Genetic Programming for Classification with Unbalanced Data

by

Urvesh Bhowan

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
2012

Abstract

In classification, machine learning algorithms can suffer a performance bias when data sets are unbalanced. Binary data sets are unbalanced when one class is represented by only a small number of training examples (called the *minority* class), while the other class makes up the rest (*majority* class). In this scenario, the induced classifiers typically have high accuracy on the majority class but poor accuracy on the minority class. As the minority class typically represents the main class-of-interest in many real-world problems, accurately classifying examples from this class can be at least as important as, and in some cases more important than, accurately classifying examples from the majority class.

Genetic Programming (GP) is a promising machine learning technique based on the principles of Darwinian evolution to automatically evolve computer programs to solve problems. While GP has shown much success in evolving reliable and accurate classifiers for typical classification tasks with balanced data, GP, like many other learning algorithms, can evolve biased classifiers when data is unbalanced. This is because traditional training criteria such as the overall success rate in the fitness function in GP, can be influenced by the larger number of examples from the majority class.

This thesis proposes a GP approach to classification with unbalanced data. The goal is to develop new internal cost-adjustment techniques in GP to improve classification performances on both the minority class and the majority class. By focusing on internal cost-adjustment within GP rather than the traditional data-balancing techniques, the unbalanced data can be used directly or "as is" in the learning process. This removes any dependence on a sampling algorithm to first artificially re-balance the input data prior to the learning process.

This thesis shows that by developing a number of new methods in GP, genetic program classifiers with good classification ability on the minority and the majority classes can be evolved. This thesis evaluates these methods on a range of binary benchmark classification tasks with unbalanced data.

This thesis demonstrates that unlike tasks with multiple balanced classes where some dynamic (non-static) classification strategies perform significantly better than the simple static classification strategy, either a static or dynamic

strategy shows no significant difference in the performance of evolved GP classifiers on these binary tasks. For this reason, the rest of the thesis uses this static classification strategy.

This thesis proposes several new fitness functions in GP to perform cost adjustment between the minority and the majority classes, allowing the unbalanced data sets to be used directly in the learning process without sampling. Using the Area under the Receiver Operating Characteristics (ROC) curve (also known as the AUC) to measure how well a classifier performs on the minority and majority classes, these new fitness functions find genetic program classifiers with high AUC on the tasks on both classes, and with fast GP training times. These GP methods outperform two popular learning algorithms, namely, Naive Bayes and Support Vector Machines on the tasks, particularly when the level of class imbalance is large, where both algorithms show biased classification performances.

This thesis also proposes a multi-objective GP (MOGP) approach which treats the accuracies of the minority and majority classes separately in the learning process. The MOGP approach evolves a good set of trade-off solutions (a *Pareto* front) in a single run that perform as well as, and in some cases better than, multiple runs of canonical single-objective GP (SGP). In SGP, individual genetic program solutions capture the performance trade-off between the two objectives (minority and majority class accuracy) using an ROC curve; whereas in MOGP, this requirement is delegated to multiple genetic program solutions along the Pareto front.

This thesis also shows how multiple Pareto front classifiers can be combined into an ensemble where individual members vote on the class label. Two ensemble diversity measures are developed in the fitness functions which treat the diversity on both the minority and the majority classes as equally important; otherwise, these measures risk being biased toward the majority class. The evolved ensembles outperform their individual members on the tasks due to good cooperation between members.

This thesis further improves the ensemble performances by developing a GP approach to ensemble selection, to quickly find small groups of individuals that cooperate very well together in the ensemble. The pruned ensembles use much fewer individuals to achieve performances that are as good as larger (unpruned) ensembles, particularly on tasks with high levels of class imbalance, thereby reducing the total time to evaluate the ensemble.

Publications Produced

The following fully-referred papers were published during this Ph.D.

1. Urvesh Bhowan, Mark Johnston, Mengjie Zhang, Xin Yao. "Evolving Diverse Ensembles using Genetic Programming for Classification with Unbalanced Data". *IEEE Transactions on Evolutionary Computation* (Accepted April 2012).

2. Urvesh Bhowan, Mark Johnston, Mengjie Zhang. "Developing New Fitness Functions in Genetic Programming for Classification with Unbalanced Data". *IEEE Transactions on Systems, Man, and Cybernetics (Part B)*, volume 42, issue 2. 2011. pp 406–421.

3. Urvesh Bhowan, Mengjie Zhang and Mark Johnston. "Ensemble Learning and Pruning in Multi-Objective Genetic Programming for Classification with Unbalanced Data". *Proceedings of the 24th Australasian Joint Conference on Artificial Intelligence (AI 2011)*. Lecture Notes in Artificial Intelligence. Vol. 7106. Springer. Perth, Australia, December, 2011. pp. 192–202.

4. Urvesh Bhowan, Mengjie Zhang, Mark Johnston. "Evolving Ensembles in Multi-objective Genetic Programming for Classification with Unbalanced Data". *Proceeding of the Genetic and Evolutionary Computation Conference (GECCO 2011)*. ACM Press. Dublin, Ireland. 2011. pp. 1331–1338.

5. Urvesh Bhowan, Mengjie Zhang, Mark Johnston. "A Comparison of Classification Strategies in Genetic Programming with Unbalanced Data". *Proceedings of the 23rd Australasian Joint Conference on Artificial Intelligence. AI 2010: Advances in Artificial Intelligence*. Lecture Notes in Artificial Intelligence. Vol. 6464. Springer. Adelaide, Australia, 2010. pp. 243–252. (Nominated for the Best Student Paper award)

6. Urvesh Bhowan, Mengjie Zhang, Mark Johnston. "AUC Analysis of the Pareto-Front using Multi-objective GP for Classification with Unbalanced

Data". *Proceedings of the 2010 Genetic and Evolutionary Computation Conference (GECCO 2010)*. ACM Press. Portland, USA. 2010. pp.845–852.

7. Urvesh Bhowan, Mengjie Zhang, Mark Johnston. "Genetic Programming for Classification with Unbalanced Data". *Proceedings of the 13th European Conference on Genetic Programming (EuroGP 2010)*. Lecture Notes in Computer Science, Vol. 6021. Springer. Istanbul, Turkey. 2010. pp. 1–13.

8. Urvesh Bhowan, Mengjie Zhang, Mark Johnston. "Multi-Objective Genetic Programming for Classification with Unbalanced Data". *Proceedings of the 22nd Australasian Joint Conference on Artificial Intelligence (AI 2009)*. Lecture Notes in Artificial Intelligence. Vol. 5866, Springer. Melbourne, Australia. 2009. pp. 370–380.

9. Urvesh Bhowan, Mengjie Zhang, Mark Johnston. "Genetic Programming for Image Classification with Unbalanced Data". *Proceeding of the 24th International Conference on Image and Vision Computing New Zealand*. IEEE Press. Wellington, NZ. 2009. pp. 316–321.

10. Urvesh Bhowan, Mark Johnston, and Mengjie Zhang. "Differentiating Between Individual Class Performance in Genetic Programming Fitness for Classification with Unbalanced Data". *Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC 2009)*. IEEE Press. Trondheim, Norway. 2009. pp. 2802–2809.

Acknowledgments

I would like to thank my supervisors, Dr Mengjie Zhang and Dr Mark Johnston, for their guidance and constant encouragement over the past three years, and constructive feedback in writing this thesis and the articles that came before it.

Thank you to Dr Mengjie Zhang, the Marsden Fund of New Zealand (under contract number VUW0806), and the BuildIT PhD Scholarship, for the financial assistance over the past 3 years.

Thank you to the rest of the Evolutionary Computation Research Group, in particular Dr Kourosh Neshatian, for the many lively and interesting discussions.

Thank you to my Dad for his encouragement. And most of all, thank you to Niamh (and Murdoch) for the support you've given me these three long years. At times it has been difficult but you have always brought me through it with your encouragement, love and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Classification is the act of placing an object into a set of classes or categories based on the object's properties or features [68]. Given the abundance of real-world information now being captured and stored digitally, systems that can automatically search for and identify valid and useful patterns in data for classification, with as little human intervention as possible, are fast becoming highly desirable.

However, creating intelligent learning systems that perform classification reliably and with a sufficient level of accuracy, is difficult. Genetic Programming (GP) is a machine learning and search technique which has been successful in building reliable classifiers to solve classification problems [104][62][176]. GP is an evolutionary learning algorithm which uses the principles of Darwinian evolution and natural selection to evolve computer programs to solve a particular problem. In GP, programs representing different solutions to a problem are combined with other programs to create new, hopefully better, programs over a number of generations, until an good solution is evolved [104].

In many real-world applications, such as fraud detection [157][67][159], medical diagnosis [80][88], bioinformatics [132], or fault diagnostics [142], it is not uncommon to have a disproportionate number of training examples in one class compared to the other class(es). This is known as *class imbalance* and occurs when at least one class is represented by only a small number of examples (called the *minority class*) while the other class(es) make up the rest (called the *majority class*) [38].

1.1 Motivation

Recent work in the machine learning community has highlighted that the class imbalance problem represents a major obstacle in classifier learning [38][172][93]. This is due to the performance *bias* that can occur when an uneven distribution of class examples is used in the learning process. Here learnt classifiers can exhibit high accuracy on the majority class(es) but poor accuracy on the important minority class(es) [141][57][173]. As the minority class usually represents the main class-of-interest in most real-world classification problems, accurately classifying examples from this class is *at least* as important as, and in some scenarios more important than, accurately classifying examples from the majority class [67][159][142].

Addressing this learning bias to correctly classify examples from both the minority and the majority classes equally well has become an important area of research [38]. Work in this area tends to focus on three main aspects. The first involves sampling [157][12][11], or transforming [173][85], the original unbalanced data set to create artificially balanced classes for the training process (so-called "external" approaches). The second aspect involves various forms of cost adjustment within the learning algorithm to utilise the original unbalanced data "as is" in the training process; these are known as "internal" approaches as the learning algorithm itself is adapted to account for the uneven distribution of class examples [27][60][34]. The third aspect uses *ensemble* learning where multiple trained classifiers are aggregated together to determine the final prediction [159][164][123]. Ensemble learning uses aspects from both external and internal approaches to train the individual base classifiers. In bagging and boosting techniques, the training data is partitioned into smaller, balanced subsets of class examples using sampling techniques [123][118][170][37]; while in other ensemble learning approaches, a diversity measure is used in the fitness function to encourage cooperation between the base classifiers [119][36][3].

While external approaches can be effective, they have major disadvantages. Sampling can add a computationally expensive overhead to the training process as, in most cases, this must be applied repeatedly for good coverage. These techniques can also require *a priori* expert knowledge about the data [157]. More importantly, sampling techniques can suffer from *poor generalisation* as potentially useful learning examples can be excluded from the learning process, and the learnt models do not capture the underlying rarities that occur in unbalanced data sets (as the training set is first artificially re-balanced).

Due to these limitations, machine learning practitioners have recently focused on internal approaches using cost adjustment in the learning algorithm. Common approaches include using fixed misclassification costs for minority and majority class examples [88][142], or developing improved training criteria that are sensitive to the unbalanced class distributions (unlike the traditional overall accuracy measure). Improved training criteria for class imbalance includes the average classification accuracy of the minority and majority class [107][139][5][116], and the Area under the Receiver Operating Characteristics (ROC) curve (also known as the AUC) [27][84][149][90]. ROC curves are a useful technique to capture the performance trade-off between the minority and majority class accuracies in the learnt models across varying classification thresholds. In GP specifically, much work has focused on adapting the fitness function to reward solutions that are accurate on both the minority and the majority classes [141][57].

While these internal cost-adjustment based approaches can substantially improve minority class accuracy, there are three main limitations. Firstly, misclassification costs for incorrect class predictions must usually be determined *a priori*, where these costs can be problem-specific and require a lengthy trial-and-error process to configure [157][88][142]. Secondly, improved performance metrics in the fitness function (such as the AUC) can substantially increase training times due to the computational overhead required to calculate these measures, particularly on large data sets [34][179]. Finally, new fitness functions can be hand-crafted to suit a particular classification problem, requiring *a priori* expert knowledge about the problem domain [157][60]. In this area there is a need to develop new performance measures in the fitness function which can evolve solutions with good classification ability on both classes, without incurring a substantial increase in training times, and which are problem-independent.

Evolutionary multi-objective optimisation (EMO) is a fast-growing area of research which offers a promising solution to learning with multiple objectives that are in conflict. Unlike single-predictor classifier induction techniques where the fittest individual is returned from the training process, in EMO a set (or *Pareto front*) of solutions is evolved to capture the performance trade-off between the different objectives. EMO accomplishes this by treating the objectives independently in the learning process using the notion of Pareto Dominance in fitness. Pareto Dominance establishes a ranking of the individuals in the population according to how well they perform on all the objective with respect to each other [42][78].

EMO has shown success in three main problem domains in classification:

model regularization, ROC optimisation and ensemble learning. The first two of these problem domains typically involves classification tasks where the class distributions are assumed to be balanced. In model regularization, the (overall) accuracy is traded-off against the complexity or size of the learnt models [70][92][50]; while in ROC optimisation, the true positive and false positive rates are traded-off each against other [162][108][65].

However, in EMO-based ensemble learning for classification with unbalanced data, most approaches use neural networks, decision trees or Naive Bayes as the base classifiers [123][118][170][37], and rely on sampling techniques to re-balance the training data during fitness evaluation [159][123][170][168]. This means that most of these works assume that the classes are balanced before the diversity between the solutions are calculated. A GP-based multi-objective GP approach where the accuracy of the minority and majority classes are traded-off against each other in the learning process for cost-adjustment, thereby allowing the original unbalanced training data to be used "as is" in the learning process (without sampling), has not previously been explored. In addition, very few works in this area investigate how to adapt the diversity measures in the fitness function to account for skewed class distributions [168].

1.2 Research goals

To address these limitations, the overall goal of this thesis is to develop new internal cost-adjustment techniques in GP for binary classification problems with unbalanced data. To achieve this goal, two GP approaches are proposed, each with a specific set of research objectives:

1. Develop new improved fitness functions in canonical single-objective ("single-predictor") GP using the ROC curves of the evolved classifiers to represent the performance trade-off between the minority and the majority class accuracies.

    a) Develop a suitable GP approach and classification strategy for binary classification tasks with unbalanced data.

    b) Develop new, improved performance measures in the fitness function which account for both the minority and the majority class accuracies in the evolved classifiers.

2. Develop a new GP-based *multi-objective* approach to representing the performance trade-off between the minority and the majority class accuracies.

   a) Develop a multi-objective GP approach to evolving a Pareto front of genetic program classifiers along the minority and majority class trade-off frontier using Pareto dominance in the fitness function.

   b) Develop an ensemble learning approach to combining Pareto front classifiers using fitness functions that promote diversity between individuals equally on both classes.

   c) Develop new cooperative classification strategies in the ensemble using small highly-cooperative groups of individuals.

Focusing on internal cost adjustment using the fitness function allows the unbalanced data to be used "as is" in the learning process, requiring no external data-balancing techniques to artificially re-balance the input data prior to the learning process. This thesis focuses on internal methods due to three important considerations.

- All learning data is assumed to be useful and should *not* be excluded from the learning process (external data-balancing techniques can exclude useful learning instances in training).
- The GP approaches should be problem-independent and not require any *a priori* data-specific or expert knowledge about the input data. This means that the new methods should work well when the unbalanced data sets are used directly (or "as is") in the GP learning process, requiring no prior pre-processing or transformations for data-balancing.
- Using the unbalanced learning data directly in the training process allows us to concentrate on the properties established by the new cost adjustment techniques in the GP algorithm. Thus, any improvements can be attributed to these properties and not a given sampling policy.

1.3 Major Contributions

This thesis makes the following major contributions.

1. The thesis shows how to address binary classification problems with unbalanced data using GP, with particular focus on cost-adjustment within GP

rather than the traditional data-balancing techniques. In the GP approach, this thesis finds that there is no major differences in performance using the traditional (static) classification strategy and a dynamic (non-static) classification strategy on these binary class imbalance tasks. This shows that GP can sufficiently tweak the mathematical expressions representing the classifiers to "shift" its outputs relative to the fixed class boundary. This is not the case for multiple-class tasks according the the current literature which shows that the dynamic strategy outperforms the traditional (static) classification strategy. Rather, this thesis shows that the configuration of the fitness function in GP is more important for evolving well-performing classifiers. These results have been published in [16].

2. This thesis proposes several new measures in the fitness function in GP to perform cost adjustment between the minority and the majority class accuracies, allowing the unbalanced data sets to be used directly in the learning process without first re-balancing the data (via sampling). By treating these two objectives as equally important in the learning process, these new measures in GP find classifiers with good classification ability on both classes (high AUC), which outperform Naive Bayes (NB) and Support Vector Machines (SVM) on tasks with very high levels of class imbalance. On these tasks, both NB and SVM methods show very biased classification results. These results have been published in [13][14][17].

3. This thesis proposes a multi-objective GP (MOGP) approach where the accuracies of the minority and the majority classes are traded-off against each other in the learning process. The novelty of this approach is that a Pareto-based fitness function is used to the treat the unbalanced classes independently (i.e. as separate objectives) for cost adjustment when the unbalanced data is used directly in the learning process. This allows multiple trade-off solutions to be evolved in a single optimisation run, leaving the final choice for the decision-maker; whereas canonical (single-objective) GP requires a much longer time to get a reasonable front as the objective preference is specified *a priori*. MOGP using the SPEA2 [188] Pareto dominance algorithm is found to perform as well as, and in some cases better than, multiple runs of canonical single-objective GP; while MOGP using the NSGAII [53] algorithm cannot achieve this to a sufficient level of accuracy. These results have been published in [15][20][21].

4. This thesis shows how to adapt the fitness function in MOGP to promote

diversity between individuals and combine Pareto front classifiers into an ensemble where members vote on the class label. Unlike traditional ensemble learning approaches (where the unbalanced data is first re-balanced via sampling), the measures are adapted to calculate diversity *separately* for the two classes (to account for the unequal classes); otherwise, the diversity measures risk being biased toward the majority class. When diverse Pareto front solutions work together to classify the data instances, the evolved ensembles, in particular MOGP with the pairwise failure crediting diversity measure, perform better than their individual members, due to good cooperation between members on the tasks. These results have been published in [22][19].

5. This thesis shows how GP can be used for ensemble selection (and pruning) to quickly find diverse subsets of Pareto front individuals that cooperate well together in the ensemble. To avoid "fine tuning" a large weight vector (as used in traditional ensemble selection), the new approach evolves composite GP solutions to represent the (final) ensemble, by combining multiple Pareto front individuals into a single composite solution. The main novelties of the new approach include using selection pressure in the evolution to find small groups of diverse members for the ensemble (by imposing a size constraint on the GP solutions), and different function sets to manipulate the outputs of the individual members (to control the final ensemble classification decision). The GP composite solutions use fewer individuals in the ensemble to produce ensemble results that are as good as, and in some cases better than, an existing approach to ensemble selection (*Off*-EEL [76]), particularly on tasks with very high levels of class imbalance. These results have been partially published in [18].

1.4 Organisation of Thesis

The remainder of this thesis is organised as follows. Chapter 2 carries out a literature review; the five main contribution chapters, Chapters 3–7, address each of the five sub-goals in this thesis; and Chapter 8 concludes this thesis.

The literature review in Chapter 2 is split into two parts: background and related work. The background covers the fundamental concepts in evolutionary computation focusing on GP, ensemble learning, and evolutionary multi-objective optimisation. The related work discusses the recent advances in GP for

classification, class imbalance learning with particular focus on GP and ensemble-based approaches, and EMO for classification. The related work also discusses the limitations of the current approaches and the challenges that the thesis attempts to address.

Chapter 3 proposes the GP framework for classification, with particular emphasis on the classification strategy used in the evolution, and evaluates three major current approaches in the fitness function to highlight the advantages and limitations of each approach and why they need to be improved. This evaluation contrasts the AUC, the overall accuracy, and individual class accuracies of the evolved GP classifiers to justify why the AUC is a better performance measure in these class imbalance scenarios, rather than the traditional overall accuracy.

Chapter 4 develops several new performance measures in the fitness function to address the limitations highlighted in Chapter 3. Focusing on the AUC of the evolved classifiers, the new methods are compared to several current approaches in the the fitness function from the literature (including those from Chapter 3), and two other popular machine learning algorithms (Naive Bayes and Support Vector Machines). These fitness functions are ranked using a new measure designed to capture the statistical significance relationships between the new fitness functions on the unbalanced data sets. Several evolved GP classifiers are analysed to gain a better understanding of how GP learns to solve a given problem.

Chapter 5 develops the multi-objective GP (MOGP) approach using the accuracies of the minority and the majority classes as the two competing learning objectives. Particular emphasis is placed on how to represent Pareto dominance in the fitness function, where two popular dominance-based ranking algorithms from the literature (SPEA2 [188] and NSGAII [53]) are compared in fitness. The performance of the evolved Pareto-approximated fronts and the AUC of the Pareto front solutions, are compared to the single-objective GP (SGP) methods (from Chapter 4) to highlight the major differences between these methods.

Chapter 6 develops an ensemble learning approach to combining the evolved set of Pareto front classifiers (from the MOGP approach in the previous chapter) into an ensemble where members vote on the class label. Two ensemble-diversity measures are developed and incorporated in the fitness function in MOGP to promote cooperation between solutions, and the diversity-based ensembles are compared to an MOGP approach using no explicit ensemble-diversity measure in fitness. The ensembles are evaluated using two combination strategies to combine the outputs of the individual members, and two selection strategies to

Table 1.1: Unbalanced classification tasks used in the experiments in the thesis.

Task	Full Name	Number of Examples			Imb.	Features	
		Total	Minority	Majority	Ratio	No.	Type
Ion	Ionosphere [8]	351	126 (35.8%)	225 (64.2%)	1:3	34	Real
Spt	SPECT Heart [8]	267	55 (20.6%)	212 (79.4%)	1:4	22	Binary
Ped	Pedestrian Images [131]	24800	4800 (19.4%)	20000 (80.6%)	1:4	22	Real
$\mathbf{Yst}_1$	Yeast (*mit*) [8]	1482	244 (16.5%)	1238 (83.5%)	1:6	8	Real
$\mathbf{Yst}_2$	Yeast (*me3*) [8]	1482	163 (10.9%)	1319 (89.1%)	1:9	8	Real
Bal	Balance Scale [8]	625	49 (7.8%)	576 (92.2%)	1:12	4	Integer

only select accurate and diverse individuals for the ensemble. Several evolved MOGP classifiers are also analysed and compared to canonical SGP classifiers.

Chapter 7 develops a new GP approach to ensemble selection to quickly find groups of diverse Pareto front individuals that cooperate well together in the ensemble, improving ensemble performances from the previous chapter. Composite solutions are developed to represent the (pruned) ensembles; these amalgamate multiple Pareto front individuals into a single genetic program. Two types of composite solutions are developed, composite voting solutions and composite logical solutions, and these are compared to the ensemble selection strategies from the previous chapter.

Chapter 8 concludes this thesis by summarising the main conclusions and research objectives achieved in the individual chapters, and provides further discussions on more general topics covered in the whole thesis, and areas of future work.

1.5 Benchmark Tasks with Unbalanced Data

Throughout this thesis, the proposed GP methods are evaluated on six real-world benchmark classification tasks with unbalanced data from the *UCI Repository of Machine Learning Databases* [8], and the Intelligent Systems Lab at the University of Amsterdam [131]. These tasks are summarised in Table 1.1; for a detailed description of each task, please refer to Appendix A.

In each task, half of the examples in each class are randomly chosen for the *training* and the *test* sets. This ensures that both training and test sets preserve the same class imbalance ratio as the original data set. While it is possible that the class distributions in the training set and test set can be different, this thesis only considers tasks with similar distributions in both sets for comparison and generalisation purposes.

These benchmark data sets are carefully selected to encompass a varied collection of problem domains to ensure that the evaluation of the different GP approaches is not problem-specific. These problems have varying levels of class imbalance (minority class ranges between 7% and 35% of total examples), and complexity where some tasks are more easily-separable than others. The training and test sets also range from being well-represented (e.g. Ped has approximately 12000 instances), to sparsely-represented (e.g. Spt only has 27 instances from the minority class). These tasks also range between high and low dimensionality (e.g. Ion has 34 features while Bal only has 4), and use different features types (binary, integer and real features). Therefore, these data sets are expected to represent class imbalance problems of varying difficulty, dimensionality, size and (feature) types reasonably well. None of these data sets contains missing attributes — this is an interesting topic but beyond the scope of this thesis.

Chapter 2

Literature Review

This chapter provides the background and related work for this thesis. The first four sections of this chapter discuss the background material, including the following four topics: machine learning and classification, evolutionary computation, genetic programming (GP), and evolutionary multi-objective optimisation (EMO). The last two sections of this chapter discuss the related work which is split into two main categories. The first outlines the related work in GP and EMO for classification with balanced data, while the second outlines the related work in the wider machine learning community (including GP) for classification with unbalanced data.

2.1 Machine Learning

Machine learning is a broad and rapidly developing area of research [114][33][59]. Different artificial intelligence experts in this field vary in their definitions of what exactly constitutes machine learning but most agree that the central idea involves computer programs which learn to solve problems without explicitly being programmed or told how to do so [97][7][24][150].

Traditionally, learning methods have been split into three main strategies: supervised, unsupervised and reinforcement learning [24][150]. *Supervised* learning is learning with labeled class examples or instances. In supervised learning, the actions or desired outputs for a problem are known in advance, and the learning system tries to find rules or a function to map its outputs to the desired (or target) outputs. *Unsupervised* learning is learning without labeled class examples. In unsupervised learning, there are no correct answers for the learner to explicitly learn from. Instead, the learner must explore underlying structures or similarities in the data to find useful patterns such as clusters. In *reinforcement* learning,

the learner receives feedback based on its actions (outputs) in terms of rewards or punishments but unlike supervised learning, the desired outputs are not explicitly provided.

2.1.1 Classification

Classification is a supervised machine learning task where the system learns from a set of labeled input examples or instances. Given a set of attributes or features and their corresponding class labels, classification involves learning a model to correctly predict the class membership of each attribute [129][32].

Common to supervised learning problems are the concepts of *training* and *test* sets. A training set is a collection of input patterns from which classification rules are induced. A test set is a similar collection of input patterns, except that these are not used during the learning process and remain unseen while learning the rules. The purpose of the test set is to evaluate the performance of the learnt rules on unseen instances of the problem. This is important as it verifies that the learnt rules are not *over-fitted* to the training set. In supervised learning, the procedure for learning is two-fold: discover/learn the rules or a function for the input-output mappings using the training set, and apply these rules or functions to the test set to determine how well the learnt concepts perform (or *generalise*) on unseen problem instances.

This thesis focuses on supervised learning. In this area, there are many different learning paradigms, some include the following (the four paradigms discussed below are used in the experimental results throughout the thesis).

Bayesian Classifiers

Bayesian classifiers use a probabilistic approach to classification based on Bayesian probability principles. *Naive Bayes* (NB) is a simple but popular Bayesian classifier which uses Bayes' theorem to compute unknown probability estimates (i.e. the class of an unseen instance) from known ones (i.e. features of known instances) [129]. NB is remarkably effective in practice and can show competitive results compared to other more-complex learning paradigms [129][178]. However, NB makes strong (naive) assumptions about the conditional independence of the features where the presence (or absence) of a feature is assumed to be completely unrelated to the presence (or absence) of another feature. Bayesian *belief networks* [94] address this issue of conditional independence by representing dependencies between features as a directed graph.

Statistical Paradigms

Support Vector Machines (SVMs) [167] is a statistical supervised learning algorithm. SVMs construct a number of *hyperplanes* in the (high-dimensional) feature-space that aim to separate the input instances from the two classes, and then try to maximise the distance between the decision hyperplanes and the input instances from both classes (this distance is called the *margin*). The original SVM algorithm was a linear classifier where the input instances are assigned a class label depending on which side of a decision hyperplane they lie on [167]. However, the current version uses *kernel functions* to construct non-linear decision surfaces [44].

Genetic Paradigms

Genetic paradigms comprise of a wide range of nature-inspired computational methodologies that incorporate the modern principles of Darwinian evolution and natural selection into machine learning. Popular genetic paradigms include genetic algorithms [86] and genetic programming [104] which is also the focus of this thesis. These paradigms and other evolutionary computational methodologies are discussed in more detail in the next section.

Ensemble Paradigms

Ensemble methods combine together *multiple* learnt models to obtain better predictive performance than could be obtained from any of the single constituent models [31][129]. In an ensemble of classifiers, a *majority vote* is typically used to combine the outputs of the individual members: for a given input, each member votes on the output (e.g. predicted class label), and the class label with most votes is chosen as the ensemble output. Ensemble methods can be used with base learners from different learning paradigms, provided that the base classifiers are accurate and diverse with respect to their outputs [54][31]. Diverse ensemble members should not make the same errors on the same inputs, otherwise the ensemble will risk misclassifying the same inputs together each time.

Other Paradigms

In addition to the above-mentioned learning paradigms, there are also many other learning paradigms. Three important categories include the following (the

three paradigms discussed below are not used in this thesis but are included to give the reader a better idea of this field).

