$	[$Icost$, $Icost$]	$Icost$
		• $DivCost$	[$Icost$, $Icost$]	$Icost$
		• E_Cost	[$Icost$]	$Ocost$ (float)
A root node				
		• $Combine$	[$Predi$, $Ocost$]	list

1: Note that $Iput$, $Predi$, $Icost$, and $Ocost$ are different data types in the evolved tree program, even though they are actually the same type (float).

type is $Predi$).

We take an example in Figure 5.3 to explain how a left subtree works. In Figure 5.3, f30, f20 and f50 are features selected by the left subtree (working as a classifier). The left subtree can be translated into an arithmetic expression $f30 - (f20 + f50)$. The output of the arithmetic expression will be used for classification predictions.

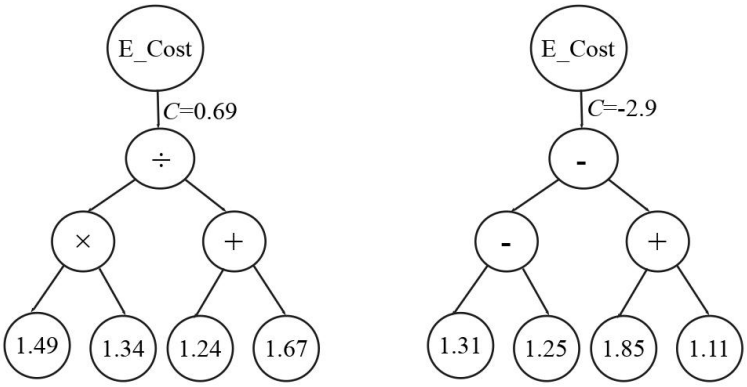
Cost optimization. In a tree, its *right subtree* is used for learning *a cost value* C in Eq. (5.2). The terminals of a right subtree are uniformly distributed random numbers in the range of [1, 2].

The function set has four arithmetic operators (i.e. $+$, $SubCost$, $\times$ and $DivCost$) and a function named E_Cost. Note that + and $\times$ are the original arithmetic operators, while $SubCost$ and $DivCost$ are slightly different from $-$ and $\div$. This is because standard $-$ and $\div$ might produce two kinds of risky cost values for the minority class:

- The generated cost value C is less than 1 (i.e. the misclassification cost of the minority class is less than that of the majority class);
- The generated cost value C is a negative value (i.e. the misclassification cost of the minority class is less than that of a correct prediction).

Figure 5.4 shows two right subtrees that use standard arithmetic operators $-$ or $\div$. In Figure 5.4 (a), the generated cost value is $C = (1.49 \times 1.34) \div (1.24 + 1.67) = 0.69$, which is less than 1. In Figure 5.4 (b), the generated cost value is $C = (1.31 - 1.25) - (1.85 + 1.11) = -2.9$, which is less than 0.

The two possible cases are dangerous. In the first case, because the evolved misclassification cost value for the minority class is less than that of the majority class (i.e. 1), the constructed classifiers by using this cost value are more likely to be biased towards the majority class. The second case is more serious than the first, because the evolved misclassification cost value for the minority class is less than that of the correct predictions (i.e. 0). This may cause the evolved classifiers to be less sensitive to correct predictions.



(a) Right sub-tree1 (b) Right sub-tree2

Figure 5.4: Two right subtrees.

To avoid the two kinds of risks, $SubCost$ and $DivCost$ are designed as:

$$SubCost(c1, c2) = max(1, abs(c1 - c2)) \tag{5.3}$$

$$DivCost(c1, c2) = \begin{cases} \frac{c_1}{c_2}, \ if \ \frac{c_1}{c_2} \geq 1 \\ \frac{c_2}{c_1}, \ otherwise \end{cases} \tag{5.4}$$

By using $SubCost$ and $DivCost$, the cost value C represented by a right subtree is greater than or equal to 1.

For E_Cost, it takes one argument $argu$ (the value of $argu$ is the evolved cost value C, with the data type of $Icost$), and then it directly returns the value of $argu$ (data type is $Ocost$).

We take an example in Figure 5.3 to explain how a right subtree works for learning a cost value C. For the right subtree in Figure 5.3, 1.59, 1.26, 1.44 and 1.11 are initially generated cost values taken from the uniformly distributed random numbers in the range of [1, 2]. Note that the initially generated cost values have sixteen decimal places (float type), and they are rounded to two decimal places in this example. The learned cost value is $C = 1.59 \times 1.26 + (1.44 + 1.11) = 4.6$. Therefore, a classification threshold TH is 0.82 (calculated by Eq. (5.2)).

A root of a tree For a tree evolved by CS-GP, a root node is a function, named $Combine$. $Combine$ takes two arguments, i.e. $argu1$ and $argu2$ ($argu1$ with the data type of $Predi$ is the output of $E\_Classifier$, and $argu2$ with the data type of $Ocost$ is the output of E_Cost). $Combine$ directly returns [$argu1$, $argu2$], and its data type is of list.

5.2.3 Classification Predictions

In order to use the threshold-moving idea, the output values of a left subtree (as a classifier) are normalized into the range of [0, 1] by min-max normalization, which is defined as:

$$p_x = 1 - \frac{Pout_x - min(PL)}{max(PL) - min(PL)} \tag{5.5}$$

where $Pout_x$ indicates the output value of a left subtree taking instance x as an input, PL is a list of $Pout_x$ for all the training instances, $min(PL)$ and $max(PL)$ are the minimum value and the maximum value in PL, respectively.

Therefore, for instance x, if $p_x \geq TH$, then x is classified into the majority class (Maj); otherwise x is classified into the minority class (Min). For example, in Figure 5.3, the output of the arithmetic expression $f30 - (f20 + f50)$ is normalized into [0, 1] by Eq. (5.5). For instance x, if p_x is greater than or equal to the threshold TH (i.e. 0.82 in the example), x is classified into Maj, otherwise it is classified into Min.

5.2.4 Fitness Function

After making classification decisions, G_Mean is used as the fitness function to evaluate the classification performance of every individual because it considers the true positive rate and the true negative rate. G_Mean is defined as:

$$G\_Mean = \sqrt{\frac{TP}{TP + FN} * \frac{TN}{TN + FP}} \tag{5.6}$$

where TP is true positive, FP is false positive, TN is true negative and FN is false negative.

Table 5.2: Dataset description.

Dataset	#Features	#Instances	Majority Class (Proportion %)	Minority Class (Proportion %)	IR (Rounding)
Armstrong-2002-v1	1081	72	66.67	33.33	2
Golub-1999-v1	1868	72	65.28	34.72	2
Colon	2000	62	64.52	35.48	2
Leukemia	7129	72	65.28	34.72	2
Shipp-2002-v1	798	77	75.32	24.68	3
DLBCL	5469	77	75.32	24.68	3
Gordon-2002	1626	181	82.87	17.13	5
Yeoh-2002-v1	2526	248	82.66	17.34	5
Tomlins-2006-v1	2315	104	88.46	11.54	8
Lung	12600	156	89.10	10.90	8

1: The proportions of the majority class and the minority class are rounded to two decimal places.

Note that the evolved cost values are not the true cost values provided by domain experts (the true cost values are totally unknown). Besides, these cost values may not be used by other cost-sensitive classification algorithms. This is because, for a tree, the cost value evolved by its right subtree is only used by the classifier represented by its left subtree to be evaluated by the fitness function for selecting good individuals.

5.3 Experiment Design

5.3.1 Datasets

In the experiments, to examine and investigate the effectiveness of CS-GP, ten gene expression datasets (the same as that in the previous contribution chapters) were used, listed in Table 5.2. A dataset is split into two sets: 70% as the training set and 30% as the test set, based on stratified sampling to make sure the same class imbalance ratio (IR) in the training set and the test set.

Table 5.3: Parameter settings.

Parameter	Values	Parameter	Values
Population size	1024	Mutation rate	0.2
Generations	50	Crossover rate	0.8
Initial population	Ramped half-and-half	Elitism	1
Maximum tree depth	10	Selection	Tournament selection (size=6)

5.3.2 Baseline Methods

CS-GP is also compared with the GP and non-GP baseline methods listed in Table 3.2 (in Chapter 3, on Page 63). Apart from these baseline methods, CS-GP is also compared with existing cost-sensitive GP methods, including a cost-sensitive GP method (denoted as GP_{RC_w}) [93], GP with a cost-based fitness function (denoted as GPCF) [140], and GP with a boundary based classification strategy (denoted as GPBC) [140]. Note that, for the used datasets, there is no available cost matrix, so the class imbalance ratio of a dataset is used as the cost information for the three cost-sensitive methods (i.e. GP_{RC_w}, GPCF and GPBC).

5.3.3 Parameter Settings

Table 5.3 reports parameter settings of the GP methods. Subtree crossover, subtree mutation and elitism are used to create a new population [143]. Since CS-GP is based on STGP, the data type constraints are used to ensure valid offspring to be generated by crossover and mutation. Note that the baseline GP methods are based on standard tree-based GP because they do not need to make subtrees to play different roles. For the baseline GP methods, their function set includes +, $-$, $\times$, $\div$ (protected) and If function; their terminal set includes all the features and a random constant. For each GP method, it was run 30 times with 30 different random seeds (all the GP methods use the same set of random seeds).

5.4 Results and Discussions

Table 5.4 reports the AUC results of CS-GP and the baseline GP methods on the test sets. Bold values in Table 5.4 are the highest AUC achieved by the GP methods on a dataset. The Wilcoxon rank-sum tests were conducted to show the significance difference between CS-GP and a baseline GP method, with the significance level of 0.05. In Table 5.4, symbols of "+", "=" and "−" are used to show that CS-GP is significantly better than, similar to, and significantly worse than a baseline method, respectively.

5.4.1 CS-GP Versus the GP Baseline Methods

Table 5.4: CS-GP versus the GP baseline methods on the test sets.

		AUC (×100)			Training time (Seconds)
Datasets	Methods	Best	Mean±Std		Mean
Armstrong-2002-v1	GP_{SMOTE}	100	91.3 ±9.83	+	144.32
	$GP_{BSMOTE1}$	100	94.16 ± 7.43	+	141.78
	$GP_{BSMOTE2}$	100	91.22 ± 10.27	+	153.6
	GP_{ADASYN}	100	92.21 ± 9.62	+	143.04
	GP_{Ave}	100	94.48 ± 8.4	+	114.86
	GP_{G_Mean}	100	92.13 ± 8.01	+	114.88
	GP_{Amse}	100	90.17 ± 7.65	+	146.24
	GP_{Corr}	100	94.67 ± 7.56	+	142.55
	GP_{Dist}	100	95.84 ± 3.93	+	141.33
	GP_{Auc_w}	100	94.46 ± 4.93	+	1917.99
	GP_{RC_w}	100	95.21 ± 5.59	+	126.03
	GPCF	100	95.24±6.03	+	233.37
	GPBC	100	97.60±5.32	=	130.18
	CS-GP	100	**98.79± 3.03**		139.12
	GP_{SMOTE}	100	92.38 ± 10.31	+	195.79
	$GP_{BSMOTE1}$	100	89.11 ± 11.08	+	187.11
	$GP_{BSMOTE2}$	100	85.8± 14.56	+	205.14
	GP_{ADASYN}	100	91.8 ±9.72	+	198.39
	GP_{Ave}	100	91.93 ± 10.09	+	158.77

Continued on next page

Table 5.4 – Continued from previous page

Golub-1999-v1	GP_{G_Mean}	100	88.99 ± 11.89	+	158.31
	GP_{Amse}	100	82.78 ± 11.62	+	226.35
	GP_{Corr}	100	96.06 ± 6.32	+	225.06
	GP_{Dist}	100	96.9 ± 5.23	+	229.09
	GP_{Auc_w}	100	98.42 ± 3.38	=	3089.78
	GP_{RC_w}	100	87.92±13.03	+	185.36
	GPCF	100	97.26±5.81	=	397.15
	GPBC	100	97.11 ± 5.16	=	175.25
	CS-GP	100	**98.95± 2.83**		187.61
Colon	GP_{SMOTE}	92.86	75.99 ± 10.55	+	222.64
	$GP_{BSMOTE1}$	88.1	71.51 ± 12.64	+	206.97
	$GP_{BSMOTE2}$	94.05	73.69 ± 15.28	+	228.62
	GP_{ADASYN}	88.1	74.6 ± 10.25	+	227.64
	GP_{Ave}	91.67	75.52 ± 10.11	+	177.08
	GP_{G_Mean}	92.86	71.51 ± 12.95	+	174.21
	GP_{Amse}	95.24	74.8 ± 10.76	+	203.33
	GP_{Corr}	96.43	75.28 ± 10.1	+	201.08
	GP_{Dist}	92.86	76.59 ± 9.63	=	203.64
	GP_{Auc_w}	91.67	78.97 ± 7.3	=	2348.72
	GP_{RC_w}	88.1	73.13±10.35	+	202.79
	GPCF	92.86	76.43 ± 8.84	=	347.44
	GPBC	94.05	75.44 ± 12.03	+	165.96
	CS-GP	90.48	**79.05 ± 7.03**		185.5
Leukemia	GP_{SMOTE}	100	87.56 ± 10.01	=	919.55
	$GP_{BSMOTE1}$	98.21	85.6 ± 13.38	=	948.52
	$GP_{BSMOTE2}$	99.11	82.47± 12.94	=	1065.41
	GP_{ADASYN}	100	89.73± 8.56	−	951.82
	GP_{Ave}	98.21	88.79 ± 7.74	=	975.7
	GP_{G_Mean}	100	81.79 ± 15.38	=	979.29
	GP_{Amse}	100	81.73 ± 11.84	=	793.34
	GP_{Corr}	100	86.16 ± 10.84	=	785.98
	GP_{Dist}	97.32	86.32 ± 8.95	=	788.22
	GP_{Auc_w}	100	86.28 ± 9.68	=	10396.7
	GP_{RC_w}	99.11	81.1± 14.52	+	871.05
	GPCF	100	**94.42±4.74**	−	1317.99
	GPBC	96.43	84.88±6.91	=	969.78

Continued on next page

Table 5.4 – Continued from previous page

	CS-GP	96.43	85.57±9.03		1006.54
Shipp-2002-v1	GP_{SMOTE}	98.15	82.15 ± 11.77	=	132.95
	$GP_{BSMOTE1}$	95.37	77.93 ± 12.35	+	126.82
	$GP_{BSMOTE2}$	100	79.46 ± 14.87	+	146.3
	GP_{ADASYN}	96.3	79.88 ±12.18	+	137.39
	GP_{Ave}	99.07	82.85 ±9.81	=	95.15
	GP_{G_Mean}	96.3	83.09 ± 9.21	=	97.85
	GP_{Amse}	96.3	75.26 ±13.2	+	225.24
	GP_{Corr}	99.07	83.02 ±13.33	=	216.72
	GP_{Dist}	99.07	**84.81 ±8.96**	=	214.44
	GP_{Auc_w}	100	82.62 ±9.45	=	3192.03
	GP_{RC_w}	97.22	82.56 ± 11.14	=	101.5
	GPCF	98.15	83.14 ± 9.26	=	200.17
	GPBC	95.37	77.42 ± 11.93	+	85.65
	CS-GP	95.37	83.3 ±8.34		100.23
DLBCL	GP_{SMOTE}	98.15	83.21 ±9.56	=	816.12
	$GP_{BSMOTE1}$	99.07	75.77 ± 18.56	+	795.67
	$GP_{BSMOTE2}$	98.15	79.41 ± 11.68	=	846.71
	GP_{ADASYN}	100	79.48± 10.92	=	831.63
	GP_{Ave}	98.15	75.4 ± 15.67	+	740.03
	GP_{G_Mean}	100	77.01 ± 15.75	=	731.45
	GP_{Amse}	100	77.19 ± 13.18	=	638.3
	GP_{Corr}	98.15	81.02± 11.42	=	643.18
	GP_{Dist}	99.07	84.35 ± 9.96	=	633.55
	GP_{Auc_w}	100	85.54 ± 10.83	−	7845.03
	GP_{RC_w}	97.22	79.32±12.99	=	651.66
	GPCF	100	**87.59±9.97**	−	1122.61
	GPBC	98.15	86.11±7.48	−	796.42
	CS-GP	94.44	80.54± 9.06		753.79
Gordon-2002	GP_{SMOTE}	100	97.49 ± 2.92	=	630.05
	$GP_{BSMOTE1}$	100	98.71 ± 2.67	=	584.44
	$GP_{BSMOTE2}$	100	96.35 ± 4.92	=	669.96
	GP_{ADASYN}	100	97.81 ± 2.9	=	598.04
	GP_{Ave}	100	98.26±2.81	=	321.69
	GP_{G_Mean}	100	98.38 ±3.01	=	323.48
	GP_{Amse}	100	96.49 ±4.46	=	734.51

Continued on next page

Table 5.4 – Continued from previous page

	GP_{Corr}	100	96.95 ±6.41	=	718.51
	GP_{Dist}	100	98.9 ±3.62	=	720.18
	GP_{Auc_w}	100	**99.23 ± 2.06**	−	21978.57
	GP_{RC_w}	100	96.86 ± 4.04	=	379.01
	GPCF	100	97.32 ± 4.56	=	697.2
	GPBC	100	95.43 ± 3.71	=	355.91
	CS-GP	100	96.71±4.2		406.25
Yeoh-2002-v1	GP_{SMOTE}	100	87.44 ± 9.24	+	1321.18
	$GP_{BSMOTE1}$	99.75	86.23 ± 10.64	+	1290.39
	$GP_{BSMOTE2}$	98.39	83.27 ± 10.15	+	1388.96
	GP_{ADASYN}	100	84.28± 10.24	+	1452.27
	GP_{Ave}	100	83.97 ± 11.91	+	773.26
	GP_{G_Mean}	95.78	66.33 ± 16.09	+	767.4
	GP_{Amse}	92.06	63.79 ± 12.06	+	717.74
	GP_{Corr}	100	93.29 ± 7.77	+	685.35
	GP_{Dist}	100	91.1 ± 8.22	+	694.73
	GP_{Auc_w}	100	**98.95 ± 2.32**	=	24174.23
	GP_{RC_w}	100	89.14 ± 10.06	+	788.65
	GPCF	100	92.84±4.84	+	1364.63
	GPBC	100	98.64±3.62	=	844.54
	CS-GP	100	97.31± 4.29		811.81
Tomlins-2006-v1	GP_{SMOTE}	100	83.27± 13.43	+	499.69
	$GP_{BSMOTE1}$	100	87.37 ± 10.14	+	521.56
	$GP_{BSMOTE2}$	100	90.18 ± 10.78	+	500.18
	GP_{ADASYN}	100	84.67± 13.16	+	478.33
	GP_{Ave}	100	88.87±13.47	+	293.08
	GP_{G_Mean}	100	84.54 ±14.55	+	296.92
	GP_{Amse}	100	92.96 ± 8.04	+	655.62
	GP_{Corr}	100	90.42 ± 14.1	+	648.78
	GP_{Dist}	100	95.83 ± 6.73	=	646.03
	GP_{Auc_w}	100	91.10 ±9.75	+	7865.15
	GP_{RC_w}	100	86.07 ± 13.11	+	306.15
	GPCF	100	83.89 ± 14.16	+	622.98
	GPBC	100	85.83 ±12.33	+	440.11
	CS-GP	100	**96.98 ±4.3**		328.66
	GP_{SMOTE}	100	81.7 ± 15.9	+	4298.87

Continued on next page

Table 5.4 – Continued from previous page

Lung	$GP_{BSMOTE1}$	100	88.89 ±10.76	+	3631.29
	$GP_{BSMOTE2}$	100	84.16± 15.78	+	3901.94
	GP_{ADASYN}	100	82.97 ± 14.98	+	4137.5
	GP_{Ave}	100	83.46 ± 14.73	+	3048.62
	GP_{G_Mean}	99.05	80.89 ± 18.41	+	3038.89
	GP_{Amse}	100	81.78 ± 16.55	+	2503.15
	GP_{Corr}	100	80.71 ±17.21	+	2490.26
	GP_{Dist}	100	84.27 ± 14.8	+	2493.37
	GP_{Auc_w}	100	92.35 ± 13.23	+	45375.33
	GP_{RC_w}	100	80.08 ± 15.46	+	2629.14
	GPCF	100	90.23±9.48	+	4763.61
	GPBC	100	89.83±11.64	+	3223.64
	CS-GP	100	**97.13± 2.57**		3241.32
Total	76 +, 48 =, 6 −				

In general, CS-GP achieves competitive AUC results compared with the baseline GP methods. Based on the mean AUC results of the 30 runs in Table 5.4, CS-GP performs best on 5 datasets (i.e. Armstrong-2002-v1, Golub-1999-v1, Colon, Tomlins-2006-v1, and Lung). According to the results of the statistical significance tests, CS-GP achieves significantly better or similar performance in 124 out of the 130 cases (significantly better performance in 76 cases and similar performance in 48 cases). When compared with the four oversampling based GP methods, in all the 40 cases, CS-GP achieves significantly better performance in 28 cases and similar performance in 11 cases.

GP_{Ave} adopts Ave as the fitness function, which treats the majority class and minority class as being equally important. Compared with GP_{Ave}, CS-GP achieves significantly better performance in 7 out of the 10 cases (similar performance in the other 3 cases). More importantly, the superiority of CS-GP over GP_{Ave} become more obvious on the highly-unbalanced datasets (i.e. $IR \geq 5$).

GP_{G_Mean} and CS-GP use the same fitness function (i.e. G_Mean), while GP_{G_Mean} uses the standard classification strategy. CS-GP achieves significantly better performance than GP_{G_Mean} on 6 datasets. On the other 4 datasets, CS-GP achieves similar performance to GP_{G_Mean}. More specifically, to compare

the similar performances on the 4 datasets, CS-GP achieves slightly better performance on Leukemia, Shipp-2002-v1 and DLBCL, while it achieves slightly worse performance on Gordon-2002.

When using Auc_w or AUC approximation measures (i.e $Corr$ and $Dist$) as a fitness function, these GP methods (i.e. GP_{Auc_w}, GP_{Corr} and GP_{Dist}) often achieve better classification performance than GP_{Ave}, GP_{G_Mean} and GP_{Amse}. It is noted that GP_{Auc_w} achieves promising classification performance. Compared with GP_{Auc_w}, CS-GP achieves significantly better or similar performance in 8 out of the 10 cases. In more detail, CS-GP achieves significantly better performance on Armstrong-2002-v1, Tomlins-2006-v1 and Lung, and achieves similar performance on 5 datasets, including Golub-1999-v1, Leukemia, Colon, Shipp-2002-v1 and Yeoh-2002-v1.

By comparing CS-GP with GP_{RC_w}, CS-GP achieves significantly better performance on 7 datasets and similar performance on the other 3 datasets. On Golub-1999-v1, Tomlins-2006-v1 and Lung, the superiority of CS-GP over GP_{RC_w} is very obvious. When compared with GPCF and GPBC, CS-GP achieves significantly better or similar performance in 17 out of the 20 cases. In GP_{RC_w}, GPCF and GPBC, the cost matrix is fixed and provided based on the class imbalance ratio, which may not always be suitable. The improved performance of CS-GP may be contributed by its ability to automatically optimize suitable cost values.

5.4.2 CS-GP Versus the Non-GP baseline Methods

In Tables 5.5, 5.6, 5.7 and 5.8, we report the results of the non-GP baseline methods on the test sets.

According to Table 5.5, by comparing the mean AUC result of CS-GP with that of the traditional classification methods using SMOTE on each dataset, CS-GP outperforms in 52 out of the 60 comparisons. According to Table 5.6, by comparing the mean AUC result of CS-GP with that of the traditional classification methods using Borderline-SMOTE1 on each dataset, CS-GP outperforms in 49 out of the 60 comparisons. According to Table 5.7, by comparing the mean AUC result of CS-GP with that of the traditional classification methods using

Table 5.5: CS-GP versus the non-GP classification methods using SMOTE (AUC$\times$100).

Dataset	SMOTE-1NN	SMOTE-DT	SMOTE-RF	SMOTE-GBDT	SMOTE-NB	SMOTE-MLP	**CS-GP**	
							Best	Mean
Armstrong-2002-v1	96.67	88.46	90.97	89.52	85.71	92.85	100	**98.79**
Golub-1999-v1	93.75	89.91	89.80	92.86	68.75	96.43	100	**98.95**
Colon	74.40	64.04	64.46	65.83	47.62	62.60	90.48	**79.05**
Leukemia	90.18	86.61	81.52	86.61	**100**	69.61	96.43	85.57
Shipp-2002-v1	72.22	68.24	72.22	69.44	58.33	76.67	95.37	**83.3**
DLBCL	69.44	67.31	77.13	72.22	**80.86**	74.63	94.44	80.54
Gordon-2002	**98.91**	86.09	92.94	90.93	88.89	85.74	100	96.71
Yeoh-2002-v1	86.29	96.03	72.66	96.15	80.58	84.10	100	**97.31**
Tomlins-2006-v1	**98.21**	80.36	88.51	83.75	75	75.95	100	96.98
Lung	85.24	95.21	82.23	**100**	80.00	82.52	100	97.13

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 5.6: CS-GP versus the non-GP classification methods using Borderline-SMOTE1 (AUC×100).

Dataset	B-SMOTE1-1NN	B-SMOTE1-DT	B-SMOTE1-RF	B-SMOTE1-GBDT	B-SMOTE1-NB	B-SMOTE1-MLP	**CS-GP**	
							Best	Mean
Armstrong-2002-v1	96.67	89.16	85.29	89.52	85.71	92.86	100	**98.79**
Golub-1999-v1	93.75	90.15	83.96	92.86	68.75	93.75	100	**98.95**
Colon	74.40	63.12	65.04	62.26	47.62	74.40	90.48	**79.05**
Leukemia	**96.43**	86.61	76.16	86.61	93.75	75	96.43	85.57
Shipp-2002-v1	75.00	68.42	68.15	69.44	66.67	75	95.37	**83.3**
DLBCL	**80.56**	70.37	70.93	73.43	**80.56**	69.44	94.44	80.54
Gordon-2002	97.83	93.94	91.96	93.36	88.89	**100**	100	96.71
Yeoh-2002-v1	96.15	96.15	66.90	96.15	82.20	87.28	100	**97.31**
Tomlins-2006-v1	**100**	84.29	80.36	81.67	62.50	**100**	100	96.98
Lung	95.24	95.75	78.55	**99.92**	70.00	90.00	100	97.13

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 5.7: CS-GP versus the non-GP classification methods using Borderline-SMOTE2 (AUC$\times$100).

Dataset	B-SMOTE2-1NN	B-SMOTE2-DT	B-SMOTE2-RF	B-SMOTE2-GBDT	B-SMOTE2-NB	B-SMOTE2-MLP	**CS-GP**	
							Best	Mean
Armstrong-2002-v1	93.33	89.79	88.86	89.52	**100**	92.86	100	98.79
Golub-1999-v1	93.75	90.03	79.11	92.86	77.68	**100**	100	98.95
Colon	70.24	76.69	62.74	77.38	43.45	60.12	90.48	**79.05**
Leukemia	96.43	86.61	78.01	86.61	**100**	81.25	96.43	85.57
Shipp-2002-v1	77.78	73.61	72.50	72.22	75	**91.67**	95.37	83.3
DLBCL	75	76.85	75.37	83.33	**88.89**	69.44	94.44	80.54
Gordon-2002	97.83	93.68	89.41	93.36	**100**	**100**	100	96.71
Yeoh-2002-v1	**100**	99.49	68.23	**100**	90.69	87.28	100	97.31
Tomlins-2006-v1	89.29	87.50	92.38	87.50	62.50	**98.21**	100	96.98
Lung	91.67	99	80.31	**100**	80.00	87.62	100	97.13

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 5.8: CS-GP versus the non-GP classification methods using ADASYN (AUC×100).

Dataset	ADASYN-1NN	ADASYN-DT	ADASYN-RF	ADASYN-GBDT	ADASYN-NB	ADASYN-MLP	**CS-GP**	
							Best	Mean
Armstrong-2002-v1	93.33	89.59	91.97	89.52	92.86	92.85	100	**98.79**
Golub-1999-v1	93.75	90.51	89.35	92.86	68.75	92.86	100	**98.95**
Colon	74.40	66.69	66.19	62.20	47.62	67.26	90.48	**79.05**
Leukemia	83.04	86.61	80.47	86.61	**100**	65.15	96.43	85.57
Shipp-2002-v1	**83.33**	80.46	75.00	**83.33**	58.33	80.92	95.37	83.3
DLBCL	77.78	67.31	74.35	72.22	**88.89**	79.44	94.44	80.54
Gordon-2002	**98.91**	83.24	95.01	83.33	88.89	83.15	100	96.71
Yeoh-2002-v1	91.13	95.64	72.66	96.15	86.04	82.07	100	**97.31**
Tomlins-2006-v1	**98.21**	80.36	86.90	82.5	62.5	85.53	100	96.98
Lung	76.90	97.83	79.74	**99.67**	80.00	74.60	100	97.13

1: Bold values are the highest AUC result achieved by methods on a dataset.

Borderline-SMOTE2 on each dataset, CS-GP outperforms in 42 out of the 60 comparisons. According to Table 5.8, by comparing the mean AUC result of CS-GP with that of the traditional classification methods using ADASYN on each dataset, CS-GP outperforms in 50 out of the 60 comparisons.

5.5 Further Analysis on the Evolved Cost Values

Figure 5.5 shows the evolved cost values (for the minority class) on the ten datasets in the 30 independent runs. The vertical axis shows 30 cost values, each of which is evolved by the best individual (its right subtree) from the final generation in a run. The horizontal axis is used to show the cost value evolved by which run (from 1 to 30).

On Armstrong-2002-v1 ($IR = 2$), most of the evolved cost values are in the interval of (1, 10), which is similar to that of being evolved for Leukemia ($IR = 2$) and Gordon-2002 ($IR = 5$). For Colon ($IR = 2$), DLBCL ($IR = 3$) and Yeoh-2002-v1 ($IR = 5$), most of the evolved cost values are in the interval of (1, 5). On Shipp-2002-v1 ($IR = 3$, the same as DLBCL), the evolved cost values are in the interval of (1, 31). For Golub-1999-v1 ($IR = 2$), the evolved cost values are mostly in the intervals of (1, 11) and (16, 25). It is interesting that, for Lung, most cost values are in the interval of (1, 3), which are less than its IR (i.e. 8). Similarly, on Tomlins-2006-v1 ($IR = 8$), the evolved cost values are in the interval of (1, 1.25), which are also less than its IR. Hence, it is hard to conclude whether the evolved cost values have any relationship with IR.

5.6 Chapter Summary

The overall goal of this chapter was to develop a new cost-sensitive GP method, which does not require manually-designed cost matrices and achieves better classification performance in high-dimensional unbalanced classification. To achieve the chapter goal, we investigate and show how cost-sensitive learning can be used with GP to construct cost-sensitive classifiers. Moreover, this chapter shows how

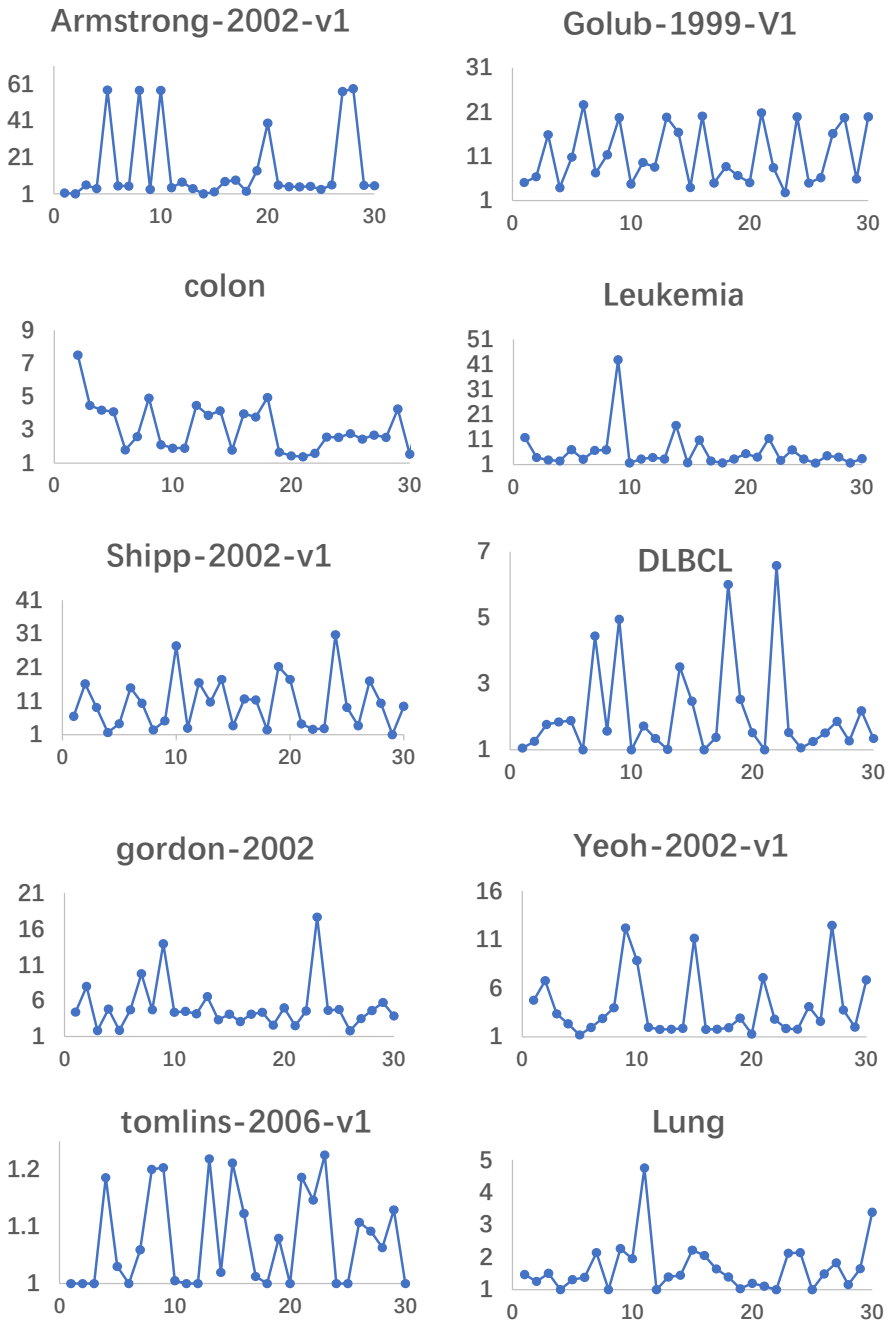


Figure 5.5: Cost values evolved from the 30 runs on the ten datasets.

cost values can be automatically learned when manually-designed cost matrices are not available.

In this chapter, we have presented a new GP method (i.e. CS-GP) to construct classifiers and learn cost values automatically and simultaneously by designing a new tree representation, terminal and function sets. In an individual, the cost value represented by its right subtree is used by the classifier represented by its left subtree in the evaluation to make this algorithm sensitive to different classification mistakes. Therefore, CS-GP does not need manually-designed cost matrices.

By comparing the performance of CS-GP with the baseline GP methods on high-dimensional unbalanced datasets, CS-GP often achieved better AUC results in both slightly and highly unbalanced cases. Therefore, the chapter goal has been successfully achieved. Based on our investigations, cost-sensitive learning could help GP address the performance bias issue and improve its effectiveness for unbalanced classification.

However, this chapter only investigates how cost values can be automatically learned for developing cost-sensitive GP classifiers. Cost intervals are another important type of cost information, which have not been investigated in GP. We will investigate how the cost intervals can be effectively learned and used to construct cost-sensitive classifiers in Chapter 6.

Chapter 6

Interval-based Cost-sensitive GP

6.1 Introduction

In cost-sensitive learning, cost information has three types, i.e. cost values, cost intervals, and cost distribution [102]. To date, most existing cost-sensitive methods are based on cost values, and the use of cost intervals to build cost-sensitive classifiers has been seldom investigated. However, when using a real number or value, a decision result is sought to be determinate and rigid [96]. Compared with cost values, the use of intervals could tolerate possible mistakes in a decision-making process [96].

Similar to cost values, cost intervals are also not always available in real-world applications. Therefore, in this chapter, we aim to use GP to automatically learn *cost intervals* to construct cost-sensitive classifiers.

6.1.1 Chapter Goals

The overall goal of this chapter is to design an interval-based cost-sensitive GP method for binary classification with high-dimensional unbalanced data. The overall goal consists of the following sub-goals:

1) Investigate how GP can be used to automatically learn *cost intervals*,

2) Investigate how the learned cost intervals can be effectively used by a GP classifier, and

3) Investigate whether the proposed method is capable of achieving significantly better or similar classification performance than baseline methods for high-dimensional unbalanced classification.

6.1.2 Chapter Organization

The rest of this chapter is organized as follows. Section 6.2 introduces the proposed GP method. Section 6.3 introduces the experiment design, and the results are analyzed for discussions in Section 6.4 and Section 6.5. The final section summarizes and concludes this chapter.

6.2 The Proposed Method

This section is devoted to introducing the proposed method, called **I**nterval-based **C**ost-**S**ensitive **G**enetic **P**rogramming (**ICS-GP**).

6.2.1 Class-dependent Misclassification Cost Intervals

The class-dependent interval-based cost matrix is given as:

$$CI\_M = \begin{bmatrix} CI_{00} & CI_{01} \\ CI_{10} & CI_{11} \end{bmatrix}$$

where CI_{10} and CI_{01} indicate cost intervals of a false negative and a false positive, respectively. CI_{00} and CI_{11} indicate cost intervals of a true positive and a true negative, respectively.

In CI_M, we assume $CI_{00} = CI_{11} = 0$, which indicates that no misclassification cost is caused by correct classification predictions. The cost of a false positive CI_{01} is set to 1, and the cost of a false negative CI_{10} is set to an interval

(C_l, C_u) ($C_l \geq 1$ and $C_u \geq 1$). Then, an interval-based cost matrix CI_M is simplified to:

$$CI\_M = \begin{bmatrix} 0 & 1 \\ (C_l, C_u) & 0 \end{bmatrix}$$

where C_l is the lower bound of CI_{10}, while C_u is the upper bound of CI_{10}.

Below, we will introduce the use of STGP to automatically construct classifiers and learn cost intervals, i.e. (C_l, C_u) in CI_M.

6.2.2 Classifier Construction and Cost Interval Optimization

ICS-GP is also based on STGP to represent a GP tree, where its left subtree is used to construct a classifier, while its right subtree is used to optimize a cost interval. The designed terminal and function sets for ICS-GP are reported in Table 6.1.

The Left Subtree. In a GP tree, the left subtree is essentially used as a classifier. A terminal set includes all the features and a random constant. There are six functions in the function set, including four basic arithmetic functions (i.e. +, $-$, $\times$ and protected division $\div$), If and $Classifier$. Note that the protected division $\div$ returns zero when dividing by zero. The If function takes three arguments (if the first argument is negative, the second argument is returned; otherwise the third argument is returned). The $Classifier$ function takes one argument $argu$ (the value of $argu$ is the output of an evolved classifier, and its data type is $Iput$), and it directly returns the value of $argu$ (its data type is $Predi$).

Figure 6.1 shows an example of evolved trees. For the tree in Figure 6.1, its left subtree is used as a classifier, which can be translated into an arithmetic expression $Classifier(f528 + f59 \times f102)$. In this arithmetic expression, $f528$, $f59$ and $f102$ are the selected features from the terminal set, and $+$ and $\times$ are taken from the function set of the left subtree. The output values of the arithmetic expression $(f528 + f59 \times f102)$ are used for predictions.

Table 6.1: Function sets and terminal sets in ICS-GP.

	Terminal Sets		**Function Sets**		
	Name	**Type**	**Name**	**Type (Input)**	**Type (Output)**
Left Subtree (for constructing classifiers)	• Features of a dataset • A random constant	$Iput$ (float)	• + • − • × • ÷ • If • $Classifier$	[$Iput$, $Iput$] [$Iput$, $Iput$] [$Iput$, $Iput$] [$Iput$, $Iput$] [$Iput$, $Iput$, $Iput$] [$Iput$]	$Iput$ $Iput$ $Iput$ $Iput$ $Iput$ $Predi$ (float)
Right Subtree (for learning cost intervals)	• The initial cost intervals (C_l, C_u) (C_l and C_u are uniformly distributed random numbers in the range of [1, 2].)	C_inter (list)	• $AddCost$ • $SubCost$ • $MulCost$ • $DivCost$ • T_Cost	[C_inter, C_inter] [C_inter, C_inter] [C_inter, C_inter] [C_inter, C_inter] [C_inter]	C_inter C_inter C_inter C_inter $Ocost$ (list)

1: $Iput$ and $Predi$ are different data types in an evolved tree program, even though they are essentially the same type (float).

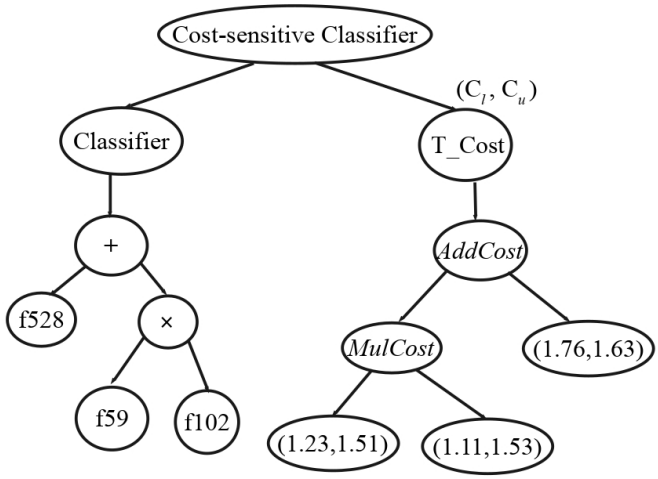


Figure 6.1: An example of the evolved trees.

The Right Subtree. The right subtree plays a role in learning a cost interval, i.e. (C_l, C_u), which will be later used to calculate classification thresholds. The terminal set includes initial cost intervals, where lower and upper bounds are taken from the uniformly distributed random numbers in the range of [1, 2]. The function set has four arithmetic functions (i.e. $AddCost$, $SubCost$, $MulCost$ and $DivCost$), and a function T_Cost. $AddCost$ and $MulCost$ are defined as follows:

$$AddCost(CI1, CI2) = \begin{cases} lower: & C1_l + C2_l \\ upper: & C1_u + C2_u \end{cases} \tag{6.1}$$

$$MulCost(CI1, CI2) = \begin{cases} lower: & C1_l \times C2_l \\ upper: & C1_u \times C2_u \end{cases} \tag{6.2}$$

where $CI1$ and $CI2$ indicate two cost intervals, and $C1_l$ (or $C2_l$) and $C1_u$ (or $C2_u$) are the lower and upper bound values of $CI1$ (or $CI2$), respectively.

$SubCost$ and $DivCost$ need to be designed very carefully because they might produce a cost interval whose lower and upper bound values are less than 1 (i.e. the misclassification cost of the minority class is less than that of the majority class) or even less than 0 (i.e. the misclassification cost of the minority class is

less than that of correct predictions). To avoid this risk, $SubCost$ and $DivCost$ are defined as follows:

$$SubCost(CI1, CI2) = \begin{cases} lower: \ max(1, abs(C1_l - C2_l)) \\ \\ upper: \ max(1, abs(C1_u - C2_u)) \end{cases} \tag{6.3}$$

$$DivCost(CI1, CI2) = \begin{cases} lower: if \ \frac{C1_l}{C2_l} > 1, then \ \frac{C1_l}{C2_l} \ else \ \frac{C2_l}{C1_l} \\ \\ upper: if \ \frac{C1_u}{C2_u} > 1, then \ \frac{C1_u}{C2_u} \ else \ \frac{C2_u}{C1_u} \end{cases} \tag{6.4}$$

The T_Cost function plays a role in transferring a learned cost interval to the root of a tree. T_Cost takes one argument $argu$ (the value of $argu$ is the learned cost interval, and its data type is C_inter), and it directly returns the value of $argu$ (its data type is $Ocost$). In addition, for the learned cost interval CI, it is possible for C_l to be greater than C_u. In that case, in the cost interval CI, its C_l and C_u need to swap their position.

Figure 6.1 also explains how a right subtree works to obtain (C_l, C_u). In Figure 6.1, (1.23, 1.51), (1.11, 1.53) and (1.76, 1.63) are the generated initial cost intervals. For each of them, its C_l and C_u are taken from the uniformly distributed random numbers in the range of [1, 2]. The right subtree is translated into $T\_Cost(AddCost(MulCost((1.23, 1.51), (1.11, 1.53)), (1.76, 1.63))$. Therefore, an evolved cost interval (C_l, C_u) is (4.35, 5.72). Note that, for C_l and C_u in a cost interval, we here keep two decimal places, but they are in fact float values with sixteen decimal places.

3) The Root Node in the Tree Representation

In a tree, for its root node, there is a function named $Cost\text{-}sensitive\ Classifier$. This function accepts the output of $Classifier$ (denoted as $Vpro$ with the data type of $Predi$) and the output of T_Cost (denoted as $Vcost$ with the data type of $Ocost$), and then combines them together as an output of a tree (i.e. return $[Vpro, Vcost]$, a list type).

6.2.3 Classification Decisions in the Training Process

In GP, the output of a program is used to predict the class labels of instances, i.e. belonging to the majority class (Maj) or the minority class (Min). The same as CS-GP in Chapter 5, we use min-max normalization to normalize the original output values of a left subtree into the range of [0, 1], defined as:

$$p_x = 1 - \frac{Pout_x - min(PL)}{max(PL) - min(PL)} \tag{6.5}$$

where $Pout_x$ is the original output value of a program taking instance x as the input, PL is a list of $Pout_x$ for all the training instances, $min(PL)$ and $max(PL)$ are the minimum value and the maximum value in PL, respectively.

In this chapter, we use cost intervals to construct cost-sensitive classifiers, and cost intervals (C_l, C_u) are learned by the right subtree in a tree. In a cost interval, two values are important, i.e. the maximum value C_max and the middle value C_middle (it is equal to $0.5 * (C_l + C_u)$) [102]. C_max indicates the maximum cost caused by a mistake (in the most pessimistic case). C_max could be used to avoid a classification mistake of a false negative at most, while it may overestimate the mistake (i.e. an instance from the minority class is predicted into the majority class). Therefore, it may not make the total cost small enough if an optimal solution minimizes the expected cost by using C_max only. In an interval, C_middle stands for a middle cost between C_min and C_max, which could be used to reflect the entire interval. In the proposed method, the classification performance of a constructed classifier is evaluated by considering C_max and C_middle in a cost interval. Based on C_max and C_middle, two classification thresholds are defined as follows:

$$TH1 = \frac{C\_max}{C\_max + 1} \tag{6.6}$$

$$TH2 = \frac{C\_middle}{C\_middle + 1} \tag{6.7}$$

Eqs. (6.6) and (6.7) are derived from Eq. (5.2) in Chapter 5 (Page 115).