Connectionist Paradigms. These include artificial neural networks (ANNs) [23] which are computational models inspired by biological neural networks. An ANN consists of an interconnected group of artificial neurons (called nodes), where information (usually numeric) travels through nodes in different layers of the network. In classification, ANNs can model complex relationships between inputs (e.g. features) and outputs (class membership) to find patterns in data. However, ANNs are typically "black-box" learners as end-users cannot easily interpret the learned concept to understand how an ANN has learned to solve a problem.

Case-Based Reasoning. These include the nearest neighbour algorithm [45] which classifies an unseen instance as the same class of the closest training instance in feature-space. These learning paradigms are *lazy* in that they do not attempt to learn or generalise a classification model using the training data.

Induction Based Reasoning. These include decision tree algorithms which seek to split features that best separate the input instances from the training set [129]. Decision trees classify instances by traversing a tree in top-down manner, starting at the root node and ending at a leaf node which represents the class label. Decision trees are easy to interpret as they represent *if-then* classification rules; popular algorithms to build decision tree include *ID3* [147] and *C4.5* [148].

2.1.2 Class Imbalance Learning

In many real-world applications, it is not uncommon to have disproportionate numbers of learning instances for one class compared to the other class(es). In classification with unbalanced data (also known as the class imbalance problem), at least one class is represented by only a small number of examples (called the *minority class*) while the other class(es) make up the rest (called the *majority class*). Research in the machine learning community has highlighted that using unbalanced data in the learning process can leave the learnt classifier with a performance bias, that is, classifiers exhibit high accuracy on the majority class(es) but poor accuracy on the minority class(es) [38][93][172][130][69]. Addressing this learning bias to find classifiers with good accuracy on both the minority and majority class is an important area of research and the focus of this thesis.

The following is list of real-world problems affected by class imbalance.

- In *fraud detection* tasks such as network intrusion detection [157], telephone fraud [67], and credit card fraud [159], fraudulent transactions are relatively small compared to the vast majority of normal transactions.
- In *medical diagnosis* of rare medical conditions, the majority of patients are healthy and only a small minority will be diseased [80][88][41].
- In *bio-informatics* tasks such as protein classification, the target protein class is small compared to non-target (normal) proteins [132].
- In *financial risk modeling* such as loan approval or insurance risk modeling, high-risk applicants are rare in comparison to normal loan or insurance applicants [143].
- In some *data mining* tasks such as direct marketing, negative responses are typically small compared to positive responses [117]; or in *churn* prediction [179], relatively few customers switch subscriptions compared to those who do not.
- In *image recognition* or object detection tasks such as target [89], face [161] or pedestrian [162] detection, the important objects-of-interest are typically in the minority class compared to non-objects (background).
- In *fault diagnostics* such as industrial defect detection [85] or network troubleshooting [142], faulty instances are typically rare compared to normal instances.

2.1.3 Evaluating Classifier Performance

The traditional measure to evaluate the goodness or success of a learnt classifier uses the overall classification accuracy (or overall error rate) on the learning instances [129]. The overall accuracy is the number of inputs correctly labeled by the classifier as a proportion of the total number of inputs seen by the classifier. Using the four different outcomes for binary classification shown in Table 2.1, the overall accuracy can be expressed by the equation below. In Table 2.1, assume that the minority class is the *positive* class.

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

Table 2.1: Outcomes of a two-class classification problem.

	Predicted Positive Class	*Predicted* Negative Class
Actual Positive Class	True Positive (TP)	False Negative (FN)
Actual Negative Class	False Positive (FP)	True Negative (TN)

However, the overall accuracy is known to be unsuitable for classification with unbalanced data [93][130][172][69]. This is because this measure considers all learning instances as equally important and does not take into account that the number of learning instances in the minority class can be much smaller than in the majority class. A *biased* classifier which has very poor accuracy on the minority class but high majority class accuracy, can also have a high overall accuracy due to the influence of majority class learning instances.

Measuring the individual classification accuracy of the minority and majority class separately such as the true positive (TP) rate and true negative (TN) rate, respectively, as shown below, can avoid this learning bias when evaluating classifier performance in class imbalance scenarios.

$$\text{TP rate} = \frac{TP}{TP+FN} \quad \text{and} \quad \text{TN rate} = \frac{TN}{TN+FP} \tag{2.2}$$

The TP and TN rates are similar to the sensitivity and specificity, or precision and recall, as used in the context of other tasks (such as information retrieval) [178]. These measures all accomplish the same goal in classification, that is, categorising the type of error made by a classifier. Precision and recall are defined as:

$$\text{Precision} = \frac{TP}{TP+FP} \quad \text{and} \quad \text{Recall} = \frac{TP}{TP+FN}$$

Sensitivity and specificity are defined as:

$$\text{Sensitivity} = \text{Recall} \quad \text{and} \quad \text{Specificity} = \frac{TN}{TN+FP}$$

The TP and TN rates are usually in conflict with each other where an improvement in one class (e.g. TP rate) produces a trade-off in the other (e.g. TN rate).

ROC Curves

Receiver Operating Characteristic (ROC) curves were originally used in signal detection theory to characterize the trade-off between hit rate and false alarm rate over a noisy channel [84]. This is now widely used in machine learning to evaluate a classifier's performance across varying decision thresholds

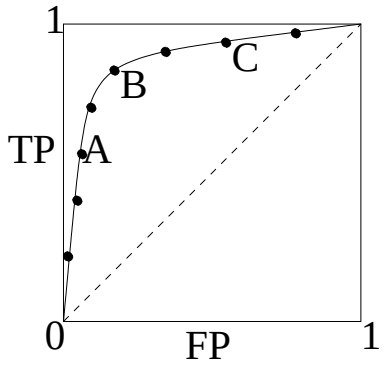


Figure 2.1: An ROC curve where operating points A, B and C represent the classifier's performance at three different decision thresholds.

[111][149][130][172]. On a ROC curve, the TP rate is plotted against the false positive (FP) rate, as seen in Figure 2.1, where each operating point (e.g. A, B and C in Figure 2.1) represents the TP and FP rates at a given decision threshold. In the context of binary classification, assuming the minority class is the positive class, the TP rate is the minority class accuracy, and FP rate is 1-minority class accuracy (or 1-TN rate).

The lower-left corner on an ROC curve (when TP and FP rates are 0) represents the decision threshold where all inputs are classified as belonging to the *negative* class. The upper-right corner (when TP and FP rates are 1) is the decision threshold where all inputs are classified as belonging to the *positive* class. Perfect classification accuracy (all inputs correctly classified) is achieved when the TP rate is 1 and FP rate is 0 (top-left corner). The line $x = y$ represents the strategy of randomly guessing the class label for a given input.

The area under an ROC curve (AUC) summarises the *classification ability* of a classifier across different decision thresholds, and represents the *probability* that an input from the positive class is correctly predicted by a given classifier [27][84]. AUC values range between 0 and 1 where the higher the value, the better the performance. Unlike the overall classification accuracy, the AUC is known to be invariant to unbalanced data and is not influenced by the larger majority class in class imbalance scenarios [173][93][130][172].

The AUC is typically approximated using the trapezoidal technique [84] where the AUC is calculated as the sum of the areas of individual trapezoids fitted under each pair of points of the ROC curve.

2.2 Evolutionary Computation

Evolutionary computation (EC) is a sub-field of artificial intelligence that comprises of nature-inspired computational methodologies. The two main categories in the area include evolutionary algorithms and swarm intelligence [97][99].

2.2.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a class of algorithms that incorporate the modern principles of biological evolution and Darwinian theories of natural selection of species into machine learning. These theories assert that only the fittest organisms will survive to reproduce while the less fit die off. As the offspring of fit parents will have similar or the same genetic code as their parents, the new generation of organisms is expected to be fitter, or at least as fit, as the current generation. An advantage of EA's is the ability to navigate through "worse" areas of the search-space in order to avoid becoming stuck in a local optima.

In EAs, an individual (or member of the population) represents a potential solution to a given problem, and these individuals are *evolved* (or improved) over generations using genetic operators. Genetic operators are mechanisms inspired by biological evolution such as crossover, mutation or reproduction, to create and improve individuals in the population. A fitness function determines the "goodness" of an individual, that is, how well an individual solves a given problem. Members of the population are selected for recombination (using the genetic operators) depending on their fitness. The goal in EAs, similar to natural selection, is to have some useful part of an individual's genetic code propagated down generations until an individual with good fitness is evolved.

The main EA techniques include the following.

Genetic Algorithms

Genetic Algorithms (GAs) are one of the earliest representations of an artificial evolutionary learning system [86][79]. In GA, individuals or *chromosomes* are typically encoded as fixed-length bit strings where each element in this string is called a *gene*, as shown in Figure 2.2.

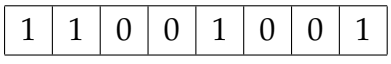


Figure 2.2: Example of an encoded 8 bit binary chromosome (individual) in GA.

However, a major limitation in GA systems is that the chromosome can require a complex encoding/decoding process. The encoding process is required to transform a task into the appropriate chromosome representation for the evolutionary process. The decoding process is required at every fitness iteration from the start of the evolution to evaluate a GA solution, to convert the individual from the chromosome representation into a more useful solution representation. For example, without the specific encoding/decoding knowledge, the meaning of a chromosome, such as the one shown in Figure 2.2, is almost impossible to interpret.

Genetic programming

Genetic Programming (GP) extends the idea of GAs by increasing the complexity of representation of individuals from fixed length bit strings to *computer programs* [104]. GP has been highly successful in automatically evolving variable length computer programs to solve a range of problems, particularly in the areas of symbolic regression [104][81], classification [176][62][111], automatic feature selection and construction [112][133][134], and object recognition and detection in images [163][89][184]. GP has two key advantages over GAs [106]. The first is the flexibility allowed in the representation (of individuals) [106]. The second is interpretability of the evolved programs; these can be easily analysed and decomposed in a meaningful way to understand how GP has learned to solve a problem [105].

Evolutionary Programming

Evolutionary Programming (EP) was originally used to evolve *finite state machines* to represent the learnt predictors [72]. Currently, there is no constraint on the representation of individuals in EP; this usually follows from the problem. However, the structure of the individuals to be optimised typically remains fixed during the evolution, while its numerical parameters are allowed to evolve. Mutation is the dominant genetic operator in EP; while crossover is not typically used. In EP, every individual in the population is mutated to generate one offspring and the level of variation in the offspring depends on a statistical distribution.

Evolution Strategies

Evolution Strategies (ES) are typically used for continuous parameter optimisation where individuals are represented as fixed-length real-valued vectors [153]. Mutation is also the dominant genetic operator in ES (similar to EP). Since all elements in the individuals are real-valued, mutation is usually performed by adding a normally distributed random value to each vector element in a given individual (to generate an offspring). Unlike GA, GP and EP which typically use a stochastic selection process, ES uses a deterministic selection process where individuals with poor fitness are removed from the population at every generation.

Learning Classifier Systems

Learning Classifier Systems (LCS) share close links with *reinforcement learning* and GAs [87]. In general, LCS is an adaptive system that learns based on actions (or outputs) that generate rewards from the environment for a given input. LCS was originally a population of binary rules, and a GA was used to update and select the best rules. Currently, the classifiers or rules can include real-valued and functional conditions (such as *S-expressions*). The evolution of classifiers takes place as the LCS system interacts with its environment. LCS systems can be split into two types: *Pittsburgh* LCS which has a population of separate rule sets, and *Michigan* LCS which only has a single set of rules in a population.

Differential Evolution

Differential Evolution (DE) was originally developed for multi-dimensional real-valued function optimisation problems [160]. Recently, DE has also been used for optimisation problems that are not continuous or which change over time since DE does not use the gradient of the problem being optimised [100]. In DE, the population of candidate solutions typically represents parameter vectors. Unlike many EAs, in DE recombination creates new candidate solutions using the weighted difference between two randomly selected population members when added to a third population member [100]. The new candidate solution is accepted as part of the population, or rejected (discarded), based on its fitness.

2.2.2 Swarm Intelligence

Swarm Intelligence (SI) is inspired by the *collective* behaviour of organisms in a biological system such as ant colonies, bird flocks and bacterial growth [25]. In SI, populations comprise of simple agents which follow simple rules; agents can interact with other agents and with their environment. Although no centralized control dictates how agents should behave in their environment, interactions between agents can lead to "intelligent" global behavior by the swarm. Two popular SI algorithms include the following.

Ant Colony Optimization

In Ant Colony Optimization (ACO) [55][56], the agents represent artificial ants. Based on the behaviour of ants seeking a path between their colony and a source of food, ACO methods are useful in problems that search for an optimal path to a goal. Agents can locate optimal paths by exploring different paths in their environment; this can be thought of as moving through the parameter-space which represents different candidate solutions. Ant agents lay down *pheromones* while exploring their environment to interact with other agents.

Particle Swarm Optimization

In Particle Swarm Optimization (PSO) [98][99], a population of candidate solutions (called particles) move around an n-dimensional search-space according to simple mathematical formulae which determine each particle's position and velocity. Each particle's movement in the search-space is influenced by two main factors: its own local best-known position according to some fitness criterion at each iteration, and the global best-known positions which are found by other particles. These movements are expected to move the swarm toward the best solution in the search-space.

2.3 Genetic Programming

This section introduces the main concepts in GP, including the GP search process, representation, creating GP programs, the genetic operators, and fitness and selection in GP.

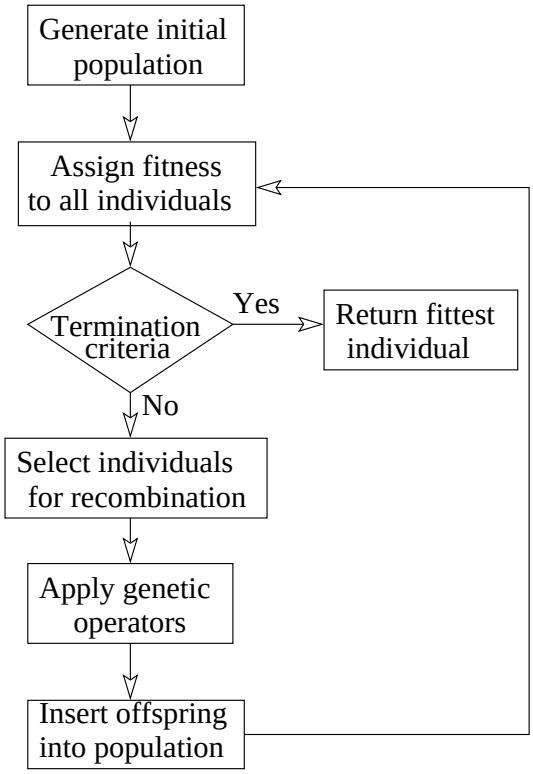


Figure 2.3: Evolutionary search in GP.

2.3.1 Overview of Evolutionary Search Process

An overview of the GP evolutionary search process is shown in Figure 2.3. The evolution starts by randomly creating an initial population of GP solutions. The fitness of each individual in the population is then evaluated using the fitness function (on the training set). If the termination criteria is met, the evolution is stopped and the fittest individual from the population is returned; otherwise, individuals are selected for recombination using the genetic operators to create the next generation of individuals. The termination criteria checks whether an individual in the population has been found which has a perfect fitness (or *close* to perfect fitness according to a pre-defined tolerance), or whether the maximum number of generations allowed in the evolution is reached.

2.3.2 Representation

An individual can be encoded as a genetic program solution in different ways. Common program representations include the following.

- *Tree-based GP*. Programs are represented as tree-like structures which can vary in length as well as what input alphabet they may contain [104]. The

leaf nodes in these genetic program trees correspond to *terminals* and the non-leaf nodes *functions*. This is the original and most common type of GP representation, and also used in this thesis.

- *Linear GP*. Programs are represented as variable-length sequences of instructions similar to an imperative programming language [10][30].
- *Graph-based GP*. Programs are represented as tree-based graph structures where partial sub-trees are allowed to be re-used in program execution such as *Cartesian GP* [127][128]. In addition to function and terminal nodes, programs have *edges* to represent the flow of data in the tree.
- *Grammar-based GP*. Individuals are represented at fixed-length strings that are encoded according using a user-specified grammar. Popular implementations include grammatical evolution (GE) [151][136] and grammar-based GP [175].

2.3.3 Creating Programs

There are three main ways of creating random individuals in tree-based GP: full, grow and ramped half-and-half [104]. The *full* method ensures that fully-formed trees are constructed according to a maximum allowed tree depth. The *grow* method allows trees of different depths and irregular shapes in the initial population. This method constructs trees by selecting each intermediate node from either the function or terminal sets, allowing part of a tree to be terminated whenever a terminal node is selected. The *ramped half-and-half* method combines the full and grow methods where half the trees are created using full method and the other half using the grow method. This method enhances the diversity of the population.

2.3.4 Genetic Operators

The genetic operators *crossover* and *mutation* alter the genetic material in the individuals to create offspring, while *reproduction* leaves the genetic makeup of an individual unaltered.

Crossover

Crossover swaps the sub-trees of two individuals at randomly selected points (called crossover points) to produce two new offspring, as shown in Figure 2.4.

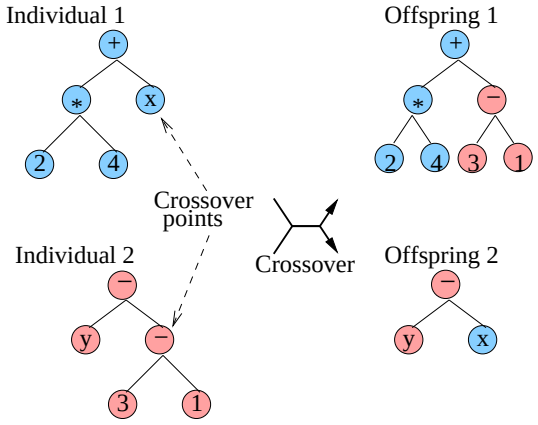


Figure 2.4: Crossover operator in GP.

Crossover works on the idea of combining the genetic makeup (i.e. sub-trees) between two individuals. This is arguably the most predominant operator used in GP [104]. A strong implication of crossover is that individuals with good fitness are crossed-over more often than those of weaker fitness. This continues the idea of "survival of the fittest" as the next generation becomes influenced by the stronger genetic makeup of the fit individuals from the current generation. The ideal case is if two individuals chosen for crossover have partially correct sub-trees, and the crossover points occur precisely on these partially correct sub-trees, then one offspring will contain both partially correct sub-trees.

Mutation

Mutation replaces a randomly selected sub-tree in a given individual with another sub-tree, as shown in Figure 2.5. It is performed on one individual, producing one offspring which is the mutated version of the parent. The new sub-tree is generated in the same way as the initial population is generated (via the grow, full or ramped half-and-half method). The mutation operator is the only one of the three main operators to introduce new genetic code into a population via an individual during evolution [104]. The motivation for mutation is to ensure diversity of the population [104].

Reproduction

Reproduction (also called *elitism*[1]) simply copies an individual from the current population into the next population. Unlike crossover and mutation, in repro-

[1]Reproduction and elitism actually have some differences but we do not distinguish them in this thesis similar to [144].

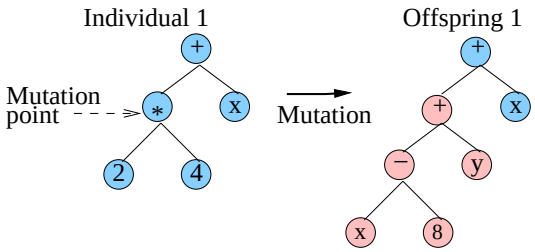


Figure 2.5: Mutation operator in GP.

duction the "offspring" in the next generation can not be worse in terms of fitness than the original "parent". For this reason, only the fittest individuals in the population at each generation are selected for reproduction. This is important as it ensures that the evolutionary search produces individuals in the next generation that are at least as fit as individuals from the previous generation.

2.3.5 Fitness and Selection

The fitness of an individual represents the goodness of that individual as determined by a suitable fitness function [104]. An individual's fitness affects its probability of selection for recombination and thus, the survival of its genetic make-up.

The two most commonly used methods for selecting individuals for recombination are *fitness-proportional* and *tournament* selection. In fitness-proportional selection, the probability of selection for an individual is proportional to its fitness [104]. This means that each solution competes with every other member in the population during selection. In tournament selection, a number of individuals in the population are chosen (sampled) at random; this number is referred to as the *tournament size* [104]. The fitnesses of individuals in a particular tournament are then compared to each other only (and not the rest of the population) during selection. This means that solutions compete with only those solutions in a given tournament during selection. GP can be used with either of these (or other) selection methods.

2.4 Evolutionary Multi-objective Optimisation

Evolutionary multi-objective optimisation (EMO) further extends traditional genetic paradigms by adapting the learning process to include multiple goals or objectives. These objectives are usually in conflict with each other which results

in a performance trade-off when one is optimised over the other(s). EMO handles each of the multiple, competing objectives independently in the optimisation process. This means that candidate individuals in EMO represent compromise (or trade-off) solutions with respect to the objectives.

2.4.1 Learning with Multiple Objectives

One of the first EMO approaches from the literature, called Vector Evaluated Genetic Algorithm (VEGA) [152], proposed a simple extension to traditional GAs. Here the population was divided into a number of different sub-populations, one for each objective. At each generation, selection was performed according to each objective function in turn. Nowadays, EMO approaches are classified into three main search techniques depending on how the objectives are treated in the learning process [42]. These include *a priori*, *a posteriori* or *progressive* objective articulation.

In *a priori* articulation, objectives are combined or aggregated together before the multi-objective (MO) search according to some pre-defined objective preference information. The evolutionary search returns a single solution that is close to optimal relative to this objective prioritisation. The focus of this technique is how to combine or aggregate the objectives, and how to determine objective relative importance prior to the search.

A posteriori objective articulation assumes that the objectives cannot be interrelated with each other. Each objective is therefore treated separately in the optimisation process. The focus of this method is to find a set of the best trade-off solutions along the objectives (called the *Pareto front*). This means that the end-user may then choose a single solution (with the desired trade-off) from the Pareto front after the EMO search has evolved a good set of set of trade-off solutions.

Progressive objective articulation integrates *a posteriori* EMO search with objective preference articulation during the optimisation process in an iterative and interactive manner.

This thesis uses concepts from both *a priori* and *a posteriori* objective articulation. These methods are discussed in more detail below. For further details on earlier EMO approaches (such as VEGA [152]), please refer to [73] and [74].

A Priori Search

The simplest and most common technique to learn with multiple objectives uses a linear aggregating function to combine the multiple objectives together into a scalar (fitness) value. Weighting coefficients are usually assigned to each objective to specify the relative importance of each objective. This can be represented by the simple aggregation function defined by Eq. (2.3) where f_i represents the performance of individual S on the ith objective, w_i represents the weighted relative importance of the ith objective ($0 \leq w_i \leq 1$), and k is the number of objectives.

$$\sum_{i=1}^{k} w_i f_i(S) \tag{2.3}$$

A major limitation of this technique is that the relative objective importance must be specified *prior* to the search. In many real-world problems, determining suitable weighting coefficients is a non-trivial task. Bad objective prioritisation can lead to poor performing solutions, and in the worst case, these objectives cannot be linearly interrelated. Another major limitation is that only one solution is returned from the search. This means that subsequent changes to the objective preference information, once a solution has been found, can often require the optimisation process to begin afresh with an updated weighting preference.

A Posteriori Search

For the reasons stated above, much work in EMO tends to focus on *a posteriori* search methods [53][188][156][171]. By assuming no prior knowledge about the objectives in the optimisation process, these EMO methods strive to find a diverse *set* of alternative competing solutions along the optimal objective trade-off surface. The goal of *a posteriori* EMO is to push the frontier of trade-off solutions (Pareto front) closer toward a point which is optimal on all objectives, while still allowing a sufficient level of diversity (which refers to the "spread" of solutions along the frontier) with respect to the objectives.

This requires two major adaptations in EMO learning systems compared to traditional (single-predictor) EAs. The first involves adapting the evolutionary search algorithm to evolve a set (or Pareto front) of solutions in parallel, where the Pareto front is the output of the EMO search. In contrast, in traditional EAs (or *a priori* search), the evolutionary search is focused on finding only a single solution with high fitness, where this individual is the output of the evolutionary

search. The second adaptation incorporates the notion of Pareto dominance in the fitness function to measure an individual's performance on all the objectives and relative to all others in the population. Pareto dominance is discussed in more detail in Section 2.4.3.

2.4.2 EMO Search Algorithms

The literature contains many *a posteriori* EMO search algorithms to evolve a Pareto front of solutions using multiple objectives [53][188][103][43][156]. All these algorithms use a Pareto-based fitness scheme which treats each objective as a separate entity in the learning process. However, each algorithm has subtle variations in the way Pareto dominance is used in fitness, or in the selection method to balance exploration or exploitation of a Pareto front. Some common EMO algorithms include the following (for a more definitive list of EMO algorithms, see [42]).

- Non-dominated Sorting GA (NSGA/NSGA-II) [53]. The fixed-size archive population is chosen by combining the parent and child population at every generation and selecting only the fittest individuals for inclusion. Dominance ranking and a "crowding" measure are used in fitness. Pareto ranking and "crowding" is discussed in more detail in Section 2.4.3.

- Strength Pareto Evolution Algorithm (SPEA/SPEA2) [188]. The archive population can be variable in size. Fitness or *strength* is determined using dominance rank and dominance count. A clustering mechanism is used when the archive size is exceeded.

- Pareto Envelope-based Selection Algorithm (PESA/PESA-II) [43]. This algorithm consists of a small internal (main) population and a larger archive (external) population, but unlike other algorithms, PESA allows the archive population to be only partially filled. Fitness is similar to NSGA/NSGA-II using dominance ranking and crowding.

- Pareto GP [156]. This GP implementation uses an archive population that is persistent across generations *and* runs to re-use good genetic material from previous independent runs in a population. For diversity in the breeding pool, crossover is performed on a member of the archive and a randomly selected member of the current local population.

- ϵ-Multi-Objective Evolutionary Algorithm (ϵ-MOEA) [52]. Unlike NSGA-II [53] and SPEA2 [188], ϵ-MOEA is a steady-state algorithm. At each generation, one offspring is created using one parent from the main population and one parent from an (external) archive population, and the offspring replaces the parent in the main population if it has a better fitness (otherwise it is discarded). The offspring is considered for the archive population based on the ϵ-dominance criteria.

- Pareto Archived Evolution Strategy (PAES) [103]. PAES is also a steady-state algorithm but unlike ϵ-MOEA [52], a *(1+1)* evolution strategy is used to generate the child population (i.e. a single parent is mutated to generate a single child) and the archive population has a fixed size.

These EMO algorithms typically use the same crossover and mutation genetic operators in the evolution as traditional EA's. However, EMO typically uses a secondary or external population, called the *archive* population. At each generation, the archive population includes the current set of non-dominated solutions (Pareto front). The archive population is used to simulate *elitism* in the population over generation, to ensure that all no non-dominated are lost as the evolution progresses. Implementation of this archive population and its interaction with the main population (e.g. for breeding) can vary between EMO algorithms.

2.4.3 EMO Fitness

In EMO, the notion of *Pareto dominance* in fitness allows individuals to be ranked according to how well they perform on all the objective with respect to all other individuals in the population. This is important as it affects the way selection is performed if the objectives are to be treated equivalently in the search process [42][78]. Pareto dominance between two solutions asserts that a single solution will *dominate* the other solution if it is at least as good as the other solution on all the objectives and better on at least one [42][78]. Solutions are *non-dominated* if they are not dominated by any solution in the population with respect to all objectives. Non-dominated solutions are usually assigned the best fitness for selection preference over other, dominated solutions. At every generation, the set of all non-dominated solutions forms the current approximation to the *Pareto front*. When a non-dominated solution can no longer be improved with respect to all objectives it is *Pareto-optimal*.

Pareto Ranking

Pareto ranking creates a partial ordering of solutions in a population according to their ability on all the objectives. This ordering can then be directly mapped to a scalar fitness value for selection. Two popular Pareto-based ranking schemes include dominance rank and dominance count. *Dominance rank* is the number of other solutions in the population that dominate a given solution (lower dominance ranks are better). This fitness scheme rewards exploration at the edges of the Pareto front (i.e. near the extremes of one objective) [42]. *Dominance count* is the number of other solutions in the population dominated by a given solution (higher dominance counts are better). This fitness scheme rewards exploitation in the middle of the Pareto front [42].

Niching Techniques

While Pareto-based ranking ensures equal selection preference to all non-dominated solutions along the Pareto front, it does not guarantee that the Pareto front will be uniformly sampled during selection [113]. This can lead to genetic drift in the population toward a single point on the front only. To avoid this, Pareto-based fitness schemes use *niching techniques* in the fitness function to encourage diversity along the Pareto front. Niching techniques typically use a "crowding" distance measure between solutions in *objective-space* to favour solutions from less crowded areas of the Pareto front and penalise solutions in densely populated regions of the Pareto front. Pareto ranking and "crowding" are typically combined in the fitness function in a hierarchical manner. Pareto ranking serves as the primary fitness, while the crowding measure resolves selection when the Pareto rank is equal between two or more solutions.

2.4.4 Evaluating Pareto Fronts in EMO

As the outcome from an EMO system is a *set* of evolved non-dominated solutions, different evaluation metrics are used to measure the "goodness" or quality of the evolved non-dominated solutions compared to traditional (single-predictor) learning paradigms where the single best solution is returned from the learning process. This thesis uses two multi-objective evaluation metrics to measure the performance of the evolved Pareto fronts: attainment functions and the hyperarea indicator (these are discussed in more detail below). Another common evaluation metric is to compare the learned (approximated) front with known fronts (this method is not used in this thesis).

Attainment Functions

Attainment summary surfaces [102][42] are a useful technique in EMO to summarise the outcome of a series of multi-objective optimisation runs, where a potentially different set of non-dominated solutions is returned from each run. For example, Figure 2.6(a) shows how different sets of non-dominated solutions can be returned from four runs of a (stochastic) EMO system, where each is obtained using a different random initial starting seed in the EMO system. In this figure, assume that the two objectives (*f1* and *f2*) are to be maximised.

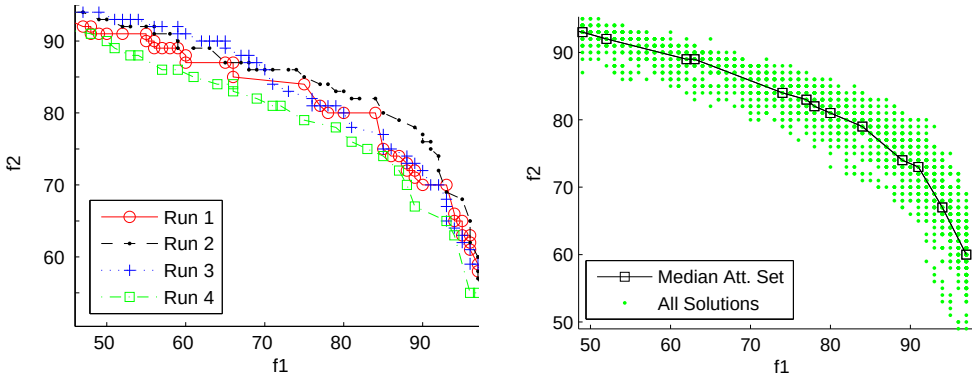
Each non-dominated solution returned from the EMO system over all optimisation runs is assigned an *attainment value*. Attainment values range between 0 and 1. This value corresponds to the probability that the EMO system will produce another solution which *weakly dominates* (i.e. is better than or equal to) the given solution on all the objectives [102][42]. Non-dominated solutions that have identical attainment values are then grouped together into *attainment surfaces*; the number of attainment surfaces corresponds to the number of optimisation runs. In general, solutions with low attainment values represent well-performing solutions as there is a low probability that the EMO system with evolve another solution that is better on the objectives. In contrast, solutions with high attainment values represent poor-performing solutions as there is a high probability that the EMO system will evolve another solution that is better on the objectives.

Solutions in the *median* attainment surface (i.e. the set of solutions with attainment values of 0.5) are of particular interest as these correspond to solutions with a 50% probability of attainment with respect to all runs. The median attainment surface approximates the *typical* (or "average") set of non-dominated solutions that an EMO system can be expected to evolve with respect to the series of optimisation runs. For example, Figure 2.6(b) shows the median attainment surface with respect to the set of non-dominated solutions obtained from 50 optimisation runs.

For more details on how the attainment values are calculated, please refer to Appendix B (Section B.1).

Hyperarea Quality Indicator

Unlike attainment summary surfaces, the *hyperarea* (also known as the hypervolume) of an evolved Pareto-approximated front represents a "single figure" measure to quantify the performance of the entire front on the objectives [102][42].



(a) (b)

Figure 2.6: (a) Different sets of non-dominated solutions returned from four EMO runs; and (b) the median attainment surface with respect to 50 EMO runs.

The hyperarea is the area under the Pareto-approximated front in *objective-space*, similar to the area under the ROC curve (or AUC) in a single learnt classifier. However, while the AUC represents the performance of a single classifier at varying decision thresholds, the hyperarea represents the classification performance of the *set* of individuals along the evolved front. The hyperarea is typically calculated in a similar way to the AUC, that is, using the trapezoidal technique [42]. Hyperarea values range between 0 and 1 where the higher the value, the better the performance.

2.5 Related Work: EAs for Classification (with Balanced Data)

This section outlines three related areas pertaining to traditional classification with balanced data. This includes GP and EMO for classification, and related aspects in EA-based ensemble learning.

2.5.1 GP for Classification

Genetic programming has been widely used to successfully evolve reliable and accurate classifiers over a range of classification problems [176][62][80][158][61][111][155]. While GP for classification (with balanced data) represents a large area of work, this section provides a brief overview of the four main concepts in that pertain specifically to this thesis. These include GP classification models, classification strategies, the fitness function, and GP for ensemble learning.

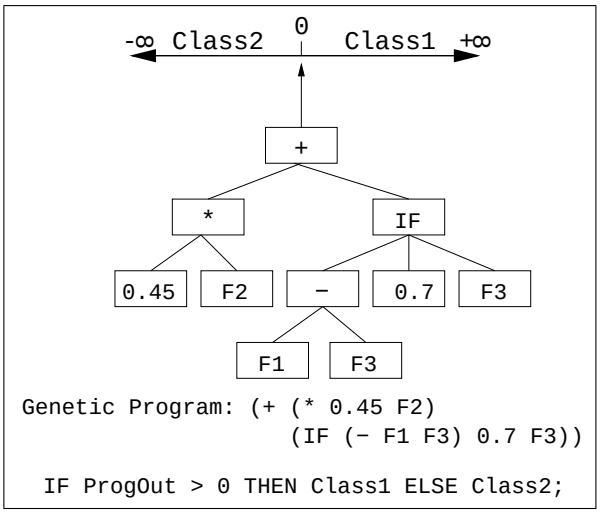


Figure 2.7: Classification strategy in GP.

Classification Models in GP

In tree-based GP, different kinds of models can be used to solve a given classification task due to the flexibility of the GP representation. Two common approaches include representing individuals as decision trees or discriminant functions for classification [64]. In *decision trees*, leaf nodes represent the class labels while internal nodes represent conditions on the features; the path from the root node to a leaf node represents the process of classifying an input instance. A *discriminant function* is when the GP classifier is represented as a mathematical expression where different operators are applied to the features of the input instances to be classified. This thesis uses discriminant functions for classification.

As a mathematical expression, a discriminant function typically computes a single floating-point number which is the output of the GP tree [176][80][158][111][155]. This single output value is then translated into a set of class labels. In binary classification, the division between positive and negative numbers is typically used as the two class boundaries to determine the class labels [181][176][80][158][111]. For example, Figure 2.7 illustrates how the numeric output of a genetic program is used for binary classification, where F_1, F_2 and F_3 represent three input features and $ProgOut$ denotes the genetic program output. Here an input instance is assigned to *class1* if the genetic program output is greater than zero; otherwise, the input instance is assigned to *class2*. Using this strategy, the class threshold is fixed at zero.

Dynamic Classification Strategies in Multi-class GP

Recently, new dynamic classification strategies have been developed to translate the numeric output of a GP individual to a set of class labels for multiple-class classification tasks (with balanced data) [186][185][154][120]. In these works, dynamic class boundaries are determined on an individual-by-individual basis for each member in the population; whereas in the traditional strategy discussed above for binary classification, the class boundaries remain fixed for all individuals in the population (i.e. zero is the class threshold). These new approaches include Dynamic Range Selection (DRS) [120], Centred Dynamic Class Boundaries (CDCB) [185], Slotted Dynamic Class Boundaries (SDCB) [154] and a probabilistic classification strategy [186]. As one of the goals in this thesis is to determine whether the traditional (static) strategy is sufficient for these binary classification tasks with unbalanced data compared to a dynamic (non-static) strategy, a brief outline of these approaches is discussed below.

In DRS [120] and SDCB [185], the number line is divided into a fixed number of "slots", and the real-valued output of an individual (when evaluated on an input instance) is mapped to a particular slot (using a truncation operator). Once all inputs are processed, the class which has the most inputs in a given slot is taken as the class label of that particular slot. However, a major limitation is that the slot sizes, slot range and the truncation operator must be determined *a priori*; these can be sensitive to the training data where poor initial settings can cause overfitting.

In CDCB [154], the class threshold is selected as the mid-point between two adjacent class *centres*, one for each class. A class centre is the average of the outputs when all individuals in the population are evaluated on all input instances from a particular class. While this approach requires no prior parameter configuration, the class threshold(s) depend on *all* individuals in the population at the current generation. This means that more training can be needed to converge on a good class threshold and accurate GP classifier.

In [186], the outputs of each individual is modeled using Gaussian distributions, one for each class, and a probabilistic technique is used to find the point(s) of *least overlap* between these class distributions. This approach requires no extra parameter configuration (unlike DRS and SDCB), is relatively fast to compute, and shows good results compared to the traditional (static) strategy on a range of multi-class tasks.

Fitness Function

In classification, the fitness function defines a measure to calculate the accuracy of a solution, by comparing the predicted class labels with the target (or actual) class labels in the training set. The traditional fitness function for classification uses the overall classification accuracy (this measure was previously shown in Section 2.1.3). Recall that this measures the number of examples correctly labeled by a classifier as a proportion of the total number of training examples.

However, using the overall classification accuracy in the fitness function is known to drive the evolutionary search toward biased classifiers which have high majority class accuracy but poor minority class accuracy, when data is unbalanced [141][57][88][172][130][69]. Also as discussed, this is because the overall accuracy can be influenced by the larger majority class. Two common approaches to address this learning bias in GP include using the average accuracy of the minority and the majority classes, or the AUC in the fitness function [141][57][88]. The reader is directed to Section 3.3 (in the next chapter) for a detailed analysis and discussion on the major advantages and disadvantages when these three fitness functions are used in GP for classification with unbalanced data.

Several other related approaches which develop new fitness functions in GP specifically for classification with unbalanced data are discussed in more detail later in this chapter in Section 2.6.2 (which focuses on related works for class imbalance problems).

GP for Ensemble Learning and Combining Classifiers

GP has also shown success is evolving ensembles of classifiers for classification with balanced data [110][111][126][29][165]. In [110][111], multiple trained classifiers obtained from several learning algorithms such as Naive Bayes, C4.5 decision trees and ANNs are combined into a single genetic program solution. Each base learner is trained using different partitions of the input data and adapted to output a real (floating-point) number (when evaluated on an input instance). Combining classifiers in this approach is shown to outperform the individual classifiers (in terms of AUC) on two benchmark binary tasks from the UCI repository. In [126], an anti-correlation measure is used in the fitness function along with the overall accuracy to encourage diversity between individuals, using a grammar-guided GP (on a 6-multiplexer problem). After the evolution, the entire population is combined into an ensemble.

Cooperative co-evolutionary methods in GP are used in [29] and [165] where

teams and individual programs are co-evolved in parallel. In [29], teams are evolved using linear GP and evaluated with several ensemble combination schemes on two benchmark classification tasks (from the UCI repository) and a regression task. To create teams, the population is divided into *demes*, and then sub-divided into teams of individual programs. The ensemble combination schemes include the average of each member's outputs, a majority vote and two winner-takes-all strategies; and two weighting schemes where teams and weights are co-evolved in parallel, or optimised after each generation (using a perceptron). The best combination scheme is found to be problem-specific (none shows the best results for all tasks) but the majority vote and weighting schemes consistly show good results.

In [165], four teaming-based selection methods are compared including a new class of "orthogonal evolutionary" selection algorithms (on two multi-class UCI classification tasks). In teaming, selection is done exclusively between teams or between individuals; while the new algorithms use individual selection with team replacement, and vice versa. The new methods produce better results than canonical methods but the best selection method is found to be problem-specific.

It is important to note two major differences between teaming in GP and the ensemble methods used in this thesis. Firstly, teaming produces teams of *weak* individuals that cooperate strongly together, as shown in both [29] and [165]. Weak individuals have very poor individual classification ability and are only effective when combined with other weak individuals in a team. Secondly, in teaming, two selection strategies are typically used (as discussed in [165]): selection of individuals within a team, and selection of teams. In this thesis, the GP classifiers are relatively strong individuals with good accuracy on the two classes, and individuals in the population are only combined into an ensemble after the training phase.

2.5.2 EMO for Classification

EMO for classification (with balanced data) can be categorised into three main areas: ROC optimisation, model regularisation, and ensemble learning. In ROC optimisation, the true and false positive rates are traded-off each against other in the learning systems. In ensemble learning, the accuracy and diversity of the base learners are typically traded-off against each other. In model regularisation, the overall classification accuracy is traded-off against a model regularisation term such as the complexity or size of the learnt models.

ROC Optimisation

The majority of related work in this area use hybrid EMO algorithms where a GA is used to optimise the parameters of an underlying base learner (such as ANNs or SVMs) [108][65][162][66]. Here individuals in GA encode free parameters in the base learners (e.g. the weights in an ANN), and each solution on the evolved Pareto front represents a particular configuration of these parameters in the base learner. These hybrid GA approaches typically use a bi-objective EMO (i.e. true positive rate *vs* false positive rate) to trace out the Pareto front. Some examples include [108] and [65], where the weights of an ANN with a *fixed* architecture are optimised using a GA, and the points along the evolved Pareto front correspond to the optimised ROC curve. Both of these works show that the GA-optimised ROC curves dominate the ROC curves produced by a traditional ANN with standard back-propagation. However, these approaches are only evaluated on simple synthetic binary classification tasks (with two features).

Hybrid GA approaches have shown success in real-world applications [162][66]. In [66], a bi-objective GA is developed to automatically tune the free parameters of a Conflicts Alert System for air traffic controllers. In [162], a two-step detection and classification approach of pedestrians in infrared images is developed for Driver Assistance Systems using SVM.

There are fewer works in this area using GP [187][140]. In [187], a bi-objective GP approach for evolving Pareto-optimal decision trees is developed. Sensitivity and specificity are used as the competing learning objectives with dominance rank in the fitness function. The AUC of the decision trees evolved from the multi-objective GP is shown to dominate traditional decision tree classifiers, ANNs and SVMs on two benchmark tasks from the UCI repository. In [140], three multi-objective formulations in GP are compared on three benchmark UCI (multiple-class) tasks. These formulations include using the error rates of each class as the competing objectives, using an additional parsimony objective with the accuracy-based objectives (from above), and trading-off the overall accuracy and size of the learnt models. The first accuracy-based formulation shows the best overall results from all three methods (and is also better than canonical GP), while the third formulation shows the worst results. However, in all three formulations, a single non-dominated solution (which has the lowest deviation of errors on all classes) is selected from the evolved Pareto front (to represent the output of the EMO system). This means that no other non-dominated solutions, nor the evolved Pareto fronts representing the different trade-off solutions, are examined or compared to traditional GP.

Ensemble Learning

In [54], the two main techniques (among others) to generate diverse and accurate ensembles are discussed. The first involves partitioning the input-space into smaller subsets (called *bootstrap samples*) which are used to train the individual base classifiers, such as bagging and boosting techniques [159][164][137]. The second involves injecting randomness into the learning algorithm, and is favoured in EMO due to its inherent stochastic and population-based nature [36][40][35].

In EMO, a common strategy to promote diversity between individuals uses an additional penalty term in the fitness function, such as Negative Correlation Learning (NCL) [36][40][35]. For more details on NCL, please refer to Section 6.2.3 (later in Chapter 6). These approaches differ from typical bagging and boosting techniques as the *full* training set is used in learning to promote interaction and cooperation in the ensemble, whereas bagging relies on sampling techniques to partition the training data into smaller subsets. This thesis uses EMO and GP for ensemble learning as this approach does not rely on sampling. Ensemble learning using bagging and boosting approaches are omitted from the discussion in this section (which focuses on EMO for ensemble learning) — a discussion on bagging and boosting approaches can be found later in this chapter (in Section 2.6.4) with other related works for classification with unbalanced data.

EMO approaches using an ensemble-diversity objective in the fitness function typically build the ensembles using the set of non-dominated individuals in the population [36][40][4][35]. In [35], the training accuracy is traded-off against the NCL in a two-objective EMO (called DIVACE) to evolve ANN ensembles. In [36], two diversity measures are compared in DIVACE to evolve ANN ensembles: NCL and a measure called Pairwise Failure Crediting (PFC). PFC is found to outperform NCL on two benchmark UCI tasks. In [40], EMO with three objectives is used to evolve a Radial Basis Function (RBF) network ensemble: the accuracy, NCL and the size of the learnt models.

Some EMO approaches use other mechanisms for ensemble diversity [95][4][3]. In [95], the structure of the ANN models (e.g. number of hidden nodes) is varied for diversity, and this is traded-off against the error rates of the ANN models. In [4][3][2], two multi-objective formulations are proposed for diversity in ANN-based ensembles. The first splits the training set into two subsets and uses the error on the subsets as the learning objectives; while the second adds Gaussian noise to the training set as the second objective. The first formulation shows better results than the second, and these methods are competitive compared to NCL on two (binary) benchmark classification tasks.

Model Regularisation

This category of work in EMO trades-off the overall accuracy and complexity of the learned models. Models with high complexity can overfit the learning data; while models with lower complexity can offer good generalisation ability, but very simple models also risk underfitting the data. *A posteriori* EMO techniques are typically used to find "compromise solutions" depending on model complexity and performance. As model regularisation techniques are not the focus of this thesis, only a brief list of related work is outlined below to give the reader an idea of the field.

In [70], a simple bi-objective hybrid GA algorithm is used to find the best set of weights for an MLP classifier using root mean square error (RMSE) and the number of free MLP weights as the regularisation term. In [92] and [50], multi-objective rule-based classification systems are developed using rule accuracy and generality as the competing objectives.

In GP, reducing code *bloat* is a large area of research [26][96][63][49][48][9]. Bloat occurs when the size of the evolved solutions grow rapidly without any clear benefit to fitness [48][63]. In classification, the overall accuracy can be traded-off against the program size (of the evolved GP classifiers) as the two competing objectives to drive the evolutionary search toward smaller high-performing solutions [26][96][49][9]. A similar EMO approach can also be used in program simplification in GP to promote smaller programs for easier program analysis [109][182].

In the above-mentioned approaches, the parsimony objective in fitness succeeds in driving the EMO search toward smaller solutions that perform as well as, or better than, models induced with no regularisation pressure, due to better generalisation from the EMO-induced solutions [26][96][70][92][50]. An advantage of the Pareto-based approaches for model regularisation is that the trade-off between the accuracy and the complexity of the learnt models can be analysed *a posteriori* via the Pareto front.

Comments on EMO for ROC Optimisation and Ensemble Learning

This thesis develops a multi-objective GP (MOGP) approach using EMO for cost adjustment (when data is unbalanced) by trading-off the minority and the majority class accuracies against each other in the learning process. This MOGP approach is different from the existing works discussed above which use EMO for ROC optimisation. Many existing approaches such as [108] and [65], use a

GA to evolve the *parameters* in a single ANN classifier. This means that each solution on the Pareto front represents a particular configuration of weights in the ANN classifier. However, in MOGP, each solution on the evolved Pareto front is a distinct GP classifier with a particular performance bias toward either the majority or minority class.

While some GP-related works in EMO also evolve a Pareto front of genetic program classifiers (such as [140]), here only a single classifier is extracted from the Pareto front to represent the learnt model from the EMO system. By discarding all but one of the evolved classifiers on Pareto front, this selection process reduces the EMO search to a traditional single-predictor learning system. This selection process also makes certain assumptions about the problem as the other (discarded) Pareto front solutions may also represent useful classifiers. However, this thesis utilises the set of genetic program classifiers on the evolved Pareto fronts in two important ways. Firstly, the full set of evolved Pareto front classifiers is used to evaluate the effectiveness of the MOGP system compared to a traditional single-predictor GP system. Secondly, the MOGP Pareto front is then combined into an ensemble, thus utilising the combined classification ability of the Pareto front of genetic program classifiers to further improve classification performances on the two classes.

While many related works in EMO (discussed above) also combine the evolved Pareto fronts into an ensemble for classification [36][40][35][95][4][3], these works all use GA to evolve ANN-based ensembles. Some works also use an ensemble diversity measure, NCL, in the fitness function to promote diversity between individuals [36][40][35]. This MOGP approach is different from the above-mentioned works as the ensemble-diversity measures used in the fitness function are adapted for genetic program classifiers, and for classification tasks with unbalanced data. In MOGP, diversity is measured separately for each class; otherwise these measures risk being biased toward the larger majority class.

2.5.3 Related Aspects in Ensemble Learning

While ensemble learning in EAs also represents a very large area of related work, this section provides a brief discussion of two key groups of related work that address specific concepts pertaining to this thesis. The first includes recent work which provides a theoretical analysis on how anti-correlation measures such as NCL work to promote diversity in a population of base learners in ensemble approaches [126][125], and the second includes recent developments in ensemble

selection strategies in EAs [76][180][138].

Recently, in [125][126], two anti-correlation measures, NCL and a new variation called root quartic NCL (rqNCL), are theoretically analysed to explain how each creates diversity in a population. The new variant rqNCL is found to create widely separated but small clusters of points in the population, while traditional NCL tries to increase the distances of the individual points to the overall mean of the points in the population. These measures are compared using a grammar-guided GP on a 6-multiplexer problem [126], and a benchmark binary classification task (Australian Credit) using an ANN ensemble [125]. The new variant (rqNCL) is found to show competitive results compared to traditional NCL, particularly for large ensembles.

Ensemble selection addresses the question of how to choose which classifiers to include in the final ensemble, given a large pool of learnt base classifiers [76][180][138]. In [76], offline and online (off-EEL and on-EEL) ensemble selection algorithms are proposed using a simple greedy search to construct the ensembles. Given a pool of learnt base classifiers sorted by fitness, each classifier is removed from the pool and copied into the ensemble where, at each step, the ensemble is evaluated (using a majority vote of the current ensemble). Once the pool is empty, the ensemble with the best performance is taken as the final ensemble. Off-EEL is run once after the training cycle; while on-EEL interleaves the ensemble learning and selection process, and is performed at each generation. Off-EEL is shown to outperform on-EEL on six benchmark UCI tasks, as the co-evolutionary approach on-EEL is found to be easily prone to noise, particularly in the early stages of the co-evolution.

In [138], the fittest individuals in the evolved population of ANN classifiers are selected for the final ensemble using a weighted average of the accuracy and diversity of each individual. In [180], a GA used to first train an ensemble of ANN classifiers, and then to optimise the weights specifies each ensemble member's contribution in the final ensemble. The GA-optimised ensembles are found to outperform two other ensemble selection schemes. The first uses a fitness-weighted majority vote, while the second uses a recursive least-square (RLS) algorithm to find the best ensemble members.

These ensemble selection approaches show that carefully selecting only the "best" individuals for the final ensemble is a non-trivial problem. Some use relatively simple ensemble selection algorithms [76][138], while others optimise the ensembles using a secondary training phase [37][180]. This thesis tries to address these issues using GP for classification with unbalanced data by devel-

oping two ensemble selection and optimisation strategies, and by comparing the effectiveness of these approaches to both off-EEL [76] and the traditional majority vote approach.

2.6 Related Work: Classification with Unbalanced Data

Traditionally, two main approaches are used to address the class imbalance problem. The first involves transforming, or sampling from, the original unbalanced data set to create a balanced class distribution in training. These are known as "external" approaches as the external training data is *re-balanced* while the learning algorithm remains relatively unchanged. The second approach uses various forms of cost adjustment within the learning algorithm to utilise the original unbalanced data *as is* in the training process. These are known as "internal" approaches as the learning algorithm is adapted to factor in the uneven class distributions. Some approaches also combine these two techniques, that is, sampling and cost adjustment in the learning algorithm [57][157][139][11][122].

A new and smaller area of work has recently emerged which focuses on gaining a better theoretical understanding of the nature of the class imbalance problem [93][159][173].

Ensemble methods using bagging and boosting with data-balanced techniques have also shown some success in class imbalance tasks, as the bootstrap samples (used to train the ensemble members) can be re-balanced using under- and over-sampling. These ensemble-based approaches are categorised separately from the three areas work discussed above as they use *multiple* classifiers in the final classification decision. In contrast, the above-mentioned works focus on canonical single-predictor learning algorithms where a single classifier is trained to represent the learnt model.

These four main approaches and their limitations are discussed below.

2.6.1 External Data-Balancing Approaches

The most common data-balancing techniques include *over-sampling* the minority class to boost representation by replicating known minority examples [41][139], and *under-sampling* the majority class to reduce majority class representation [11][122]. However, as over-sampling does not introduce any new information into the learning process, and under-sampling can discard potentially useful learning examples from the majority class, more robust sampling techniques

such as synthetic over-sampling and *editing* are also common [107][12][5][91]. Synthetic over-sampling of the minority class (known as SMOTE) creates "new" minority examples by interpolating between several similar examples [12], while *editing* carefully removes noisy or atypical majority class examples [107][5]. Editing techniques include using Euclidean distance measures between majority class examples in feature-space to eliminate noisy or atypical examples [5], and removing majority class examples along the border-line between opposite-class examples nearest to each other [107].

Data-transformation techniques such as self organising maps (SOMs) can also reduce the size of the majority class while preserving data topology [85]. SOMs reduce the complexity of the decision boundaries in feature-space to transform the majority class into a smaller set of examples.

Sampling in GP

In GP, other effective, although more complex, sampling approaches include Random Subset Selection (RSS) and Dynamic Subset Selection (DSS) [158][46][77]. In [158], a hierarchical two-tier sampling approach is used in LGP to identify good or bad internet connections for an intrusion detection system with more than half a million learning examples. First *blocks* of training examples are sampled using RSS, and then examples within those blocks are sampled using DSS. In [46], this work is extended by developing a family of hierarchical DSS algorithms such as cascaded RSS-DSS and balanced-block DSS for very large data sets. In [77], DSS is adapted to incorporate a bias toward difficult-to-classify examples while RSS is scaled toward minority instances; these methods are applied to benchmark multi-class tasks from the UCI repository.

Limitations of External Data-Balancing Approaches

While sampling techniques are effective in improving minority class performance, they have major limitations. Some sampling approaches can suffer from *poor generalisation* as potentially useful learning examples can be excluded from the learning process. Similarly, as many sampling techniques aim to artificially re-balance the training data prior to the learning process, the induced classification model might not be able to capture or learn the underlying rarities that occur in a particular problem. In other words, if the problem domain is inherently unbalanced, a model induced with artifically balanced training data may not learn to correctly identify the underlying rarities in the original problem domain.

For these reasons, recent work comparing sampling techniques to cost-adjustment shows that the latter can often outperform sampling methods across a variety of learning algorithms and problem domains [11][122][93]. These works also show that good results can be achieved using a combination of sampling and cost-adjustment as opposed to sampling on its own [139].

Another limitation is that many sampling or data transformation techniques require *a priori* expert knowledge about the data to develop appropriate sampling algorithms. For example, *a priori* knowledge can be needed about which learning instances are more important in the learning process and should not be omitted from the sampled sets, compared to other learning instances which can be omitted. Likewise, data transformation techniques can require complex representation to ensure minimum information loss.

Sampling can also add a computational overhead to the training process as in many cases, sampling must be applied repeatedly for good coverage.

2.6.2 Internal Cost Adjustment

For the reasons described above, much work within the machine learning community focuses on cost adjustment within the learning algorithm to factor in the uneven representation of class examples. Common approaches include assigning fixed misclassification costs to incorrect class predictions [88][142], or developing improved training criteria that are more sensitive to the unbalanced class distributions (compared to the standard overall accuracy or overall error rate). Improved training criteria include: the average classification accuracy of the minority and majority classes where the performance of both classes is considered equally important in fitness [107][139][5][116], the Area under the ROC curve (AUC) [27][84][149][90], statistical measures of accuracy such as the Wilcoxon-Mann-Whitney (WMW) statistic (to approximate the AUC) [179][57], and the F-measure widely used in information retrieval [34][69].

Fixed Costs

Fixed cost-based approaches tend to specify these costs *a priori* [88][142]. In [88], different ratios of penalty factors are compared which control the rate at which false negatives (FN) are penalised over false positives (FP), using a population-based rule-inducing classifier. Using real-world medical data with class imbalance, they show larger penalties for FNs can improve minority class accuracy. In [142], fixed class costs for misclassified examples are compared to the overall error rate to train three classifiers for a multi-class network

troubleshooting problem. Results show that the cost-reducing method reduced overall costs but produced higher error rates compared to the traditional error-reducing method. However, fixed class cost can be difficult to determine *a priori*, typically requiring a trial-and-error process on a task-by-task basis.

Class Sensitive Training Criteria

Much research in this area also studies what effects different training criteria have on the learned classifiers in class imbalance scenarios [130][173][179][27][34][69]. In four separate studies [130][173][179][27], the authors show that when data sets are unbalanced, using the overall accuracy as the training criteria can lead to biased classification performances by the learnt classifiers. However, when the class distributions are unbalanced, using the AUC in training can produce classifiers with better classification ability, where the AUC becomes increasingly disassociated from the overall error rate as the level of class imbalance increases. In [179], a Multilayer Perceptron (MLP) classifier, and two benchmark UCI tasks and a real-world churn-prediction data set, are used in the experiments. In [27], five machine learning algorithms such as k-nearest neighbour (KNN), decision trees and MLPs, and five tasks (with balanced and unbalanced data) from the UCI repository are used on the experiments. In [172], decision trees learners and 26 benchmark unbalanced data sets are used in the experiments.