During the training process, the classification performance of a classifier is evaluated at two thresholds, i.e. $TH1$ and $TH2$. When $TH1$ is used as a classification threshold, for instance x, if $p_x \geq TH1$, then x is classified into Maj; otherwise it is classified into Min. Similarly, when using $TH2$ as a classification threshold, for instance x, if $p_x \geq TH2$, then x is classified into Maj; otherwise it is classified into Min. We later explain the process of how instances from a test set are classified, which is slightly different from that in the training process.

6.2.4 Fitness Function

In GP, the evolutionary learning process is guided by a fitness function. Usually, the goodness of every individual in a population needs to be evaluated by the fitness function. In ICS-GP, for each tree, the learned cost interval (represented by its right subtree) is used to calculate $TH1$ and $TH2$. When $TH1$ (or $TH2$) is used as a classification threshold to predict Maj and Min, the predictions on a training set are measured by G_Mean (Eq. (2.6), Page 24).

The fitness function is defined as:

$$Fitness = G\_Mean1 + G\_Mean2 \tag{6.8}$$

where G_Mean1 (or G_Mean2) is the G_Mean value when using $TH1$ (or $TH2$) as a classification threshold.

6.2.5 The Overall Design of ICS-GP

In the training process of ICS-GP, for a tree, the main steps of making the classification predictions are summarized as follows:

Step 1: The output values of the left subtree are normalized into the range of [0,1] by Eq. (6.5).

Step 2: The output of the right subtree is an evolved cost interval (C_l, C_u), based on which C_max and C_middle are obtained.

Step 3: $TH1$ and $TH2$ are calculated by Eqs. (6.6) and (6.7), respectively.

Step 4: The constructed classifier (represented by the left subtree) separately uses

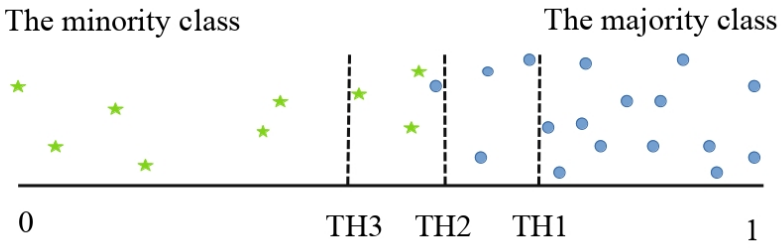


Figure 6.2: The process of classification predictions on a test set.

$TH1$ and $TH2$ as classification thresholds to predict Maj and Min. After that, G_Mean1 and G_Mean2 are calculated and summed together as the fitness value (according to Eq. (6.8)).

After the evaluation process, the fitness values are used by tournament selection to select good individuals. The genetic operators, e.g. mutation, crossover and elitism, are used to generate offspring for the new population. The evolutionary learning process is stopped when a termination criterion is satisfied. Finally, the best individual from the final generation is selected to make classification predictions on a test set.

The Test Process

During the test process, a new classification threshold $TH3$ is defined as:

$$TH3 = \frac{C\_min}{C\_min + 1} \tag{6.9}$$

where C_min stands for the minimum cost value in a cost interval. $TH3$ is defined by Eq.(6.9), which is also derived from Eq. (5.2) (Page 115) in Chapter 5. $TH3$ is used with $TH1$ and $TH2$ to predict class labels of unseen instances. Note that $TH3 \leq TH2 \leq TH1$ (because $0 < C\_min \leq C\_middle \leq C\_max$).

Figure 6.2 explains the classification process on a test set. During the test process, the main steps of making classification predictions are summarized as follows:

Step 1: The output values of the left subtree are normalized into the range of [0,1] by Eq. (6.5).

Step 2: According to the output of the right subtree, $TH1$, $TH2$ and $TH3$ are calculated.

Step 3: At the beginning, P_{Maj} and P_{Min} are two empty sets and will save program outputs (p_x) when x is directly classified into Maj or Min.

Step 4: For instance x in the test set,

- If $p_x \geq TH1$, x is directly classified to Maj and p_x is appended into P_{Maj};
- If $p_x \leq TH3$, x is directly classified to Min and p_x is appended into P_{Min};
- If $TH3 < p_x < TH1$, x is temporarily saved in a set called Tem.

Step 5: It calculates the maximum value in P_{Min} and the minimum value in P_{Maj}, denoted as $max(P_{Min})$ and $min(P_{Maj})$, respectively.

Step 6: For instance x in Tem,

- If $p_x \geq TH2$ and p_x is nearer to $min(P_{Maj})$ than $max(P_{Min})$, then x is classified into Maj; otherwise it is classified into Min.

6.3 Experiment Design

6.3.1 Datasets

In the experiments, ten gene expression datasets were used to examine the performance of ICS-GP. The used datasets are the same as those used in the previous contribution chapters. The details of these datasets are reported in Table 6.2. To ensure the same class imbalance ratio in both training and test sets, stratified sampling is employed to split a dataset into the training set (70%) and the test set (30%).

Table 6.2: Dataset description.

Dataset	#Features	#Instances	Proportion % (Majority Class)	Proportion % (Minority Class)	IR (Rounding)
Armstrong-2002-v1	1081	72	66.67	33.33	2
Golub-1999-v1	1868	72	65.28	34.72	2
Colon	2000	62	64.52	35.48	2
Leukemia	7129	72	65.28	34.72	2
Shipp-2002-v1	798	77	75.32	24.68	3
DLBCL	5469	77	75.32	24.68	3
Gordon-2002	1626	181	82.87	17.13	5
Yeoh-2002-v1	2526	248	82.66	17.34	5
Tomlins-2006-v1	2315	104	88.46	11.54	8
Lung	12600	156	89.10	10.90	8

1: The proportions of the majority class and the minority class are rounded to two decimal places.

6.3.2 Baseline Methods

The baseline methods in Chapter 5 are also used in this chapter (see Page 122), and ICS-GP is also compared with CS-GP designed in Chapter 5.

6.3.3 Parameter Settings

For GP methods, the population size is 1024 and the number of generations is 50. Population sizes of 512 and 1024 are the common settings for GP methods [82]. Because of the complexity of classification problems to be solved (i.e. with a large number of features), we chose 1024 as the population size to search for optimal solutions. The initial population is generated by ramped half-and-half, which is the most commonly used initialization method in GP [143]. For each generation, good individuals are selected by tournament selection (the tournament size is set to 6). Tournament selection amplifies small differences of individuals in their fitness

values to select better individuals [143]. To generate a new population (with 1024 individuals), 1023 individuals are generated by standard subtree crossover and subtree mutation [143], and 1 individual is directly inherited from the previous generation (i.e. elitism). The probabilities of crossover and mutation are set to 0.8 and 0.2, respectively, which are also very common settings for GP methods [188]. The maximum tree depth is limited to 10. After 50 generations, the evolutionary learning process of a GP method is terminated.

6.4 Results and Discussions

AUC is the most widely used metric in unbalanced classification because it is invariant to uneven data distributions [11]. The AUC results of the proposed method (i.e. ICS-GP) and the baseline GP methods are reported in Table 6.3, where bold values are the highest AUC result achieved by these methods on a dataset. Furthermore, the Wilcoxon rank-sum tests were conducted to compare ICS-GP with a baseline GP method, with the significance level of 0.05. In Table 6.3, "+", "=" and "−" are used to show that ICS-GP is significantly better than, similar to, and significantly worse than a compared method, respectively.

Table 6.3: ICS-GP versus the baseline GP methods on the test sets.

Method	AUC (×100)			Training Time (Seconds)	AUC (×100)			Training Time (Seconds)
	Best	Mean±Std		Mean	Best	Mean±Std		Mean
	Armstrong-2002-v1				Golub-1999-v1			
GP_{SMOTE}	100	91.3 ±9.83	+	144.32	100	92.38 ± 10.31	+	195.79
$GP_{BSMOTE1}$	100	94.16 ± 7.43	+	141.78	100	89.11 ± 11.08	+	187.11
$GP_{BSMOTE2}$	100	91.22 ± 10.27	+	153.6	100	85.8± 14.56	+	205.14
GP_{ADASYN}	100	92.21 ± 9.62	+	143.04	100	91.8 ±9.72	+	198.39
GP_{Ave}	100	94.48 ± 8.4	+	114.86	100	91.93 ± 10.09	+	158.77
GP_{G_Mean}	100	92.13 ± 8.01	+	114.88	100	88.99 ± 11.89	+	158.31

Continued on next page

Table 6.3 – Continued from previous page

GP_{Amse}	100	90.17 ± 7.65	+	146.24	100	82.78 ± 11.62	+	226.35
GP_{Corr}	100	94.67 ± 7.56	+	142.55	100	96.06 ± 6.32	+	225.06
GP_{Dist}	100	95.84 ± 3.93	+	141.33	100	96.9 ± 5.23	+	229.09
GP_{Auc_w}	100	94.46 ± 4.93	+	1917.99	100	98.42 ± 3.38	=	3089.78
GP_{RC_w}	100	95.21 ± 5.59	+	126.03	100	87.92±13.03	+	185.36
GPCF	100	95.24±6.03	+	233.37	100	97.26±5.81	=	397.15
GPBC	100	97.60±5.32	=	130.18	100	97.11 ± 5.16	=	175.25
CS-GP	100	**98.79± 3.03**	=	139.12	100	98.95± 2.83	=	187.61
ICS-GP	100	98.22 ±3.1		134.85	100	**99.11 ± 2.9**		185.61
	Colon				Leukemia			
GP_{SMOTE}	92.86	75.99 ± 10.55	=	222.64	100	87.56 ± 10.01	=	919.55
$GP_{BSMOTE1}$	88.1	71.51 ± 12.64	+	206.97	98.21	85.6 ± 13.38	=	948.52
$GP_{BSMOTE2}$	94.05	73.69 ± 15.28	+	228.62	99.11	82.47± 12.94	+	1065.41
GP_{ADASYN}	88.1	74.6 ± 10.25	+	227.64	100	89.73± 8.56	=	951.82
GP_{Ave}	91.67	75.52 ± 10.11	+	177.08	98.21	88.79 ± 7.74	=	975.7
GP_{G_Mean}	92.86	71.51 ± 12.95	+	174.21	100	81.79 ± 15.38	+	979.29
GP_{Amse}	95.24	74.8 ± 10.76	+	203.33	100	81.73 ± 11.84	+	793.34
GP_{Corr}	96.43	75.28 ± 10.1	=	201.08	100	86.16 ± 10.84	=	785.98
GP_{Dist}	92.86	76.59 ± 9.63	=	203.64	97.32	86.32 ± 8.95	=	788.22
GP_{Auc_w}	91.67	78.97 ± 7.3	=	2348.72	100	86.28 ± 9.68	=	10396.7
GP_{RC_w}	88.1	73.13±10.35	+	202.79	99.11	81.1± 14.52	+	871.05
GPCF	92.86	76.43 ± 8.84	=	347.44	100	**94.42±4.74**	−	1317.99
GPBC	94.05	75.44 ± 12.03	=	165.96	96.43	84.88±6.91	=	969.78
CS-GP	90.48	**79.05 ± 7.03**	=	185.5	96.43	85.57±9.03	=	1006.54
ICS-GP	92.86	78.69 ±6.93		196.21	100	87.56 ±9.51		861.85
	Shipp-2002-v1				DLBCL			
GP_{SMOTE}	98.15	82.15 ± 11.77	=	132.95	98.15	83.21 ±9.56	=	816.12
$GP_{BSMOTE1}$	95.37	77.93 ± 12.35	+	126.82	99.07	75.77 ± 18.56	+	795.67
$GP_{BSMOTE2}$	100	79.46 ± 14.87	+	146.3	98.15	79.41 ± 11.68	=	846.71
GP_{ADASYN}	96.3	79.88 ±12.18	+	137.39	100	79.48± 10.92	=	831.63

Continued on next page

Table 6.3 – Continued from previous page

GP_{Ave}	99.07	82.85 ±9.81	=	95.15	98.15	75.4 ± 15.67	+	740.03
GP_{G_Mean}	96.3	83.09 ± 9.21	=	97.85	100	77.01 ± 15.75	+	731.45
GP_{Amse}	96.3	75.26 ±13.2	+	225.24	100	77.19 ± 13.18	+	638.3
GP_{Corr}	99.07	83.02 ±13.33	=	216.72	98.15	81.02± 11.42	=	643.18
GP_{Dist}	99.07	**84.81 ±8.96**	=	214.44	99.07	84.35 ± 9.96	=	633.55
GP_{Auc_w}	100	82.62 ±9.45	=	3192.03	100	85.54 ± 10.83	−	7845.03
GP_{RC_w}	97.22	82.56 ± 11.14	=	101.5	97.22	79.32±12.99	=	651.66
GPCF	98.15	83.14 ± 9.26	=	200.17	100	**87.59±9.97**	−	1122.61
GPBC	95.37	77.42 ± 11.93	+	85.65	98.15	86.11±7.48	−	796.42
CS-GP	95.37	83.3 ±8.34	=	100.23	94.44	80.54± 9.06	=	753.79
ICS-GP	100	84.46±8.91		135.99	100	81.56±11.95		761.97
	Gordon-2002				Yeoh-2002-v1			
GP_{SMOTE}	100	97.49 ± 2.92	=	630.05	100	87.44 ± 9.24	+	1321.18
$GP_{BSMOTE1}$	100	98.71 ± 2.67	=	584.44	99.75	86.23 ± 10.64	+	1290.39
$GP_{BSMOTE2}$	100	96.35 ± 4.92	=	669.96	98.39	83.27 ± 10.15	+	1388.96
GP_{ADASYN}	100	97.81 ± 2.9	=	598.04	100	84.28± 10.24	+	1452.27
GP_{Ave}	100	98.26±2.81	=	321.69	100	83.97 ± 11.91	+	773.26
GP_{G_Mean}	100	98.38 ±3.01	=	323.48	95.78	66.33 ± 16.09	+	767.4
GP_{Amse}	100	96.49 ±4.46	=	734.51	92.06	63.79 ± 12.06	+	717.74
GP_{Corr}	100	96.95 ±6.41	=	718.51	100	93.29 ± 7.77	+	685.35
GP_{Dist}	100	98.9 ±3.62	=	720.18	100	91.1 ± 8.22	+	694.73
GP_{Auc_w}	100	**99.23** ± 2.06	=	21978.57	100	**98.95** ± **2.32**	=	24174.23
GP_{RC_w}	100	96.86 ± 4.04	=	379.01	100	89.14 ± 10.06	+	788.65
GPCF	100	97.32 ± 4.56	=	697.2	100	92.84±4.84	+	1364.63
GPBC	100	95.43 ± 3.71	=	355.91	100	98.64±3.62	=	844.54
CS-GP	100	96.71±4.2	=	406.25	100	97.31± 4.29	=	811.81
ICS-GP	100	97.19± 2.39		441.93	100	97.87±3.86		894.9
	Tomlins-2006-v1				Lung			
GP_{SMOTE}	100	83.27± 13.43	+	499.69	100	81.7 ± 15.9	+	4298.87
$GP_{BSMOTE1}$	100	87.37 ± 10.14	+	521.56	100	88.89 ±10.76	+	3631.29

Continued on next page

Table 6.3 – Continued from previous page

$GP_{BSMOTE2}$	100	90.18 ± 10.78	+	500.18	100	84.16± 15.78	+	3901.94
GP_{ADASYN}	100	84.67± 13.16	+	478.33	100	82.97 ± 14.98	+	4137.5
GP_{Ave}	100	88.87±13.47	+	293.08	100	83.46 ± 14.73	+	3048.62
GP_{G_Mean}	100	84.54 ±14.55	+	296.92	99.05	80.89 ± 18.41	+	3038.89
GP_{Amse}	100	92.96 ± 8.04	+	655.62	100	81.78 ± 16.55	+	2503.15
GP_{Corr}	100	90.42 ± 14.1	+	648.78	100	80.71 ±17.21	+	2490.26
GP_{Dist}	100	95.83 ± 6.73	=	646.03	100	84.27 ± 14.8	+	2493.37
GP_{Auc_w}	100	91.10 ±9.75	+	7865.15	100	92.35 ± 13.23	+	45375.33
GP_{RC_w}	100	86.07 ± 13.11	+	306.15	100	80.08 ± 15.46	+	2629.14
GPCF	100	83.89 ± 14.16	+	622.98	100	90.23±9.48	+	4763.61
GPBC	100	85.83 ±12.33	+	440.11	100	89.83±11.64	+	3223.64
CS-GP	100	96.98 ±4.3	=	328.66	100	97.13± 2.57	=	3241.32
ICS-GP	100	**97.43±2.46**		368.59	100	**98.46± 3.61**		2581.3
Total	**78 +, 58 =, 4 −**							

6.4.1 ICS-GP Versus the GP Baseline Methods

In general, ICS-GP achieves significantly better or similar performance than the baseline GP methods in 136 out of the 140 cases (significantly better performance in 78 cases and similar performance in 58 cases, respectively). By comparing the best AUC results of the 30 runs, ICS-GP achieves better or the same performance than other GP methods in 136 out of the 140 cases.

GP_{G_Mean} uses G_Mean as the fitness function and standard classification strategy. By comparing with GP_{G_Mean}, ICS-GP achieves significantly better performance in 8 out of the 10 cases (similar performance in the other 2 cases). In fact, for ICS-GP, it does not require a pre-defined classification threshold prior to a classification process. Two classification thresholds are calculated by an evolved cost interval, which are essentially flexible, to predict the majority class and the minority class.

GP_{Ave} also adopts the standard classification strategy, using Ave ($W = 0.5$)

as the fitness function. GP$_{Ave}$ treats the majority class and minority class as being equally important. By comparing with GP$_{Ave}$, ICS-GP achieves significantly better performance in 7 out of the 10 cases (similar performance in the other 3 cases). In fact, in many real-world applications, the majority class and the minority class are not equally important.

Unlike the accuracy-based fitness functions, such as Ave and G_Mean, using AUC as the fitness function does not require a fixed classification threshold in the fitness evaluation process. This is because, for AUC, building the receiver operating characteristic (ROC) curve requires the re-evaluation of TP rate and FP rate many times by varying thresholds to provide an accurate rendition of the curve. However, GP using AUC as a fitness function is much more time-consuming than that of using accuracy measures as a fitness function. Based on Table 6.3, GP$_{Auc_w}$ often achieves promising classification performance, while its training time is also much longer than other GP methods. Compared with GP$_{Auc_w}$, ICS-GP achieves significantly better or similar performance in 9 out of the 10 cases (significantly better performance in 3 cases and similar performance in 6 cases, respectively). On the two datasets with $IR = 8$, ICS-GP achieves significantly better performance than GP$_{Auc_w}$. In addition, the training time of ICS-GP is significantly shorter than GP$_{Auc_w}$ on all datasets. As acknowledged, GP$_{Auc_w}$ is time-consuming due to $|Maj| \times |Min|$ times pairwise comparisons in an evaluation. ICS-GP is faster mainly because ICS-GP requires less computational costs than GP$_{Auc_w}$ in evaluations.

Compared with cost-sensitive GP methods (i.e. GP$_{RC_w}$, GPCF, GPBC and CS-GP), ICS-GP performs better in 33 out of the 40 cases. GP$_{RC_w}$, GPCF and GPBC use the class imbalance ratio IR of a dataset to construct a cost matrix. In most cases, the three cost-sensitive methods could achieve comparable AUC results. However, it is noted that ICS-GP achieves significantly better performance on datasets with $IR = 8$, i.e. tomlins-2006-v1 and Lung, than the three cost-sensitive methods. This reveals that the use of IR as the cost information may not always be suitable. Compared to CS-GP specifically, ICS-GP performs slightly better than CS-GP on eight out of the ten datasets.

ICS-GP shows a good classification performance on both slightly-unbalanced datasets (i.e. $2 \leq IR < 5$) and highly-unbalanced datasets (i.e. $IR \geq 5$). For the slightly-unbalanced datasets, such as Armstrong-2002-v1 and Golub-1999-v1, ICS-GP often outperforms the baseline GP methods. In total, on these slightly-unbalanced datasets, ICS-GP achieves significantly better or similar performance in 80 out of the 84 cases. On the highly-unbalanced datasets, ICS-GP achieves significantly better or similar performance in all the 56 cases.

There are two main reasons for explaining the improved classification performance. The first reason is that cost-sensitive learning could help GP address its performance bias issue in unbalanced classification. As can be seen from Table 6.3, the cost-sensitive GP methods (i.e. GP_{RC_w}, GPCF, GPBC, CS-GP and ICS-GP) often achieve comparable AUC results to other GP methods that employ fitness functions or sampling methods to solve the issue of class imbalance. The second reason is that ICS-GP is independent of a manually-designed cost matrix that is fixed during the whole evolutionary learning process. In ICS-GP, the needed cost interval is co-learned with a classifier, so the evolved classifier could select a suitable cost interval for use to improve its performance.

Unfortunately, in a few cases, the classification performance has not improved as expected. The main reason is a possible risk of discarding some offspring which have been evolved as good classifiers by the left subtree, but have an associated terrible cost interval evolved by the right subtree. However, this issue does not matter that much in most cases because GP is a population-based method and in the experiments, the population size is 1024. With a large population size, losing some offspring (i.e. a good classifier with a terrible cost interval) might not significantly influence the performance. If we attempt to protect these kinds of offspring from being discarded, additional computational costs will be required. This is because, in the fitness evaluation process, the classifier evolved by the left subtree would need to be evaluated twice, i.e. with and without the evolved cost information.

In addition, in the proposed ICS-GP method, the search space of possible solutions is larger than that of other baseline GP methods. This is because ICS-GP

needs to construct classifiers as well as to optimize cost intervals, while the baseline GP methods only need to construct classifiers (except for CS-GP). However, to ensure relatively fair comparisons in the experiments, ICS-GP has the same parameter settings as the baseline GP methods (i.e. the population size and the number of generations of ICS-GP are the same as that of the baseline GP methods, to ensure each GP method to have the same number of evaluations).

6.4.2 ICS-GP Versus the Non-GP Baseline Methods

The results of the non-GP baseline methods on the test sets are reported in Tables 6.4, 6.5, 6.6 and 6.7.

As can be seen from Table 6.4, by comparing the mean AUC result of ICS-GP with that of the traditional classification methods using SMOTE on each dataset, ICS-GP achieves better performance in 55 out of the 60 comparisons. As can be seen from Table 6.5, to compare the mean AUC result of ICS-GP with that of the traditional classification methods using Borderline-SMOTE1 on each dataset, ICS-GP achieves better performance in 53 out of the 60 comparisons. Based on Table 6.6, by comparing the mean AUC result of ICS-GP with that of the traditional classification methods using Borderline-SMOTE2 on each dataset, ICS-GP outperforms in 44 out of the 60 comparisons. As can be seen from Table 6.7, by comparing the mean AUC result of ICS-GP with that of the traditional classification methods using ADASYN on each dataset, ICS-GP outperforms in 55 out of the 60 comparisons.

6.5 Analysis on Evolved GP Trees

In this section, we show two trees evolved by the proposed ICS-GP method on Lung (a highly-unbalanced dataset, $IR = 8$) and Armstrong-2002-v1 (a slightly-unbalanced dataset, $IR = 2$), respectively, for further analysis. The evolved tree on Lung is shown in Figure 6.3, and the evolved tree on Armstrong-2002-v1 is shown in Figure 6.4.