In [34], nine well-known training criteria such as the AUC, F-measure, and mean square error (MSE) are compared and analysed in *metric-space* on three tasks from the UCI repository with unbalanced data. A new composite measure based on the average class accuracy, AUC and RMSE is found to be the most effective in training good solutions. Similarly, in [69], the correlation between 18 training measures such as the overall accuracy, several variants of the AUC, and several averaging functions (e.g. the F-measure and geometric mean), are studied using 30 binary and multi-class tasks from the UCI repository with balanced and unbalanced data. Ranking measures (e.g. AUC) are found to be the most effective in training classifiers with good class separability, and also the least correlated to both qualitative (e.g. accuracy) and probabilistic (e.g. WMW statistic or logloss) measures when data is unbalanced; whereas the different measures are all closely correlated when data is balanced. Both [34] and [69] used a variety of learning algorithms (e.g. NB, SVM, ANN, KNN and decision trees, but not any genetic-based learners) in their experiments.

Cost Adjustment in the Fitness Function in GP

Cost-adjustment in GP focuses on developing new fitness functions to reward solutions which have good accuracy on both classes with better fitness, and penalise those solutions which have poor accuracy on one class with poor fitness.

In [60], an adaptive cost-based fitness function is developed in GP to assign and periodically re-weight the error associated with certain hard-to-classify examples (similar to boosting), typically focused on minority class examples. This method improves overall classification performance compared to canonical GP on four benchmark UCI tasks but determining good initial costs is non-trivial.

To address this limitation, a fixed-cost fitness function is used in [6] where incorrect minority class predictions are penalised by the factor of the class imbalance ratio. This approach is applied to a bankruptcy prediction problem using data from Spanish companies generated between 1999 and 2000, and is shown to outperform canonical approaches without class-specific cost adjustment.

In [157], three new fitness functions are developed for a multi-class network intrusion detection problem using real-world transmission control protocol (TCP) dump data. The first uses the average accuracy for each minority class, the second uses a cost-based measure which dynamically re-adjusts weights for minority class examples, and the third uses a two-level fitness function where the accuracies on the two smallest classes are used to resolve ties for the primary criterion (the accuracies of the two largest classes). These methods, particularly the first two fitness functions, show good accuracy on some of the minority classes. However, the many free parameters in these approaches are considered a hindrance where performance is very sensitive to the initial settings; these methods also use sampling.

In [176], the weighted sum of three criteria are used in the fitness function for three multi-class tasks with unbalanced data (from the UCI repository). These measures include the overall error (weighted the highest), a new measure based on the difference between predicted and expected classifier outputs, and a separability-based measure (similar to the AUC). Their results show good accuracy across all minority classes. Similarly, in [57], two metrics are combined with an equal weighting in the fitness function for five multi-class tasks (from the UCI repository). A binary decomposition-based approach is used to split the multiple classes into n binary tasks; the two metrics include the geometric mean of the two accuracies of the two classes, and an estimate of the AUC using the WMW statistic. However, this approach also uses sampling in the learning process.

In [141], three new fitness functions for binary class imbalance tasks are developed. These use the average classification accuracy of the minority and the majority classes, except each used an increasing penalty for poor accuracy on one class only. Using two benchmark tasks (from UCI) and two synthetic tasks, the results show that, not surprisingly, the improved fitness functions show better minority class accuracies (but poorer majority accuracies) compared to the standard GP fitness function, and that larger penalties lead to better minority class performance. However, neither the AUC of the evolved classifiers nor the statistical difference between the three fitness functions is explored.

Limitations of Cost-based Approaches

While these approaches for cost-adjustment in GP are effective, there are four main limitations. The first is that misclassification costs for incorrect class predictions must usually be determined *a priori* [157][88][142]. These can be problem-specific and often require a trial-and-error process to determine an appropriate set of costs for each class. The second is that improved metrics in the fitness function (such as the AUC) can increase training times due to the computational overhead required to calculate these measures, particularly on large data sets [34][179]. The third limitation is that many new fitness functions are hand-crafted to suit a particular classification problem [157][60]. These can require expert or *a priori* knowledge about the problem domain, whereas problem-independent fitness functions are more desirable. Finally, while many approaches improve minority class performance at the expense of overall accuracy, this performance trade-off is not examined in any depth nor are any techniques proposed to address or exploit this trade-off. This thesis will try to address some of these limitations.

2.6.3 Theoretical Analysis in Class Imbalance Tasks

Work in this area tries to gain a better understanding of the nature of the class imbalance problem, and includes investigating the influence of other factors in the learning phase [93][145]. In [93], the level of class imbalance, complexity (class overlap), and the training set size, are varied in training using three learners (decision tree, MLP and SVM) for a synthetic (binary) classification task. It is shown that class imbalance is less of a hindrance in larger training sets and lower complexity problems for all three learners, while SVM is the least susceptible to the learning bias. However, a limitation in this study is that the test set

remains balanced in all the experiments; this can bias the performance of the learnt classifiers. A similar conclusion, i.e. that performance degradation is not solely due to the level of class imbalance but is related to the degree of complexity, is also shown in [145] for decision tree learners on another synthetic (binary) classification task.

2.6.4 Ensemble Methods

As discussed, combining bagging and boosting with sampling (such as under-sampling, over-sampling and SMOTE), to create *balanced* bootstrap samples is a popular approach to classification with unbalanced data [123][118][170][37]. This means that the base learners are trained using traditional measures such as the overall error or classification accuracy, as the bootstrap samples are artificially balanced. While bagging and boosting with balanced bootstrap sampling represents a large area of work for classification with unbalanced data; bagging has also recently been combined with EMO [123][124], and NCL in the fitness function [168] for class imbalance. These two main approaches (traditional bagging and boosting, and bagging with NCL and EMO) and their limitations are discussed below.

Bagging and Boosting with Balanced Bootstrap Sampling

Most work in this area uses ANNs, decision trees, NB or SVMs as the base classifiers in the ensembles. Some examples include the following. In [118], two new under-sampling methods are developed to create balanced bootstrap samples for a boosting algorithm with decision trees; these are compared to several other boosting approaches from the literature. A similar under-sampling approach is developed in [164] using SVMs where the support-vectors are iteratively learned on balanced bootstrap samples. Both [118] and [164] use benchmark tasks from the UCI repository. In [135], an ensemble of linear regression classifiers are trained using under-sampling and AdaBoost for a (binary) classification task from the PAKDD [1] data mining competition. In [159], a pool of decision tree, NB and rule-based base classifiers are combined into a "meta-classifier" for an e-Commerce fraud detection task. Here the base classifiers are trained using a combination of balanced and unbalanced bootstrap samples.

In [170] and [169], a decision trees-based bagging approach is developed where ensemble performances on balanced and unbalanced tasks are compared when the level of diversity in the ensembles are varied during training. Diversity

is varied by increasing the balanced bootstrap sample sizes where the smaller the bootstrap size, the better the diversity. These findings show that, as expected, the accuracies on the minority and majority class improve together when ensemble diversity is increased in *balanced* data sets. However, on the unbalanced test sets, ensembles with high diversity rates show high minority class accuracies but poor majority class accuracies. These experiments used eight binary and multi-class benchmark tasks from the UCI repository.

Bagging with NCL, and Bagging with EMO

Recently, bagging (with balanced bootstrap sampling) has been compared with NCL in the fitness function (for ensembles-diversity) to train ANN-based ensembles [168]. In this work, two formulations of NCL are evaluated in the fitness function (and compared to bagging) on the same UCI tasks as [169]. However, the original unbalanced data set is first *re-balanced* using sampling before NCL is measured. In the first formulation, NCL is applied to all training instances in the (re-balanced) training set; in the second formulation, NCL is only applied to minority class instances in the (re-balanced) training set while majority class instances are ignored. Both NCL-trained ensembles show better minority class accuracies than the bagging approach, and the first NCL formulation shows the best overall diversity from all three approaches. The authors attribute this to very high diversity on the minority class alone (e.g. the second NCL formulation) producing high minority class accuracies but poor majority class accuracies.

In [123][124], a co-evolutionary approach with bagging and EMO has been used in ensemble learning with grammatical evolution (GE). These works use a problem-decomposition approach (e.g. one-vs-rest) to evolve a population of classifiers for two multi-class tasks (from the UCI repository) with many minority classes. Two populations are co-evolved for ensemble diversity: (binary) classifiers and "points" (which are balanced bootstrap samples). The Pareto based learning objectives in fitness include the overall error of each classifier, the level of overlap between correctly learned "points", and a parsimony objective favouring smaller solutions. A winner-takes-all approach of the Pareto front determines the final ensemble prediction. This approach is shown to outperform traditional single-objective GP on the tasks.

In one related work [37], a bagging approach with unbalanced bootstrap samples are used for ensemble learning where base classifiers from 16 different learning algorithms are trained with the F-measure as the training criteria. However, this work focuses on ensemble selection where a second training

phase (using GA) optimises the weights that specify which base classifiers are represented in the final ensemble. Experiments on five tasks with unbalanced data from the UCI repository show that the GA-based ensemble selection strategy outperforms two previous approaches: a fitness-weighted majority vote strategy, and a traditional majority voting approach where all members contribute equally in the ensembles. However, this approach represents the only related work which does not use balanced bootstrap sampling.

Limitations of Ensemble Methods

While these approaches show good results on some unbalanced data sets, there are some limitations which this thesis tries to address. Most works uses ANNs, decision trees and NB as the base classifiers [159][168][40][170][37]. In addition, these works rely on sampling techniques to either create balanced bootstrap samples in bagging [159][123][124][170], or re-balance the training data when diversity measures (such as NCL) are used in fitness evaluation [168].

GP has shown much success in evolving reliable and accurate classifiers for traditional single-predictor classification [176][157][60][141][57][77]. However, there is very little related work, particularly in GP, which does not rely on sampling techniques (for cost adjustment) when data is unbalanced. This thesis uses the original unbalanced training data directly in the GP learning process (using the EMO component for cost adjustment), without the need to first artificially re-balance the data. This allows us to concentrate on the cost-adjustment and diversity measures in GP, and remove the dependence on a sampling algorithm.

There has also been very little work which focuses on adapting the ensemble diversity measures in fitness to account for the skewed class distributions [168]. As discussed, most related works measure diversity relative to all examples, irrespective of class, as the classes are first *re-balanced* using sampling [168]. While the work in [168] measures NCL separately for the two classes, diversity on the majority class is ignored. In contrast, this thesis compares two ensemble-diversity measures in the fitness function (NCL and PFC) where diversity is calculated separately for each class using the original unbalanced training data, and diversity on the minority and the majority classes then contributes *equally* in fitness evaluation. This is to ensure that the ensembles are equally diverse on both classes.

2.7 Summary

This chapter presents the background and related work for the main topics used this thesis. The background covers the fundamental concepts in machine learning and classification, evolutionary computation focusing on genetic programming (GP), evolutionary multi-objective optimisation (EMO), and ensemble learning. The related work focuses on recent advances in GP and EMO for classification, and class imbalance learning with particular emphasis on GP and ensemble-based approaches. The related work in each of these areas also highlights the limitations/criticisms of the current approaches, and the challenges that this thesis attempts to address.

In classification with balanced data (the first part of the related work), emphasis is given to how to represent classifiers in GP, developing "improved" classification strategies and fitness functions in GP, and GP for ensemble learning. EMO for classification is categorised into three main areas: ROC optimisation where the true and false positive rates are traded-off each against other during learning, ensemble learning where the accuracy and diversity of the base learners are traded-off against each other, and model regularisation where the accuracy (of a learnt model) is traded-off against a model regularisation term.

The second part of the related work for class imbalance learning is categorised into three main areas. These include sampling-based techniques used to artificially re-balance the training data prior to learning, cost adjustment techniques used within the learning algorithm to factor in the unbalanced classes during learning, and ensemble-based approaches using bagging and boosting (and other) techniques.

2.7.1 Next Chapter

The next chapter outlines the GP approach to classification where the evolved GP classifiers are represented as mathematical expressions. Particular emphasis is placed on how to develop classification strategies and fitness functions in GP. A classification strategy determines how an output value for a genetic program is mapped to a class label, while the fitness function defines a measure to calculate the overall performance of a solution. The next chapter compares whether the traditional (static) classification strategy in GP is good enough on the (binary) unbalanced tasks compared to an "improved" (non-static) strategy, and highlights the main advantages and limitations of the three main current approaches in the fitness function in GP on the classification tasks.

Chapter 3

GP Approach for Classification

This chapter is organised as follows. The first section outlines the chapter introduction and goals. The second section discusses the GP approach to classification, and introduces the two main classification strategies in GP. The third section introduces three existing fitness functions for classification. The fourth section presents the experimental results comparing the classification strategies and fitness functions in GP on the tasks. The fifth section summarises the main ideas in this chapter.

3.1 Introduction

This chapter develops a GP approach to classification focusing on two important aspects in the evolution. The first aspect is how to develop an effective classification strategy, and the second is how to develop a good fitness function, for classification with unbalanced data. By representing the evolved GP classifiers as mathematical expressions, a GP classifier computes a single output value (floating-point number) when evaluated on an input instance from the data set. The classification strategy determines how this output value is translated to a set of binary class labels (for the minority and the majority classes) in these tasks. Generally speaking, the fitness function determines how well an evolved solution solves a given problem.

In classification tasks, the standard GP fitness function measures the overall accuracy of a solution, that is, the number of input instances assigned the correct class label by a given solution as a proportion of all input instances in the training set. However, when data sets are unbalanced, biased classifiers with high majority class accuracies but poor minority class accuracies are often evolved using this fitness function in GP [141][57][173]. This is because this standard

measure of performance can be influenced by the larger number of examples from the majority class. Research shows that adapting the fitness function to use improved training criteria that are more sensitive to the smaller minority class can improve performances on the important minority class when data is unbalanced. Two common approaches for cost adjustment to account for the unbalanced classes use the average accuracy of the minority and majority classes, and the area under the ROC curve (also known as the AUC), in fitness. The AUC is a particularly useful measure of performance when data is unbalanced as it represents how well a learned classifier approximates the trade-off between the minority and majority classes across multiple classification thresholds (using the ROC curve) [27][84][90]. As a result, it is invariant to the unbalanced classes (unlike the standard measure, the overall accuracy).

3.1.1 Classification Strategies

The traditional classification strategy in binary classification tasks defines a static class threshold for all solutions in the population, typically using the natural division between positive and negative numbers to represent the two class labels (i.e. zero is fixed as the class threshold). However, recent improvements to the classification strategy in GP uses dynamic class boundaries, determined on a solution-by-solution basis for each solution in the population [186][185][154][120]. This dynamic (non-static) strategy has been shown to evolve better-performing solutions than the traditional static strategy on a range of tasks with multiple balanced classes, as it offers greater flexibility in the evolution. That is to say, the non-static class boundaries do not need to be defined *a priori*, since these are determined a solution-by-solution basis.

However, the effectiveness of a dynamic (non-static) or static strategy has not previously been compared in the context of binary classification with unbalanced data, i.e., where there are only two classes and one class has a smaller number of examples than the other. Previous works (such as [186][185][154][120]) have compared these two strategies on classification tasks with multiple *balanced* classes and, as a result, only the standard GP fitness function was considered in the evolution. However, this particular fitness function is not suitable when data sets are unbalanced (as discussed above). Therefore, it is not clear whether the improved non-static strategy will also outperform the traditional static strategy (in terms of performance of the evolved solutions) on these binary tasks with unbalanced data, and when different fitness functions are used in the evolution

to account for the unbalanced classes.

This chapter compares which of these two classification strategies in GP finds solutions with better AUC on these tasks, when the standard fitness function and the two improved fitness functions are used in the evolution. To be more specific, this chapter investigates whether the traditional (static) strategy is sufficient on these tasks compared to the dynamic (non-static) strategy, in terms of the AUC of the evolved solutions. This chapter also examines the main differences between the evolved GP solutions using the three fitness functions (in terms of their AUC, minority and majority class accuracies, and overall classification accuracy) to highlight the advantages and limitations of these fitness functions, and why they need to be improved. This chapter also demonstrates how the overall accuracy of a solution can be a misleading measure of performance on these unbalanced tasks compared to the AUC which is invariant to the unbalanced classes.

3.1.2 Chapter Goals

This chapter has two mains goals. The first goal is to develop a GP approach to classification, with particular emphasis on the classification strategy to investigate whether the traditional (static) strategy is sufficient for these tasks compared to the dynamic (non-static) strategy.

The second goal is to highlight the advantages and limitations of three well-known current approaches in the fitness function, by examining the performances of the evolved classifiers on the unbalanced tasks.

3.2 GP Approach to Classification

This section outlines the GP representation, discusses the main difference between static and non-static classification strategies in GP, and presents the non-static GP strategy used in this study.

3.2.1 GP Representation

A tree-based structure is used to represent genetic programs [104]. The terminal set consists of feature terminals (features from the data set) and constant terminals (randomly generated floating point numbers). The function set consists of the four standard arithmetic operators and a conditional operator (`if`) as defined below:

$$\{+, -, \%, \times, \texttt{if}\}$$

The $+$,$-$ and $\times$ operators have their usual meanings (addition, subtraction and multiplication) while % means *protected* division. Protected division is the typical division except that a divide by zero gives a result of zero. The four arithmetic operators all take two arguments and return one. The conditional `if` function takes three arguments. If the first is negative, the second argument is returned; otherwise it returns the third argument. The `if` function is included in the function set as this allows the solutions to contain a different expression in different regions of feature space, and allows discontinuities rather than insisting on smooth functions.

As the terminals and the return types of all the functions are numeric, the genetic programs represent mathematical expressions. For example, the Lisp expression `(- (+ F1 F2) 0.5)` for a genetic program represents the mathematical expression $(F_1 + F_2) - 0.5$. In this expression, the arithmetic operators ($-$ and $+$) are the functions, and F_1, F_2 and 0.5 are the feature terminals and the constant terminal.

3.2.2 Classification Strategies in GP

Using this GP representation, a genetic program classifier represents a mathematical expression which computes a single output value (floating-point number) when evaluated on a particular input instance from the training or test sets. This floating-point number must then be mapped or translated to a set of class labels. As previously discussed, the traditional technique to translate this number into a binary class label defines *zero* as the class threshold, using the natural division between positive and negative numbers to represent the two class labels. For the zero-threshold classification strategy, a data example will be assigned to the majority class if the classifier output is negative, otherwise it will be assigned to the minority class.

Non-static Class Threshold

Recent research in GP, particularly for classification with more than two classes, has shown that the static zero-threshold (ZT) strategy can place additional constraints on the solutions during evolution [186][185][154][120]. This is because the class boundaries are not relative to the output values unique to each solution, but remain fixed throughout the evolution. The evolved solutions

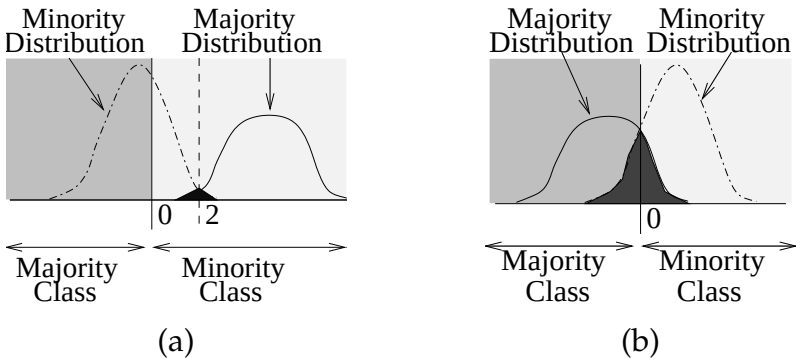


Figure 3.1: Distributions of minority and majority class outputs for two GP solutions (output values along the horizontal axis) and target class regions.

are required to not only separate their outputs for each class, but also ensure that these outputs adhere to a pre-determined ordering, i.e., minority class outputs must be non-negative while majority class outputs must be negative. These previous works argue that a *non-static* classification strategy, where the class boundaries are different for each solution (based on its output values), can lead to better performing classifiers being discovered in the evolutionary process [186][185][154][120]. It is argued that non-static classification strategies can potentially identify good solutions early in evolution, whereas the static ZT strategy risks overlooking/disregarding solutions which do not conform to the ZT class boundaries.

For example, consider two evolved genetic program classifiers whose outputs on the two classes are represented by Figures 3.1(a) and 3.1(b). Each figure shows the distributions of output values when the classifier is evaluated on the input instances from the two classes, where the horizontal axis corresponds to the output values and the height of each distribution represents the relative *frequency* of observations for the output values. In Figure 3.1(a), the two class distributions are separated with relatively little overlap between the distributions (darkly-shaded area in figure). However, using the ZT strategy, this classifier maps most of its outputs to the incorrect class labels (as shown by the two class boundary regions in this figure). Consequently, this classifier will exhibit a poorer fitness (as deemed by the fitness function) than the classifier in Figure 3.1(b), which maps more of its outputs to the correct class labels but which also has poorer class separability (larger overlap between distributions) than Figure 3.1(a).

Clearly, selecting 2 as the class threshold in Figure 3.1(a) represents a better "split point" between the two class distributions. In this case, the majority class label is returned when an output value is greater than 2; otherwise, the minority

class label is returned. Using these class boundaries, more outputs are mapped to the correct class labels than the ZT strategy for the same classifier.

Static or Non-static Classification Strategy in GP?

The above-mentioned examples show how a non-static classification threshold can potentially identify better-performing solutions than the traditional (static) strategy. However, in binary classification, GP should be able to automatically "shift" the outputs of the evolved solutions in the population to conform to the ZT strategy using the evolutionary process. GP can accomplish this for two important reasons. Firstly, in binary classification, only two distinct class ranges are required; while in multi-class tasks, there are more ranges (or intervals) to consider. Secondly, GP can tweak the mathematical expressions representing the genetic program solutions using the genetic operators in the evolution.

For example, let a genetic program p represent the expression $(F_1 + F_2) - 0.5$ where F_1 and F_2 are features terminals and 0.5 is a constant terminal. Assuming that p outputs values in the range $[5, 10]$ when it is evaluated on examples from majority class, and values in the range $[10, 15]$ for the minority class. If a mutation or crossover operation on the root node of p creates a new solution p' during evolution where $p' = p - 10$, then the outputs of p' will lie in the range $[-5, 0]$ for the majority class and $[0, 5]$ for minority class. The solution p' will now output negative values for majority class examples (except those outputs that are exactly 0) and non-negative numbers for minority class examples. The new genetic program p' represents the expression $((F_1 + F_2) - 0.5) - 10$ which can simplified to $(F_1 + F_2) - 10.5$.

The first goal of this chapter tests whether GP can accomplish this, i.e., "shift" the outputs of the evolved solutions relative to the ZT strategy during the evolution, to a sufficient level of accuracy compared to the non-static strategy. This hypothesis is tested by comparing the AUC performances of the evolved GP solutions for both strategies, when both strategies are given the same number of generations to evolve a good solution.

3.2.3 Non-static Classification Strategies

Recently, several non-static classification strategies in GP have been proposed for classification with multiple classes [186][185][154][120]. These strategies implement different techniques to accomplish the same end-goal in the evolution, i.e., to automatically determine dynamic class boundaries for a given solution in

the population. This study uses the approach from [186] to determine the best "split point" between two class distributions for a given solution. This approach uses a probabilistic technique to find the point of *least overlap* between the class distributions for a given solution, to represent the class threshold. This approach is described in more detail below, followed by a brief discussion on why this approach is selected over the other non-static classification strategies from the literature.

Probability-based Non-static Classification Strategy

The probability-based classification strategy [186] models the outputs of the genetic program classifiers using two Gaussian distributions, one for each class, and uses the *probability density function* of the class distributions to determine the class label for a given input instance. The probability density function ϕ is shown by Eq. (3.1), where N_c is the number of examples in class c, and μ_c and σ_c are the mean and standard deviation, respectively, of the genetic program outputs $P_{c,i}$ (when a solution p is evaluated on all training examples i from class c). In this equation, x is the (real-valued) genetic program output when a solution is evaluated on an unknown input instance (i.e. instance to be classified).

$$\phi(\mu_c, \sigma_c, x) = exp\left(\frac{-(x-\mu_c)^2}{2\sigma_c^2}\right).\frac{1}{\sigma_c\sqrt{2\pi}} \tag{3.1}$$

where

$$\mu_c = \frac{\sum_{i=1}^{N_c} P_{c,i}}{N_c} \quad \text{and} \quad \sigma_c = \sqrt{\frac{1}{N_c}\sum_{i=1}^{N_c}(P_{c,i}-\mu_c)^2}$$

In the training phase, the two class distributions are constructed for a given solution, by evaluating the solution on all training examples from the two classes (to obtain the μ_c and σ_c values for ϕ for both classes). To assign a class label to a particular input instance x from the training set, two ϕ values are calculated, one for each class distribution, for the given input instance. The class with the higher ϕ value is taken as the class of that particular input instance, where the higher ϕ value reflects which of the two class distributions the output value for x is more likely to belong to.

This technique finds the point of *least overlap* between these two class distributions, as shown in Figure 3.2. This figure[1] shows the two class distributions for a GP solution where the output values are plotted along the horizontal axis, and

[1]The two class distributions in this figure correspond to the actual output values for an evolved classifier from a particular GP run on the Ped task.

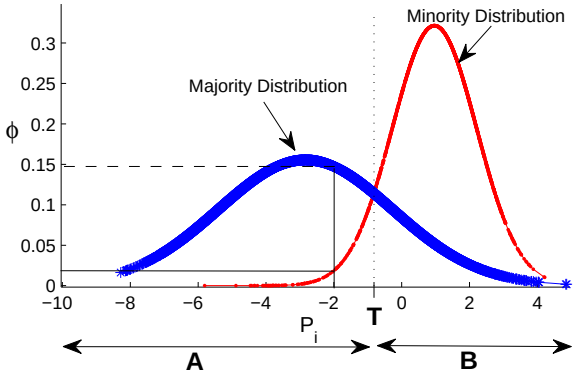


Figure 3.2: Distributions of class outputs for a GP solution and ϕ values of the outputs $P_{c,i}$ for the two classes.

the vertical axis shows the probability density function (ϕ) for the output values. When an output value lies in the region A, the majority class label is returned as ϕ is larger for the majority class distribution (than the minority class distribution). Similarly, when an output value lies in the region B, ϕ is larger for the minority class distribution (than the majority class distribution), thus the minority class label is returned. For example, when the output value is -2, ϕ for the majority class distribution will be approximately 0.15; whereas ϕ for the minority class distribution is under 0.05, as shown in Figure 3.2.

At the point of overlap between these two class distributions, T in Figure 3.2, both ϕ values will be the same; this represents the class threshold. When an output value is identical to the class threshold T, the minority class label is returned (similar to the case when an output value is exactly zero for the static ZT strategy).

A similar procedure is followed to obtain the class label for an unseen input instance from the *test* set. Two ϕ values are calculated for the genetic program output x, one for each class distribution using the μ_c and σ_c values of the solution's outputs on the *training set*. The class with the higher ϕ value is taken as the class of that particular example.

Justification of Probabilistic Strategy for Binary Class Imbalance Tasks

Other popular non-static classification strategies in GP include Centred Dynamic Class Boundaries (CDCB) [185], Slotted Dynamic Class Boundaries (SDCB) [154], and Dynamic Range Selection (DRS) [120] (see Section 2.5.1 in Chapter 2 for more details on these works). The probabilistic approach described above is chosen for

this study for three main reasons.

Firstly, this method is not influenced by the unbalanced classes in these tasks since the output values for the two classes as modelled using two Gaussian distributions. This allows the outputs from both classes to be treated as equally important when calculating the probability density function (ϕ), as each ϕ value is calculated relative to the μ_c and σ_c values for the minority and majority classes alone. In contrast, both the SDCB and DRS methods are "slot"-based; this means that each slot contains the output values from both classes. When the classes are unbalanced, the larger majority class can influence the class label of each slot, as the class with the most number of examples in a given slot determines the slot's class label. Secondly, this probabilistic method requires no extra parameter configuration whereas both "slot"-based methods require some *a priori* parameter configuration, e.g., size of each slot, total number of slots, and range values of slots. These parameters can be problem-specific and require a trial and error process to configure. Finally, this non-static classification strategy is relatively fast to compute and does not add a significant cost to the GP training times compared to the ZT strategy. This probabilistic strategy requires only one additional pass through the fitness cases to compute the μ_c and σ_c values for the two class distributions during fitness evaluation.

3.3 Fitness Functions in GP

The static and non-static classification strategies (discussed above) determine *how* a class label is assigned to a particular input instance. The fitness function is different; this defines a measure to calculate the accuracy of a solution by comparing the predicted class labels (as returned by a particular classification strategy) with the target (or actual) class labels.

This section outlines three typical current approaches in the fitness function and discusses the advantages and limitations of each. The first is the standard fitness function for classification: the overall classification accuracy. The other two are improved fitness functions for classification with unbalanced data: the average accuracy of the minority and majority classes, and the AUC.

3.3.1 Overall Accuracy in Fitness

The traditional measure for classification, Acc shown below, uses the overall classification accuracy in the fitness function (as discussed in Section 2.5.1 in

Table 3.1: Outcomes of a two-class classification problem.

	Predicted Positive Class	*Predicted* Negative Class
Actual Positive Class	True Positive (TP)	False Negative (FN)
Actual Negative Class	False Positive (FP)	True Negative (TN)

Chapter 2). Using Table 3.1, this corresponds to the number of examples correctly predicted by a classifier as a proportion of the total number of training examples. Note that the same confusion matrix as Table 3.1 is also shown in the the previous chapter (Table 2.1 on page 16) but is repeated here for convenience.

$$Acc = \frac{TP+TN}{TP+TN+FP+FN} \tag{3.2}$$

In Acc, fitness values range between 0 and 1 where 0 is very poor overall accuracy and 1 is perfect classification accuracy. As Acc treats all correct predictions as equally important in fitness, the larger number of examples from the majority class can influence the overall accuracy, rewarding *biased* solutions with high fitness values [173][179][130].

For example, consider a data set that contains 100 instances where 10 belong to the minority class and the rest (90) belong to the majority class. Using this fitness function, a trivial solution p which classifies *all* the instances as belonging to the majority class, can score a relatively high fitness, 0.9 (shown by Acc_p below). An alternative solution q with better discrimination ability between examples from the two classes, e.g., which correctly classifies 8 minority class examples and 72 majority class examples, scores a lower fitness value, 0.8 (shown by Acc_q below). Even though q has good accuracy rates on both classes, its fitness is lower than the biased solution p and thus, q will have a lower selection probability than p in the evolution.

$$Acc_p = \frac{0+90}{0+90+10+0} = \frac{90}{100} = 0.9$$
$$Acc_q = \frac{8+72}{8+72+2+8} = \frac{80}{100} = 0.8$$

3.3.2 Average Class Accuracy in Fitness

To promote solutions which have better accuracy on *both* classes, the fitness measure Ave (Eq. 3.3) uses the average classification accuracy of the minority and majority classes in fitness.

$$Ave = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right) \tag{3.3}$$

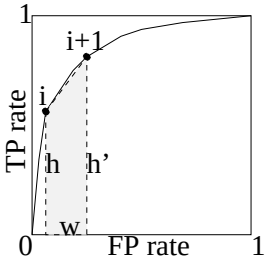


Figure 3.3: (a) Shaded area is the trapezoid fitted under two points on an ROC curve where w is the width, and h and h' are heights of the trapezoid.

In Eq. (3.3), the accuracy of each class is treated separately in the fitness function, where both contribute equally to the final fitness value. Using the example above, the biased classifier p will now have a poorer fitness of 0.5 (shown by Ave_p) than solution q which has a fitness of 0.8 (shown by Ave_q). Solution q has a higher fitness because it has a better accuracy across both classes.

$$Ave_q = \tfrac{1}{2}(\tfrac{0}{0+8} + \tfrac{90}{90+0}) = \tfrac{1}{2}(\tfrac{0}{10} + \tfrac{90}{90}) = \tfrac{1}{2}(0+1) = 0.5$$
$$Ave_q = \tfrac{1}{2}(\tfrac{8}{8+2} + \tfrac{72}{72+18}) = \tfrac{1}{2}(\tfrac{8}{10} + \tfrac{72}{90}) = \tfrac{1}{2}(0.8+0.8) = 0.8$$

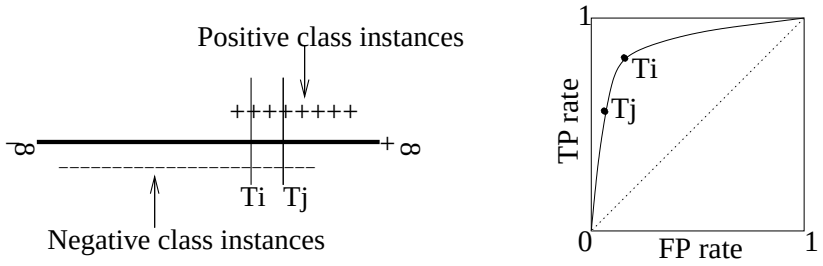
3.3.3 Area under the ROC curve

While Ave can find solutions with better minority class accuracies than the standard Acc, a major limitation of both these measures is that they represent the performance of a solution when it is evaluated using a *single* class threshold. In contrast, the area under the ROC curve (or AUC) measures the classification performance at *multiple* class thresholds. The AUC measures the overall quality of a classifier when the threshold parameter biasing the final classification decision is varied [130].

$$Auc = \sum_{i=1}^{N-1} \tfrac{1}{2} \left(FP_{i+1} - FP_i\right)\left(TP_{i+1} + TP_i\right) \tag{3.4}$$

In Eq. (3.4), N is number of class thresholds and TP_i/FP_i represent the performance of the solution at class threshold i. The equation sums the area of the individual trapezoids[2] fitted under the ROC points, as shown in Figure 3.3 for two ROC points. This measure returns values between 0 and 1 where the higher the value, the better the performance. The AUC corresponds to the

[2]The area of a trapezoid is $\frac{1}{2}w(h+h')$ where w is the width, and h and h' are heights the trapezoid [27].



(a) (b)

Figure 3.4: (a) Numeric outputs of a GP solution when it is evaluated on the input instances, where + and - denote the positive (minority) class and negative (majority) class outputs, respectively, and T_i and T_j are two different class thresholds; (b) an ROC curve with two points.

probability that a minority class example is correctly predicted across different class thresholds [84]. As mentioned, the AUC is a particularly useful and common measure of performance in classification tasks with unbalanced data as it represents how well a learned classifier approximates the trade-off between the minority and the majority classes across multiple classification thresholds. The following procedure is used to generate an ROC curve for a given GP solution.

a) Evaluate the solution on all the input instances from both classes to obtain the numeric output values (this requires one full pass through the input instances). Store the numeric output values *separately* for the two classes (e.g. in two separate array structures).

b) For each class, sort the numeric output values (stored in the arrays) in ascending order. For example, Figure 3.4(a) shows the (sorted) numeric output values for the two classes when a GP solution is evaluated on the input instances, where + and - denote the positive (minority) class and negative (majority) class outputs, respectively.

c) Build an ROC curve for each classification threshold value T:

   1. Initialise T (i.e. the first class threshold) as the lowest output value for the positive (minority) class.
   2. Iterate through the *positive* class outputs to count the number of outputs that are greater than, or equal to, T (i.e. TPs). For example, using T_i as the current threshold in Figure 3.4(a), seven out of eight positive class outputs satisfy this constraint (i.e. $\geq T_i$) giving a TP rate of $\frac{7}{8}$ (0.875).

3. Similarly, iterate through the *negative* class outputs to count the number of outputs that are greater than, or equal to, T (i.e. FPs). For example, using T_i as the class threshold in Figure 3.4(a), six out of 26 negative class outputs satisfy this constraint (i.e. $\geq T_i$), producing a FP rate of $\frac{6}{26}$ (0.23). The TP rate (from the previous step) and this FP rate correspond to one point on the ROC curve, as shown in Figure 3.4(b).
4. Update the threshold T for the next iteration. The new threshold is the lowest output value from either the positive class or negative class output values that is greater than T.
5. Repeat steps (2) to (4) until the largest output value for the positive (minority) class is reached. Each step produces one ROC point, e.g., using T_j is the class threshold in Figure 3.4(a) will produce another ROC point, as shown in Figure 3.4(b) for T_j.

d) Use Eq. (3.4) to calculate the final AUC value for the ROC curve.

A major limitation of Auc in the fitness function is the increased training times, due to the computational overhead required to construct an ROC curve. Once a solution in evaluated on the input instances (i.e. after step (a) in the above procedure), the extra computational overhead is due to two main factors that are *not required* in the calculation of the Acc and Ave measures. These factors are: sorting the output values for the two classes (i.e. step (b) in the above procedure), and *multiple* iterations over these output values to obtain the different ROC points (i.e. step (c) in the above procedure). When the distance between class thresholds T_i and T_{i+1} is small, more points on the ROC curve are generated (compared to larger distances), allowing for a highly accurate AUC estimation.

While the above procedure can be optimised using more efficient programming/optimisation techniques to speed up the calculation, this is not used in this thesis. This is because the full procedure (shown above) represents the traditional "out-of-the-box" method to calculate the AUC (as outlined in [27]). For example, a more efficient technique to count the TPs and FPs at a given class threshold T (after the output values are sorted) in step (c) would be to only count the number of TPs and FPs in region r where $T_{i-1} < r \leq T_i$, and reconcile these with the TPs and FPs from the previous iteration (for the class threshold T_{i-1}). In this case, given the number of TPs from the previous iteration (call this tp_{i-1}) and the number of TPs in the region r for the current iteration (call this tp_r), the final number of TPs in the current iteration (call this tp_i) would then be $tp_i = tp_{i-1} - tp_r$ (and likewise for the FPs).

However, in the next chapter, several new measures are developed to approximate the AUC in fitness but with faster training times, and these compared the traditional AUC measure. One of these includes a lower-precision AUC measure where the distance between class thresholds is increased (to speed up the calculation).

3.4 GP Experimental Results

This section first outlines the evolutionary parameters used in the GP experiments, then presents the full experimental results and analysis. The full experimental results are split into two parts to address the two chapter goals. In the first part, the AUC of the evolved GP solutions using the traditional ZT strategy is compared to the non-static strategy for the different fitness functions, to establish that the traditional ZT strategy performs as well as the non-static strategy on these tasks. In the second part, the main differences between the evolved GP solutions are examined (in terms of their AUC, minority and majority class accuracies, and overall accuracies) for the three fitness functions with the traditional ZT classification strategy.

3.4.1 Evolutionary Parameters

The ramped half-and-half method is used for generating programs in the initial population and for the mutation operator [104]. Crossover, mutation and elitism rates were 60%, 35% and 5%, respectively, and the tournament selection method is used with a tournament size of 7. The maximum program depth is 8 to restrict very large programs in the population. The population size is 500. The evolution is allowed to run for a maximum of 50 generations, or is terminated early if a solution with an *optimal* fitness value is found. For the three fitness functions defined above, the optimal fitness value is 1 for a given solution.

This configuration reflects the recommended settings from the literature [111][144][26], and is sufficient to train GP classifiers with good AUC on the tasks. The experimental results aim to compare the different fitness functions and classification strategies using a good configuration of evolutionary parameters for all tasks. For this reason, *fine-tuning* this configuration of evolutionary parameters is outside the scope of this work.

3.4.2 Comparing Classification Strategies

To address the first goal in this chapter, the experimental results in this section compare the traditional (static) ZT strategy and non-static threshold (NST) strategy in GP. This comparison focuses on the AUC of the evolved classifiers for both strategies using Acc (Eq. 3.2), Ave (Eq. 3.3) and Auc (Eq. 3.4) in the fitness function. As GP is a stochastic algorithm, each GP experiment is repeated 50 times using a different random starting seed in each run. Table 3.2 shows the average AUC of the fittest evolved classifiers for both methods on the *test* sets over 50 GP runs. The AUC of an evolved solution is calculated using Eq. (3.4).

The common random numbers technique [115] is used to test which classification strategy (for a given fitness function and task) achieves a statistically *significantly better* AUC at a 5% level of significance over 50 runs. The common random numbers method compares the difference in AUC between the two GP systems on a run-by-run basis (over 50 runs for a task), and outputs a 95% confidence interval of the AUC differences [115]. In Table 3.2, the classification strategy with the (statistically) significantly better AUC for given fitness function is highlighted in bold, according to the confidence intervals for the tasks. If neither strategy is highlighted in bold for a given fitness function and task, then there is *no significant difference* in AUC.

Analysis of the Differences Between ZT and NST

Table 3.2 shows that for each fitness function, neither classification strategy *consistently* outperforms the other (in terms of AUC) on the tasks. In two tasks (Ion and Spt), the ZT strategy is at least as good as, or statistically significantly better than, the NST for all fitness functions. In one task (Ped), the NST strategy is better than the ZT strategy for two fitness functions (Acc and Auc), although the difference in AUC for fitness function Auc is very small (0.01). In two tasks (Yst_1 and Yst_2), the ZT strategy is significantly better for fitness function Auc; while the NST strategy is significantly better for fitness function Ave. This shows that while the NST strategy is better than ZT for classification with multiple balanced classes, this is not always the case for these binary tasks with unbalanced data. Rather, the differences between the ZT and NST strategies in these tasks does not follow any clear pattern, where both strategies are competitive with respect to each other on different tasks. This suggests that GP can sufficiently tweak the mathematical expressions representing the GP solutions to "shift" its outputs relative to the ZT strategy in the evolution in many tasks.

Table 3.2: Average AUC (± standard deviation) of evolved classifiers using a fixed zero-threshold (ZT) and non-static threshold (NST) classification strategies (statistically significantly better AUC highlighted in bold) over 50 GP runs.

Task	Fitness Function	Static (ZT) Strategy	Non-Static (NST) Strategy
Ion	Acc	**0.82 ± 0.06**	0.74 ± 0.11
	Ave	**0.80 ± 0.06**	0.77 ± 0.10
	Auc	0.85 ± 0.04	0.86 ± 0.06
Spt	Acc	**0.72 ± 0.06**	0.64 ± 0.09
	Ave	**0.71 ± 0.05**	0.68 ± 0.10
	Auc	0.77 ± 0.04	0.76 ± 0.05
Ped	Acc	0.80 ± 0.12	**0.85 ± 0.10**
	Ave	0.87 ± 0.04	0.88 ± 0.09
	Auc	0.92 ± 0.01	**0.93 ± 0.01**
Yst1	Acc	0.76 ± 0.07	0.73 ± 0.11
	Ave	0.79 ± 0.03	**0.82 ± 0.03**
	Auc	**0.83 ± 0.02**	0.80 ± 0.01
Yst2	Acc	0.91 ± 0.06	0.90 ± 0.11
	Ave	0.93 ± 0.04	**0.95 ± 0.03**
	Auc	**0.95 ± 0.03**	0.93 ± 0.03
Bal	Acc	0.55 ± 0.09	0.56 ± 0.08
	Ave	**0.71 ± 0.15**	0.54 ± 0.11
	Auc	0.84 ± 0.09	**0.89 ± 0.07**

It must be mentioned that no GP solutions with *perfect* fitness values on either the training or test sets are found in any of the runs. Perfect fitness values are indicated by AUC values of 1, i.e., 100% accuracy on the minority and majority class. This may be because these tasks represent difficult classification problems to solve. This means that in all the GP experiments, the evolution is terminated after 50 generations.

In those cases where the ZT strategy has significantly better AUC results than the NST strategy, it can be assumed that solutions in the ZT populations are improving *faster* (in terms of fitness) than solutions in the NST populations over generations, or that solutions in the NST populations are not improving *sufficiently* (in terms of fitness) compared to solutions in ZT populations. This is assumed because, in the initial populations, the NST strategy should identify solutions that perform at least as well as the solutions identified by the ZT strategy. As the i^{th} GP run for either strategy (for a given fitness function and task) uses the same *initial* population of solutions, the NST strategy can select zero as the class threshold for a solution if this happens to be the best "split point" between the class outputs. In contrast, the ZT strategy is always fixed at zero.

In those cases where the ZT strategy has significantly better AUC results than the NST strategy, the ZT strategy may be introducing more *uniformity* in the evolved populations, enforced through selection pressure. In other words, as the evolution progresses (over generations), more solutions are evolved whose outputs lie within the target class boundaries (for the ZT strategy), as these represent fitter solutions. In contrast, the populations for the NST strategy can lack this uniformity, as different solutions are allowed individualised (non-static) class boundaries, i.e., one solution's output values on the two classes can be very different to another while both can be equally fit. This uniformity between solutions in the population may have contributed to the better AUC results for the ZT strategy in some tasks. Indeed, on a case-by-case basis over all six tasks and three fitness functions (18 cases in Table 3.2), the ZT strategy is statistically significantly better than the NST strategy in 7 cases; while the NST strategy is statistically significantly better that the ZT strategy in 5 cases. In the remaining 6 cases, the AUC for both strategies are very similar (not statistically significantly different to each other).

For these reasons, the GP system in the subsequent section, and in the rest of this study, will use the traditional static ZT strategy to map a given solution's outputs to the target class labels.

3.4.3 Comparing Fitness Functions with ZT Strategy

The analysis in the previous section focuses on the AUC between the two classification strategies in GP. Using the same experimental results, the analysis in this section focuses on the statistically significant differences in AUC between the three fitness functions (Acc, Ave and Auc) on a task-by-task basis (over 50 GP runs), using the ZT classification strategy in GP. Table 3.3 summarises these statistically significant differences (which are explained in the subsequent section) for each task, alongside the AUC values for the fitness functions (repeated from Table 3.2 for convenience). Also included in Table 3.3 are the overall classification accuracies, the individual minority and majority class accuracies of the evolved GP classifiers, and the GP training times in seconds (s) or minutes (m). The overall accuracy and individual class accuracies in Table 3.3 represent the performance (on the test sets) of the fittest evolved solution (over 50 GP runs) when these solutions are evaluated using *zero* as the class threshold. These performance measures are included in this analysis to gain a better understanding of the major differences between the fitness functions, and to contrast the AUC with

Table 3.3: Full classification results using the fitness functions (for the ZT classification strategy) over 50 GP runs.

Task	Fit. Func.	AUC Average	Class Accuracy Minority	Class Accuracy Majority	Overall Accuracy	Train Time
Ion	Acc	0.82 ± 0.06 ◦	73.8 ± 7.7	95.3 ± 3.9	88.6 ± 3.7	$2.7s \pm 0.8$
	Ave	0.80 ± 0.06 ◦	76.6 ± 6.3	91.3 ± 6.1	87.0 ± 4.6	$2.8s \pm 0.9$
	Auc	0.85 ± 0.04	81.1 ± 5.2	81.3 ± 6.5	81.2 ± 5.8	$20.0s \pm 5.3$
		$p = 3.1 \times 10^{-5}$				
Spt	Acc	0.72 ± 0.06 ◦	47.4 ± 4.6	88.6 ± 2.5	81.6 ± 2.0	$2.3s \pm 0.6$
	Ave	0.71 ± 0.05 ◦	56.7 ± 8.3	82.7 ± 3.6	78.8 ± 2.1	$2.6s \pm 1.0$
	Auc	0.77 ± 0.04	70.2 ± 6.7	70.0 ± 5.8	70.0 ± 5.2	$12.8s \pm 3.8$
		$p = 1.2 \times 10^{-6}$				
Ped	Acc	0.80 ± 0.12	43.3 ± 14.5	96.6 ± 1.6	86.3 ± 1.9	$5.4m \pm 1.8$
	Ave	0.87 ± 0.04	87.7 ± 2.3	85.6 ± 2.8	86.0 ± 2.2	$5.0m \pm 3.0$
	Auc	0.92 ± 0.01	86.2 ± 1.5	86.1 ± 1.6	86.1 ± 1.6	$71.3m \pm 9.9$
		$p = 1.4 \times 10^{-13}$				
Yst1	Acc	0.76 ± 0.07	40.8 ± 4.2	94.6 ± 1.4	86.0 ± 1.1	$13.5s \pm 5.7$
	Ave	0.79 ± 0.03	60.2 ± 4.6	83.1 ± 3.8	79.6 ± 2.7	$13.3s \pm 4.7$
	Auc	0.83 ± 0.02	73.0 ± 1.4	72.8 ± 1.5	72.8 ± 1.4	$2.1m \pm 0.6$
		$p = 1.7 \times 10^{-11}$				
Yst2	Acc	0.91 ± 0.06 ◦	64.0 ± 8.1	97.4 ± 0.6	94.0 ± 0.8	$11.5s \pm 3.5$
	Ave	0.93 ± 0.04 ◦ •	85.9 ± 4.0	93.0 ± 2.1	92.4 ± 1.8	$12.6s \pm 7.9$
	Auc	0.95 ± 0.03 •	86.8 ± 2.7	88.2 ± 4.1	88.1 ± 3.8	$1.6m \pm 0.3$
		$p = 5.9 \times 10^{-5}$				
Bal	Acc	0.55 ± 0.09	9.0 ± 17.5	98.9 ± 1.1	92.6 ± 1.8	$5.1s \pm 2.0$
	Ave	0.71 ± 0.15	85.6 ± 11.4	84.6 ± 11.7	85.3 ± 11.3	$4.7s \pm 1.5$
	Auc	0.84 ± 0.09	82.8 ± 8.3	87.1 ± 11.0	86.8 ± 10.7	$28.1s \pm 7.6$
		$p = 6.6 \times 10^{-23}$				

the overall accuracy.

Significance Tests for AUC of Fitness Functions

An ANOVA *F-test* of the AUC verifies the *null hypothesis* between these three GP systems for each task (5% level of significance). The null hypothesis is that there is no difference in the distribution of AUC values between the fitness functions. The outcomes of the F-test for each task, i.e., the p-value under the null hypothesis, are shown in Table 3.3 for each task. These p-values indicate that there is at least one fitness function whose AUC is significantly different to the other fitness functions for each task, i.e., null hypothesis rejected, as these are all lower than 0.05 (5% level of significance).

Therefore, a *post-hoc* multiple comparisons test using Tukey's Honestly Signif-

icant Difference (HSD) is used to determine the statistically significant differences between the group means. Recall that Tukey's HSD test conducts a series of pairwise comparisons using the mean AUC from the different GP systems, and outputs a set of 95% confidence intervals for each comparison. A Shapiro-Wilk test verified that our experiment data is normally distributed (required for Tukey's HSD).

The outcomes of Tukey's multiple comparisons are shown in Table 3.3 alongside the AUC value for a particular fitness function. The symbols $\circ$ or $\bullet$ denote that there is *no significant difference* in the AUC for the corresponding pair of fitness functions. This means that between two or more GP systems (on a particular task), the system with the higher average AUC is statistically *significantly better* than the system with the lower average AUC, unless these systems are marked by the $\circ$ and $\bullet$ symbols. In other words, two systems that are marked with either of these symbols show no (statistically) significant difference in their AUC, even if one system has a higher average AUC than the other.

For example, in Table 3.3 for the Ion and Spt tasks, the fitness function Auc has a (statistically) significantly better AUC than both Acc and Ave. However, the AUC for Acc and Ave is not significant different. In Yst_2, the AUC for the fitness function Auc is significantly better than Acc alone; while *Acc* and *Ave*, and *Ave* and Auc, show no significant differences. In the rest of the tasks (Ped, Yst_1 and Bal), Ave is significantly better than Acc; while fitness function Auc is significantly better than both Acc and Ave.

Analysis of the Learning Bias using Acc

As expected, Table 3.3 shows that the classifiers evolved using the standard GP fitness function Acc performs poorly on these tasks compared to Ave and Auc. The AUC for Acc is (statistically) significantly poorer than the fitness function Auc in all tasks, and significantly poorer than Ave in three tasks. On some tasks such as Ped, Yst_1 and in particular Bal (which has the highest class imbalance ratio), this difference in AUC between Acc and Ave/Auc is substantial. The poor AUC for Acc is due to poor minority class accuracies but high majority class accuracies on the tasks. In fact, in the three tasks mentioned above, the minority class accuracies are very poor by comparison. This indicates that Acc tends to find classifiers that are biased toward the majority class with poor minority class accuracies on these tasks, while Acc and Auc have more balanced accuracies on both classes.

Notice that the overall accuracy for Acc is higher than the other two fitness

functions in all tasks. This shows that this performance measure can be misleading in these tasks, as it reflects high accuracy rates even for biased classification performances. The Ped task represents a good example of how the overall accuracy can be the misleading in these class imbalance problems. Here all three fitness functions show very similar overall accuracy rates (86%) but Acc is highly biased toward the majority class, whereas Ave and Auc have relatively high (and balanced) accuracy rates on both classes. In contrast, the AUC is better able to reflect a classifier's ability to *discriminate* between examples from the two classes, particularly for the important minority class, and associates biased performances with poor (low) AUC values.

Advantages and Limitations of Fitness Functions Ave and Auc

While Ave evolves solutions with relatively high accuracies on both classes (unlike Acc), Ave does not consistently show good AUC results compared to Acc on these tasks. Table 3.3 shows that there is no significant difference in AUC between these two fitness functions in three tasks (Ion, Spt and Yst_2). This suggests that Ave in the fitness function does not always guarantee that the evolved classifiers will have better AUC than the standard Acc on these tasks, particularly when the imbalance ratio is not large.

Not surprisingly, the fitness function Auc achieves the best AUC results in all tasks. However, a major limitation of Auc in the fitness function is the increased training times. Table 3.3 shows that Auc takes approximately 5–8 times longer than the two other fitness functions on the smaller data sets (fewer than 1500 training examples). On largest data set, Ped, which more than 12000 training examples, Auc takes approximately 15 times longer. This represents a substantial increase in training time compared to the two other fitness functions.

As discussed, the increased training times for Auc is due to the additional computational effort required to construct an ROC curve. During fitness evaluation, each classifier is evaluated on all fitness cases for every distinct class threshold (to obtain the true positive and false positive rates). This additional computational cost for Auc can be reduced by using fewer class thresholds in the AUC calculation in the fitness function, but this also reduces the quality of the AUC estimate. This aspect is investigated further in the next chapter.

3.5 Summary

This chapter develops a GP approach to classification representing the evolved GP classifiers as mathematical expressions. Particular emphasis is placed on how to develop the classification strategy and fitness function in GP. The classification strategy translates an output value (floating-point number) when a genetic program is evaluated on an input instance from the data set, to a (predicted) class label. The fitness function defines a measure to calculate the overall performance of a solution, by comparing a solution's predicted class label to the target (or actual) class label for all input instances in the training set.

3.5.1 Static Classification Strategy in GP

This chapter compares the performance of the traditional (static) classification strategy to an improved (non-static) strategy, using three different fitness functions on the unbalanced data sets. Focusing on the AUC of the evolved classifiers, the experimental GP results show that for all three fitness functions, there is no major differences between the two strategies on these tasks. While the non-static strategy can be better in problems with multiple (balanced) classes, this is not the case for these binary class imbalance tasks. Rather, the results show that GP can tweak the mathematical expressions representing the GP classifiers to "shift" the outputs of the evolved classifiers relative to the fixed class boundaries, to a sufficient level of accuracy compared to a non-static strategy. An advantage of the traditional static strategy is that more uniformity is introduced in the population. This is enforced through selection pressure where solutions whose output values lie within the desired class boundaries are assigned with better fitness values. In contrast, the non-static strategy defines different class boundaries for different solutions and can lack this uniformity in the population.

This chapter shows that the traditional static classification strategy is good enough for binary tasks with unbalanced data. For this reason, the rest of the thesis uses this static classification strategy.

3.5.2 AUC is a Good Measure

This chapter also shows that the AUC is a good measure of performance on these classification tasks with unbalanced data. The traditional measure, the overall classification accuracy, is shown to be misleading on these tasks, giving the appearance of "good looking" results even for biased classification performances.

In contrast, the AUC is better able to reflect a classifier's ability to *discriminate* between examples from the two classes, particularly for the important minority class, and associates biased performances with poor (low) AUC values.

3.5.3 Limitations of the Fitness Functions

This chapter uses empirical results on the six real-world classification tasks with unbalanced data to highlight the main advantages and limitations of the three current approaches in the fitness function in GP. The following summarises these limitations to highlight what aspects need to be improved.

The standard GP fitness function Acc, which measures the overall classification accuracy, finds genetic program classifiers with high accuracy rates on the majority class but poor accuracy rates on the minority class. These biased classifiers show poor AUC on the tasks, particularly when the level of class imbalance in a data set is high. This chapter shows that this is because Acc can be influenced by the larger number of examples in the majority class.

A common approach to address this learning bias uses the average classification accuracy of the minority and the majority classes (Ave) in the fitness function. This fitness measure shows greater sensitivity to the smaller minority class compared to the standard Acc. While Ave typically finds solutions with similarly high accuracy rates on both classes, Ave does not consistently show very good AUC on tasks. How can this widely-used measure be improved to evolve better-performing classifiers (with higher AUC) on these unbalanced data sets?

Another common approach to address this learning bias uses the actual AUC in the fitness function (Auc). This measure is able to find solutions with high AUC on these tasks but incurs substantially longer training times than the two other fitness functions, due to the computational cost of constructing an ROC curve in fitness evaluation. Can new and faster measures to approximate the AUC in fitness be developed to evolve solutions with good AUC, but with better (faster) training times? How will these new measures compare to the Auc on these tasks?

The experimental results in this chapter show that the choice of fitness function is important for evolving well-performing classifiers with good AUC in these tasks. For this reason, the next chapter will focus on developing several new fitness functions in GP to address the above issues, using the traditional static (zero threshold) strategy in the GP system to translate a solution's output to the predicted class label.