Table 6.4: ICS-GP versus the non-GP classification methods using SMOTE (AUC×100).

Dataset	SMOTE-1NN	SMOTE-DT	SMOTE-RF	SMOTE-GBDT	SMOTE-NB	SMOTE-MLP	**ICS-GP**	
							Best	Mean
Armstrong-2002-v1	96.67	88.46	90.97	89.52	85.71	92.85	100	**98.22**
Golub-1999-v1	93.75	89.91	89.80	92.86	68.75	96.43	100	**99.11**
Colon	74.40	64.04	64.46	65.83	47.62	62.60	89.29	**78.69**
Leukemia	90.18	86.61	81.52	86.61	**100**	69.61	98.21	87.56
Shipp-2002-v1	72.22	68.24	72.22	69.44	58.33	76.67	96.3	**84.46**
DLBCL	69.44	67.31	77.13	72.22	80.86	74.63	100	**81.56**
Gordon-2002	**98.91**	86.09	92.94	90.93	88.89	85.74	100	97.19
Yeoh-2002-v1	86.29	96.03	72.66	96.15	80.58	84.10	100	**97.87**
Tomlins-2006-v1	**98.21**	80.36	88.51	83.75	75	75.95	100	97.43
Lung	85.24	95.21	82.23	**100**	80.00	82.52	100	98.46

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 6.5: ICS-GP versus the non-GP classification methods using Borderline-SMOTE1 (AUC$\times$100).

Dataset	B-SMOTE1-1NN	B-SMOTE1-DT	B-SMOTE1-RF	B-SMOTE1-GBDT	B-SMOTE1-NB	B-SMOTE1-MLP	**ICS-GP**	
							Best	Mean
Armstrong-2002-v1	96.67	89.16	85.29	89.52	85.71	92.86	100	**98.22**
Golub-1999-v1	93.75	90.15	83.96	92.86	68.75	93.75	100	**99.11**
Colon	74.40	63.12	65.04	62.26	47.62	74.40	89.29	**78.69**
Leukemia	**96.43**	86.61	76.16	86.61	93.75	75	98.21	87.56
Shipp-2002-v1	75.00	68.42	68.15	69.44	66.67	75	96.3	**84.46**
DLBCL	80.56	70.37	70.93	73.43	80.56	69.44	100	**81.56**
Gordon-2002	97.83	93.94	91.96	93.36	88.89	**100**	100	97.19
Yeoh-2002-v1	96.15	96.15	66.90	96.15	82.20	87.28	100	**97.87**
Tomlins-2006-v1	**100**	84.29	80.36	81.67	62.50	**100**	100	97.43
Lung	95.24	95.75	78.55	**99.92**	70.00	90.00	100	98.46

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 6.6: ICS-GP versus the non-GP classification methods using Borderline-SMOTE2 (AUC×100).

Dataset	B-SMOTE2-1NN	B-SMOTE2-DT	B-SMOTE2-RF	B-SMOTE2-GBDT	B-SMOTE2-NB	B-SMOTE2-MLP	**ICS-GP**	
							Best	Mean
Armstrong-2002-v1	93.33	89.79	88.86	89.52	**100**	92.86	100	98.22
Golub-1999-v1	93.75	90.03	79.11	92.86	77.68	**100**	100	99.11
Colon	70.24	76.69	62.74	77.38	43.45	60.12	89.29	**78.69**
Leukemia	96.43	86.61	78.01	86.61	**100**	81.25	98.21	87.56
Shipp-2002-v1	77.78	73.61	72.50	72.22	75	**91.67**	96.3	84.46
DLBCL	75	76.85	75.37	83.33	**88.89**	69.44	100	81.56
Gordon-2002	97.83	93.68	89.41	93.36	**100**	**100**	100	97.19
Yeoh-2002-v1	**100**	99.49	68.23	**100**	90.69	87.28	100	97.87
Tomlins-2006-v1	89.29	87.50	92.38	87.50	62.50	**98.21**	100	97.43
Lung	91.67	99	80.31	**100**	80.00	87.62	100	98.46

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 6.7: ICS-GP versus the non-GP classification methods using ADASYN (AUC$\times$100).

Dataset	ADASYN-1NN	ADASYN-DT	ADASYN-RF	ADASYN-GBDT	ADASYN-NB	ADASYN-MLP	ICS-GP	
							Best	Mean
Armstrong-2002-v1	93.33	89.59	91.97	89.52	92.86	92.85	100	**98.22**
Golub-1999-v1	93.75	90.51	89.35	92.86	68.75	92.86	100	**99.11**
Colon	74.40	66.69	66.19	62.20	47.62	67.26	89.29	**78.69**
Leukemia	83.04	86.61	80.47	86.61	**100**	65.15	98.21	87.56
Shipp-2002-v1	83.33	80.46	75.00	83.33	58.33	80.92	96.3	**84.46**
DLBCL	77.78	67.31	74.35	72.22	**88.89**	79.44	100	81.56
Gordon-2002	**98.91**	83.24	95.01	83.33	88.89	83.15	100	97.19
Yeoh-2002-v1	91.13	95.64	72.66	96.15	86.04	82.07	100	**97.87**
Tomlins-2006-v1	**98.21**	80.36	86.90	82.5	62.5	85.53	100	97.43
Lung	76.90	97.83	79.74	**99.67**	80.00	74.60	100	98.46

1: Bold values are the highest AUC result achieved by methods on a dataset.

Lung has 156 instances, described by 12600 features. In Figure 6.3, the evolved tree has 16 nodes in total, where 6 features are selected from 12600 features to construct a classifier by its left subtree. The cost interval represented by its right subtree is (1.25, 1.50). Based on the evolved cost interval, two classification thresholds are calculated and used by the left subtree (works as a classifier). On the test set, the AUC score of this tree is 1, and the accuracies of the majority class and the minority class are 100%. Note that, for the lower and upper bound values in a cost interval, we here keep two decimal places, but they are in fact float values with sixteen decimal places.

Armstrong-2002-v1 has 72 instances and 1081 features, and its class imbalance ratio IR is 2. Shown in Figure 6.4, the evolved tree has 11 nodes in total. For its left subtree, only one feature (i.e. f245) is selected from 1081 features. The evolved cost interval, i.e. (5.52, 5.54), is represented by its right subtree. The AUC score of this tree on the test set is 1, and the accuracies of the majority class and the minority class are 100%.

By comparing the two trees, the evolved tree on Lung is a bit more complicated than that of Armstrong-2002-v1. The reason is due mainly to more features included in Lung. It is interesting that, for the evolved cost interval on Lung, its upper bound value is less than the imbalance ratio IR of Lung (i.e. 8). However, for the evolved cost interval on Armstrong-2002-v1, i.e. (5.52, 5.54), its lower bound value is greater than IR of Armstrong-2002-v1 (i.e. 2). Therefore, it is hard to conclude whether there is a direct or indirect relationship between IR and the evolved cost information.

6.6 Chapter Summary

The overall goal of this chapter was to investigate how cost intervals can be automatically learned and used by GP to effectively construct interval-based cost-sensitive classifiers for high-dimensional unbalanced classification. To achieve this goal, in the proposed ICS-GP method, new terminal and function sets were designed to make a GP individual with the capability to construct a classifier and

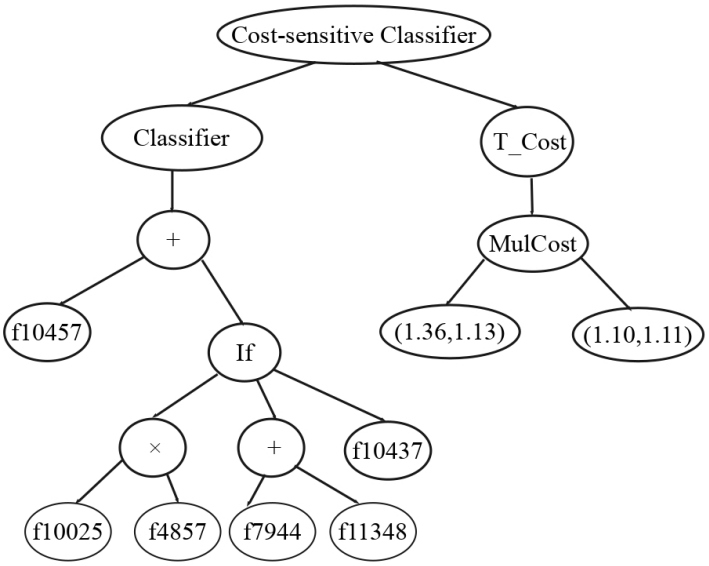


Figure 6.3: An evolved tree by ICS-GP on Lung.

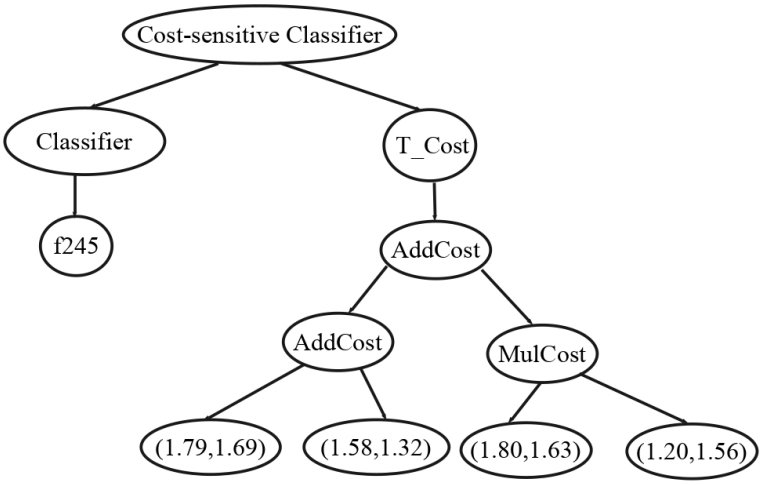


Figure 6.4: An evolved tree by ICS-GP on Armstrong-2002-v1.

learn a cost interval simultaneously. For a tree in a population, the cost interval represented by its right subtree will be later used by the classifier represented by its left subtree in the fitness evaluation process to make it sensitive to different classification mistakes.

In the experiments, we tested the effectiveness of ICS-GP on high-dimensional unbalanced datasets. By comparing ICS-GP with the baseline GP methods, ICS-GP achieved significantly better or similar performance in almost all cases. In addition, by comparing ICS-GP with the non-GP baseline methods, ICS-GP also achieved better performance in most cases. Hence, we have achieved the chapter goal successfully.

In this chapter and the previous contribution chapters, the class overlap issue is not typically considered to address. However, when class overlap is intertwined with the issue of class imbalance, it is more challenging to discover useful patterns due to an ambiguous boundary between the majority class and the minority class. Cost-sensitive GP methods treat the minority class as being more important than the majority class, but this may cause an accuracy decrease in overlapping areas. In the next chapter, we will investigate how overlapping areas can be detected, and then cost-sensitive classifiers are developed by GP.

Chapter 7

GP with Detection of Overlapping Areas Using Rough Set

7.1 Introduction

In classification, one class overlaps with another when instances of different classes have similar characteristics [105, 157]. This is known as *class overlap* and acknowledged as one of the hard problems in classification because of ambiguous boundaries between different classes. For every instance in a dataset, its neighbors are expected to be also from the same class, but in an overlapping area, some of its neighbors are quite likely from a different class. In an overlapping area, the prior probabilities of two classes are often approximately equal [57, 105].

The class overlap issue often makes it more difficult to address the issue of class imbalance [63]. In the worst case, the minority class is completely overlapped with the majority class, such as an example visualized in Figure 7.1. Moreover, in a complicated binary classification task with unbalanced data, a class may have several sub-clusters, each of which may overlap with the other class. This may produce multiple overlapping areas in a binary classification task. Figure 7.2 shows two situations of class overlaps in a two-dimensional data space. In Figure 7.2 (a), there is a single overlapping area (marked by a red circle), which is a relatively easy case. However, in 7.2 (b), there are three overlapping areas. This is

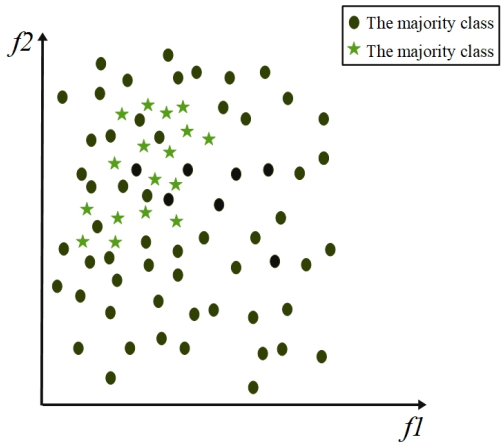


Figure 7.1: Two classes are fully overlapped with each other.

because the minority class has three sub-clusters, each of which overlaps with the majority class. Note that, instances in a sub-cluster of the minority class can be closer to the majority class.

Although standard classification algorithms attempt to correctly separate different classes, they usually learn patterns from the whole training set. If a task has a serious class overlap issue, these algorithms do not specifically detect overlapping instances, and treat overlapping instances and non-overlapping instances equally. In fact, the dissimilarity between two overlapping instances of different classes is usually small, so it is often difficult for classifiers to discover the hidden patterns in the overlapping instances in order to correctly classify them. Accordingly, it is important to distinguish overlapping instances from a training set and treat them differently from non-overlapping instances [90]. In addition, instances in overlapping areas could also be called *borderline instances*, since they are usually close to a decision boundary [175].

Most existing cost-sensitive studies do not specifically consider the influence of overlapping areas. Usually, in a cost matrix, a misclassification cost value for the minority class is greater than or equal to that of the majority class, to avoid mistakenly predicting instances from the minority class into the majority class. Nevertheless, this may cause an accuracy decrease in the overlapping areas, where

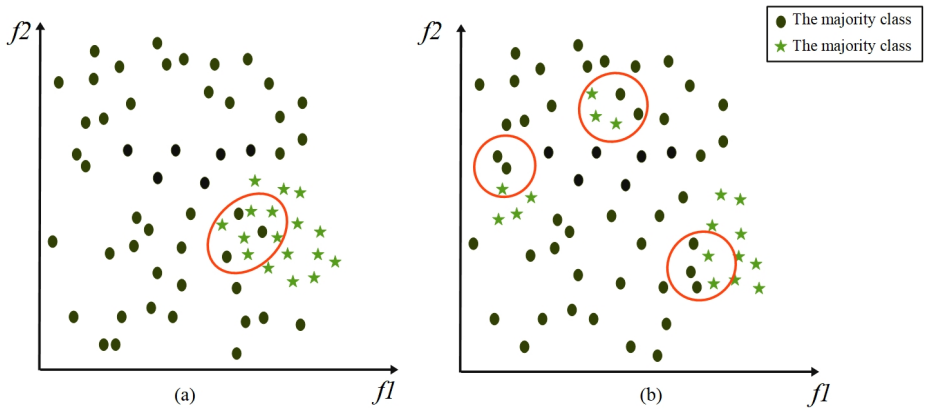


Figure 7.2: Two situations of class overlaps in binary classification with unbalanced data.

the prior probabilities of two class are about the same. Therefore, it is necessary to detect the overlapping areas and use different strategies for classification in the non-overlapping and overlapping areas.

7.1.1 Chapter Goals

In this chapter, we aim to tackle the issue of class overlap in order to enhance the classification performance of GP in high-dimensional unbalanced classification. To achieve this goal, the sub-goals are introduced as follows:

1) Design and develop a method to detect overlapping areas,

2) Develop classification strategies for GP classifiers in order to classify instances from non-overlapping and overlapping areas,

3) Investigate whether the proposed method can perform better than baseline methods, and

4) Investigate the efficiency of the proposed method.

7.1.2 Chapter Organization

The rest of this chapter is organized as follows. Sections 7.2 introduces the proposed GP method. Section 7.3 presents the experiment design. The experimental results are discussed in Section 7.4 and Section 7.5. In Section 7.6, we summarize this chapter.

7.2 The Proposed Method

In this section, we introduce a new method, called **C**ost-**S**ensitive **G**enetic **P**rogramming with **N**eighborhood-based **O**verlapping **D**etection (CSGPNOD).

7.2.1 Detection of Overlapping Areas in CSGPNOD

In CSGPNOD, overlapping areas are detected by using the neighborhood rough set (it was introduced in Section 2.6.1, on Page 42). Note that the standard neighborhood operator is reflexive, symmetric, and transitive, based on the equivalence relation [207]. However, for high-dimensional data, the equivalence relation is very strong and finds it very hard to analyze numerical high-dimensional data. The main reason is that the granules (equivalence classes) become very small in size when too many features are associated with instances. Therefore, the granules are very likely to be a subset of a set X, i.e. belonging to the lower approximation of X, based on **Definition 2.1** (introduced on Page 42). When all of the granules belong to X, no instance is detected into a boundary. However, this result is most likely due to high dimensionality of the data. To avoid this, in the proposed method, we do not use the standard neighborhood operator, but use the *ball tree* algorithm that is a nearest neighbor search method to find out the neighborhood of each instance [8, 99, 131]. Another reason for choosing the ball tree algorithm is because of its high efficiency [99].

For each training instance x, the ball tree algorithm is used to find out the k nearest neighbors, denoted as $\sigma(x)$. After that, we use the neighborhood rough set to detect overlapping areas. The main idea is straightforward, i.e. for an instance

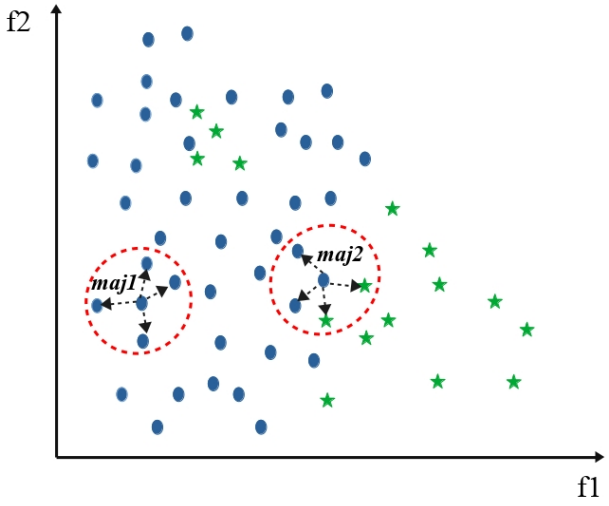


Figure 7.3: Neighborhood-based overlapping areas detection.

x, if its $\sigma(x)$ includes instances from two classes (i.e. the majority class Maj and the minority class Min) and is not a subset of any class (i.e. not in the lower approximation set of Maj or Min), then this instance is located in the overlapping areas. The idea is explained by Figure 7.3 (the number of neighbors $k = 4$ in this example). In Figure 7.3, for instance $maj1$, its 4 nearest neighbors are all from Maj, so this instance is located in the lower approximation of Maj. However, for instance $maj2$, 2 of the 4 neighbors are from Maj and the other 2 neighbors are from Min. The instance $maj2$ is very likely to be from overlapping areas since its neighbors are from both Maj and Min.

For a training set U, $Maj \subseteq U$, based on **Definition 2.2** (it was introduced on Page 43), we obtain:

$$N_{-}(Maj) = \{x \in U | \sigma(x) \subseteq Maj\} \tag{7.1}$$

$$N^{-}(Maj) = \{x \in U | \sigma(x) \cap Maj \neq \varnothing\} \tag{7.2}$$

$$Bn_n(Maj) = N^{-}(Maj) - N_{-}(Maj) \tag{7.3}$$

Therefore, the overlapping areas in the training set is:

$$Overlap = Bn_n(Maj) \tag{7.4}$$

Note that $Maj \cup Min = U$ and $Maj \cap Min = \varnothing$, so if $x \in Bn_n(Maj)$, then $x \in Bn_n(Min)$ (the proof is in Appendix). It means that we can either use Maj or Min to calculate $Overlap$.

An example: Given $U = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\}$, $Min = \{x_2, x_3, x_{10}\}$, and $Maj = \{x_1, x_4, x_5, x_6, x_7, x_8, x_9\}$, for instance x_1, its $\sigma(x1) = \{x_2, x_5, x_{10}\}$. Obviously, $\sigma(x_1) \cap Maj \neq \varnothing$ and $\sigma(x_1) \nsubseteq Maj$ (i.e. x_1 is in the $N^-(Maj)$, but it is not in the $N_-(Maj)$). Therefore, x_1 belongs to $Bn_n(Maj)$, so it is an overlapping instance.

The steps of detecting overlapping areas in a training set are summarized as follows:

(a) For instance x, $\sigma(x)$ is identified;

(b) $N_-(Maj)$ and $N^-(Maj)$ are calculated by Eqs. (7.1) and (7.2), respectively.

(c) $Bn_n(Maj)$, i.e. $Overlap$, is calculated by Eq. (7.3).

The detection method is expected to detect instances in the overlapping areas as described in Figure 7.2 (a) and Figure 7.2 (b) (i.e. the presence of multiple overlapping areas in a binary classification task). This is because the neighborhood information of an instance is used to judge whether the instance is located in overlapping areas or not. The nearest neighbors of each instance can be from either ordinary instances of a class or instances of a particular sub-cluster of a class.

After identifying the overlapping areas in the training data, GP is used to evolve cost-sensitive classifiers. The general classification steps are described in Figure 7.4. A cost-sensitive GP classifier uses one classification strategy to classify instances in the non-overlapping areas and uses a different classification strategy to classify the instances in the overlapping areas. We will introduce the classification strategies later in more detail.

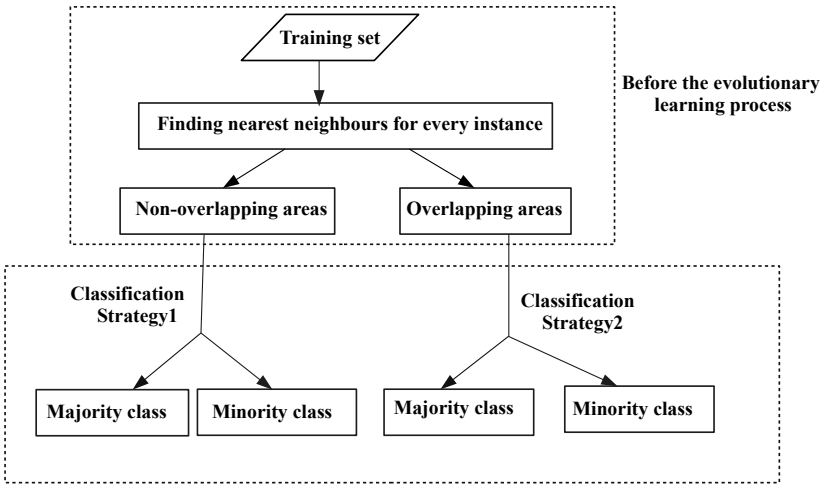


Figure 7.4: Classification steps in CSGPNOD.

7.2.2 Construction of Cost-sensitive Classifiers when Considering Overlapping Areas

The construction of cost-sensitive classifiers is based on the CS-GP method introduced in Chapter 5 (Page 112). For a GP individual (i.e. a GP tree), its left subtree works as a classifier, while its right subtree is used to learn a cost value. The learned cost value is employed to calculate a classification threshold that is used to separate the outputs of the left subtree for predicting class labels of instances.

7.2.3 Classification Strategies

When an instance is fed into a GP classifier, an output value of this classifier could show the probability of which class this instance is from. In CSGPNOD, after inputting all the training instances into a GP classifier, the output values of the left subtree are normalized into the range of [0,1] by max-min normalization, which is defined as:

$$p_x = 1 - \frac{ProgOut_x - min(ProgOut_{all})}{max(ProgOut_{all}) - min(ProgOut_{all})} \tag{7.5}$$

where $ProgOut_x$ indicates an output value of a program when x is input to the program, $ProgOut_{all}$ is a list that includes all the program outputs for all of the training instances, $min(ProgOut_{all})$ and $max(ProgOut_{all})$ indicate the minimum value and the maximum value in $ProgOut_{all}$, respectively.

Classification Strategy 1 for the non-overlapping areas

In the fitness evaluation process, $Pro(Maj)$ and $Pro(Min)$ are two empty sets at the beginning and will save the program outputs (p_x) when this program is used to classify instances from the non-overlapping areas into Maj or Min.

For an individual, after obtaining C represented by its right subtree, the classification threshold TH is calculated by Eq. (5.2) (introduced in Chapter 5, Page 115). For an instance x from the non-overlapping areas, if $p_x \geq TH$, then x is predicted into Maj and p_x is appended into $Pro(Maj)$; otherwise x is predicted into Min and p_x is appended into $Pro(Min)$. The information in $Pro(Maj)$ and $Pro(Min)$ will be used to classify instances in the overlapping areas by using Classification Strategy 2 described below.

Classification Strategy 2 for the overlapping areas

After all of the instances in the non-overlapping areas have been classified, the minimum value in $Pro(Maj)$ and the maximum value in $Pro(Min)$ are calculated, denoted as $min(Pro(Maj))$ and $max(Pro(Min))$, respectively.

For instance x from the overlapping areas, if $p_x \geq TH$ and p_x is closer to $min(Pro(Maj))$ than $max(Pro(Min))$, then x is classified into Maj; otherwise x is classified into Min.

7.2.4 Fitness Function

The goodness of each individual in a population is evaluated by a fitness function. On the basis of the obtained fitness values, good individuals are selected and then used by genetic operators for generating new offspring. In CSGPNOD, after

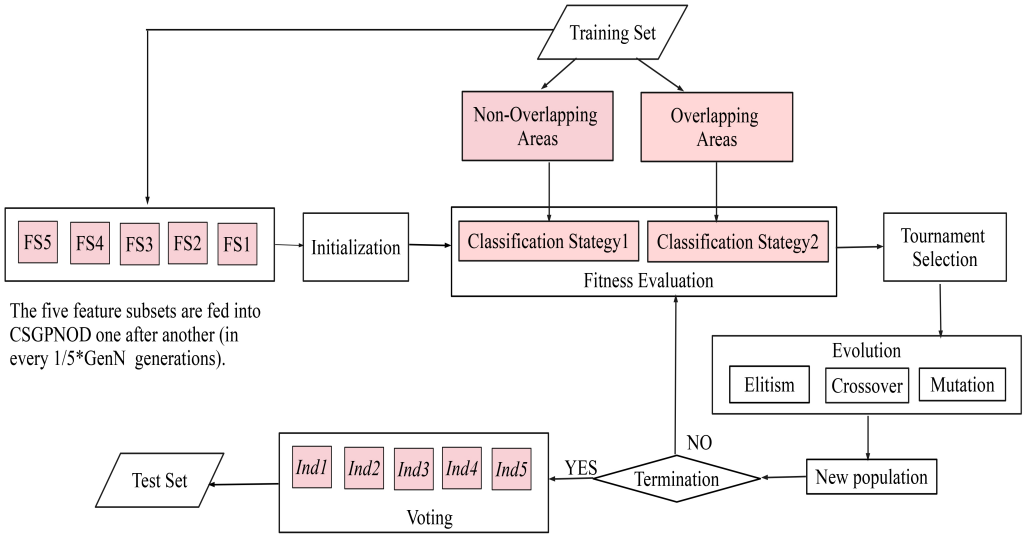


Figure 7.5: Overall design of CSGPNOD.

making classification decisions for all the training instances, the geometric mean, G_Mean, is used as the fitness function, defined as:

$$G\_Mean = \sqrt{\frac{TP}{TP+FN} * \frac{TN}{TN+FP}} \tag{7.6}$$

where TP is true positive, FP is false positive, TN is true negative and FN is false negative.

7.2.5 The Overall design of CSGPNOD

The overall design of CSGPNOD is shown in Figure 7.5. The details are described below.

The training process

During the training process, after distinguishing between overlapping and non-overlapping areas, the whole feature set in a dataset is divided into five feature subsets at random, denoted as $FS1$, $FS2$, $FS3$, $FS4$ and $FS5$. These feature

subsets are fed to CSGPNOD as terminals to train cost-sensitive classifiers one after another in every $(1/5) * Gen_N$ generations, where Gen_N stands for the number of generations. For example, if the number of generations is 50, the five feature subsets are sequentially fed to CSGPNOD one after another in every 10 generations.

At the beginning of the evolutionary learning process, only $FS1$ is fed to CSGPNOD to train cost-sensitive classifiers. After initializing a population, every cost-sensitive classifier uses classification strategy 1 to classify instances from the non-overlapping areas and classification strategy 2 to classify instances from the overlapping areas. The goodness of each individual is evaluated by Eq. (7.6). According to the obtained fitness values, better individuals are selected by tournament selection. To create a new population, the top one individual from a previous generation is directly copied into the new generation, and other new offspring are generated by subtree crossover and mutation.

The best individual from the $((1/5) * Gen_N)th$ generation is saved, denoted as $Ind1$. For the $((1/5) * Gen_N + 1)th$ generation, $FS1$ is replaced by $FS2$ (i.e. $FS2$ is fed to CSGPNOD), and the population is reinitialized. Afterwards, the learning process is the same as the previous $(1/5) * Gen_N$ generations. The best individual from the $((2/5) * Gen_N)th$ generation is saved, denoted as $Ind2$. Similarly, for the following feature subsets (i.e. $FS3$, $FS4$ and $FS5$), each of them is fed to CSGPNOD as terminals successively in every $(1/5) * Gen_N$ generations. The best individuals from the $((3/5) * Gen_N)th$ generation, the $((4/5) * Gen_N)th$ generation and the final generation are saved, denoted as $Ind3$, $Ind4$ and $Ind5$, respectively.

The test process

The overlapping areas in the training set are identified based on the neighborhood information of every instance. On the test set, the neighborhood information of an unseen instance is unknown. Therefore, for an unseen instance, we find out its nearest neighbor from the training set. If its nearest neighbor is from the overlapping areas in the training set, this unseen instance is placed into overlapping areas,

Table 7.1: Datasets.

Dataset	#Features	#Instances	IR (Rounded)
Armstrong-2002-v1	1081	72	2
Golub-1999-v1	1868	72	2
Colon	2000	62	2
Leukemia	7129	72	2
Shipp-2002-v1	798	77	3
DLBCL	5469	77	3
Gordon-2002	1626	181	5
Yeoh-2002-v1	2526	248	5
Tomlins-2006-v1	2315	104	8
Lung	12600	156	8

otherwise it is placed into non-overlapping areas. After the evolutionary learning process, $Ind1$, $Ind2$, $Ind3$, $Ind4$ and $Ind5$ (from the training process) are used together (based on the majority voting) to predict the class labels of unseen instances in a test set.

7.3 Experiment Design

7.3.1 Datasets

The used datasets are the same as those used in the previous contribution chapters, listed in Table 7.1. A dataset is split into the training set (70%) and the test set (30%) by using stratified sampling to guarantee the same class imbalance ratio (IR) in the training set, the test set and the whole original dataset.

7.3.2 Baseline Methods

Baseline methods include two groups of methods, i.e. 11 GP methods and 24 non-GP methods. These baseline methods were introduced in the previous contribution chapters. We list all of the baseline methods in Table 7.2.

Table 7.2: Baseline methods.

GP Methods	Oversampling based	GP$_{SMOTE}$ [16]
		GP$_{BSMOTE1}$ [60]
		GP$_{BSMOTE2}$ [60]
		GP$_{ADASYN}$ [62]
	Fitness function based	GP$_{Ave}$
		GP$_{G_Mean}$
		GP$_{Dist}$ [11]
		GP$_{Corr}$ [11]
		GP$_{Auc_w}$ [205]
	Cost-sensitive GP	ICS-GP (the proposed method in Chapter 6)
		CS-GP (the proposed method in Chapter 5)
Non-GP Methods		SMOTE-1NN B-SMOTE1-1NN B-SMOTE2-1NN ADASYN-1NN
		SMOTE-DT B-SMOTE1-DT B-SMOTE2-DT ADASYN-DT
		SMOTE-RF B-SMOTE1-RF B-SMOTE2-RF ADASYN-RF
		SMOTE-GBDT B-SMOTE1-GBDT B-SMOTE2-GBDT ADASYN-GBDT
		SMOTE-NB B-SMOTE1-NB B-SMOTE2-NB ADASYN-NB
		SMOTE-MLP B-SMOTE1-MLP B-SMOTE2-MLP ADASYN-MLP

7.3.3 Parameter Settings

For all the GP methods in the experiments, the population size is 1024 and the number of generations is 50. Ramped half-and-half is used to initialize a population. After evaluating the goodness of every individual in a population, good individuals are selected by tournament selection (the tournament size is 6). Elitism is used, and the top one individual in a current population is directly copied into the next population. Other individuals of the new population are generated by subtree crossover and mutation (where the crossover and mutation rates are 0.8 and 0.2, respectively). The maximum tree depth is limited to 10. Note that the baseline GP methods (except for ICS-GP and CS-GP) are based on standard tree-based GP. For them, the function set includes +, $-$, $\times$, protected $\div$ and If function; the terminal set includes all the features and a random constant. To detect the overlapping areas in the proposed CSGPNOD method, the number of nearest neighbors k is set to 2. Every GP method has been run 30 times independently, and each run is associated with a random seed (note that all the GP methods use the same set of 30 different random seeds).

7.4 Results and Discussions

7.4.1 CSGPNOD versus the baseline GP Methods

Table 7.3 reports the results of CSGPNOD and the baseline GP methods on the 10 datasets, where bold values are the highest AUC achieved by these methods on each dataset. To show the significance of the differences, the Wilcoxon rank-sum test (with the significance level of 0.05) was conducted to compare CSGPNOD with each of the baseline GP methods. The indicators "$+$", "$-$" and "=" are used to show that CSGPNOD is significantly better/worse than, or similar to a compared method. Note that, we compare and discuss the results of CSGPNOD and CS-GP in Section 7.5.

Table 7.3: CSGPNOD versus the baseline GP methods on the test sets.

Method	AUC (×100)			Training Time (Seconds)	AUC (×100)			Training Time (Seconds)
	Best	Mean±Std		Mean	Best	Mean±Std		Mean
	Armstrong-2002-v1				Golub-1999-v1			
GP_{SMOTE}	100	91.3 ±9.83	+	144.32	100	92.38 ± 10.31	+	195.79
$GP_{BSMOTE1}$	100	94.16 ± 7.43	+	141.78	100	89.11 ± 11.08	+	187.11
$GP_{BSMOTE2}$	100	91.22 ± 10.27	+	153.6	100	85.8± 14.56	+	205.14
GP_{ADASYN}	100	92.21 ± 9.62	+	143.04	100	91.8 ±9.72	+	198.39
GP_{Ave}	100	94.48 ± 8.4	+	114.86	100	91.93 ± 10.09	+	158.77
GP_{G_Mean}	100	92.13 ± 8.01	+	114.88	100	88.99 ± 11.89	+	158.31
GP_{Corr}	100	94.67 ± 7.56	+	142.55	100	96.06 ± 6.32	+	225.06
GP_{Dist}	100	95.84 ± 3.93	+	141.33	100	96.9 ± 5.23	+	229.09
GP_{Auc_w}	100	94.46 ± 4.93	+	1917.99	100	98.42 ± 3.38	=	3089.78
ICS-GP	100	98.22 ±3.1	=	134.85	100	99.11 ± 2.9	=	185.61
CSGPNOD	100	**99.7 ± 0.76**		116.49	100	**99.97 ± 0.16**		120.26
	Colon				Leukemia			
GP_{SMOTE}	92.86	75.99 ± 10.55	+	222.64	100	87.56 ± 10.01	=	919.55
$GP_{BSMOTE1}$	88.1	71.51 ± 12.64	+	206.97	98.21	85.6 ± 13.38	+	948.52
$GP_{BSMOTE2}$	94.05	73.69 ± 15.28	+	228.62	99.11	82.47± 12.94	+	1065.41
GP_{ADASYN}	88.1	74.6 ± 10.25	+	227.64	100	89.73± 8.56	=	951.82
GP_{Ave}	91.67	75.52 ± 10.11	+	177.08	98.21	88.79 ± 7.74	=	975.7
GP_{G_Mean}	92.86	71.51 ± 12.95	+	174.21	100	81.79 ± 15.38	+	979.29
GP_{Corr}	96.43	75.28 ± 10.1	+	201.08	100	86.16 ± 10.84	=	785.98
GP_{Dist}	92.86	76.59 ± 9.63	+	203.64	97.32	86.32 ± 8.95	=	788.22
GP_{Auc_w}	91.67	78.97 ± 7.3	=	2348.72	100	86.28 ± 9.68	=	10396.7
ICS-GP	92.86	78.69 ±6.93	=	196.21	100	87.56 ±9.51	=	861.85
CSGPNOD	91.07	**81.03 ±4.7**		114.96	96.43	**89.81 ± 3.74**		427.16
	Shipp-2002-v1				DLBCL			
GP_{SMOTE}	98.15	82.15 ± 11.77	+	132.95	98.15	83.21 ±9.56	+	816.12

Continued on next page

Table 7.3 – Continued from previous page

$GP_{BSMOTE1}$	95.37	77.93 ± 12.35	+	126.82	99.07	75.77 ± 18.56 +	795.67
$GP_{BSMOTE2}$	100	79.46 ± 14.87	+	146.3	98.15	79.41 ± 11.68 +	846.71
GP_{ADASYN}	96.3	79.88 ±12.18	+	137.39	100	79.48± 10.92 +	831.63
GP_{Ave}	99.07	82.85 ±9.81	+	95.15	98.15	75.4 ± 15.67 +	740.03
GP_{G_Mean}	96.3	83.09 ± 9.21	+	97.85	100	77.01 ± 15.75 +	731.45
GP_{Corr}	99.07	83.02 ±13.33	+	216.72	98.15	81.02± 11.42 +	643.18
GP_{Dist}	99.07	84.81 ±8.96	+	214.44	99.07	84.35 ± 9.96 +	633.55
GP_{Auc_w}	100	82.62 ±9.45	+	3192.03	100	85.54 ± 10.83 +	7845.03
ICS-GP	100	84.46±8.91	+	135.99	100	81.56±11.95 +	761.97
CSGPNOD	99.54	**89.44±7.28**		113.08	100	**89.34± 6.84**	203.61
	Gordon-2002				Yeoh-2002-v1		
GP_{SMOTE}	100	97.49 ± 2.92	+	630.05	100	87.44 ± 9.24 +	1321.18
$GP_{BSMOTE1}$	100	98.71 ± 2.67	=	584.44	99.75	86.23 ± 10.64 +	1290.39
$GP_{BSMOTE2}$	100	96.35 ± 4.92	+	669.96	98.39	83.27 ± 10.15 +	1388.96
GP_{ADASYN}	100	97.81 ± 2.9	+	598.04	100	84.28± 10.24 +	1452.27
GP_{Ave}	100	98.26±2.81	=	321.69	100	83.97 ± 11.91 +	773.26
GP_{G_Mean}	100	98.38 ±3.01	=	323.48	95.78	66.33 ± 16.09 +	767.4
GP_{Corr}	100	96.95 ±6.41	+	718.51	100	93.29 ± 7.77 +	685.35
GP_{Dist}	100	98.9 ±3.62	=	720.18	100	91.1 ± 8.22 +	694.73
GP_{Auc_w}	100	99.23 ± 2.06	=	21978.57	100	98.95 ± 2.32 =	24174.23
ICS-GP	100	97.19± 2.39	+	441.93	100	97.87±3.86 +	894.9
CSGPNOD	100	**99.63±0.74**		267.1	100	**99.8 ± 0.36**	438.53
	Tomlins-2006-v1				Lung		
GP_{SMOTE}	100	83.27± 13.43	+	499.69	100	81.7 ± 15.9 +	4298.87
$GP_{BSMOTE1}$	100	87.37 ± 10.14	+	521.56	100	88.89 ±10.76 +	3631.29
$GP_{BSMOTE2}$	100	90.18 ± 10.78	+	500.18	100	84.16± 15.78 +	3901.94
GP_{ADASYN}	100	84.67± 13.16	+	478.33	100	82.97 ± 14.98 +	4137.5
GP_{Ave}	100	88.87±13.47	+	293.08	100	83.46 ± 14.73 +	3048.62
GP_{G_Mean}	100	84.54 ±14.55	+	296.92	99.05	80.89 ± 18.41 +	3038.89
GP_{Corr}	100	90.42 ± 14.1	+	648.78	100	80.71 ±17.21 +	2490.26

Continued on next page

Table 7.3 – Continued from previous page

GP_{Dist}	100	95.83 ± 6.73	+	646.03	100	84.27 ± 14.8	+	2493.37
GP_{Auc_w}	100	91.10 ±9.75	+	7865.15	100	92.35 ± 13.23	+	45375.33
ICS-GP	100	97.43±2.46	+	368.59	100	98.46± 3.61	=	2581.3
CSGPNOD	100	**99.17 ± 2.32**		172.98	100	**99.86 ± 0.56**		761.06
Total	**81 +, 19 =, 0 −**							

Overall, according to the results of statistical significance tests in Table 7.3, the proposed CSGPNOD method performs significantly better than or similar to the baseline GP methods in all the 100 cases (significantly better performance in 81 cases and similar performance in 19 cases, respectively). On all of the 10 datasets, CSGPNOD achieves the highest mean AUC than the baseline GP methods, and the standard deviation (Std) of AUC results from the 30 runs is smaller than that of the baseline GP methods.

GP_{SMOTE}, $GP_{BSMOTE1}$, $GP_{BSMOTE2}$ and GP_{ADASYN} are oversampling-based GP methods. The used oversampling methods generate synthetic instances for the minority class to re-balance a training set. By comparing with the four GP methods on the 10 datasets, CSGPNOD achieves significantly better or similar performance in all the 40 cases (significantly better performance in 37 cases and similar performance in 3 cases). Oversampling methods have risks of generating noisy instances and overfitting (overfitting to the generated noisy instances). Moreover, because the generated instances are also used in the fitness evaluation process, additional computational costs are required.

GP_{Ave} and GP_{G_Mean} use the standard classification strategy (the classification threshold TH is set to 0 in advance). Note that, CSGPNOD also employs G_Mean as the fitness function, but the classification strategy is different from that in GP_{G_Mean} (i.e. TH is not given in advance, and it is calculated based on the cost value C that is represented by the right subtree). Compared with the two methods (GP_{Ave} and GP_{G_Mean}), CSGPNOD performs significantly better in 17 out of the 20 cases (in the other 3 cases, CSGPNOD achieves similar performance).

GP_{Corr}, GP_{Dist} and GP_{Auc_w} use AUC-based fitness functions. The three GP methods are also independent of a fixed classification threshold during the evolutionary learning process. Compared with GP_{Auc_w}, CSGPNOD achieves significantly better performance on 5 datasets and similar performance on the other 5 datasets. To compare the similar performances, CSGPNOD performs slightly better than GP_{Auc_w} on Golub-1999-v1, Colon, Leukemia, Gordon-2002 and Yeoh-2002-v1. Therefore, CSGPNOD does not perform worse than GP_{Auc_w} on these datasets, and its training time is much shorter than GP_{Auc_w}.

ICS-GP is a cost-sensitive GP method based on cost intervals. By comparing with ICS-GP on the 10 datasets, CSGPNOD achieves significantly better performance on 5 datasets and similar performance on the other 5 datasets. In Section 7.5, we will further investigate CSGPNOD in order to reveal contributions of different components to improve the effectiveness and efficiency.

7.4.2 CSGPNOD versus the Non-GP Baseline Methods

We report the AUC results of the non-GP baseline methods on the test sets in Tables 7.4, 7.5, 7.6 and 7.7. According to Table 7.4, by comparing the mean AUC results of CSGPNOD with classification algorithms using SMOTE, CSGPNOD achieves better performance in 57 out of the 60 comparisons. Based on results in Table 7.5 and Table 7.6, by comparing the mean AUC results of CSGPNOD with classification algorithms using Borderline-SMOTE1 or Borderline-SMOTE2, CSGPNOD achieves better performance in 104 out of the 120 comparisons. As can be seen from Table 7.7, by comparing the mean AUC results of CSGPNOD with classification algorithms using ADASYN, CSGPNOD achieves better performance in 59 out of the 60 comparisons.

7.5 Further Analysis

In Table 7.8, we report the number of overlapping instances detected by the proposed method in a training set. As can be seen from Table 7.8, these datasets have

Table 7.4: CSGPNOD versus the non-GP classification methods using SMOTE (AUC$\times$100).

Dataset	SMOTE-1NN	SMOTE-DT	SMOTE-RF	SMOTE-GBDT	SMOTE-NB	SMOTE-MLP	**CSGPNOD**	
							Best	Mean
Armstrong-2002-v1	96.67	88.46	90.97	89.52	85.71	92.85	100	**99.7**
Golub-1999-v1	93.75	89.91	89.80	92.86	68.75	96.43	100	**99.97**
Colon	74.40	64.04	64.46	65.83	47.62	62.60	91.07	**81.03**
Leukemia	90.18	86.61	81.52	86.61	**100**	69.61	96.43	89.81
Shipp-2002-v1	72.22	68.24	72.22	69.44	58.33	76.67	99.54	**89.44**
DLBCL	69.44	67.31	77.13	72.22	80.86	74.63	100	**89.34**
Gordon-2002	98.91	86.09	92.94	90.93	88.89	85.74	100	**99.63**
Yeoh-2002-v1	86.29	96.03	72.66	96.15	80.58	84.10	100	**99.8**
Tomlins-2006-v1	98.21	80.36	88.51	83.75	75	75.95	100	**99.17**
Lung	85.24	95.21	82.23	**100**	80.00	82.52	100	99.86

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 7.5: CSGPNOD versus the non-GP classification methods using Borderline-SMOTE1 (AUC$\times$100).

Dataset	B-SMOTE1-1NN	B-SMOTE1-DT	B-SMOTE1-RF	B-SMOTE1-GBDT	B-SMOTE1-NB	B-SMOTE1-MLP	**CSGPNOD**	
							Best	Mean
Armstrong-2002-v1	96.67	89.16	85.29	89.52	85.71	92.86	100	**99.7**
Golub-1999-v1	93.75	90.15	83.96	92.86	68.75	93.75	100	**99.97**
Colon	74.40	63.12	65.04	62.26	47.62	74.40	91.07	**81.03**
Leukemia	**96.43**	86.61	76.16	86.61	93.75	75	96.43	89.81
Shipp-2002-v1	75.00	68.42	68.15	69.44	66.67	75	99.54	**89.44**
DLBCL	80.56	70.37	70.93	73.43	80.56	69.44	100	**89.34**
Gordon-2002	97.83	93.94	91.96	93.36	88.89	**100**	100	99.63
Yeoh-2002-v1	96.15	96.15	66.90	96.15	82.20	87.28	100	**99.8**
Tomlins-2006-v1	**100**	84.29	80.36	81.67	62.50	**100**	100	99.17
Lung	95.24	95.75	78.55	**99.92**	70.00	90.00	100	99.86

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 7.6: CSGPNOD versus the non-GP classification methods using Borderline-SMOTE2 (AUC$\times$100).