Chapter 4

Developing New GP Fitness Functions

This chapter is organised as follows. The first section outlines the chapter introduction and goals. The second section outlines several existing fitness functions (from the literature) which are used in the experiments. The third section develops several new functions for classification with unbalanced data. The fourth and fifth sections present the experimental results. The sixth section analyses several evolved GP classifiers. The last section provides a summary of this chapter.

4.1 Introduction

The previous chapter has shown that typical training criteria such as the overall classification accuracy in the fitness function in GP can evolve biased classifiers with strong accuracies on the larger (majority) class but poor accuracies on the smaller (minority) class when data sets are unbalanced. This is because this fitness function (Acc) can be influenced by the larger number of examples from the majority class. Two common approaches which adapt the fitness function for cost adjustment to factor in the uneven representation of class examples, Ave and Auc, are shown to find solutions with better minority class accuracies and AUC (than Acc) on the tasks. As the minority class typically represents the main class-of-interest in many real-world tasks with unbalanced data, developing new fitness functions which effectively perform cost adjustment between the two classes to find solutions with good minority class accuracies (and good AUC), is an important research goal.

Other common approaches to address this goal includes using fixed mis-

classification costs for incorrect class predictions (which are determined *a priori*) [88][6], and statistical measures to approximate the AUC such as Wilcoxon-Mann-Whitney (WMW) statistic [57]. Some approaches also develop "specialist" fitness functions which are applied to specific classification tasks with unbalanced data, such as network intrusion detection problems [157] or medical diagnostics [176]. See Chapter 2 for details on these and other works which adapt the fitness function for unbalanced data.

While most of these approaches can improve the classification ability of evolved solutions, they have three main limitations. Firstly, fixed misclassification costs for each class must be determined *a priori*. These costs can be problem-specific and require a lengthy trial-and-error process to configure [157][88][142]. Using equal costs for both classes (such as the average accuracy of the minority and majority classes, Ave, from the previous chapter) can address this limitation. However, as shown in the previous chapter, using Ave in the fitness function does not consistently find solutions with good AUC (that is significantly better than Acc) on the tasks. Secondly, using the AUC directly in the fitness function (such as Auc from the previous chapter) can find solutions with high AUC on the tasks, but can also incur long training times due to the computational overhead required to calculate the AUC, particularly on large data sets. Finally, some "specialist" fitness functions which are hand-crafted to suit a particular classification problem (such as [157][60][176]), require *a priori* expert knowledge about the problem domain. Fitness functions that are domain-independent, i.e., which can be applied to a range of classification tasks with unbalanced data without any *a priori* knowledge about the input data, are more desirable.

In this area there is a need to develop new domain-independent performance measures in the fitness function to find solutions with good classification ability on both classes but which do not incur a substantial increase in the GP training times. This chapter tries to address these issues by developing several new fitness functions to promote classifiers with good accuracy on both classes and penalise biased solutions in the evolutionary process. These measures aim to evolve classifiers with high AUC but with better (faster) training times than the traditional approach Auc. By developing new measures in the fitness function to perform cost adjustment between the two classes (irrespective of the problem domain), the original unbalanced input data can be "as is" in the GP approach, requiring no *a priori* knowledge about the input data. This alleviates the need for a sampling algorithm to first artificially re-balanced the input data before the training process.

This chapter develops several new fitness functions for classification with unbalanced data. The AUC (of the evolved classifiers) and GP training times of the new fitness functions are compared to other approaches from the previous chapter, and two other machine learning algorithms (Naive Bayes and Support Vector Machines) on the tasks.

4.1.1 Chapter goals

This chapter has two mains goals. The first goal is to develop new measures in the fitness function to find solutions with high AUC on these tasks. These new fitness functions aim to improve AUC performances over the traditional Ave, and improve training times over the traditional Auc. The second goal investigates whether an equal weighting of the minority and majority class accuracy in a weighted-average fitness function, or a non-equal weighting where one class has a greater cost in fitness than the other, finds solutions with better overall AUC on the tasks.

4.2 Current Approaches in Fitness

This section summarises the GP framework defined in the previous chapter (also used in this chapter), and presents several *baseline* fitness functions in GP to evaluate the effectiveness of the new measures (developed in the next section). These baseline fitness functions include the three approaches from the previous chapter, Acc, Ave and Auc, and four other useful approaches from the literature, namely, $Wave$, $AveM$, Auc_E and Wmw.

4.2.1 GP Framework

The same GP framework is used in these experiments as outlined in the previous chapter. To recap, the genetic program solutions use a tree-based structure for representation. Feature and constant terminals are used in the terminal set, and the function set consists of the four standard arithmetic operators ($+, -, \%$ and $\times$) and a conditional operator (`if`). As the previous chapter establishes that the standard zero-threshold (ZT) strategy performs as well as the non-static strategy on these tasks, the ZT strategy is used to translate the real-valued output of a solution (when evaluated on an input instance) to the target class labels. Recall that this strategy uses the natural division between positive and

Table 4.1: Outcomes of a two-class classification problem.

	Predicted Positive Class	*Predicted* Negative Class
Actual Positive Class	True Positive (TP)	False Negative (FN)
Actual Negative Class	False Positive (FP)	True Negative (TN)

negative numbers to represent the two class labels, i.e., an input instance will be assigned to the majority class if the solution output is negative, otherwise it will be assigned to the minority class.

4.2.2 Baseline GP Fitness Functions

This chapter presents several baseline fitness functions in GP. These include the three measures discussed in the previous chapter (Acc, Ave and Auc), two average-based measures ($AveM$ and $Wave$), and two alternative approaches to calculate the AUC is fitness (Auc_E and Wmw).

Fitness Functions Acc, Ave and Auc

The three fitness functions from the previous chapter are repeated below for convenience. These include the standard GP fitness function Acc (the overall classification accuracy), and average accuracy of the minority and majority class Ave, and the AUC in the fitness function AUC. These fitness functions are defined using the four outcomes for binary classification shown in Table 4.1. Note that the same confusion matrix is also shown in the earlier chapters (Chapters 2 and 3) but is repeated here for convenience.

$$Acc = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.1}$$

$$Ave = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right) \tag{4.2}$$

$$Auc = \sum_{i=1}^{N-1} \frac{1}{2}\left(FP_{i+1} - FP_i\right)\left(TP_{i+1} + TP_i\right) \tag{4.3}$$

Average-Based Fitness Function $Wave$

An alternative form of Ave uses the weighted-average of the minority and majority class accuracy in the fitness function, as shown by Eq. (4.4). In $Wave$, the weighting coefficient controls the trade-off between the minority and majority

Table 4.2: Minority and majority class accuracies of three solutions, and the corresponding $AveM$ fitness values.

Solution	Minority Accuracy	Majority Accuracy	$AveM$ Fitness
a_1	70%	70%	24.5%
a_2	60%	80%	24.0%
a_3	85%	55%	23.4%

class accuracy where $0 < W < 1$. When W is 0.5, the accuracy of both classes is considered as equally important in fitness (with this setting, $Wave$ and Ave are the same). When $W > 0.5$, minority class accuracy will contribute more in the fitness function than majority class accuracy by factor W. Similarly, majority class accuracy will contribute more when $W < 0.5$.

$$Wave = W \times \left(\frac{TP}{TP+FN}\right) + (1 - W) \times \left(\frac{TN}{TN+FP}\right) \tag{4.4}$$

This fitness function addresses the second goal of this chapter, and investigates whether an equal weighting of the minority and majority class accuracy ($W = 0.5$), or a non-equal weighting ($W \neq 0.5$), finds solutions with better overall AUC on the tasks.

Average-Based Fitness Function $AveM$

The fitness function $AveM$ or Eq. (4.5) is based on the geometric mean of the minority and majority class [57][141]. Similar to Ave and $Wave$, the two components in Eq. (4.5) corresponds to the minority and majority class accuracy, respectively.

$$AveM = \frac{1}{2}\left(\frac{TP}{TP+FN} \times \frac{TN}{TN+FP}\right) \tag{4.5}$$

This fitness function has the two useful properties compared to the arithmetic average (used in Ave and $Wave$). Firstly, as the geometric mean multiplies the minority and majority class accuracies, if the accuracy on a single class is zero, then zero is returned. Secondly, $AveM$ can produce more fine-grained fitness values when Ave evaluates to the same fitness value for some combinations of the two components.

For example, consider the minority and majority class accuracies of three solutions (a_1, a_2 and a_3) shown in Table 4.2. Each solution has the same average accuracy, 70%, and therefore the same fitness values according to Ave. In contrast, $AveM$ ranks a_1 as the fittest and a_3 as the least fit according to the fitness values

shown in Table 4.2. This fitness function aims to investigate how these differences affect the AUC of evolved solutions.

AUC-Based Fitness Functions Auc_F and Auc_E

As discussed, a major limitation of using Auc or Eq. (4.3) in the fitness function is the increased training times, due to the computational effort required to construct an ROC curve in fitness evaluation. Each classifier must be evaluated on all fitness cases N times to obtain N distinct TP/FP points on an ROC curve. A useful technique to speed-up training times for Auc uses fewer TP/FP points on the ROC curves [84]. In [84], exactly seven distinct TP/FP points are used in the ROC curves; this number is recommended for a fast and accurate approximation to the full AUC [84]. To simulate this in GP, seven ROC points are generated by choosing seven distinct class thresholds spread uniformly over the range of a given genetic program's output values (when evaluated on all training instances). The genetic program is then evaluated at each threshold to produce the seven TP/FP points. Naturally, this faster approximation of the AUC will have a lower precision than the full AUC. Recall (from the previous chapter) that in the full AUC, every distinct value in the range of output values for a given genetic program is taken as a separate class threshold.

This chapter compares the GP training times and AUC of the evolved solutions using both AUC calculations in the fitness function, i.e., the full AUC Auc_F, and and the faster estimation Auc_E (which uses exactly seven ROC points).

AUC-Based Fitness Function Wmw

An alternative technique to calculate the AUC in the fitness function uses a statistical approximation based on the Wilcoxon-Mann-Whitney (WMW) statistic [84][179][57], as shown in Eq. (4.6). The WMW statistic uses a series of pairwise comparisons between the genetic program outputs (when evaluated on examples from the two classes), effectively measuring the ordering of minority to majority class outputs. In Eq. (4.6), P_i and P_j represent the output of a genetic program when evaluated on an example from the minority and majority class, respectively, and N_{min} and N_{maj} are the number of examples in the the minority and majority class respectively.

The indicator function I_{wmw} enforces two constraints on the ordering of output values for each class. The first constraint ($P_i \geq 0$) checks whether the minority class outputs are zero or positive. The second constraint ($P_i > P_j$) checks whether

the minority class outputs are greater than the majority class outputs, to establish an ordering of class outputs (using zero as the class threshold). The denominator ensures that Wmw returns values between 0 and 1, where 1 indicates good class separability (high AUC) and 0 indicates poor separability (low AUC).

$$Wmw = \frac{\sum_{i=1}^{N_{min}} \sum_{j=1}^{N_{maj}} I_{wmw}(P_i, P_j)}{N_{min} \times N_{maj}} \tag{4.6}$$

where

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

4.3 New Fitness Functions

Five new fitness functions for classification with unbalanced data are developed to address the first goal of this chapter. These are split into two categories. The first category develops three new measures, $Amse$, $Incr$ and $Bands$, aiming to improve the traditional measure Ave (Eq. 4.2). The second category develops two novel separability-based measures, $Corr$ and $Dist$, aiming to evolve solutions with high AUC but with faster training times than the AUC-based functions.

4.3.1 Improving the Average-Based Measure

These three new fitness functions, $Amse$, $Incr$ and $Bands$, all use variations of the average accuracy of each class in fitness.

Fitness Function *Amse*

Eq. (4.7) or $Amse$ is based on the mean squared error (MSE) function, a popular machine learning measure for determining the difference between predicted and target output patterns [34][178][177]. This fitness function is similar to Ave except that the *magnitude* or genetic program output values are also factored into the fitness functions; whereas the traditional Ave only considers the true positive and true negative rates (magnitude of genetic program outputs ignored). The goal of $Amse$ is to evolve classifiers whose outputs are closely "calibrated" to the desired or target values for each class, where solutions with smaller deviations between the target and classifier outputs are rewarded with better fitness over solutions with larger differences.

In Eq. (4.7), $P_{c,i}$ represents the output of a genetic program classifier when evaluated on the ith example belonging to class c, N_c is the number of examples in class c, and K is the number of classes. The target T_c values for the majority and minority classes are -0.5 and 0.5, respectively, as zero is the class boundary (i.e. majority and minority class outputs should be negative and non-negative respectively).

$$Amse = \frac{1}{K}\sum_{c=1}^{K}\left(1 - \frac{\sum_{i=1}^{N_c}(sig(P_{c,i}) - T_c)^2}{2N_c}\right) \tag{4.7}$$

where

$$sig(x) = \tfrac{2}{1+e^{-x}} - 1$$

This fitness function is different to the traditional MSE in two important ways. Firstly, $Amse$ uses the *average* error for each class to account for the unbalanced data in these tasks, whereas many other approaches (such as [34][177]) use the overall MSE on all training examples. Secondly, MSE is typically used in approaches where the raw output values of the learned classifiers are bound in a fixed range, e.g., output values range between 0 and 1. As the raw outputs of a genetic program classifier $P_{c,i}$ has no bounds (can by anything between $-\infty$ or $+\infty$), these raw outputs must first be bound (or scaled) for consistency in fitness values across the population. If this is not enforced, genetic programs which produce large output values risk inflating the difference between target and actual values (i.e. poorer fitness), compared to other genetic programs which have similar accuracies but which produce smaller output values. For example, if two classifiers, S_1 and S_2, have the same class accuracy but S_1 outputs values in the range $[-100, 100]$ and S_2 in the range $[-5, 5]$; then the difference between target outputs (T_c) and genetic program outputs will be larger for S_1 by virtue of the larger genetic program outputs alone.

For this reason, Eq. (4.7) uses a sigmoid function (sig) to scale the raw genetic program outputs to the range $[-1, 1]$. This sigmoid function is applied to the value returned from the root node of the genetic program during the fitness evaluation, and serves only to scale the range of genetic program outputs to -1 and $+1$ (sign of genetic program output values unaltered). The scaling ensures that positive output values are "spread out" between 0 and 1, and not simply "cut off" at 1; likewise for negative output values between 0 and -1.

The denominator in Eq. (4.7) corresponds to the maximum difference between target and actual outputs for each class, where 2 (in $2N_c$) is the maximum (absolute) difference between the smallest (-1) and largest ($+1$) output value

allowed by the sigmoid function. This serves to normalise the MSE for each class to values between 0 and 1. The normalised MSE for each class is then inverted to make the fitness values returned from this function consistent with the other fitness functions (0 worst and 1 best).

Fitness Function *Incr*

Eq. (4.8) extends the function Ave by assigning greater *rewards* to solutions whose output values fall further away from the class boundary. $Incr$ improves the traditional Ave by differentiating between solutions which have the same class accuracy, but which use different internal classification models. By counting the average number of incremental rewards earned per class, $Incr$ favours solutions whose output values are further away from the class boundary.

In Eq. (4.8), $P_{c,i}$ represents the output of a genetic program classifier when evaluated on the ith example belonging to class c, N_c is the number of examples in class c, and K is the number of classes. The term $D_{c,j}$ represents the jth *cluster* of genetic program output values in class c, and M_c is the number of clusters of output values in class c. The denominator in Eq. (4.8) corresponds to the maximum reward a solution can obtain for each class. This serves to normalise the rewards earned in each class to values between 0 and 1. As a result, fitness values for $Incr$ range between 0 (worst fitness) and 1 (best fitness).

$$Incr = \frac{1}{K}\sum_{c=1}^{K}\left(\frac{\sum_{j=1}^{M_c}\left[I_{zt}(j, D_{c,j}, c)\ .\ \sum_{i=1}^{N_c} Eq(D_{c,j}, P_{c,i})\right]}{\frac{1}{2}N_c(N_c+1)}\right) \tag{4.8}$$

where

$$I_{zt}(r,k,c) = \begin{cases} r & \text{if } (k \geq 0 \text{ and } c \in Min) \text{ or if } (k < 0 \text{ and } c \in Maj) \\ 0 & \text{otherwise} \end{cases}$$

and

$$Eq(p,q) = \begin{cases} 1 & \text{if } p = q \text{ to 2 decimal places} \\ 0 & \text{otherwise} \end{cases}$$

Eq. (4.8) uses two main components to calculate the incremental rewards for each class; these correspond to the two indicator functions, Eq and I_{zt}. The indicator function Eq returns 1 if two genetic program outputs are the same or 0 otherwise. This indicator function is used to count the number of different output values in a *cluster* of outputs. A cluster of outputs are different input instances

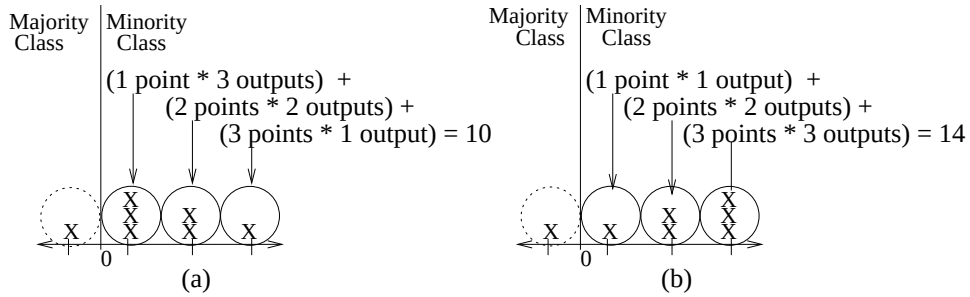


Figure 4.1: Genetic program outputs for two classifiers; X denotes the solution outputs for seven (minority class) instances where equivalent X values are stacked above each other. Solid circle shows correctly predicted clusters of outputs and dotted circle shows incorrect clusters. Solution (b) is better as it earns 14 rewards while (a) only earns 10, as (b) has more outputs that lie further away from the class boundary (0).

for a given solution that evaluate to the *same* floating point number (to 2 decimal places).

The indicator function I_{zt} returns its first argument (j) if its second argument ($D_{c,j}$ is the j^{th} output cluster in class c) lies within the target class region, or 0 otherwise. When indicator function I_{zt} is satisfied (i.e. j is returned and not 0), j is the incremental reward earned by cluster $D_{c,j}$. Provided that the genetic program outputs are processed in ascending order for each class, the rewards earned will increase as the clusters of output values lie further and further away from the class boundary (zero).

Figure 4.1 provides an example of how the incremental reward for a particular class is calculated using two different genetic program solutions. In Figure 4.1, X denotes the genetic program outputs (along the horizontal axis) when evaluated on seven minority class instances. Notice that these seven genetic program outputs only correspond to four distinct clusters (solid and dotted circles) where equivalent X values in the same cluster are stacked above each other in Figure 4.1. Using zero as the class boundary, both solutions (a) and (b) in Figure 4.1 have three "correct" clusters of output values (solid circles) and one incorrect cluster (dotted circle). As there are exactly three "correct" clusters in both (a) and (b), the incremental rewards are calculated as follows: 1 point for each X value in the *first* cluster (nearest to the zero), 2 points for each X value in the second cluster, and 3 points for each X value in the third cluster.

Using these incremental rewards, solution (a) accumulates a total of 10 points: 3 points in the first cluster (1 point for each prediction), 4 points in the second cluster (2 points for each prediction), and 3 points in the third cluster (3 points

Majority Accuracy (%)	Minority Accuracy (%)										
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
100%	0	1	2	3	4	5	6	7	8	9	10
90%	0	1	2	3	4	5	6	7	8	9	9
80%	0	1	2	3	4	5	6	7	8	8	8
70%	0	1	2	3	4	5	6	7	7	7	7
60%	0	1	2	3	4	5	6	6	6	6	6
50%	0	1	2	3	4	5	5	5	5	5	5
40%	0	1	2	3	4	4	4	4	4	4	4
30%	0	1	2	3	3	3	3	3	3	3	3
20%	0	1	2	2	2	2	2	2	2	2	2
10%	0	1	1	1	1	1	1	1	1	1	1
0%	0	0	0	0	0	0	0	0	0	0	0

Figure 4.2: Regions of fitness bands (for fitness function $Bands$) where the objective-space is divided into a 10×10 grid and each grid square represent the fitness value for the minority and majority class accuracy of a solution).

for the single prediction). Similarly, solution (b) accumulates a total of 14: 1 point in the first cluster, 4 points in the second cluster, and 9 points in the third cluster. Solution (b) is therefore rewarded with a higher fitness than solution a (for this particular class), as b has more predictions that are further away from the class boundary than a.

Using the fitness function Ave, solutions (a) and (b) both have identical accuracy rates for this class (equally fit), as six out seven instances are assigned the correct class label. On the other hand, $Incr$ will rank solution (b) as fitter than (a) for this particular class.

Fitness Function *Bands*

The fitness function $Bands$ maps a solution's minority and majority class accuracy to a single value, similar to the average of these objectives (such as Ave and $AveM$). However, good accuracy on both classes is rewarded to a greater extent than in Ave, while poor accuracy on one class is strongly discouraged. To achieve this, the *objective-space* for the minority and majority class accuracy is divided into 100×100 grid where each square represents 1% of the objective-space. For example, Figure 4.2 shows a simplified instance of this scenario where the objective-space is divided into a 10×10 grid. In this figure, the minority class accuracy lies along the horizontal axis and majority accuracy along the vertical axis. Note that to obtain the larger 100×100 grid, each grid square in Figure 4.2 is further divided into another 10×10 grid.

Each grid square is assigned a fitness value based on its distance to the "target" region of the grid, that is, the top-right corner of the grid which represents 100%

Table 4.3: Minority and majority class accuracies, and corresponding fitness values for Ave, $AveM$ and $Bands$ for four solutions.

Solution	Minority Accuracy	Majority Accuracy	Ave Fitness	$AveM$ Fitness	$Bands$ Fitness	
					10×10	100 ×100
a_1	70%	70%	70%	24.5%	7	70
a_2	60%	80%	70%	24.0%	6	60
a_3	70%	80%	75%	28.0%	7	70
a_4	65%	85%	75%	27.6%	6	65

accuracy on both objectives. Fitness values in the grid increase as the distances to the target region decrease, as shown in Figure 4.2. These fitness values represent the "goodness" of the minority and majority class accuracy for a given solution, where the higher the fitness, the better the two accuracies. For example, if a solution has 60% accuracy on the minority class and 80% on the majority class, the final fitness value (using the simplified Figure 4.2) is 6. This is obtained by looking-up the grid value in the 9^{th} row (80% majority accuracy) and 7^{th} column (60% minority accuracy).

This fitness function is different to Ave and $AveM$ as it rewards solutions which have *equally high* accuracy on both classes with higher fitness values (than Ave and $AveM$). In Ave and $AveM$, fitness values can be high only if one factor in the average is high (the other factor can be low). For example, consider the minority and majority class accuracies of four solutions (a_1, a_2, a_3 an a_4) as shown in Table 4.3. The fitness values for the $Bands$ fitness function is shown for both the example 10×10 grid (in Figure 4.2) and the 100×100 grid (actual fitness values).

The actual fitness values for $Bands$ in Table 4.3 rank these solutions in a different order compared to $AveM$. Most noticeably, a_1 has a high fitness for $Bands$ as both factors (minority and majority accuracy) are equally high; while both a_2 and a_4 have a lower fitness as one factor (minority accuracy) is low. In contrast, a_1 has a lower fitness value than a_3 and a_4 for $AveM$ as one factor in these two solutions (majority accuracy) is high but the other is low.

4.3.2 New Separability-based Measures in Fitness

The two new fitness functions, $Corr$ and $Dist$ (discussed below), use separability-based measures in fitness (similar to the AUC-based fitness functions).

Fitness Function *Corr*

Eq. (4.9) is a novel fitness function based on the statistical measure, the correlation ratio [71], which measures linear dispersal between two populations of data. Assuming that the genetic program outputs (when evaluated on the examples from the two classes) are two populations of data, the correlation ratio can be used to measure how well these populations are separated with respect to each other. The higher the dispersal between these two populations, the better the separability of the genetic program outputs for the two classes. The correlation ratio outputs values between 0 (poor separability) and 1 (good separability).

$$Corr = r + I_{zt}(1, \mu_{maj}, \mu_{min}) \tag{4.9}$$

where

$$r = \sqrt{\frac{\sum_{c=1}^{K} N_c(\mu_c - \bar{\mu})^2}{\sum_{c=1}^{K}\sum_{i=1}^{N_c}(P_{c,i} - \bar{\mu})^2}}$$

and

$$\mu_c = \frac{\sum_{i=1}^{N_c} P_{c,i}}{N_c} \text{ and } \bar{\mu} = \frac{\sum_{c=1}^{K} N_c \mu_c}{\sum_{c=1}^{K} N_c}$$

In Eq. (4.9), r computes the correlation ratio where $P_{c,i}$ is the output of a solution when evaluated on the i^{th} example belonging to class c, N_c is the number of examples in class c, and K is the number of classes. In this equation, μ_c represents the mean of a solution's outputs for class c only, and $\bar{\mu}$ represents the mean of μ_c for the minority and majority classes.

The function r measures the level of separability of the output values for the two classes. The final fitness value for $Corr$ uses indicator function I_{zt} (from Eq. 4.8 in the previous section) to encourage solutions to order their outputs (for the two classes) according to the target class boundaries. Indicator function I_{zt} takes, as inputs, a reward value and the means of the outputs on the majority and minority class instances for a given solution (μ_{maj} and μ_{min}, respectively), and returns the reward if the majority and minority class means are negative and non-negative, respectively (or 0 otherwise).

As r returns values between 0 and 1, and I_{zt} returns either 0 or 1, the final fitness values returned by $Corr$ will range between 0 (worst fitness) and 2 (best fitness).

Fitness Function *Dist*

Eq. (4.10) models the genetic program outputs for the two classes as two independent class distributions, and uses the distance between these two class

distributions to represent the level of class separability. $Dist$ was originally developed for multiple class problems with relatively balanced class distributions [186], and has not previously been evaluated on binary class imbalance tasks. Eq. (4.10) computes the point that is equi-distant from the means of two distributions, measured in terms of standard deviations away from the mean (where the standard deviations can be different for the two distributions). In the worst case where the means and standard deviations of both class distributions are the same (poor separability), this distance will be 0. In the ideal case, where there is no overlap between the two class distributions (high separability), this distance will be large (go to $+\infty$).

In Eq. (4.10), μ_c and σ_c correspond to the mean and standard deviation of the class distribution c, respectively, where c is either the minority (min) and majority (maj) class. Similarly, $P_{c,i}$ is the output of the classifier when evaluated on the i^{th} example belonging to class c and N_c is the number of examples in class c. $Dist$ also uses indicator function I_{zt} (in a similar manner to $Corr$) to encourage solutions to order their outputs (for the two classes) according to the target class boundaries. In $Dist$, the estimated distance value for a solution is doubled when the indicator function is true (i.e. when μ_{maj} and μ_{min} are negative and non-negative, respectively).

$$Dist = \frac{|\mu_{min} - \mu_{maj}|}{\sigma_{min} + \sigma_{maj}} \times (1 + I_{zt}(1, \mu_{min}, \mu_{maj})) \tag{4.10}$$

where

$$\mu_c = \frac{\sum_{i=1}^{N_c} P_{c,i}}{N_c} \text{ and } \sigma_c = \sqrt{\frac{1}{N_c} \sum_{i=1}^{N_c} (P_{c,i} - \mu_c)^2}$$

Indicator Function I_{zt} in *Corr* and *Dist*

Both fitness functions *Corr* and *Dist* use an indicator function I_{zt} to reward solutions which order their output values (on the two classes) according to the target class boundaries, with higher fitness values. This is accomplished by checking if the mean of the output values for each class (μ_c for class c) lie within the target class region where majority class outputs should be negative and minority class outputs should be non-negative. An alternative approach initially considered for $Corr$ and $Dist$, used Ave (Eq. 4.2) to measure how well the output values for a solution adheres to the above class ordering of outputs. This Ave-based approach can produce smoother fitness values which reflect the proportion of class outputs that lie within the target class regions for a given solution; whereas the indicator function-based approach only returns a reward

value (such as 1) if the mean of the outputs for both classes falls within the target class regions, or 0 otherwise.

However, preliminary experiments which compared $Corr$ and $Dist$ using either Ave or indicator function I_{zt} on the tasks, found that the Ave-based approach produced slightly lower AUC results in the evolved solutions than the indicator function-based approach. These preliminary results are omitted here as they are not the main focus of this chapter but can be seen in Appendix B (in Section B.2.1). The indicator function I_{zt} outperforms the Ave-based approach because I_{zt} has the desirable property that solutions whose outputs do not adhere to the desired class ordering are assigned poor fitness values early in the evolution. These solutions are then phased-out out of the evolution relatively early in the process due to selection pressure. In contrast, the Ave-based approach adopts a "fairer" strategy which assigns moderate-level fitness values to solutions whose outputs only *partially* adhere to the desired class ordering. These solutions then remain in the population for longer.

For these reasons, the indicator function I_{zt} is the preferred method in $Corr$ and $Dist$ to ensure that majority and minority class outputs are negative and non-negative, respectively, in the evolved solutions.