Dataset	B-SMOTE2-1NN	B-SMOTE2-DT	B-SMOTE2-RF	B-SMOTE2-GBDT	B-SMOTE2-NB	B-SMOTE2-MLP	**CSGPNOD**	
							Best	Mean
Armstrong-2002-v1	93.33	89.79	88.86	89.52	**100**	92.86	100	99.7
Golub-1999-v1	93.75	90.03	79.11	92.86	77.68	**100**	100	99.97
Colon	70.24	76.69	62.74	77.38	43.45	60.12	91.07	**81.03**
Leukemia	96.43	86.61	78.01	86.61	**100**	81.25	96.43	89.81
Shipp-2002-v1	77.78	73.61	72.50	72.22	75	**91.67**	99.54	89.44
DLBCL	75	76.85	75.37	83.33	88.89	69.44	100	**89.34**
Gordon-2002	97.83	93.68	89.41	93.36	**100**	**100**	100	99.63
Yeoh-2002-v1	**100**	99.49	68.23	**100**	90.69	87.28	100	99.8
Tomlins-2006-v1	89.29	87.50	92.38	87.50	62.50	98.21	100	**99.17**
Lung	91.67	99	80.31	**100**	80.00	87.62	100	99.86

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 7.7: CSGPNOD versus the non-GP classification methods using ADASYN (AUC×100).

Dataset	ADASYN-1NN	ADASYN-DT	ADASYN-RF	ADASYN-GBDT	ADASYN-NB	ADASYN-MLP	**CSGPNOD**	
							Best	Mean
Armstrong-2002-v1	93.33	89.59	91.97	89.52	92.86	92.85	100	**99.7**
Golub-1999-v1	93.75	90.51	89.35	92.86	68.75	92.86	100	**99.97**
Colon	74.40	66.69	66.19	62.20	47.62	67.26	91.07	**81.03**
Leukemia	83.04	86.61	80.47	86.61	**100**	65.15	96.43	89.81
Shipp-2002-v1	83.33	80.46	75.00	83.33	58.33	80.92	99.54	**89.44**
DLBCL	77.78	67.31	74.35	72.22	88.8	79.44	100	**89.34**
Gordon-2002	98.91	83.24	95.01	83.33	88.89	83.15	100	**99.63**
Yeoh-2002-v1	91.13	95.64	72.66	96.15	86.04	82.07	100	**99.8**
Tomlins-2006-v1	98.21	80.36	86.90	82.5	62.5	85.53	100	**99.17**
Lung	76.90	97.83	79.74	99.67	80.00	74.60	100	**99.86**

1: Bold values are the highest AUC result achieved by methods on a dataset.

Table 7.8: The number of the detected overlapping instances in the training sets.

Dataset	The number of overlapping instances
Armstrong-2002-v1	11
Golub-1999-v1	19
Colon	13
Leukemia	10
Shipp-2002-v1	16
DLBCL	14
Gordon-2002	3
Yeoh-2002-v1	26
Tomlins-2006-v1	1
Lung	7

the class overlap issue. In order to investigate the contribution of every component of CSGPNOD, CSGPNOD is further compared with another two methods, introduced as follows:

- CS-GP: CS-GP is a cost-sensitive GP method that is able to automatically optimize cost values, but it does not consider addressing the class overlap issue. CS-GP was introduced in Chapter 5 (Page 112).

- ACSGPNOD: ACSGPNOD uses all the features at the beginning, which is the only difference between CSGPNOD and ACSGPNOD.

The comparison between ACSGPNOD and CS-GP is to investigate whether it is particularly useful to resolve the class overlap issue. The comparison between CSGPNOD and ACSGPNOD is to test the effectiveness of dividing the whole feature set and using each feature subset one after another in CSGPNOD.

In Table 7.9, we report the results of CS-GP and ACSGPNOD. As can be seen from Table 7.9, ACSGPNOD performs better than CS-GP on all of the datasets, but on 8 datasets, ACSGPNOD consumes longer training time than CS-GP. There-

Table 7.9: Results of the variants of CSGPNOD on the test sets.

		AUC ×100		Training Time (Seconds)
Datasets	Methods	Best	Mean±Std	Mean
Armstrong	CS-GP	100	97.60±5.32	130.18
	ACSGPNOD	100	99.16 ±2.11	183.17
	CSGPNOD	100	99.7 ± 0.76	116.49
Golub-1999-v1	CS-GP	100	98.95± 2.83	187.61
	ACSGPNOD	100	99.55 ± 2.24	258.19
	CSGPNOD	100	99.97 ± 0.16	120.26
Colon	CS-GP	90.48	79.05 ± 7.03	185.5
	ACSGPNOD	94.05	82.14 ±6.03	243.1
	CSGPNOD	91.07	81.03 ±4.7	114.96
Leukemia	CS-GP	96.43	85.57±9.03	1006.54
	ACSGPNOD	96.43	88.51±5.73	959.77
	CSGPNOD	96.43	89.81 ± 3.74	427.16
Shipp-2002-v1	CS-GP	95.37	83.3 ±8.34	100.23
	ACSGPNOD	98.15	84.17±11.88	150.63
	CSGPNOD	99.54	89.44±7.28	113.08
DLBCL	CS-GP	94.44	80.54± 9.06	753.79
	ACSGPNOD	98.15	81.44±9.84	783.96
	CSGPNOD	100	89.34± 6.84	203.6
Gordon-2002	CS-GP	100	96.71±4.2	406.25
	ACSGPNOD	100	97.35± 2.17	490.47
	CSGPNOD	100	99.63±0.74	267.1
Yeoh-2002-v1	CS-GP	100	97.31± 4.29	811.81
	ACSGPNOD	100	99.20±1.66	898.45
	CSGPNOD	100	99.8 ± 0.36	438.53
Tomlins-2006-v1	CS-GP	100	96.98 ±4.3	328.66
	ACSGPNOD	100	98.2±1.57	379.85
	CSGPNOD	100	99.17 ± 2.32	172.98
Lung	CS-GP	100	97.13± 2.57	3241.32
	ACSGPNOD	100	98.71± 1.18	2724.39
	CSGPNOD	100	99.86 ± 0.56	761.06

fore, the classification performance is improved after addressing the class overlap issue. However, the classification performance improvement is at the expense of efficiency. This is because ACSGPNOD needs to detect class overlapping areas, and then uses the different strategies to classify instances from the overlapping and non-overlapping areas in the fitness evaluation process, which require more computational costs.

Table 7.9 shows that CSGPNOD performs better than ACSGPNOD in almost all cases (except for Colon, the classification performance of CSGPNOD is slightly worse than ACSGPNOD). However, the training time of CSGPNOD is significantly shorter than ACSGPNOD. This is because, in ACSGPNOD, for every generation, every feature from a dataset has a chance to be used by an individual in the population. However, in CSGPNOD, all the features are randomly grouped into 5 feature subsets, which are successively fed into CSGPNOD in every 10 generations. Therefore, the search space is reduced and the risk of generating large-sized individuals is also possibly reduced.

Evolved Programs

In Figure 7.6, we draw the cost-sensitive classifiers evolved by CSGPNOD from a GP run on Golub-1999-v1 (including 1868 features and 72 instances). The five cost-sensitive classifiers in Figure 7.6 work together to predict class labels of unseen instances. Note that the initially generated cost values are rounded to two decimal places in the figure. For the five cost-sensitive classifiers, the associated cost values are rounded to 4.4, 6.1, 20.6, 15.4 and 5.0. Although the five cost-sensitive classifiers are from the same GP run, the associated cost values are clearly different. The main reason is explained as follows. For a GP tree, it attempts to construct a classifier as well as to learn a cost value. The learned cost value is used by the constructed classifier, and then the performance of an individual is evaluated. As a result, the associated cost value has a close relationship with the constructed classifier in the tree.

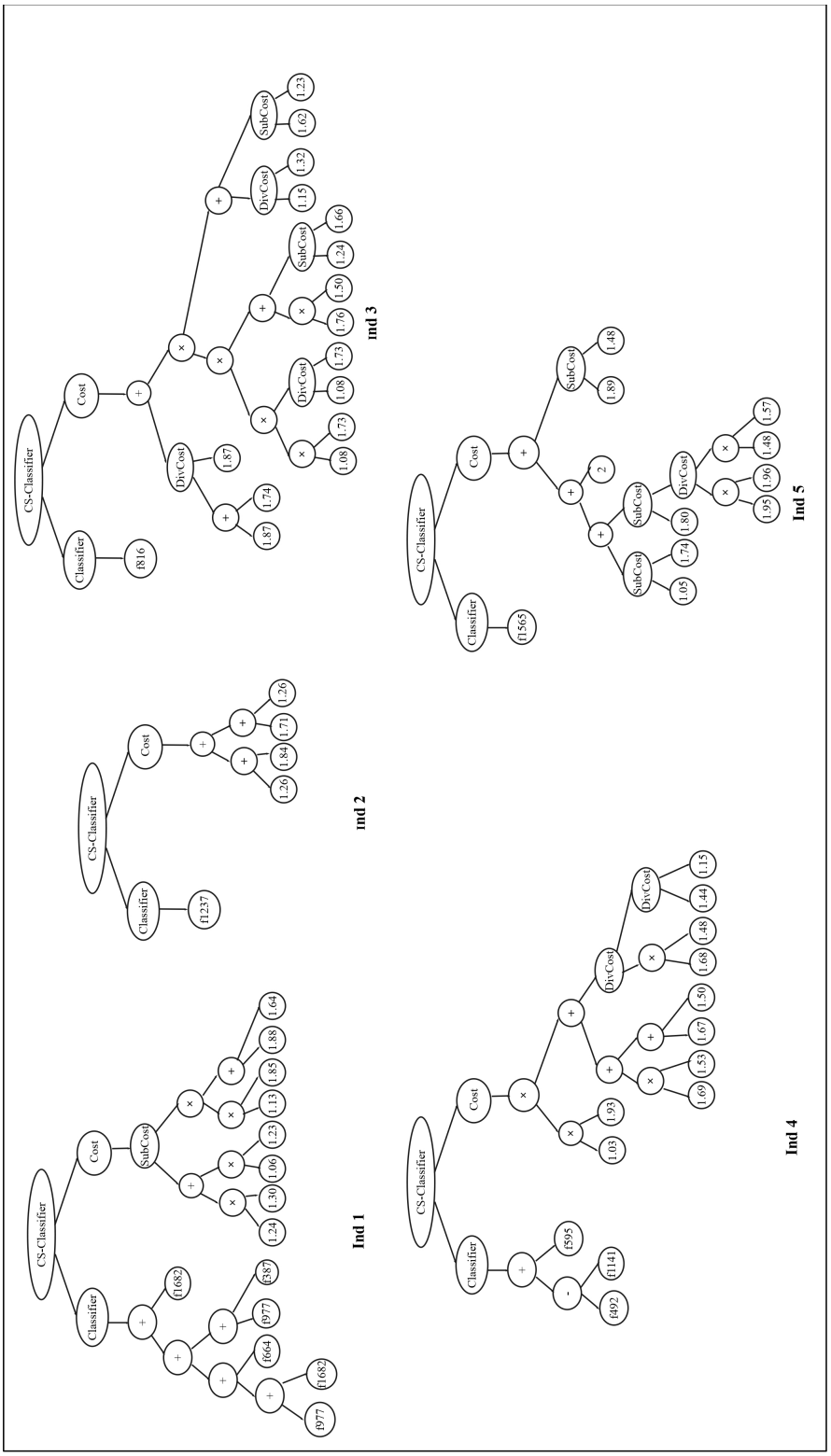


Figure 7.6: The evolved programs by CSGPNOD on Golub-1999-v1.

7.6 Chapter Summary

The goal of this chapter was to resolve the issue of class overlap in classification with unbalanced high-dimensional data. The chapter goal has been achieved by designing the CSGPNOD method. In CSGPNOD, the overlapping areas are detected based on the neighborhood rough set method. After distinguishing between overlapping and non-overlapping areas, cost-sensitive GP classifiers are trained based on different strategies to classify instances from the overlapping and non-overlapping areas, separately.

In the experiments, the effectiveness and efficiency of CSGPNOD were compared with the 11 GP baseline methods on the 10 unbalanced high-dimensional datasets. The experimental results show that CSGPNOD achieved better classification performance and consumed less training time than the baseline GP methods. Furthermore, compared with the 24 non-GP baseline methods, CSGPNOD also achieved better performance in almost all comparisons.

Chapter 8

Conclusions

This thesis focuses on the use of GP for binary classification with high-dimensional unbalanced data, with the overall goal of enhancing the classification performance. The goal has been successfully achieved by designing a number of new GP methods for classification with high-dimensional unbalanced data. The proposed methods were examined and compared with existing popular methods on high-dimensional unbalanced datasets. The experimental results indicated that the proposed GP methods achieved better or similar classification performance than the compared methods in almost all cases.

The rest of this chapter will draw conclusions and provide main findings and insights for each of the contribution chapters. Afterwards, we will introduce potential research areas for future works.

8.1 The Achieved Objectives

The achieved research objectives are introduced as follows:

- A new AUC approximation measure was designed, which has been proven to be relatively effective and more efficient in evaluating the goodness of individuals than AUC. Moreover, a new program reuse mechanism was designed to reuse previous effective GP individuals in order to further im-

prove the efficiency. A new GP method was proposed by combining the new AUC approximation measure and the program reuse mechanism. According to the experimental results, the new GP method can significantly reduce the computational time and often achieve at least similar classification performance to GP methods that employ different fitness functions, e.g. G_Mean and AUC. Moreover, the proposed method outperforms traditional classification algorithms with oversampling methods to address the class imbalance issue in almost all comparisons.

- A new multi-criterion fitness evaluation method and a selection operator were developed, which could avoid manually setting up weights to combine multiple criteria into a fitness function in the evolutionary learning process. The proposed method evaluates an individual by two criteria, and the obtained values on the two criteria are combined in pairs (called a fitness tuple). To deal with fitness tuples, a new multi-criterion tournament selection was designed, which could allow a set of solutions to be filtered according to a cascading set of priorities. A combination of the new evaluation method and the selection operator enables GP to effectively identify and select good individuals to create new offspring in the evolutionary learning process. The experimental results show that the proposed GP method achieves significantly better (or similar) classification performance than GP methods that employ different fitness functions in almost all cases. Further analysis reveals the contributions of different components in improving the effectiveness and efficiency, and also shows that the proposed method is able to produce small trees.

- The thesis investigated and showed how cost-sensitive learning can be used with GP to develop cost-sensitive GP classifiers. In the thesis, two new cost-sensitive GP methods were presented based on a new tree representation, new terminal and function sets. More importantly, the two proposed cost-sensitive GP methods do not require manually-designed cost matrices when they are not available. When constructing classifiers, the first GP method

is able to automatically learn the needed cost values, while the second GP method can learn cost intervals. The experimental results indicate that the proposed GP methods achieve better performance than the compared methods in almost all comparisons.

- The thesis investigated how the issue of class overlap can be addressed and thereby designed a new GP method in classification with high-dimensional unbalanced data. In the proposed GP method, the overlapping areas are detected based on the neighborhood rough set before the evolutionary learning process starts. After distinguishing between the overlapping and non-overlapping areas, cost-sensitive GP classifiers are trained based on different classification strategies to classify instances from the overlapping and non-overlapping areas, separately. The experimental results show that the classification performance of GP is improved after the class overlap issue is well-addressed.

8.2 Main Conclusions

The thesis investigates how GP can be effectively used to develop classifiers for binary classification when data is high-dimensional and unbalanced. The thesis finds that the performance bias issue of GP can be effectively addressed by designing new fitness functions or using cost-sensitive learning in classification with high-dimensional unbalanced data. This section draws the main conclusions for each of the five research objectives from Chapters 3-7.

8.2.1 GP with a New Fitness Function and Program Reuse Mechanism

Chapter 3 proposed a new GP based classification method (i.e. GPFRM) to evolve classifiers for classification with high-dimensional unbalanced data, based on a new fitness function and a program reuse mechanism.

Fitness Function

In GP, a fitness function is employed to evaluate the goodness of each individual for guiding the evolutionary learning process towards finding better solutions. In unbalanced classification, the choice of a suitable fitness function can significantly influence the classification performance of GP.

GP using AUC as the fitness function has been proved to work effectively in addressing the class imbalance issue, but it is very time-consuming. Therefore, to reduce the training time, AUC is approximated in the newly designed fitness function. Besides, the newly-designed fitness function considers the classification clarity to further distinguish the effectiveness of the programs.

Program Reuse Mechanism

It is found that the proposed program reuse mechanism is able to significantly reduce the training time and further improve the classification performance in almost all cases because of the following reasons. Firstly, in the program reuse mechanism, not all the features are fed into GP at the beginning of the evolutionary learning process. Therefore, the number of input features in each sub-process is significantly reduced, which may avoid generating large GP trees that require more computational costs. Secondly, the program reuse mechanism suggests to reuse both the good features previously selected and good trees in the initialization of the later sub-process. This enables the following generations to take benefit from the previous good trees for further improving the effectiveness.

By combining the proposed fitness function and program reuse mechanism, the GPFRM method can simultaneously enhance the classification performance and reduce training time in almost all cases for high-dimensional unbalanced classification.

8.2.2 GP with Multi-criterion Fitness Evaluation and Selection

Chapter 4 proposed a novel GP based classification method (i.e. GPMFS) for classification with high-dimensional unbalanced data, based on a new multi-criterion

fitness evaluation method and a selection operator.

Two-criterion Fitness Evaluation

The fitness function in Chapter 3 treated the AUC approximation and classification clarity criteria as equal and summed them together. In reality, the two criteria are not always equally important, and it is usually hard to set an accurate weight without domain knowledge. Basically, to distinguish between two programs, the classification clarity of a program becomes important when the two programs achieve the same AUC performance. To avoid providing a weight, we designed a two-criterion fitness evaluation method in Chapter 4. In the new evaluation method, the obtained fitness values on the two criteria are combined in pairs in the fitness evaluation process, instead of summing them together. A pair of two values are used to show the goodness of a program on the two criteria, and also used as an input for the following selection process.

Three-criterion Tournament Selection

In GP, the selection process is also crucial since the selected individuals are used as parents to generate new offspring by crossover and mutation for the next generation. The standard tournament selection is designed mainly for dealing with fitness values, so it needs to be changed to deal with fitness tuples. In Chapter 4, a new tournament selection was designed to deal with fitness tuples, and it considered the three criteria, i.e. AUC approximation measure $C1$, classification clarity $C2$, and program sizes $C3$. The three-criterion tournament selection allows a set of solutions to be filtered according to a cascading set of criterion priorities to identify a best solution in a tournament.

By combining the designed multi-criterion fitness evaluation and selection methods, the GPMFS method achieves superior classification performance and produces small programs that only use several out of a large number of features for high-dimensional unbalanced classification. It is found that the improved effectiveness is due mainly to the cooperation of $C1$, $C2$ and program size $C3$ in

the three-criterion tournament selection, and the high efficiency of GPMFS is due mainly to the smaller sizes of programs in the population.

8.2.3 Value-based Cost-sensitive GP

In Chapter 5, we investigated how GP can be used with cost-sensitive learning. In cost-sensitive learning, cost matrices are usually provided by domain experts or end users, while when using cost-sensitive learning to address the class imbalance issue over a wide range of unbalanced datasets in many real-world applications, the cost matrices are not always available. We proposed a new cost-sensitive GP method (i.e. CS-GP) by designing a new tree representation, terminal and function sets for classification with high-dimensional unbalanced data, to learn cost values and construct classifiers simultaneously for the case that the cost matrices are not available.

Tree Representation, Terminal and Function Sets

In the CS-GP method, a tree representation is introduced, which allows a candidate solution to learn a cost value and develop a classifier simultaneously. For an individual, its *left subtree* is used to evolve a classifier, meanwhile its *right subtree* is used to learn a cost value. After designing terminal and function sets, the right subtree in an individual can choose suitable functions from the function set to increase or decrease the initial cost values (note that subtraction and division operators in the function set were designed to guarantee that the evolved cost value for the minority class is greater than or equal to 1). The cost value represented by the right subtree is used to calculate a classification threshold for use by the evolved classifier (i.e. the left subtree) to make classification predictions.

CS-GP is the first cost-sensitive GP method that is independent of manually-designed cost matrices. Based on the experimental results, CS-GP achieves superior classification performance in both slightly-unbalanced and highly-unbalanced cases. Therefore, cost-sensitive learning could improve the performance of GP in unbalanced classification.

8.2.4 Interval-based Cost-sensitive GP

In Chapter 6, we investigated how cost intervals can be automatically learned to construct cost-sensitive classifiers. Based on that, we proposed an interval-based cost-sensitive GP method (i.e. ICS-GP) by designing a tree representation, terminal and function sets, classification strategies and a fitness function.

Tree Representation, Terminal and Function Sets

In the ICS-GP method, for each individual, its *left subtree* is used to represent a classifier while its *right subtree* is used to represent a cost interval (C_min, C_max). For the function sets designed for learning cost intervals, the subtraction and division operators need to ensure that both C_min and C_max in an evolved cost interval are greater than or equal to 1.

Cost Interval based Classification Strategy

In the ICS-GP method, the classification strategy was designed based on a threshold-moving idea and it considered two important values in a cost interval, i.e. the maximum value C_max and the middle value C_middle. C_max is the maximum cost caused by a mistake. C_max could be used to avoid the classification mistake of a false negative at most, while it may overestimate the mistake. Therefore, it may not make the total cost small enough. In an interval, C_middle stands for a middle cost value between C_min and C_max, which could be used to reflect an entire interval. In the ICS-GP method, based on C_max and C_middle, two classification thresholds are defined and calculated, and then the classification performance of a constructed classifier is evaluated at the two thresholds.

By comparing between CS-GP and ICS-GP on the high-dimensional unbalanced datasets, the experimental results show that ICS-GP is superior to CS-GP in almost all comparisons, but it consumes longer training time than CS-GP because of more computational costs to calculate cost intervals and to derive the two thresholds to evaluate the classification performance of a cost-sensitive classifier.

8.2.5 GP with Detection of Overlapping Areas Using Rough Set

In Chapter 8, we investigated how the class overlap issue can be addressed in GP for classification with high-dimensional unbalanced data. To achieve this goal, we designed a new GP method that detects overlapping areas before the evolutionary learning process.

Neighborhood-based Overlapping Area Detection Method

A neighborhood rough set based method was designed to detect overlapping areas. After distinguishing between overlapping and non-overlapping areas, GP is used to develop cost-sensitive GP classifiers, where different strategies were designed to classify instances from the overlapping and non-overlapping areas, separately. Note that the instances from the non-overlapping areas are classified prior to classifying the instances from the overlapping areas.

The neighborhood-based detection method is able to detect one or more overlapping areas because the neighborhood information of an instance is used to judge whether the instance is located in overlapping areas or not. The nearest neighbors of each instance can be from either ordinary instances of a class or instances of a particular sub-cluster of a class. It is discovered that the classification performance of GP is further improved over the CS-GP method in Chapter 5 after resolving the class overlap issue.

8.3 Summaries on the Proposed Methods

In this subsection, we summarize and compare the proposed methods in the contribution chapters, and discuss their major limitations.

8.3.1 Comparisons on the Proposed Methods

Chapters 3-4 investigated how the class imbalance issue can be resolved by means of the fitness function in GP. In chapter 3, we proposed a new fitness function based on two criteria, i.e. AUC approximation $C1$ and classification clarity $C2$. The newly-designed fitness function could evaluate individuals effectively and efficiently. A program reuse mechanism was proposed, which suggested not only reusing good features previously selected but also reusing previous effective trees in the initialization of the later GP sub-process, to further improve the efficiency. With the new fitness function and program reuse mechanism, the proposed GPFRM method has demonstrated its effectiveness and efficiency on the high-dimensional unbalanced datasets used in the experiments. Based on our further investigations, the classification performance of GPFRM was enhanced due mainly to the new fitness function, and the efficiency was improved due mainly to the program reuse mechanism.

However, the major drawback of GPFRM is that $C1$ and $C2$ are weighted equally in the proposed fitness function. To address the drawback, in chapter 4, we proposed a GP method (called GPMFS), where a new fitness evaluation method and a selection operator were developed. The GPMFS method could avoid weighting multiple criteria in the evaluation process and effectively identify good programs in the selection process to be parents to breed offspring. According to the experimental results, the GPMFS method achieved better performance than the GPFRM method in chapter 3. Based on our further investigations, the classification performance of GPMFS was improved mainly due to the cooperation of $C1$, $C2$ and program size $C3$ in the three-criterion tournament selection. The good efficiency of GPMFS was gained mainly due to the small size of programs in a population.

Differently from methods in chapters 3-4, chapters 5-6 investigated the use of cost-sensitive learning to address the class imbalance issue in GP when cost matrices are not available from domain experts or end users. In chapter 5, we designed a new tree representation, terminal and function sets, so that the proposed CS-GP method could automatically construct cost-sensitive classifiers. According

to the experimental results, the proposed CS-GP method addressed the class imbalance issue successfully, achieving a good classification performance. On the basis of chapter 5, we conducted further investigations on the use of cost intervals to construct cost-sensitive GP classifiers and proposed a new ICS-GP method in chapter 6. By comparing between CS-GP and ICS-GP on the datasets in the experiments, ICS-GP achieved better classification performance than CS-GP in almost all cases, but it consumed a longer training time than CS-GP. The CS-GP and ICS-GP methods usually achieved better classification performance on highly-unbalanced high-dimensional data than the GPFRM and GPMFS methods proposed in chapter 3-4.

In chapters 3-6, the proposed methods did not consider addressing the class overlap issue. The proposed cost-sensitive methods treat the minority class as being more important than the majority class. However, in class overlapping areas, the prior probabilities of the two classes are roughly the same [57]. This may decrease the performance of a cost-sensitive classifier in the overlapping areas. To improve the classification performance, in chapter 7, we proposed a CSGPNOD method, which could detect overlapping areas before the evolutionary learning process and classify instances from the overlapping and non-overlapping areas, separately. According to the experimental results, the classification performance was further enhanced after the class overlap issue was well addressed. The CSGPNOD method achieved better performance than the other 4 methods proposed in the previous chapters.

8.3.2 Major Limitations of the Proposed Methods

The limitations of the proposed methods are listed as follows:

1) All of the proposed methods are targeted at binary classification with high-dimensional unbalanced data. For using them in a multi-class classification task, the task must be decomposed into multiple binary classification tasks to deal with.

2) The proposed cost-sensitive methods (CS-GP and ICS-GP) are designed

only for the case that there is no cost information provided by domain experts or end users. Besides, the learned cost information could not be decoupled from the classifier which has been evolved together, because the cost information represented by the right subtree is only used by the classifier represented by the left subtree for evaluating the performance of an individual in the fitness evaluation process.

3) All of the proposed methods are based on tree representation of GP for classification. This thesis has not investigated other types of representations.

8.4 Future Work

In this section, we discuss some key areas of future work.

8.4.1 GP for Multi-class Classification with Unbalanced Data

Many real-world unbalanced data tasks are related to multi-class classification. Although a multi-class classification task can be conducted by decomposing it into multiple binary classification tasks, it is still worth investigating how the multi-class classification task can be conducted directly and efficiently.

8.4.2 Improving the Generality of Learned Cost information

In Chapters 5 and 6, we designed two cost-sensitive GP methods, where cost information (cost values or cost intervals) was co-learned with classifiers. However, the learned cost information could not be decoupled from the classifier which has been evolved together. In the future, it is interesting to explore how the learned cost information can be used by different kinds of classifiers, and how the performance of these cost-sensitive classifiers can be evaluated and be effectively combined to improve the classification performance.

8.4.3 Improving the Interpretability of GP in High-dimensional Unbalanced Classification

In general, GP has a relatively good interpretability due mainly to its tree representation. However, GP may suffer from the issue of bloat, particularly when data is high-dimensional. The size increase of a program may not always guarantee the performance improvement, and it leads to complicated models. An over-complicated model is usually hard for end users to understand when determining whether the predictions are reliable or not. In the future, it is crucial to investigate how the interpretability of GP can be further improved.

8.4.4 Multi-objective GP Approach to Classification and Feature Selection with High-dimensional Unbalanced Data

EMO has been investigated to develop multi-objective GP methods for unbalanced classification [9,10,12,13], while most of them have not typically considered high-dimensional data. GP can simultaneously evolve classifiers and select informative features, while for high-dimensional data, the search space of feature selection is huge. It is necessary to investigate how EMO can be used with GP to develop new multi-objective GP approaches to classification and feature selection with high-dimensional unbalanced data.

8.4.5 Data-level GP Approaches to Unbalanced Classification

In the thesis, all of the designed GP methods are algorithm-level approaches, i.e. GP is used as a classification algorithm and is directly improved for use in unbalanced classification. It is acknowledged that sampling methods are very popular because they are not limited to a specific classification algorithm to address the class imbalance issue. However, for popular oversampling methods, e.g. SMOTE, Borderline-SMOTE and ADASYN, they use a pre-defined linear model structure to generate a synthetic instance for the minority class. GP has the ability to automatically evolve a model structure to generate synthetic instances. To date, there

is no existing work that attempts to use GP as a sampling method. In this regard, it is interesting to investigate how GP can be used as an oversampling method for unbalanced classification.

Bibliography

[1] ACHARYA, D., GOEL, S., ASTHANA, R., AND BHARDWAJ, A. A novel fitness function in genetic programming to handle unbalanced emotion recognition data. *Pattern Recognition Letters 133* (2020), 272–279.

[2] AL-MADI, N., FARIS, H., AND ABUKHURMA, R. Cost-sensitive genetic programming for churn prediction and identification of the influencing factors in telecommunication market. *International Journal of Advanced Science and Technology* (2018), 13–28.

[3] ALPAYDIN, E. *Introduction to machine learning*. MIT press, 2009.

[4] BÄCK, T., FOGEL, D. B., AND MICHALEWICZ, Z. *Evolutionary computation 1: Basic algorithms and operators*, vol. 1. CRC press, 2000.

[5] BAHNSEN, A. C., AOUADA, D., AND OTTERSTEN, B. Example-dependent cost-sensitive decision trees. *Expert Systems with Applications 42*, 19 (2015), 6609–6619.

[6] BEADLE, L., AND JOHNSON, C. G. Semantically driven mutation in genetic programming. In *2009 IEEE Congress on Evolutionary Computation* (2009), IEEE, pp. 1336–1342.

[7] BEYER, H.-G., AND SCHWEFEL, H.-P. Evolution strategies–a comprehensive introduction. *Natural computing 1*, 1 (2002), 3–52.

[8] BHATIA, N., ET AL. Survey of nearest neighbor techniques. *arXiv preprint arXiv:1007.0085* (2010).

[9] BHOWAN, U., JOHNSTON, M., AND ZHANG, M. Ensemble learning and pruning in multi-objective genetic programming for classification with unbalanced data. In *Australasian Joint Conference on Artificial Intelligence* (2011), Springer, pp. 192–202.

[10] BHOWAN, U., JOHNSTON, M., AND ZHANG, M. Evolving ensembles in multi-objective genetic programming for classification with unbalanced data. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation* (2011), ACM, pp. 1331–1338.

[11] BHOWAN, U., JOHNSTON, M., AND ZHANG, M. Developing new fitness functions in genetic programming for classification with unbalanced data. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 42*, 2 (2012), 406–421.

[12] BHOWAN, U., JOHNSTON, M., ZHANG, M., AND YAO, X. Evolving diverse ensembles using genetic programming for classification with unbalanced data. *IEEE Transactions on Evolutionary Computation 17*, 3 (2013), 368–386.

[13] BHOWAN, U., JOHNSTON, M., ZHANG, M., AND YAO, X. Reusing genetic programming for ensemble selection in classification of unbalanced data. *IEEE Transaction on Evolutionary Computation 18*, 6 (2014), 893–908.

[14] BHOWAN, U., ZHANG, M., AND JOHNSTON, M. Genetic programming for classification with unbalanced data. In *European Conference on Genetic Programming* (2010), Springer, pp. 1–13.

[15] BRUZZONE, L., ROLI, F., AND SERPICO, S. B. An extension of the jeffreys-matusita distance to multiclass cases for feature selection. *IEEE Transactions on Geoscience and Remote Sensing 33*, 6 (1995), 1318–1321.

[16] CHAWLA, N. V., BOWYER, K. W., HALL, L. O., AND KEGELMEYER, W. P. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research 16* (2002), 321–357.

[17] CHEN, L., AND GUO, G. Nearest neighbor classification of categorical data by attributes weighting. *Expert Systems with Applications 42*, 6 (2015), 3142–3149.

[18] CHEN, S., GUO, G., AND CHEN, L. A new over-sampling method based on cluster ensembles. In *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops* (2010), IEEE, pp. 599–604.

[19] CHEN, S., HE, H., AND GARCIA, E. A. Ramoboost: ranked minority oversampling in boosting. *IEEE Transactions on Neural Networks 21*, 10 (2010), 1624–1642.

[20] CIESIELSKI, V. Linear genetic programming. *Genetic Programming and Evolvable Machines 9*, 1 (2008), 105–106.

[21] COELLO, C. C. Evolutionary multi-objective optimization: a historical view of the field. *IEEE computational intelligence magazine 1*, 1 (2006), 28–36.

[22] CURRY, R., LICHODZIJEWSKI, P., AND HEYWOOD, M. I. Scaling genetic programming to large datasets using hierarchical dynamic subset selection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 37*, 4 (2007), 1065–1073.

[23] DASH, M., AND LIU, H. Consistency-based search in feature selection. *Artificial intelligence 151*, 1-2 (2003), 155–176.

[24] DEB, K., PRATAP, A., AGARWAL, S., AND MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation 6*, 2 (2002), 182–197.

[25] DEMIDOVA, L., AND KLYUEVA, I. Svm classification: Optimization with the smote algorithm for the class imbalance problem. In *2017 6th Mediterranean Conference on Embedded Computing (MECO)* (2017), IEEE, pp. 1–4.

[26] DENG, Z., ZHU, X., CHENG, D., ZONG, M., AND ZHANG, S. Efficient KNN classification algorithm for big data. *Neurocomputing 195* (2016), 143–148.

[27] DEVARRIYA, D., GULATI, C., MANSHARAMANI, V., SAKALLE, A., AND BHARDWAJ, A. Unbalanced breast cancer data classification using novel fitness functions in genetic programming. *Expert Systems with Applications 140* (2020), 112866.

[28] DEVI, D., BISWAS, S. K., AND PURKAYASTHA, B. Learning in presence of class imbalance and class overlapping by using one-class svm and undersampling technique. *Connection Science 31*, 2 (2019), 105–142.

[29] DEVI, D., PURKAYASTHA, B., ET AL. Redundancy-driven modified tomek-link based undersampling: a solution to class imbalance. *Pattern Recognition Letters 93* (2017), 3–12.

[30] DIETTERICH, T., DOMINGOS, P., MITCHELL, T., PAGE, D., AND SHAVLIK, J. Instance-based learning.

[31] DONG, S., AND WU, Y. A genetic algorithm-based approach for class-imbalanced learning. In *Third International Workshop on Pattern Recognition* (2018), vol. 10828, International Society for Optics and Photonics, p. 108281D.

[32] DORIGO, M., BIRATTARI, M., AND STUTZLE, T. Ant colony optimization. *IEEE computational intelligence magazine 1*, 4 (2006), 28–39.

[33] DOUCETTE, J., AND HEYWOOD, M. I. GP classification under imbalanced data sets: Active sub-sampling and auc approximation. In *European Conference on Genetic Programming* (2008), Springer, pp. 266–277.

[34] DROWN, D. J., KHOSHGOFTAAR, T. M., AND NARAYANAN, R. Using evolutionary sampling to mine imbalanced data. In *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)* (2007), IEEE, pp. 363–368.

[35] ELKAN, C. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence* (2001), vol. 17, Lawrence Erlbaum Associates Ltd, pp. 973–978.

[36] ERTEKIN, S., HUANG, J., AND GILES, C. L. Active learning for class imbalance problem. In *SIGIR* (2007), vol. 7, pp. 823–824.

[37] ESPEJO, P. G., VENTURA, S., AND HERRERA, F. A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 40*, 2 (2010), 121–144.

[38] FERNÁNDEZ, A., GARCÍA, S., GALAR, M., PRATI, R. C., KRAWCZYK, B., AND HERRERA, F. Cost-sensitive learning. In *Learning from Imbalanced Data Sets*. Springer, 2018, pp. 63–78.

[39] FERNÁNDEZ, A., GARCÍA, S., GALAR, M., PRATI, R. C., KRAWCZYK, B., AND HERRERA, F. *Learning from imbalanced data sets*, vol. 11. Springer, 2018.

[40] FISHBURN, P. C. Continua of stochastic dominance relations for bounded probability distributions. *Journal of Mathematical Economics 3*, 3 (1976), 295–311.

[41] FISHER, R. A. Statistical methods for research workers. In *Breakthroughs in statistics*. Springer, 1992, pp. 66–70.

[42] FLEURET, F. Fast binary feature selection with conditional mutual information. *Journal of Machine Learning Research 5*, Nov (2004), 1531–1555.

[43] FOGEL, D. B., AND FOGEL, L. J. An introduction to evolutionary programming. In *European Conference on Artificial Evolution* (1995), Springer, pp. 21–33.

[44] FOGEL, L. J. *Intelligence through simulated evolution: forty years of evolutionary programming*. John Wiley & Sons, Inc., 1999.

[45] GALAR, M., FERNANDEZ, A., BARRENECHEA, E., BUSTINCE, H., AND HERRERA, F. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42*, 4 (2012), 463–484.

[46] GALAR, M., FERNÁNDEZ, A., BARRENECHEA, E., AND HERRERA, F. Eusboost: Enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling. *Pattern recognition 46*, 12 (2013), 3460–3471.

[47] GÁMEZ, J. A., PUERTA, J. M., ET AL. Migration of probability models instead of individuals: An alternative when applying the island model to edas. In *International Conference on Parallel Problem Solving from Nature* (2004), Springer, pp. 242–252.

[48] GAO, L., YANG, L., AREFAN, D., AND WU, S. One-class classification for highly imbalanced medical image data. In *Medical Imaging 2020: Imaging Informatics for Healthcare, Research, and Applications* (2020), vol. 11318, International Society for Optics and Photonics, p. 113181C.

[49] GARCÍA, S., AND HERRERA, F. Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy. *Evolutionary computation 17*, 3 (2009), 275–306.

[50] GARCÍA, V., SÁNCHEZ, J., AND MOLLINEDA, R. An empirical study of the behavior of classifiers on imbalanced and overlapped data sets. In *Iberoamerican Congress on Pattern Recognition* (2007), Springer, pp. 397–406.

[51] GARNIER, S., GAUTRAIS, J., AND THERAULAZ, G. The biological principles of swarm intelligence. *Swarm Intelligence 1*, 1 (2007), 3–31.

[52] GATHERCOLE, C., AND ROSS, P. Dynamic training subset selection for supervised learning in genetic programming. In *International Conference on Parallel Problem Solving from Nature* (1994), Springer, pp. 312–321.

[53] GU, B., SHENG, V. S., AND LI, S. Bi-parameter space partition for cost-sensitive svm. In *Twenty-Fourth International Joint Conference on Artificial Intelligence* (2015).

[54] GUNN, S. R., ET AL. Support vector machines for classification and regression. *ISIS technical report 14*, 1 (1998), 5–16.

[55] GUYON, I., AND ELISSEEFF, A. An introduction to variable and feature selection. *Journal of machine learning research 3*, Mar (2003), 1157–1182.

[56] HAESELEER D, P. Context preserving crossover in genetic programming. In *IEEE World Congress on Computational Intelligence* (1994), IEEE, pp. 256–261.

[57] HAIXIANG, G., YIJING, L., SHANG, J., MINGYUN, G., YUANYUE, H., AND BING, G. Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications 73* (2017), 220–239.

[58] HALL, M. A. Correlation-based feature selection for machine learning.

[59] HAMIDA, S. B., HMIDA, H., BORGI, A., AND RUKOZ, M. Adaptive sampling for active learning with genetic programming. *Cognitive Systems Research 65* (2021), 23–39.

[60] HAN, H., WANG, W.-Y., AND MAO, B.-H. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing* (2005), Springer, pp. 878–887.

[61] HARDING, S. L., MILLER, J. F., AND BANZHAF, W. Self-modifying cartesian genetic programming. In *Cartesian Genetic Programming*. Springer, 2011, pp. 101–124.

[62] HE, H., BAI, Y., GARCIA, E. A., AND LI, S. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *IEEE International Joint Conference on Computational Intelligence* (2008), IEEE, pp. 1322–1328.

[63] HE, H., AND GARCIA, E. A. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 9 (2009), 1263–1284.

[64] HOFMANN, T., SCHÖLKOPF, B., AND SMOLA, A. J. Kernel methods in machine learning. *The annals of statistics* (2008), 1171–1220.

[65] HOLLAND, J. H., ET AL. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[66] HORNIK, K., STINCHCOMBE, M., AND WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks 2*, 5 (1989), 359–366.

[67] HUNT, R., JOHNSTON, M., BROWNE, W., AND ZHANG, M. Sampling methods in genetic programming for classification with unbalanced data. In *Australasian Joint Conference on Artificial Intelligence* (2010), Springer, pp. 273–282.

[68] HUNT, R., NESHATIAN, K., AND ZHANG, M. A genetic programming approach to hyper-heuristic feature selection. In *Asia-Pacific Conference on Simulated Evolution and Learning* (2012), Springer, pp. 320–330.

[69] IRANMEHR, A., MASNADI-SHIRAZI, H., AND VASCONCELOS, N. Cost-sensitive support vector machines. *Neurocomputing 343* (2019), 50–64.

[70] JIANG, K., LU, J., AND XIA, K. A novel algorithm for imbalance data classification based on genetic algorithm improved smote. *Arabian journal for science and engineering 41*, 8 (2016), 3255–3266.

[71] JOSHI, A., DANGRA, J., AND RAWAT, M. A decision tree based classification technique for accurate heart disease classification and prediction. *International Journal of Technology Research and Management 3* (2016), 1–4.

[72] KENNEDY, J., AND EBERHART, R. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks* (1995), vol. 4, IEEE, pp. 1942–1948.

[73] KEPHART, J. O. A biologically inspired immune system for computers. In *In proc. Of the fourth international workshop on synthesis and simulation of living systems, artificial life IV* (1994), Citeseer.

[74] KHAN, S. S., AND MADDEN, M. G. One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review 29*, 3 (2014), 345–374.

[75] KHANCHI, S., HEYWOOD, M. I., AND ZINCIR-HEYWOOD, A. N. Properties of a gp active learning framework for streaming data with class imbalance. In *Proceedings of the genetic and evolutionary computation conference* (2017), pp. 945–952.

[76] KHANCHI, S., VAHDAT, A., HEYWOOD, M. I., AND ZINCIR-HEYWOOD, A. N. On botnet detection with genetic programming under streaming data label budgets and class imbalance. *Swarm and evolutionary computation 39* (2018), 123–140.

[77] KIM, K., SHAN, Y., NGUYEN, X. H., AND MCKAY, R. I. Probabilistic model building in genetic programming: a critical review. *Genetic Programming and Evolvable Machines 15*, 2 (2014), 115–167.

[78] KNOWLES, J., AND CORNE, D. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Proceedings of the 1999 Congress on Evolutionary Computation* (1999), vol. 1, IEEE, pp. 98–105.

[79] KOHAVI, R., ET AL. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conferences on Artificial Intelligence* (1995), vol. 14, Montreal, Canada, pp. 1137–1145.

[80] KONONENKO, I. Semi-naive bayesian classifier. In *European Working Session on Learning* (1991), Springer, pp. 206–219.

[81] KOVASHKA, A. Classification: Nearest neighbors.

[82] KOZA, J. R., AND KOZA, J. R. *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[83] 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.

[84] KRISHNAKUMAR, A., AMRITA, D., AND PRIYA, N. S. Mining association rules between sets of items in large databases. *International Journal of Science and Modern Engineering* (2013), 2319–6386.

[85] LANGDON, W. B. The evolution of size in variable length representations. In *IEEE World Congress on Computational Intelligencen* (1998), IEEE, pp. 633–638.

[86] LANGDON, W. B. Size fair and homologous tree crossovers for tree genetic programming. *Genetic programming and evolvable machines 1*, 1-2 (2000), 95–119.

[87] LARRAÑAGA, P., AND BIELZA, C. Estimation of distribution algorithm (EDA). *Dictionary of Bioinformatics and Computational Biology* (2004).

[88] LARRAÑAGA, P., AND LOZANO, J. A. *Estimation of distribution algorithms: A new tool for evolutionary computation*, vol. 2. Springer Science & Business Media, 2001.

[89] LAST, M., MAIMON, O., AND MINKOV, E. Improving stability of decision trees. *International Journal of Pattern Recognition and Artificial Intelligence 16*, 2 (2002), 145–159.

[90] LEE, H. K., AND KIM, S. B. An overlap-sensitive margin classifier for imbalanced and overlapping data. *Expert Systems with Applications 98* (2018), 72–83.

[91] LI, F., ZHANG, X., ZHANG, X., DU, C., XU, Y., AND TIAN, Y.-C. Cost-sensitive and hybrid-attribute measure multi-decision tree over imbalanced data sets. *Information Sciences 422* (2018), 242–256.

[92] LI, H., ZHANG, L., HUANG, B., AND ZHOU, X. Cost-sensitive dual-bidirectional linear discriminant analysis. *Information Sciences 510* (2020), 283–303.

[93] LI, J., LI, X., AND YAO, X. Cost-sensitive classification with genetic programming. In *The 2005 IEEE Congress on Evolutionary Computation* (2005), vol. 3, IEEE, pp. 2114–2121.

[94] LIN, C.-F., AND WANG, S.-D. Fuzzy support vector machines. *IEEE transactions on neural networks 13*, 2 (2002), 464–471.

[95] LIN, W.-C., TSAI, C.-F., HU, Y.-H., AND JHANG, J.-S. Clustering-based undersampling in class-imbalanced data. *Information Sciences 409* (2017), 17–26.

[96] LIU, F., PEDRYCZ, W., AND LIU, X.-W. Flexibility degree of fuzzy numbers and its implication to a group-decision-making model. *IEEE Transactions on Cybernetics* (2018).

[97] LIU, H., SETIONO, R., ET AL. A probabilistic approach to feature selection- A filter solution. In *ICML* (1996), vol. 96, Citeseer, pp. 319–327.

[98] LIU, H., AND YU, L. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on knowledge and data engineering 17*, 4 (2005), 491–502.