4.4 Experimental Setup

This section outlines the GP evolutionary parameters and the statistical significance testing techniques used in the experimental results.

4.4.1 GP Evolutionary Parameters

The same evolutionary parameters from the previous chapter are also used in these experiments. To recap, crossover, mutation and elitism rates are 60%, 35% and 5%, respectively, and tournament selection is used with a tournament size of 7. The maximum program depth is 8 to restrict very large programs in the population, and the population size is 500. The evolution is allowed to run for a maximum of 50 generations, or is terminated early if a solution with a maximum fitness value on the training set is found.

As discussed in the previous chapter, this configuration of parameters is recommended in the literature. To concentrate on the effects of the fitness functions in the GP algorithm, it is important that the configuration of evolutionary parameters is kept consistent. As the goal of this chapter is to compare

the different fitness functions in the evolution, fine-tuning this configuration of evolutionary parameters for better classification performances is outside the scope of this study.

4.4.2 Statistical Significance Testing of the AUC

Similar to the experimental results in the previous chapter, Tukey's Honestly Significant Difference (HSD) [166] is used to find the statistically significant differences in AUC for the solutions evolved using the fitness functions. Tukey's multiple comparisons test compares the average AUC of the fittest evolved solutions from each GP system (using a particular fitness function) to all others, and outputs a confidence interval for each pairwise comparison between GP systems. However, as these experimental results compare 11 different fitness functions, 55 confidence intervals are returned from Tukey's multiple comparisons test when each GP system is compared to all others, as shown below.

$$n = \frac{k(k-1)}{2} = \frac{11(11-1)}{2} = 55$$

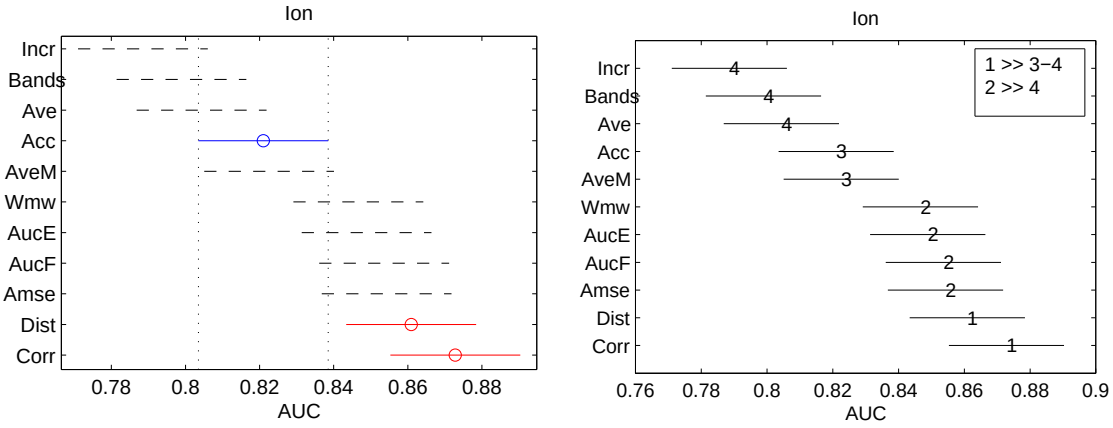
where k is the number of GP fitness functions. This means that 55 confidence intervals (of the AUC) for the different fitness functions must be compared to one another to find the statistically *significantly better* mean AUC values. A confidence interval between two fitness functions is calculated using Eq. (4.11) below for all fitness functions.

$$\bar{y_i} - \bar{y_j} \pm \frac{q(\alpha,k,M-k)}{\sqrt{2}} SE\sqrt{\frac{2}{n}} \quad \text{for all } i,j = 1,2,...,k \text{ where } i \neq j \tag{4.11}$$

In Eq. (4.11), $\bar{y_i}$ and $\bar{y_j}$ are the mean AUC for two fitness functions, n is the number of GP runs (50), k is the number of fitness functions (11), and SE is the standard deviation of the entire sample. The constant value q is the critical value for the studentised range statistic Q [121]. This is obtained using a look-up table[1] for three variables: α, k and M. Here are α is the level of significance, k is the number of fitness functions, and M is the total sample size (i.e. total number of experiments for all fitness functions for a given task). As k is 11, M is 550 (50 runs of k fitness functions), and α is 0.05 (5% level of significance), the look-up value for q according to [121] is 4.51, as shown below.

$$q(\alpha, k, M-k) = q(0.05, 11, 539) = 4.51$$

[1]The distribution of Q has been tabulated and appears in many textbooks on statistics or online such as the online Statistical Table Entries Calculator at Vassar College [121].



(a) (b)

Figure 4.3: Confidence intervals of the AUC for the different fitness functions for the Ion task. In (a), the interval for Acc is statistically significantly poorer than $Dist$ and $Corr$. In (b), the confidence intervals are labelled with their s-ranks where the legend shows significantly better s-ranks.

In Eq. (4.11), $\frac{q(\alpha,k,M-k)}{\sqrt{2}} SE \sqrt{\frac{2}{n}}$ remains constant for all pairwise comparisons between fitness functions. As a result, these confidence intervals can be visualised for easier interpretability, as shown in Figure 4.3(a) for the Ion task. In Figure 4.3(a), each bar represents the 95% confidence interval of the mean AUC for a particular fitness function, where the horizontal axis shows the AUC. Two fitness functions are significantly different to one another only if their intervals are disjoint, and are not significantly different to one another if their intervals overlap.

For example, Figure 4.3(a) shows that the fitness function Acc is significantly different to $Dist$ and $Corr$ (in terms of average AUC), as the interval for Acc (highlighted in blue) does not overlap with the intervals for $Dist$ and $Corr$ (highlighted in red). However, as the interval for Acc does overlap with all other intervals (dashed), Acc is not statistically significantly different to these fitness functions.

Figure 4.3(a) allows each interval to be easily compared to all other intervals to determine the statistically significant AUC values for the different fitness functions.

4.4.3 Significance Ranking using *S-rank*

To summarise which fitness functions have a significantly better AUC compared to others (i.e. when each interval is compared to all others) for a given task,

an identifying number is assigned to each fitness function. This number, called the *significance rank* (or s-rank), represents a *group* of fitness functions that are statistically significantly different to other groups on a particular task. The fitness function(s) with the highest average AUC is assigned the best r-rank (1) and s-rank values will increase (s-rank gets worse) as the average AUC of the fitness functions also gets worse, as shown in Figure 4.3(b).

In Figure 4.3(b), the fitness function intervals are shown for Ion when each interval has been labelled with the corresponding s-rank. The legend in Figure 4.3(b) shows which s-rank values are statistically *significantly better* than other s-rank values, where the symbol $\gg$ denotes a significantly better s-rank. For example, "$1 \gg 3-4$" shows that the fitness function(s) with an s-rank of 1 ($Dist$ and $Corr$ in this case) have a significantly better AUC than the fitness functions with s-ranks 3 and 4. Likewise, "$2 \gg 4$" shows that the fitness function(s) with an s-rank of 2 have a significantly better AUC than those with an s-rank of 4.

The following procedure assigns s-rank values to the fitness functions for a given task.

1. Sort the fitness functions in ascending order (using their average AUC values), as shown in Figure 4.3(a). Select the fitness function (or interval) with the highest AUC as the current interval, and initialise the s-rank to 1.

2. Find all other intervals that are significantly worse than the current interval (i.e. other intervals that no not overlap with the current interval). For example, if the current interval is $Corr$ in Figure 4.3(b), the other intervals that no not overlap with $Corr$ are those with s-rank values of 3 and 4.

3. Find all other intervals that are not significantly different from the current interval (i.e. other intervals that overlap with the current interval). For example, if the current interval is $Corr$ in Figure 4.3(b), the other intervals that overlap with $Corr$ are those with s-rank values of 1 and 2.

4. Using the intervals from Step (3), find those intervals that do not overlap with *all* the intervals in the set from Step (2). For example, if the current interval is $Corr$ in Figure 4.3(b), the only interval from Step (3) that does not overlap with all the intervals from Step (2) is $Dist$ since all other intervals from Step (3) (Wmw, Auc_E, Auc_F, and $Amse$) overlaps with at least one interval from Step (2).

5. Assign the intervals from Step (4) with the current s-rank value (e.g. this is 1 on the first iteration), and then increment the s-rank. For example, if the

current interval is $Corr$, and $Dist$ is selected from the previous step, $Dist$ is assigned an s-rank of 1. Note that if no intervals are found matching the criteria from the previous step, then just increment the s-rank.

6. Select the next interval (from the ordered list from Step (1)) that has not yet been assigned an s-rank. Repeat steps (2) to (6) using this interval as the current interval, until all intervals are processed. In the example discussed in the previous step, the *new* current interval to be processed will be $Amse$ since both $Corr$ and $Dist$ have already been assigned s-rank-values.

This process of visualising the confidence intervals for each fitness function and then assigning s-rank values to each interval (using the above procedure) is repeated for all tasks. The s-rank values, and the statistical significance relationships between groups of fitness functions, are presented and analysed in the next section for the tasks.

4.5 Experimental Results

This section shows the GP experimental results and consists of four main parts. The first part examines the AUC (and s-ranks) of the different fitness functions on each task. The second part analyses the overall AUC behaviour of the fitness functions over *all* tasks. The third part presents typical ROC curves for the fitness functions. The final part compares the GP results to Naive Bayes and Support Vector Machines on the tasks.

4.5.1 AUC of Fitness Functions

Table 4.4 reports the average AUC ($\pm$ standard deviation) and the best AUC achieved by an evolved classifier over 50 GP runs on the *test* set for each task. Also shown in this table are the average GP training times, reported in second (s) or minutes (m). The results of Tukey's multiple comparisons significance testing of the AUC is shown in the "Stat. Test" column in Table 4.4, as summarised by the new *significance rank* (or s-rank) measure. This corresponds to the s-rank ("SR") for a given fitness function, where "Beats" shows the set of other s-ranks with a (statistically) significantly poorer AUC. For example, the first line in Table 4.4 for Ion (for "Stat. Test") shows that fitness function $Corr$ achieves the (best) s-rank of 1 in this task, and "$\{3-4\}$" means that $Corr$ is significantly better than fitness functions with s-ranks 3 and 4. Both $Corr$ and $Dist$ have equivalent s-ranks of

Table 4.4: Full classification results of the GP fitness functions for the tasks. The *SR* denotes the significance rank (s-rank) of a fitness function and *beats* denotes other s-rank(s) with a (statistically) significantly poorer AUC.

Fitness	AUC		Stat. Test		Training	Fitness	AUC		Stat. Test		Training
Func.	Average	Best	SR	Bats	Times	Func.	Average	Best	SR	Beats	Times
	Ion						*Spt*				
Corr	0.87 ± 0.04	0.94	1	{3-4}	$2.4s \pm 0.5$	AucF	0.77 ± 0.04	0.83	1	{4-5}	$12.8s \pm 3.8$
Dist	0.86 ± 0.05	0.95	1	{3-4}	$1.4s \pm 0.5$	Incr	0.76 ± 0.05	0.86	1	{4-5}	$4.4s \pm 1.8$
Amse	0.85 ± 0.05	0.94	2	{4}	$2.8s \pm 0.9$	AucE	0.76 ± 0.04	0.86	1	{4-5}	$2.8s \pm 0.9$
AucF	0.85 ± 0.04	0.94	2	{4}	$20.0s \pm 5.3$	Bands	0.76 ± 0.04	0.87	1	{4-5}	$2.6s \pm 0.8$
AucE	0.85 ± 0.05	0.93	2	{4}	$3.1s \pm 0.9$	Amse	0.75 ± 0.04	0.84	2	{5}	$2.2s \pm 0.4$
Wmw	0.85 ± 0.06	0.96	2	{4}	$17.8s \pm 4.6$	Wmw	0.74 ± 0.05	0.86	3	{6}	$15.3s \pm 3.6$
AveM	0.82 ± 0.05	0.94	3		$3.0s \pm 1.1$	Corr	0.74 ± 0.05	0.84	3	{6}	$2.3s \pm 0.7$
Acc	0.82 ± 0.06	0.93	3		$2.7s \pm 0.8$	Dist	0.73 ± 0.05	0.82	4		$1.2s \pm 0.4$
Ave	0.80 ± 0.06	0.92	4		$2.8s \pm 0.9$	Acc	0.72 ± 0.06	0.85	5		$2.3s \pm 0.6$
Bands	0.79 ± 0.06	0.91	4		$2.9s \pm 1.1$	Ave	0.71 ± 0.05	0.82	5		$2.6s \pm 1.0$
Incr	0.79 ± 0.07	0.90	4		$3.5s \pm 0.7$	AveM	0.70 ± 0.06	0.82	6		$2.4s \pm 0.8$
			$(p = 5.5 \times 10^{-21})$						$(p = 1.5 \times 10^{-17})$		
	Ped						Yst_1				
Wmw	0.93 ± 0.01	0.94	1	{4-8}	$49.2m \pm 7.1$	Wmw	0.84 ± 0.02	0.88	1	{4-7}	$1.8m \pm 0.4$
AucF	0.92 ± 0.01	0.94	2	{5-8}	$71.3m \pm 9.9$	AucF	0.83 ± 0.02	0.87	2	{5-7}	$2.1m \pm 0.6$
AucE	0.92 ± 0.01	0.94	3	{6-8}	$5.8m \pm 1.9$	Dist	0.83 ± 0.03	0.87	2	{5-7}	$6.0s \pm 1.6$
Dist	0.90 ± 0.02	0.92	4	{7-8}	$2.4m \pm 1.1$	Amse	0.82 ± 0.02	0.86	2	{5-7}	$13.2s \pm 4.7$
Corr	0.89 ± 0.01	0.92	5	{7-8}	$4.5m \pm 2.9$	AucE	0.82 ± 0.02	0.87	2	{5-7}	$13.3s \pm 3.3$
Amse	0.88 ± 0.02	0.91	6	{8}	$4.5m \pm 0.9$	Bands	0.82 ± 0.02	0.86	3	{6-7}	$21.8s \pm 10.7$
Bands	0.87 ± 0.03	0.93	6	{8}	$4.2m \pm 4.5$	Corr	0.81 ± 0.02	0.86	4	{7}	$12.8s \pm 3.0$
Ave	0.87 ± 0.04	0.92	6	{8}	$5.0m \pm 3.0$	AveM	0.79 ± 0.04	0.87	5		$12.3s \pm 4.0$
AveM	0.86 ± 0.04	0.91	7	{8}	$5.2m \pm 1.9$	Incr	0.79 ± 0.05	0.87	6		$15.7s \pm 4.9$
Incr	0.86 ± 0.04	0.92	7	{8}	$6.1m \pm 2.1$	Ave	0.79 ± 0.03	0.85	6		$13.3s \pm 4.7$
Acc	0.80 ± 0.12	0.92	8		$5.4m \pm 1.8$	Acc	0.76 ± 0.07	0.84	7		$13.5s \pm 5.7$
			$(p = 1.7 \times 10^{-49})$						$(p = 3.5 \times 10^{-34})$		
	Yst_2						*Bal*				
Amse	0.96 ± 0.01	0.98	1	{3-6}	$11.4s \pm 2.9$	Wmw	0.86 ± 0.08	0.98	1	{3-7}	$26.9s \pm 8.0$
Corr	0.95 ± 0.02	0.98	1	{3-6}	$10.3s \pm 3.1$	AucF	0.84 ± 0.09	0.98	2	{5-7}	$28.1s \pm 7.6$
Wmw	0.95 ± 0.02	0.98	2	{5-6}	$1.4m \pm 0.3$	AucE	0.84 ± 0.11	0.98	2	{5-7}	$5.0s \pm 1.3$
AucF	0.95 ± 0.03	0.98	2	{5-6}	$1.6m \pm 0.3$	AveM	0.84 ± 0.12	0.98	2	{5-7}	$4.9s \pm 1.4$
Bands	0.95 ± 0.02	0.98	2	{5-6}	$19.2s \pm 6.8$	Incr	0.83 ± 0.11	0.98	2	{5-7}	$5.8s \pm 2.2$
Dist	0.94 ± 0.03	0.97	2	{5-6}	$5.8s \pm 2.3$	Amse	0.78 ± 0.10	0.97	3	{6-7}	$5.2s \pm 1.4$
AveM	0.93 ± 0.03	0.97	3	{6}	$13.4s \pm 5.1$	Bands	0.77 ± 0.11	0.98	4	{7}	$5.0s \pm 3.8$
Ave	0.93 ± 0.04	0.97	4		$12.6s \pm 7.9$	Dist	0.77 ± 0.13	0.96	4	{7}	$2.7s \pm 1.3$
AucE	0.92 ± 0.03	0.98	5		$15.2s \pm 5.3$	Corr	0.75 ± 0.11	0.98	5	{7}	$4.9s \pm 1.9$
Incr	0.92 ± 0.04	0.97	5		$15.5s \pm 5.0$	Ave	0.71 ± 0.15	0.98	6	{7}	$4.7s \pm 1.5$
Acc	0.91 ± 0.04	0.97	6		$12.6s \pm 7.9$	Acc	0.55 ± 0.09	0.90	7		$5.1s \pm 2.0$
			$(p = 8.5 \times 10^{-19})$						$(p = 1.1 \times 10^{-46})$		

1 (in Ion) as these fitness functions are not (statistically) significantly different to one other.

A blank entry for "Beats" means that the given fitness function is not statistically significantly better than any other fitness function. For example, in Table 4.4 for Ion, the fitness functions $AveM$ and Acc both have an s-rank of 3 and these are not significantly better than any other fitness function. Each fitness

function in Table 4.4 is ordered from best to worst average AUC in each task, and the higher the s-rank, the better the average AUC for a fitness function.

Table 4.4 also shows the p-values under the null hypothesis from the ANOVA F-test of the AUC for each task. As the p-values are all substantially lower than 0.05 (5% level of significance) in each task, the null hypothesis is rejected at a 5% level of significance.

Analysis of AUC-based Fitness Functions.

Table 4.4 shows that the two AUC-based fitness functions Wmw and Auc_F achieve the best AUC results on four out of six tasks (Spt, Ped, Yst_1 and Bal). In the remaining two tasks, Auc_F is not (statistically) significantly different to the fitness function that achieves the best AUC in these tasks ($Corr$ for Ion and $Amse$ for Yst_2). These high AUC results are not unexpected for Wmw and Auc_F as both these measures use approximations of the AUC directly in fitness. These two fitness functions incur the longest average training times on the tasks, as expected. The functions Wmw and Auc_F take approximately 5-8 times longer than the other fitness functions on these tasks.

On the other hand, Auc_E shows substantially faster training times than Auc_F and Wmw in all tasks, while the AUC performances for Auc_E is not significantly different to Auc_F and Wmw in five out of six tasks. The only exception is Yst_1 where Auc_E has a statistically significantly lower AUC than Auc_F and Wmw. However, the difference in average training times between Auc_E and Auc_F/Wmw is substantial, particularly in the largest task, Ped (which has more than 10,000 training examples). Here Auc_E takes approximately 5 minutes on average compared to 71 and 49 minutes for Auc_F and Wmw, respectively. This suggests that while Wmw gives a very close approximation to the full AUC in the fitness function (Auc_F), no substantial gain can be made in terms of reducing the training time. However, Auc_E offers a significant reduction in training time while still evolving solutions with high AUC.

Analysis of New Fitness Functions.

Table 4.4 shows that the AUC for the new fitness functions $Dist$, $Amse$ and $Corr$ are as good as Auc_E in five out of six tasks. Each is statistically significantly better than Auc_E in exactly one task (Yst_2), and not significantly different to Auc_E in exactly four tasks. The training times using these three fitness functions are also faster than Auc_E in all tasks. This is most apparent using $Dist$, where

the average training time is approximately twice as fast as Auc_E in all tasks. This suggests that these new fitness functions are fast and effective measures of classifier separability, competitive to AUC-based measure Auc_E. Of particular interest is $Dist$ which scores good AUC results on the tasks, while consistently showing the fastest training times from all the fitness functions on the tasks.

Interestingly, Table 4.4 shows that the AUC using $Amse$ (Eq. 4.7) is significantly better than the traditional measure Ave (Eq. 4.2) in all tasks except Ped (where AUC is similar). This is interesting as both $Amse$ and Ave are relatively similar classification measures. The only difference being that $Amse$ utilises the magnitude of the genetic program output in fitness to "calibrate" a classifier's outputs to target values for each class, whereas Ave uses only the true positive and true negative rates in fitness (magnitude ignored).

The AUC for $Dist$, $Amse$ and $Corr$ are not statistically significantly different to each another in all tasks. However, $Amse$ outperforms the traditional Ave more often than $Dist$ and $Corr$. $Amse$ is significantly better than Ave in five tasks (Ped is the only exception), whereas $Dist$ and $Corr$ are significantly better than Ave in only two tasks each. These two tasks are Ion and Yst_1 for $Dist$, and Ion and Yst_2 for $Corr$. This suggests that $Amse$ produces relatively good AUC performances *consistently* across the six tasks; while $Dist$ and $Corr$ have very good results in some tasks (such as Ion, Ped and Yst_2), but poorer results in other tasks (such as Spt and Bal). This subtle difference may be due to properties of the $Dist$ and $Corr$ measures compared to $Amse$. $Amse$ is based on a traditional measure in machine learning (mean squared error) which tries to minimise the differences between input and target patterns. On the other hand, $Dist$ and $Corr$ are new separability-based measures; $Corr$ uses the correlation ratio of class outputs (for a given solution), and $Dist$ uses the distance between the two class distributions.

Interestingly, the new measure $Incr$ achieves very good performances on the two tasks with smallest number of minority class examples, namely, Spt and Bal. These tasks only have 24 and 27 training examples, respectively. Table 4.4 shows that $Incr$ has the best s-rank of 1 in Spt and 2 in Bal. This is as good as both Auc_F and Auc_E, and significantly better than Ave, in these tasks. This suggests that the incremental reward scheme in $Incr$ is particularly useful for tasks with very few minority class examples.

The new measure $Bands$ also shows very good AUC results on the Spt task achieving the best an s-rank of 1 (along with Auc_F, Auc_E and $Incr$). This measure also achieves a significantly better AUC than the traditional Ave in three tasks (Spt, Yst_1 and Yst_2). This suggests that rewarding solutions which

have equally high accuracies on both classes in $Bands$ to a greater extend than in Ave, can improve AUC performances over Ave on some tasks. $Band$ also shows significantly better AUC results than $AveM$ in two tasks, Spt and Yst_1 (no significantly difference in AUC in the remaining tasks).

Analysis of Traditional Measures.

As expected, the traditional measures Acc and Ave have among the poorest AUC results on the tasks compared to the AUC-based functions and new fitness functions (particularly $Dist$, $Amse$ and $Corr$). This is particularly noticeable when the level of class imbalance in a task is high (such as Ped, Yst_1, Yst_2 and Bal). Interestingly, in the Bal task, the alternative average-based measure $AveM$ significantly outperforms Ave, and achieves a higher average AUC than all of the new fitness functions (only significantly better than $Corr$). This suggests that $AveM$ is particularly effective in this task (Bal has the highest level of class imbalance of the tasks). This may be because highly biased solutions (with 0% accuracy on one class) are given very poor selection probabilities in the evolution, as $AveM$ returns a fitness value of zero for all these solutions. However, on the remaining five tasks, Ave and $AveM$ produce very similar AUC results, suggesting that either of these averaging functions finds similar-performing solutions (compared to the other fitness functions).

4.5.2 Overall AUC Behaviour

To gain an overall picture of how the fitness functions perform relative to each other over *all* GP runs and tasks, the fitness functions with the three best (highest) AUC performances on a *run-by-run* basis are counted over all runs. As each GP run (using a particular fitness function) is repeated 50 times for each task, the i^{th} run for all fitness functions (for a task) share the same initial starting seed and initial population. This means that the fitness functions that produce the three highest AUC performances in a particular run for a task (i.e. highest AUC, next highest AUC and third highest AUC), can be summed over all runs and tasks (i.e. 50 runs $\times$ 6 tasks $=$ 300 total GP runs).

Table 4.5 shows the number of top-three AUC performances on a run-by-run basis (on the *test* sets) for all 300 GP runs. The percentage values correspond to the number of first, second or third place totals for a fitness function, as a fraction of the total number of runs (300). These percentages (columns in Table 4.5) sum to

Table 4.5: Total number and percentage of first, second and third place AUC positions on a run-by-run basis over 50 GP runs and six tasks (300 total runs).

Fitness	First Place		Second Place		Third Place		Total Number of
Function	%	Number	%	Number	%	Number	Top-Three Positions
Wmw	20.3	61	21.0	63	12.7	38	162
AucF	17.7	53	18.7	56	17.3	52	161
AucE	13.7	41	13.0	39	13.7	41	121
Dist	10.0	30	10.7	32	11.7	35	97
Amse	9.7	29	7.7	23	7.7	23	75
Corr	8.3	25	8.3	25	6.0	18	68
Incr	7.0	21	5.7	17	7.7	23	61
Bands	4.0	12	7.3	22	7.7	23	57
AveM	6.3	19	3.3	10	5.7	17	46
Ave	3.0	9	3.3	10	5.3	16	35
Acc	1.3	4	2.0	6	4.7	14	24

100% over all fitness functions[2]. For example, the first line in Table 4.5, under the "First Place" column, shows that the solutions evolved using the fitness function Wmw score the highest AUC in 20.3% of all runs (in 61 of 300 runs). The fitness function that scored the next highest number of first-place AUC positions is Auc_F in 17% of all runs (in 53 of 300 runs).

The fitness functions in Table 4.5 are ordered according to the total number of top three-positions (shown in the right-most column).

Table 4.5 shows a distinctive pattern in overall AUC behaviour for the fitness functions across the tasks. The three AUC-based functions, Wmw, Auc_F and Auc_E, achieve the highest total number of top-three placements across all GP experiments, as these appear at the top of Table 4.5. This confirms the analysis from the previous section that these fitness functions typically achieved the best AUC result on these tasks. This is not surprising as these measures, particularly Auc_F and Auc_E, use the AUC directly in fitness. It is interesting that Wmw achieves a greater number of first and second place AUC positions than even Auc_F over all runs and tasks, as Wmw uses a different statistical-based equation to calculate the AUC.

[2]The reader will notice that the total number of first place and second place rankings over *all* fitness functions in Table 4.5 does not sum to exactly 300. This is due to joint first, second or third place placements for two or more fitness functions on a given run, e.g., if two fitness functions score equivalent AUC values that are also the highest in a given run, both are counted in the first-place position (the fitness function with the next highest AUC is then counted in the third-place position). This means that the corresponding percentages in these two columns are also slightly over 100% (by at most, 1.3%).

The new fitness function $Dist$ achieves the highest percentage of first, second and third place AUC positions relative to the other new fitness functions ($Amse$, $Corr$, $Incr$ and $Bands$). This suggests that this fitness function shows the best overall performance over all runs and tasks compared to the other new fitness functions. Another advantage of $Dist$ is the fast training times on the tasks (as shown in Table 4.4). Table 4.5 shows that $Amse$, followed by $Corr$, achieve the next best overall AUC performance ranking from all the new fitness functions; while $Incr$ and $Bands$ have the fewest total number of top-three AUC positions from the new fitness functions. This shows that $Incr$ and $Bands$ generally show poorer overall AUC results than the other new fitness functions ($Dist$, $Amse$ and $Corr$) in these tasks.

Also not surprising, the traditional measures, $AveM$, Ave and Acc, show the worst AUC performances from all the fitness functions, ranking at the bottom of Table 4.5. As expected, the standard GP fitness function Acc is the worst-ranked fitness function in Table 4.5. A closer analysis of these results reveals that the first, second and third-place positions for Acc in Table 4.5 is only from the Ion and Spt tasks, as these tasks have relatively low levels of class imbalance. $AveM$ achieves a higher overall ranking than Ave because $AveM$ achieves very good AUC results in the Bal task (as discussed in the section above).

4.5.3 Typical GP ROC Curves

To illustrate how the GP classifiers capture the trade-off between the minority and majority accuracy, Figure 4.4 shows typical ROC curves from the evolved GP classifiers using the fitness functions. In these figures the true positive (TP) rate is the minority accuracy, and false positive (FP) rate is $1-$the majority accuracy. Figure 4.4 shows ROC curves for the two AUC-based fitness functions Auc_F and Auc_E (Wmw is omitted as its AUC is very similar to Auc_F in these tasks); the three "best" overall new fitness functions ($Dist$, $Amse$ and $Corr$ as these are the highest ranked in Table 4.5); and the two traditional measures Ave and Acc. The ROC curves in Figure 4.4 are generated using the evolved solutions from a particular run whose AUC is similar to the average AUC performance reported in Table 4.4 (to provide an indication of a typical ROC curve). The ROC curves for the Ion, Ped, Yst_1 and Bal tasks are shown in Figure 4.4. The remaining two tasks are omitted for space constraints.