[99] LIU, T., MOORE, A. W., AND GRAY, A. New algorithms for efficient high-dimensional nonparametric classification. *Journal of Machine Learning Research 7*, Jun (2006), 1135–1158.

[100] LIU, X.-Y., WU, J., AND ZHOU, Z.-H. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 39*, 2 (2008), 539–550.

[101] LIU, X.-Y., AND ZHOU, Z.-H. The influence of class imbalance on cost-sensitive learning: An empirical study. In *Sixth International Conference on Data Mining (ICDM'06)* (2006), IEEE, pp. 970–974.

[102] LIU, X.-Y., AND ZHOU, Z.-H. Learning with cost intervals. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining* (2010), ACM, pp. 403–412.

[103] LIU, X.-Y., AND ZHOU, Z.-H. Towards cost-sensitive learning for real-world applications. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2011), Springer, pp. 494–505.

[104] LÓPEZ, V., FERNÁNDEZ, A., DEL JESUS, M. J., AND HERRERA, F. A hierarchical genetic fuzzy system based on genetic programming for addressing classification with highly imbalanced and borderline data-sets. *Knowledge-Based Systems 38* (2013), 85–104.

[105] LÓPEZ, V., FERNÁNDEZ, A., GARCÍA, S., PALADE, V., AND HERRERA, F. An insight into classification with imbalanced data: Empirical results

and current trends on using data intrinsic characteristics. *Information sciences 250* (2013), 113–141.

[106] LU, W., LI, Z., AND CHU, J. Adaptive ensemble undersampling-boost: a novel learning framework for imbalanced data. *Journal of systems and software 132* (2017), 272–282.

[107] LUKE, S., AND PANAIT, L. Lexicographic parsimony pressure. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation* (2002), pp. 829–836.

[108] LUNA, J. M., PECHENIZKIY, M., AND VENTURA, S. Mining exceptional relationships with grammar-guided genetic programming. *Knowledge and Information Systems 47*, 3 (2016), 571–594.

[109] LUSA, L., ET AL. Evaluation of smote for high-dimensional class-imbalanced microarray data. In *2012 11th International Conference on Machine Learning and Applications* (2012), vol. 2, IEEE, pp. 89–94.

[110] LUSA, L., ET AL. Smote for high-dimensional class-imbalanced data. *BMC bioinformatics 14*, 1 (2013), 106.

[111] MAHANIPOUR, A., NEZAMABADI-POUR, H., AND NIKPOUR, B. Using fuzzy-rough set feature selection for feature construction based on genetic programming. In *Conference on Swarm Intelligence and Evolutionary Computation (CSIEC)* (2018), IEEE.

[112] MAUŠA, G., AND GRBAC, T. G. Co-evolutionary multi-population genetic programming for classification in software defect prediction: An empirical case study. *Applied soft computing 55* (2017), 331–351.

[113] MESTER, D., AND BRÄYSY, O. Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers and Operations Research 32*, 6 (2005), 1593–1614.

[114] MICHALEWICZ, Z. Evolution strategies and other methods. In *Genetic algorithms+ data structures= evolution programs*. Springer, 1996, pp. 159–177.

[115] MICHALSKI, R. S., CARBONELL, J. G., AND MITCHELL, T. M. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[116] MILLER, J., AND TURNER, A. Cartesian genetic programming. In *Proceedings of the Companion Publication of the Annual Conference on Genetic and Evolutionary Computation* (2015), ACM, pp. 179–198.

[117] MITRA, P., MURTHY, C., AND PAL, S. K. Unsupervised feature selection using feature similarity. *IEEE transactions on pattern analysis and machine intelligence 24*, 3 (2002), 301–312.

[118] MONTANA, D. J. Strongly typed genetic programming. *Evolutionary computation 3*, 2 (1995), 199–230.

[119] MUJA, M., AND LOWE, D. G. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence 36*, 11 (2014), 2227–2240.

[120] MUNI, D. P., PAL, N. R., AND DAS, J. Genetic programming for simultaneous feature selection and classifier design.

[121] MURPHY, K. P. Naive bayes classifiers. *University of British Columbia 18* (2006).

[122] NAG, K., AND PAL, N. R. A multiobjective genetic programming-based ensemble for simultaneous feature selection and classification. *IEEE transactions on cybernetics 46*, 2 (2016), 499–510.

[123] NARENDRA, P. M., AND FUKUNAGA, K. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers 26*, 09 (1977), 917–922.

[124] NESHATIAN, K., AND ZHANG, M. Dimensionality reduction in face detection: A genetic programming approach. In *24th International Conference on Image and Vision Computing New Zealand* (2009), IEEE, pp. 391–396.

[125] 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* (2009), ACM, pp. 1027–1034.

[126] NESHATIAN, K., ZHANG, M., AND ANDREAE, P. A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming. *IEEE Transactions on Evolutionary Computation 16*, 5 (2012), 645–661.

[127] NESHATIAN, K., ZHANG, M., AND JOHNSTON, M. Feature construction and dimension reduction using genetic programming. In *Australasian Joint Conference on Artificial Intelligence* (2007), Springer, pp. 160–170.

[128] NEYMAN, J. On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. In *Breakthroughs in Statistics*. Springer, 1992, pp. 123–150.

[129] OFEK, N., ROKACH, L., STERN, R., AND SHABTAI, A. Fast-cbus: A fast clustering-based undersampling method for addressing the class imbalance problem. *Neurocomputing 243* (2017), 88–102.

[130] OLSON, D. L., AND DELEN, D. *Advanced data mining techniques*. Springer Science & Business Media, 2008.

[131] OMOHUNDRO, S. M. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[132] OTERO, F. E., SILVA, M. M., FREITAS, A. A., AND NIEVOLA, J. C. Genetic programming for attribute construction in data mining. In *European Conference on Genetic Programming* (2003), Springer, pp. 384–393.

[133] PADURARIU, C., AND BREABAN, M. E. Dealing with data imbalance in text classification. *Procedia Computer Science 159* (2019), 736–745.

[134] PARSONS, L., HAQUE, E., AND LIU, H. Subspace clustering for high dimensional data: a review. *Acm sigkdd explorations newsletter 6*, 1 (2004), 90–105.

[135] PATTERSON, G., AND ZHANG, M. Fitness functions in genetic programming for classification with unbalanced data. In *Australasian Joint Conference on Artificial Intelligence* (2007), Springer, pp. 769–775.

[136] PAWLAK, Z., AND SKOWRON, A. Rough sets: some extensions. *Information sciences 177*, 1 (2007), 28–40.

[137] PAWLAK, Z., AND SKOWRON, A. Rudiments of rough sets. *Information sciences 177*, 1 (2007), 3–27.

[138] PEI, W., LIN, H., AND LI, L. Optimal-neighborhood statistics rough set approach with multiple attributes and criteria. In *International Conference on Rough Sets and Knowledge Technology* (2014), Springer, pp. 683–692.

[139] PEI, W., XUE, B., SHANG, L., AND ZHANG, M. New fitness functions in genetic programming for classification with high-dimensional unbalanced data. In *2019 IEEE Congress on Evolutionary Computation (CEC)* (2019), IEEE, pp. 2779–2786.

[140] PEI, W., XUE, B., ZHANG, M., AND SHANG, L. A cost-sensitive genetic programming approach for high-dimensional unbalanced classification. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)* (2019), IEEE, pp. 1770–1777.

[141] PERKIS, T. Stack-based genetic programming. In *IEEE World Congress on Computational Intelligence* (1994), IEEE, pp. 148–153.

[142] PERRY, T., BADER-EL-DEN, M., AND COOPER, S. Imbalanced classification using genetically optimized cost sensitive classifiers. In *2015 IEEE Congress on Evolutionary Computation (CEC)* (2015), IEEE, pp. 680–687.

[143] POLI, R., LANGDON, W., AND MCPHEE, N. *A field guide to genetic programming*. 2008.

[144] POLI, R., PAGE, J., AND LANGDON, W. B. Smooth uniform crossover, sub-machine code gp and demes: A recipe for solving high-order boolean parity problems. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation* (1999), vol. 2, Morgan Kaufmann Publishers Inc., pp. 1162–1169.

[145] POWERS, D. M. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies* (2011), 37–62.

[146] PRATI, R. C., BATISTA, G. E., AND MONARD, M. C. Class imbalances versus class overlapping: an analysis of a learning system behavior. In *Mexican international conference on artificial intelligence* (2004), Springer, pp. 312–321.

[147] PRICE, K. V. Differential evolution: a fast and simple numerical optimizer. In *Biennial Conference of the North American on Fuzzy Information Processing Society (NAFIPS)* (1996), IEEE, pp. 524–527.

[148] PUROHIT, A., CHAUDHARI, N. S., AND TIWARI, A. Construction of classifier with feature selection based on genetic programming. In *IEEE Congress on Evolutionary Computation* (2010), IEEE, pp. 1–5.

[149] QUINLAN, J. R. Induction of decision trees. *Machine learning 1*, 1 (1986), 81–106.

[150] QUINLAN, J. R. Improved use of continuous attributes in C4. 5. *Journal of artificial intelligence research 4* (1996), 77–90.

[151] RAMENTOL, E., CABALLERO, Y., BELLO, R., AND HERRERA, F. Smote-rsb*: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using smote and rough sets theory. *Knowledge and information systems 33*, 2 (2012), 245–265.

[152] RATLE, A., AND SEBAG, M. Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. In *International Conference on Parallel Problem Solving from Nature* (2000), Springer, pp. 211–220.

[153] RECHENBERG, I. Evolution strategy: Nature's way of optimization. In *Optimization: Methods and applications, possibilities and limitations*. Springer, 1989, pp. 106–126.

[154] REN, F., CAO, P., LI, W., ZHAO, D., AND ZAIANE, O. Ensemble based adaptive over-sampling method for imbalanced data learning in computer aided detection of microaneurysm. *Computerized Medical Imaging and Graphics 55* (2017), 54–67.

[155] REY HORN, J., NAFPLIOTIS, N., AND GOLDBERG, D. E. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the first IEEE conference on evolutionary computation, IEEE world congress on computational intelligence* (1994), vol. 1, Citeseer, pp. 82–87.

[156] RUSSELL, S., AND NORVIG, P. Artificial intelligence: A modern approach (second edition).

[157] SÁEZ, J. A., GALAR, M., AND KRAWCZYK, B. Addressing the overlapping data problem in classification using the one-vs-one decomposition strategy. *IEEE Access 7* (2019), 83396–83411.

[158] SAFAVIAN, S. R., AND LANDGREBE, D. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics 21*, 3 (1991), 660–674.

[159] SALUSTOWICZ, R., AND SCHMIDHUBER, J. Probabilistic incremental program evolution. *Evolutionary Computation 5*, 2 (1997), 123–141.

[160] SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of research and development 3*, 3 (1959), 210–229.

[161] SANTOS, M. S., ABREU, P. H., GARCÍA-LAENCINA, P. J., SIMÃO, A., AND CARVALHO, A. A new cluster-based oversampling method for improving survival prediction of hepatocellular carcinoma patients. *Journal of biomedical informatics 58* (2015), 49–59.

[162] SASTRY, K., AND GOLDBERG, D. E. Probabilistic model building and competent genetic programming. In *Genetic programming theory and practice*. Springer, 2003, pp. 205–220.

[163] SAVIC, D. A., WALTERS, G. A., AND DAVIDSON, J. W. A genetic programming approach to rainfall-runoff modelling. *Water resources management 13*, 3 (1999), 219–231.

[164] SETTLES, B. Active learning literature survey.

[165] SHAKYA, S., MCCALL, J., AND BROWN, D. Updating the probability vector using mrf technique for a univariate EDA. In *Proceedings of the Second Starting AI Researchers' Symposium* (2004), vol. 109, pp. 15–25.

[166] SHAKYA, S., MCCALL, J., AND BROWN, D. Estimating the distribution in an EDA. In *Adaptive and Natural Computing Algorithms*. Springer, 2005, pp. 202–205.

[167] SHEN, H., AND CAO, J. Imbalanced research of deep belief network based on dynamic cost sensitive. In *Proceedings of the 2019 5th International Conference on Computing and Data Engineering* (2019), pp. 15–19.

[168] SHIN, H. J., EOM, D.-H., AND KIM, S.-S. One-class support vector machines—an application in machine fault detection and classification. *Computers & Industrial Engineering 48*, 2 (2005), 395–408.

[169] 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.

[170] SOK, H. K., OOI, M. P.-L., KUANG, Y. C., AND DEMIDENKO, S. Multivariate alternating decision trees. *Pattern Recognition 50* (2016), 195–209.

[171] SONG, D., HEYWOOD, M. I., AND ZINCIR-HEYWOOD, A. N. A linear genetic programming approach to intrusion detection. In *Genetic and Evolutionary Computation Conference* (2003), Springer, pp. 2325–2336.

[172] SOUFAN, O., KLEFTOGIANNIS, D., KALNIS, P., AND BAJIC, V. B. DWFS: a wrapper feature selection tool based on a parallel genetic algorithm. *PloS one 10*, 2 (2015), e0117988.

[173] STAŃCZYK, U. Ranking of characteristic features in combined wrapper approaches to selection. *Neural Computing and Applications 26*, 2 (2015), 329–344.

[174] STAPLETON, F., AND GALVÁN, E. Semantic neighborhood ordering in multi-objective genetic programming based on decomposition. *arXiv preprint arXiv:2103.00480* (2021).

[175] STEFANOWSKI, J. Overlapping, rare examples and class decomposition in learning classifiers from imbalanced data. In *Emerging paradigms in machine learning*. Springer, 2013, pp. 277–306.

[176] STEFANOWSKI, J. Dealing with data difficulty factors while learning from imbalanced data. In *Challenges in computational statistics and data mining*. Springer, 2016, pp. 333–363.

[177] STORN, R., AND PRICE, K. Differential evolution- A simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization 11*, 4 (1997), 341–359.

[178] SUBUDHI, S., PATRO, R. N., AND BISWAL, P. K. Pso-based synthetic minority oversampling technique for classification of reduced hyperspectral image. In *Soft computing for problem solving*. Springer, 2019, pp. 617–625.

[179] SUMATHI, S., AND SIVANANDAM, S. *Introduction to data mining and its applications*, vol. 29. Springer, 2006.

[180] SUTTON, R. S., AND BARTO, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

[181] TANG, J., ALELYANI, S., AND LIU, H. Feature selection for classification: A review. *Data Classification: Algorithms and Applications* (2014), 37.

[182] TANG, W., MAO, K., MAK, L. O., AND NG, G. W. Classification for overlapping classes using optimized overlapping region detection and soft decision. In *2010 13th International Conference on Information Fusion* (2010), IEEE, pp. 1–8.

[183] TARIQ, H., ELDRIDGE, E., AND WELCH, I. An efficient approach for feature construction of high-dimensional microarray data by random projections. *PloS one 13*, 4 (2018), e0196385.

[184] THAI-NGHE, N., GANTNER, Z., AND SCHMIDT-THIEME, L. Cost-sensitive learning methods for imbalanced data. In *The 2010 International Joint Conference on Neural Networks* (2010), IEEE, pp. 1–8.

[185] THEARLING, K. An introduction to data mining. 2017.

[186] TIBSHIRANI, R., HASTIE, T., NARASIMHAN, B., AND CHU, G. Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proceedings of the National Academy of Sciences 99*, 10 (2002), 6567–6572.

[187] TING, K. M. An instance-weighting method to induce cost-sensitive trees. *IEEE Transactions on Knowledge and Data Engineering 14*, 3 (2002), 659–665.

[188] TRAN, B., XUE, B., AND ZHANG, M. Genetic programming for feature construction and selection in classification on high-dimensional data. *Memetic Computing 8*, 1 (2016), 3–15.

[189] TRAN, B., XUE, B., AND ZHANG, M. Class dependent multiple feature construction using genetic programming for high-dimensional data. In *Australasian Joint Conference on Artificial Intelligence* (2017), Springer, pp. 182–194.

[190] TRAN, B., XUE, B., AND ZHANG, M. Using feature clustering for GP-based feature construction on high-dimensional data. In *European Conference on Genetic Programming* (2017), Springer, pp. 210–226.

[191] TRAN, B., XUE, B., AND ZHANG, M. Genetic programming for multiple-feature construction on high-dimensional classification. *Pattern Recognition 93* (2019), 404–417.

[192] TRAN, B., ZHANG, M., AND XUE, B. Multiple feature construction in classification on high-dimensional data using gp. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)* (2016), IEEE, pp. 1–8.

[193] TRAN, B. N. Evolutionary computation for feature manipulation in classification on high-dimensional data.

[194] TU, J. V. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of clinical epidemiology 49*, 11 (1996), 1225–1231.

[195] VUTTIPITTAYAMONGKOL, P., AND ELYAN, E. Neighbourhood-based undersampling approach for handling imbalanced and overlapped data. *Information Sciences 509* (2020), 47–70.

[196] WALKER, J. A., AND MILLER, J. F. The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation 12*, 4 (2008), 397–417.

[197] WANG, J. *Data mining: opportunities and challenges*. IGI Global, 2003.

[198] WANG, Y., CAI, Z., AND ZHANG, Q. Differential evolution with composite trial vector generation strategies and control parameters. *IEEE transactions on evolutionary computation 15*, 1 (2011), 55–66.

[199] WEI, G. Study on genetic algorithm and evolutionary programming. In *2012 IEEE International Conference on Parallel Distributed and Grid Computing (PDGC)* (2012), IEEE, pp. 762–766.

[200] WHITLEY, D. A genetic algorithm tutorial. *Statistics and computing 4*, 2 (1994), 65–85.

[201] WONG, M. L., AND LEUNG, K. S. *Data mining using grammar based genetic programming and applications*, vol. 3. Springer Science & Business Media, 2006.

[202] WU, X., KUMAR, V., QUINLAN, J. R., GHOSH, J., YANG, Q., MOTODA, H., MCLACHLAN, G. J., NG, A., LIU, B., PHILIP, S. Y., ET AL. Top 10 algorithms in data mining. *Knowledge and information systems 14*, 1 (2008), 1–37.

[203] XIONG, H., WU, J., AND LIU, L. Classification with classoverlapping: A systematic study. In *Proceedings of the 1st International Conference on E-Business Intelligence (ICEBI2010),* (2010), Atlantis Press.

[204] XUE, B., ZHANG, M., BROWNE, W. N., AND YAO, X. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation 20*, 4 (2015), 606–626.

[205] YAN, L., DODIER, R. H., MOZER, M., AND WOLNIEWICZ, R. H. Optimizing classifier performance via an approximation to the wilcoxon-mann-whitney statistic. In *Proceedings of the 20th international conference on machine learning (icml-03)* (2003), pp. 848–855.

[206] YANAI, K., AND IBA, H. Estimation of distribution programming: EDA-based approach to program generation. In *Towards a New Evolutionary Computation*. Springer, 2006, pp. 103–122.

[207] YAO, Y. Relational interpretations of neighborhood operators and rough set approximation operators. *Information sciences 111*, 1-4 (1998), 239–259.

[208] YU, L., AND LIU, H. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th international conference on machine learning (ICML)* (2003), pp. 856–863.

[209] ZHANG, C., TAN, K. C., LI, H., AND HONG, G. S. A cost-sensitive deep belief network for imbalanced classification. *IEEE transactions on neural networks and learning systems 30*, 1 (2018), 109–122.

[210] ZHANG, C., TAN, K. C., AND REN, R. Training cost-sensitive deep belief networks on imbalance data problems. In *2016 international joint conference on neural networks (IJCNN)* (2016), IEEE, pp. 4362–4367.

[211] ZHANG, G. P. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 30*, 4 (2000), 451–462.

[212] ZHANG, J., LI, T., RUAN, D., AND LIU, D. Neighborhood rough sets for dynamic data mining. *International Journal of Intelligent Systems 27*, 4 (2012), 317–342.

[213] ZHANG, L., AND ZHANG, D. Evolutionary cost-sensitive extreme learning machine. *IEEE transactions on neural networks and learning systems 28*, 12 (2016), 3045–3060.

[214] ZHANG, Q., AND LI, H. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation 11*, 6 (2007), 712–731.

[215] ZHANG, S., LI, X., ZONG, M., ZHU, X., AND WANG, R. Efficient knn classification with different numbers of nearest neighbors. *IEEE Transactions on Neural Networks and Learning Systems 29*, 5 (2018), 1774–1785.

[216] ZHANG, X., WANG, D., ZHOU, Y., CHEN, H., CHENG, F., AND LIU, M. Kernel modified optimal margin distribution machine for imbalanced data classification. *Pattern Recognition Letters 125* (2019), 325–332.

[217] ZHANG, Y., AND ZHOU, Z.-H. Cost-sensitive face recognition. *IEEE transactions on pattern analysis and machine intelligence 32*, 10 (2009), 1758–1769.

[218] ZHOU, Z. H. Cost-sensitive learning. In *International Conference on Modeling Decisions for Artificial Intelligence* (2011), Springer, pp. 17–18.

[219] ZHOU, Z.-H., AND LIU, X.-Y. On multi-class cost-sensitive learning. *Computational Intelligence 26*, 3 (2010), 232–257.

[220] ZHU, J., WANG, H., TSOU, B. K., AND MA, M. Active learning with sampling by uncertainty and density for data annotations. *IEEE Transactions on audio, speech, and language processing 18*, 6 (2009), 1323–1331.

[221] ZHU, Z., ONG, Y.-S., AND DASH, M. Markov blanket-embedded genetic algorithm for gene selection. *Pattern Recognition 40*, 11 (2007), 3236–3248.

[222] ZIKEBA, M., AND TOMCZAK, J. M. Boosted svm with active learning strategy for imbalanced data. *Soft Computing 19*, 12 (2015), 3357–3368.

[223] ZITZLER, E., LAUMANNS, M., AND THIELE, L. SPEA2: Improving the strength pareto evolutionary algorithm. *TIK-report 103* (2001).

[224] ZITZLER, E., AND THIELE, L. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation 3*, 4 (1999), 257–271.

Appendix

Theorem 1. Given U, $Maj \cup Min = U$ and $Maj \cap Min = \varnothing$, $\forall x \in Bn_n(Maj)$, so $x \in Bn_n(Min)$.

Proof: $\because x \in Bn_n(Maj)$

$\therefore \sigma(x) \cap Maj \neq \varnothing$ and $\sigma(x) \subsetneqq Maj$

① $\sigma(x) \cap Maj \neq \varnothing \Rightarrow \sigma(x) \cap Min \neq \varnothing$ (This is proved by contradiction.)

We assume $\sigma(x) \cap Min = \varnothing$.

$\because Maj \cap Min = \varnothing \therefore$ if $\sigma(x) \cap Min = \varnothing$, then $\sigma(x) \subseteq Maj$, which is contradicted with $\sigma(x) \subsetneqq Maj$, so the assumption does not hold.

Therefore, $\sigma(x) \cap Min \neq \varnothing$.

② $\sigma(x) \subsetneqq Maj \Rightarrow \sigma(x) \subsetneqq Min$ (This is proved by contradiction.)

We assume $\sigma(x) \subseteq Min$.

$\because Maj \cap Min = \varnothing \therefore$ if $\sigma(x) \subseteq Min$, then $\sigma(x) \cap Maj = \varnothing$, which is contradicted with $\sigma(x) \cap Maj \neq \varnothing$, so the assumption does not hold.

Therefore, $\sigma(x) \subsetneqq Min$.

Based on ① and ②, $\sigma(x) \cap Min \neq \varnothing$ and $\sigma(x) \subsetneqq Min$, so $x \in Bn_n(Min)$.

□