Figure 4.4 clearly shows why the AUC for some fitness functions (such as Auc_F) are better than others, while some (such as Acc) are much worse, in terms of

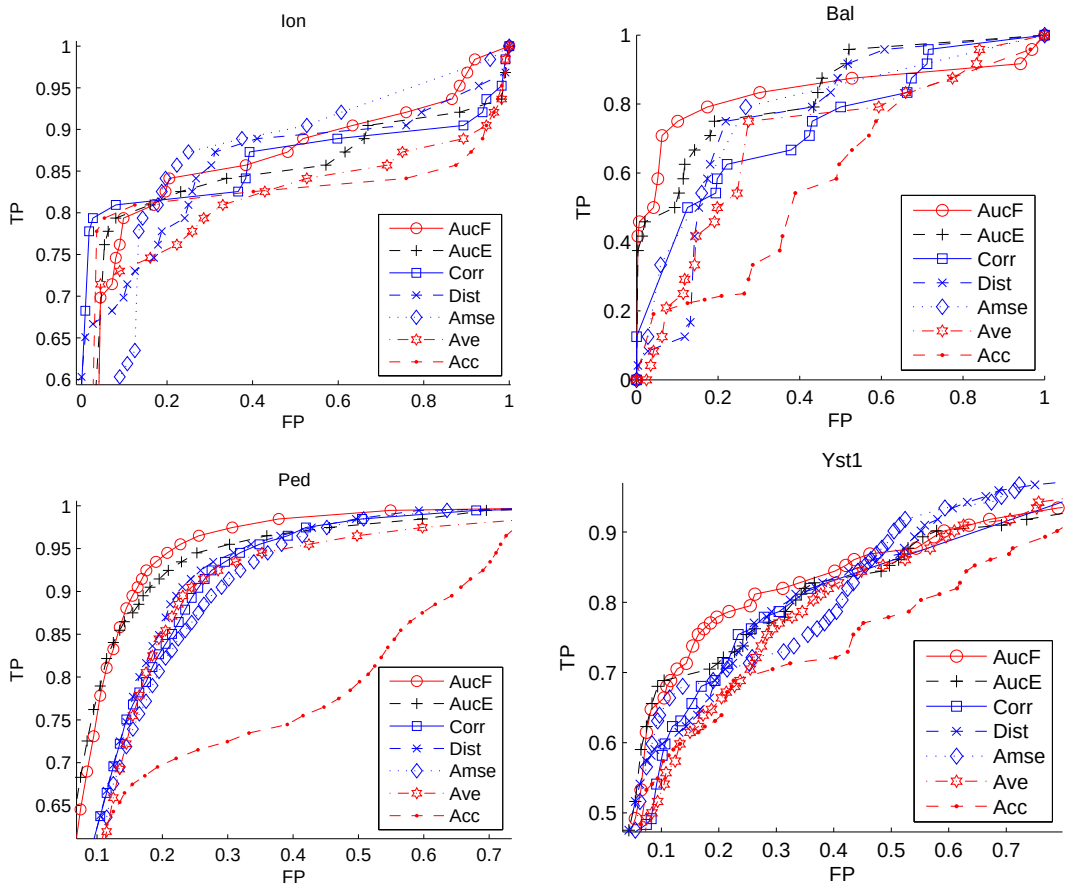


Figure 4.4: Typical ROC Curves (test set) for the GP fitness functions on four tasks. The true positive (TP) rate is the minority class accuracy, and false positive (FP) rate is 1−the majority class accuracy. The axis scopes are different in each figure.

the trade-offs between the TP and FP rates. Interestingly, these ROC curves show that very good FP rates can be achieved (e.g. $FP \geq 0.2$) while the corresponding TP rates also remain relatively good, e.g., between 0.6–0.7 in Bal and Yst_1, and 0.8 (or higher) in Ped and Ion. In contrast, very good TP rates (e.g. $TP \geq 0.8$) show poor (high) FP rates on these four tasks. This means that high majority class accuracies can be achieved with relatively little resistance in the corresponding minority class accuracies, but not for the opposite case. Here high minority class accuracies cause a more significant trade-off in majority class accuracies.

4.5.4 Naive Bayes and Support Vector Machines

This section compares the GP performances with two other popular machine learning approaches, namely, Naive Bayes (NB) and Support Vector Machine (SVM). Table 4.6 shows the AUC and training time using NB and SVM on the tasks, generated using the WEKA [82] machine learning package.

Table 4.6: AUC and training time for a single run using Naive Bayes (NB) and Support Vector Machines (SVM) on the tasks.

	Ion		Spt		Ped		Yst_1		Yst_2		Bal	
	Auc	Time	Auc	Time	Auc	Time	Auc	Time	Auc	Time	Auc	Time
NB	0.91	0.02s	0.83	0.04s	0.92	20.1s	0.83	0.03s	0.95	0.02s	0.5	0.001s
SVM	0.93	0.08s	0.68	0.2s	0.93	3.8m	0.71	1.2s	0.85	1.4s	0.5	0.05s

Table 4.7: GP fitness function results on the tasks (repeat of Table 4.4). The column *SR* denotes the significance rank (s-rank) of a fitness function and the column *Beats* denotes other s-rank(s) with a (statistically) significantly poorer AUC.

Fitness	AUC		Stat. Test		Training	Fitness	AUC		Stat. Test		Training
Func.	Average	Best	SR	Bats	Times	Func.	Average	Best	SR	Beats	Times
	Ion						*Spt*				
Corr	0.87 ± 0.04	0.94	1	{3-4}	$2.4s \pm 0.5$	AucF	0.77 ± 0.04	0.83	1	{4-5}	$12.8s \pm 3.8$
Dist	0.86 ± 0.05	0.95	1	{3-4}	$1.4s \pm 0.5$	Incr	0.76 ± 0.05	0.86	1	{4-5}	$4.4s \pm 1.8$
Amse	0.85 ± 0.05	0.94	2	{4}	$2.8s \pm 0.9$	AucE	0.76 ± 0.04	0.86	1	{4-5}	$2.8s \pm 0.9$
AucF	0.85 ± 0.04	0.94	2	{4}	$20.0s \pm 5.3$	Bands	0.76 ± 0.04	0.87	1	{4-5}	$2.6s \pm 0.8$
AucE	0.85 ± 0.05	0.93	2	{4}	$3.1s \pm 0.9$	Amse	0.75 ± 0.04	0.84	2	{5}	$2.2s \pm 0.4$
Wmw	0.85 ± 0.06	0.96	2	{4}	$17.8s \pm 4.6$	Wmw	0.74 ± 0.05	0.86	3	{6}	$15.3s \pm 3.6$
AveM	0.82 ± 0.05	0.94	3		$3.0s \pm 1.1$	Corr	0.74 ± 0.05	0.84	3	{6}	$2.3s \pm 0.7$
Acc	0.82 ± 0.06	0.93	3		$2.7s \pm 0.8$	Dist	0.73 ± 0.05	0.82	4		$1.2s \pm 0.4$
Ave	0.80 ± 0.06	0.92	4		$2.8s \pm 0.9$	Acc	0.72 ± 0.06	0.85	5		$2.3s \pm 0.6$
Bands	0.79 ± 0.06	0.91	4		$2.9s \pm 1.1$	Ave	0.71 ± 0.05	0.82	5		$2.6s \pm 1.0$
Incr	0.79 ± 0.07	0.90	4		$3.5s \pm 0.7$	AveM	0.70 ± 0.06	0.82	6		$2.4s \pm 0.8$
	Ped						*Yst_1*				
Wmw	0.93 ± 0.01	0.94	1	{4-8}	$49.2m \pm 7.1$	Wmw	0.84 ± 0.02	0.88	1	{4-7}	$1.8m \pm 0.4$
AucF	0.92 ± 0.01	0.94	2	{5-8}	$71.3m \pm 9.9$	AucF	0.83 ± 0.02	0.87	2	{5-7}	$2.1m \pm 0.6$
AucE	0.92 ± 0.01	0.94	3	{6-8}	$5.8m \pm 1.9$	Dist	0.83 ± 0.03	0.87	2	{5-7}	$6.0s \pm 1.6$
Dist	0.90 ± 0.02	0.92	4	{7-8}	$2.4m \pm 1.1$	Amse	0.82 ± 0.02	0.86	2	{5-7}	$13.2s \pm 4.7$
Corr	0.89 ± 0.01	0.92	5	{7-8}	$4.5m \pm 2.9$	AucE	0.82 ± 0.02	0.87	2	{5-7}	$13.3s \pm 3.3$
Amse	0.88 ± 0.02	0.91	6	{8}	$4.5m \pm 0.9$	Bands	0.82 ± 0.02	0.86	3	{6-7}	$21.8s \pm 10.7$
Bands	0.87 ± 0.03	0.93	6	{8}	$4.2m \pm 4.5$	Corr	0.81 ± 0.02	0.86	4	{7}	$12.8s \pm 3.0$
Ave	0.87 ± 0.04	0.92	6	{8}	$5.0m \pm 3.0$	AveM	0.79 ± 0.04	0.87	5		$12.3s \pm 4.0$
AveM	0.86 ± 0.04	0.91	7	{8}	$5.2m \pm 1.9$	Incr	0.79 ± 0.05	0.87	6		$15.7s \pm 4.9$
Incr	0.86 ± 0.04	0.92	7	{8}	$6.1m \pm 2.1$	Ave	0.79 ± 0.03	0.85	6		$13.3s \pm 4.7$
Acc	0.80 ± 0.12	0.92	8		$5.4m \pm 1.8$	Acc	0.76 ± 0.07	0.84	7		$13.5s \pm 5.7$
	Yst_2						*Bal*				
Amse	0.96 ± 0.01	0.98	1	{3-6}	$11.4s \pm 2.9$	Wmw	0.86 ± 0.08	0.98	1	{3-7}	$26.9s \pm 8.0$
Corr	0.95 ± 0.02	0.98	1	{3-6}	$10.3s \pm 3.1$	AucF	0.84 ± 0.09	0.98	2	{5-7}	$28.1s \pm 7.6$
Wmw	0.95 ± 0.02	0.98	2	{5-6}	$1.4m \pm 0.3$	AucE	0.84 ± 0.11	0.98	2	{5-7}	$5.0s \pm 1.3$
AucF	0.95 ± 0.03	0.98	2	{5-6}	$1.6m \pm 0.3$	AveM	0.84 ± 0.12	0.98	2	{5-7}	$4.9s \pm 1.4$
Bands	0.95 ± 0.02	0.98	2	{5-6}	$19.2s \pm 6.8$	Incr	0.83 ± 0.11	0.98	2	{5-7}	$5.8s \pm 2.2$
Dist	0.94 ± 0.03	0.97	2	{5-6}	$5.8s \pm 2.3$	Amse	0.78 ± 0.10	0.97	3	{6-7}	$5.2s \pm 1.4$
AveM	0.93 ± 0.03	0.97	3	{6}	$13.4s \pm 5.1$	Bands	0.77 ± 0.11	0.98	4	{7}	$5.0s \pm 3.8$
Ave	0.93 ± 0.04	0.97	4		$12.6s \pm 7.9$	Dist	0.77 ± 0.13	0.96	4	{7}	$2.7s \pm 1.3$
AucE	0.92 ± 0.03	0.98	5		$15.2s \pm 5.3$	Corr	0.75 ± 0.11	0.98	5	{7}	$4.9s \pm 1.9$
Incr	0.92 ± 0.04	0.97	5		$15.5s \pm 5.0$	Ave	0.71 ± 0.15	0.98	6	{7}	$4.7s \pm 1.5$
Acc	0.91 ± 0.04	0.97	6		$12.6s \pm 7.9$	Acc	0.55 ± 0.09	0.90	7		$5.1s \pm 2.0$

The SVM uses a sequential minimal optimisation algorithm with an RBF kernel and Gamma value[3] of 10. For convenience, the GP results for the different fitness functions are repeated in Table 4.7 (under Table 4.6). Note that these GP results are identical to those shown previously in Table 4.4.

Table 4.7 shows that the *best* classifiers evolved by GP (over 50 experiments) are as good as, or in most case better than, NB and SVM (in Table 4.6) for these tasks, particular with the AUC-based and new fitness functions ($Dist$, $Amse$, $Corr$) in GP. This suggests that these GP fitness functions succeeded in evolving good classifiers with little overlap between two class distributions (or high AUC) on the tasks compared to NB and SVM. In Bal in particular, NB and SVM show very poor classification results compared to GP. The AUC for NB and SVM is 0.5 in Bal, indicating that these methods show highly biased classification results that are no better than random guessing on this test set. This indicates that the high level of class imbalance in this task represents a difficult challenge for NB and SVM but not for GP with the new fitness functions.

Comparing the average AUC of the GP approaches with NB and SVM, all the GP fitness functions, including Acc, achieve better AUC results than NB and SVM on Bal. On the tasks with minority class representation between 10–20% of all examples (Ped, Yst_1 and Yst_2), the GP fitness function with the highest average AUC in Table 4.7 (Wmw in Ped and Yst_1, and $Amse$ in Yst_2), shows similar average AUC performances to NB; for these tasks, both GP and NB are better than SVM. These results suggests that the ability to choose/develop an effective fitness function to evolve classifiers with high AUC gives GP an advantage over NB and SVM particularly when data sets are highly unbalanced.

Table 4.6 also shows that a single run of SVM, and particularly NB, is faster than the average GP training times on these tasks. However, this is not a serious concern as GP only takes a few seconds in nearly all tasks. The only exception is Ped which is the largest data set (more than 24000 examples). Here most of the GP methods and SVM take a few minutes.

Further Analysis in Ion and Spt Tasks

In two tasks (Ion and Spt), the GP fitness function with the *highest* average AUC in Table 4.7 is slightly worse than a single run of either SVM or NB. In Ion, SVM and NB show a better AUC than GP with $Corr$ (on average); and in Spt, NB has a higher AUC than GP with Auc_F (on average). This may be due to the

[3]Gamma=10 generally gave the best classification results from experiments using 0.1, 1, 10, and 100.

complexity of this problem, rather than the relatively low level of class imbalance in these tasks. Ion has 34 features; this is the largest number of features from the tasks and represents a very large search-space of classifiers for GP, given that the maximum program depth of GP classifiers is restricted to 8. Adjusting the evolutionary parameters to allow GP to more effectively explore this search-space, such as increasing the maximum GP program depth parameter (e.g., to 12) or the population size (e.g. to 1000), should improve GP performances.

To test this hypothesis, i.e., whether the average AUC performance for GP can be further improved using a maximum program depth of 12 and population size of 1000, the experiments are repeated for Ion and Spt using these new parameters. Only the GP fitness functions $Corr$ and Auc_F for Ion and Spt, respectively, are considered with the new evolutionary parameters as these fitness functions have the highest average AUC on these tasks in Table 4.7.

New GP Results for Ion and Spt

For the new GP experiments, the average AUC ($\pm$ standard deviation) for $Corr$ on Ion is 0.91 ($\pm$ 0.03), and the best AUC is 0.98 (over 50 runs). Likewise, the average AUC for Auc_F on Spt is 0.82 ($\pm$ 0.02), and the best AUC is 0.84 (over 50 runs). This shows that on average, the AUC for GP on Ion is as good as NB, and only slightly lower than SVM. Similarly, the best AUC achieved by GP in Ion (over 50 runs) is substantially higher than both NB and SVM. Likewise, the average AUC for GP on Spt is only slightly lower than NB (and much better than SVM), but the best AUC achieved by GP (over 50 runs) is much better than both methods.

These new GP results confirm the hypothesis discussed above that the original not very good AUC results in Table 4.7. are more due to the complexity of these problems than the class imbalance factor. When the search-space is increased in GP (by updating the evolutionary parameters), performances are improved as the new evolved classifiers are more competitive in terms of AUC compared to NB and SVM in these two tasks.

4.6 Results for Weighted-Average Fitness Function

This section investigates whether different configurations in the weighted-average GP fitness function $Wave$ (Eq. 4.4) significantly affects the AUC of the evolved solutions. In other words, we check whether different configurations

Table 4.7: Average ($\pm$ standard deviation) AUC for weighted-average fitness function Ave (Eq. 4.2) on the tasks. The *SR* denotes the significance rank (s-rank) for a weight value and *beats* denotes other s-rank(s) with a (statistically) significantly poorer AUC.

Weight (W)	AUC	Stat. Test SR	Stat. Test Beats	AUC	Stat. Test SR	Stat. Test beats	AUC	Stat. Test SR	Stat. Test Beats
	Ion			*Spt*			*Ped*		
0.2	0.83 ± 0.05	1	{2-3}	0.70 ± 0.09	3		0.80 ± 0.09	3	
0.3	0.82 ± 0.05	1	{2-3}	0.73 ± 0.05	1	{4-5}	0.86 ± 0.06	1	{3}
0.4	0.82 ± 0.05	1	{2-3}	0.72 ± 0.05	2	{5}	0.86 ± 0.05	1	{3}
0.5	0.80 ± 0.06	1	{2-3}	0.71 ± 0.05	3		0.87 ± 0.04	1	{3}
0.6	0.80 ± 0.05	1	{2-3}	0.69 ± 0.06	4		0.86 ± 0.03	1	{3}
0.7	0.76 ± 0.07	2	{3}	0.69 ± 0.06	4		0.85 ± 0.05	2	
0.8	0.71 ± 0.09	3		0.68 ± 0.06	5		0.82 ± 0.05	3	
	$p = 1.7\times10^{-26}$			$p = 0.0013$			$p = 1.1\times10^{-9}$		
	Yst_1			Yst_2			*Bal*		
0.2	0.76 ± 0.07	3		0.92 ± 0.05	1		0.61 ± 0.13	3	
0.3	0.78 ± 0.05	2		0.92 ± 0.05	1		0.72 ± 0.14	1	{3}
0.4	0.80 ± 0.04	1	{3}	0.93 ± 0.04	1		0.72 ± 0.13	1	{3}
0.5	0.79 ± 0.03	2		0.93 ± 0.04	1		0.71 ± 0.15	1	{3}
0.6	0.78 ± 0.05	2		0.92 ± 0.04	1		0.69 ± 0.14	2	
0.7	0.77 ± 0.06	3		0.93 ± 0.04	1		0.67 ± 0.14	2	
0.8	0.73 ± 0.10	3		0.92 ± 0.05	1		0.67 ± 0.14	2	
	$p = 3.2\times10^{-3}$			$p = 0.49$			$p = 5.5\times10^{-5}$		

affect how well the class outputs are separated with respect to each other in the evolved solutions. Table 4.7 shows the average AUC of the evolved GP classifiers experimental using the seven different weighting configurations in $Wave$ on the tasks. The weighting configurations for W are between 0.2 and 0.8 at intervals of 0.1. Recall that in $Wave$, *W* specifies the weight for the minority class accuracy and $1 - W$ for the majority class accuracy.

Similar to the previous experimental results, an ANOVA F-test is first used to statistically test the null hypothesis (i.e., no difference in AUC for the different W values over 50 runs) at a 5% level of significance. The p-values from the F-test, shown in Table 4.7 for each task, are lower than than 0.05 in all tasks except Yst_2 (where p is 0.49). This indicates that the null hypothesis is rejected (at a 5% level of significance) in these tasks except Yst_2. In Yst_2, all weighting configurations show very similar AUC results (that are not statistically significantly different).

Tukey's HSD test [166] is also used as the multiple comparisons test, to find the statistically significant differences between AUC values in the tasks (except

Yst_2). An s-rank is also assigned to each W value in Table 4.7 to summarise which weighting configurations have statistically significantly better AUC values than other configurations. In Table 4.7, the *SR* denotes the s-rank for a given W configuration and *Beats* denotes other s-rank(s) with a (statistically) significantly poorer AUC. For example, the first line in Table 4.7 for Ion (for "Stat. Test") shows that $W = 0.2$ achieves the best s-rank of 1, and that this AUC is significantly better than s-ranks 2 and 3 (W values of 0.7 and 0.8, respectively).

4.6.1 Analysis of Results

According to Table 4.7, *no* configuration of W where $W \neq 0.5$ shows a statistically significantly better AUC than an equal weighting ($W = 0.5$) on the tasks. This means that no other configuration of W where $W \neq 0.5$ improves the AUC sufficiently to be statistically significantly better than an equal weighting configuration. In fact, the W configuration with the highest average AUC (on a task-by-task basis) is still not as good as the AUC-based and new fitness functions from Table 4.4 in any of the tasks. This suggests that the tweaking the weighting configuration in $Wave$ does not significantly improve the AUC in the evolved solutions on these tasks.

As expected, "extreme" weighting configurations (such as W of 0.2 or 0.8) generally show the worst AUC results. In four tasks (Spt, Ped, Yst_1 and Bal), the W configuration with the highest average AUC is statistically significantly better than these two extreme W values. This is not surprising as extreme weights in $Wave$ favour biased solutions which have high accuracy rates on one class alone. Only in Ion does the extreme W value of 0.2 show good AUC results, most likely due to the relatively low level of class imbalance in Ion. Weighting configurations slightly favouring majority class accuracy over minority class accuracy ($0.3 < W \leq 0.5$) produce slightly better AUC than the opposite case ($W > 0.5$), but this difference is only statistically significant in one task (Spt).

These results show that the choice of weights in $Wave$ will not significantly affect the AUC in the evolved solutions unless "extreme" weights are selected.

However, it must be mentioned that while the AUC of the evolved solutions are not statistically significantly different (except for extreme weights), the main advantage of $Wave$ is the *frontier* produced by the evolved solutions, as shown in Figure 4.5. These figures show the performances of the evolved solutions on the minority and majority classes (on the test set) when these solutions are evaluated using zero as the class threshold, for three tasks (Ped, Yst_2 and Bal). These

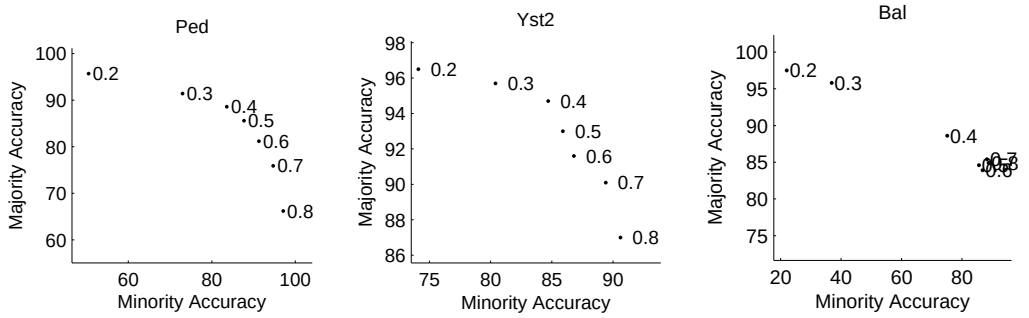


Figure 4.5: Minority and majority class accuracies (on the test sets) for weighting coefficient W in fitness function $Wave$ (axis scopes are different in each figure).

performances represent the average performance over 50 GP runs for the different W configurations; and the vertical and horizontal axis in these figures correspond to the minority and majority class accuracy, respectively. The remaining tasks are omitted for space constraints but these show very similar frontiers to Ped and Yst_2 (in fact, the $Wave$ frontier for these tasks are shown and discussed in more detail in the next chapter).

However, an major limitation of $Wave$ is that multiple GP runs are required (each with a different W configuration in the fitness function) to produce the frontiers shown in Figure 4.5. This can be a time consuming process, e.g., Figure 4.5 needed a total of 350 GP experiments (assuming 50 GP experiments are used for each W configuration). Another limitation of $Wave$ is that there is no guarantee that the points along the frontier (i.e. for the different W values) will be uniformly "spread out" along the two objectives, as seen for Bal in Figure 4.5. Here W values between 0.5 and 0.8 produce a point along the frontier that is very similar in objective-space.

4.7 Evolved GP Programs

This section examines four evolved GP classifiers using the fitness function Auc_E on the Bal task. Bal is chosen since the high level of class imbalance in Bal makes this a difficult classification problem for canonical GP, NB and SVM to solve (as demonstrated by the biased results in Table 4.4 for canonical GP, and Table 4.6 for NB and SVM). This particular fitness function is also chosen as it finds solutions with good AUC performances on this task.

Note that while this analysis is not the primary goal of this thesis, it is included to provide an overall indication on the kinds of solutions evolved by the GP approaches. As the representation of the evolved classifiers is a major advantage

```
(if<0
   (-
      (- (+ 0.94 f1) (- f3 -0.89))
      (*
         (if<0 (* f3 0.63) (+ f3 f0) f3)
         (% (- f2 f0) (if<0 f2 (% (% -0.89 -0.34) (+ f3 f1)) f0))))
   (+
      (* (- 0.02 0.09)
          (%
             (% 0.82 (+ f0 (if<0 f0 f1 0.95)))
             (* (+ (% -0.94 -0.98) 0.24) (* (- -0.62 0.37) (* -0.56 -0.34)))))
      (- (* f0 0.06) (if<0 f2 -0.89 f0)))
   (%
      (%
         (- (% (if<0 (- f2 f2) (* 0.16 f1) (if<0 f2 f3 -0.99)) f2) f3)
         (if<0 0.94 f2 f0))
      (% (- f1 -0.18) (% f2 (- 0.02)))))
```

Figure 4.6: Evolved GP classifier with a high AUC (0.98) on Bal.

of GP, examining the evolved GP trees can provide useful insights into how GP learns to solve a particular problem.

The four evolved GP classifiers analysed below are shown in Figures 4.6 – 4.9. These programs are categorised into classifiers with above average AUC (Figures 4.6 and 4.7), and classifiers with average AUC (Figures 4.8 and 4.9). In these figures, the four input features in Bal correspond to $f0$—$f3$ in the evolved programs. For convenience, these programs have been indented for easier interpretability.

4.7.1 Programs with high AUC

The first program that is analysed, shown in Figure 4.6, has an AUC of 0.98 which represents one of the best AUC performance achieved for Bal (in Table 4.4 on page 94). This program contains three distinct parts which form the input arguments to the `if` function in the root node of the tree. Recall that in the `if` function, if the first argument is negative, the second argument is evaluated and the corresponding value is returned; otherwise, the third argument is evaluated. As this program has very high AUC, the logic expressed by this high level conditional function may be a successful strategy discovered in the GP evolutionary process to classify the data inputs. Interestingly, 8 other GP classifiers out of the 50 GP runs for Bal with this fitness function (Auc_E) share a similar overall structure, i.e., trees with an `if` function at the root node (but with potentially different subtrees). These 9 GP classifiers with this overall structure also generally have above-average AUC performances.

```
(%
  (if<0                                                                    \\left-side
    (*
      (if<0
        (+ (if<0 (if<0 f1 f1 -0.54) (- -0.02 0.67) -0.11) (+ (- f0 f3) (* f2 0.27)))
        0.87
        (- f0 f3))
      (- f3 f1))
    (-
       (+ -0.66 0.93)
       (% 0.95
          (%
            (+ (* f2 0.63) (% 0.75 0.47))
            (* (if<0 -0.82 -0.54 -0.61) (* -0.25 f2)))))
    (-
       (% 0.63 0.25)
       (if<0
           0.25
           (- (% 0.28 0.25) (if<0 (- f2 0.82) (+ f0 -0.16) 0.44))
           (% 0.27 (+ (if<0 f2 -0.25 -0.51) (- f3 f0))))))
  (-                                                                       \\right-side
       (+ (- f1 f1) (- f1 f3))
       (+
           (* (- 0.25 -0.47) -0.44)
           (% (- f0 f2) -0.95))))
```

Figure 4.7: Evolved GP classifier with an above average AUC (0.92) on Bal.

The second program, shown in Figure 4.7, has an AUC of 0.92 on Bal. This is higher than the average AUC achieved by this fitness function over 50 GP runs (0.84 according to Table 4.4), but lower than the AUC of the first program (Figure 4.6). Interestingly, this program also has a similar structure to the program in Figure 4.6, except that the outer-most `if` function is embedded a level deeper in the tree than in Figure 4.6. This means that the left side of the program (as shown in Figure 4.7) uses a similar logic to the first program (e.g. the `if` function), but this logic is incorporated into another function (the protected division operator, %, in the root node). The (above average) AUC of this program affirms the analysis discussed above that this high-level logic in the tree is a successful strategy for good performance. Other evolved GP classifiers with a similar overall structure, i.e., trees with a high-level `if` function (but with potentially different subtrees) embedded one level deep from the root node, are found in 15 out of the 50 GP runs. These programs also generally have good AUC performances.

The poorer performance by this program (compared to Figure 4.6) may be due to the additional structure above the outer-most `if` function, or to different subtrees within the `if` function.

```
(%
   (+                                                                    \\left-side
      f0
      (%
         (+ (+ (% (- -0.01 f3) (% 0.28 0.02)) (if<0 f3 (% f3 0.39) f0))  f1)
         (- f0 -0.12)))
   (-                                                                    \\right-side
      (+ f1 0.16)
      (if<0
         (%
            (if<0
                 (- (- f0 f2) f0)
                 (if<0 (+ -0.83 f0) 0.042 (- f3 f0))
                 (if<0 0.91 (+ -0.83 f0) f2))
            (+ (+ -0.83 f0) (+ (* f3 -0.81) (* 0.91 0.37))))
         (- f3 (* (- f0 (* f2 0.97)) (- (+ f1 f0) (- f1 0.47))))
         f2)))
```

Figure 4.8: Evolved GP classifier with a typical AUC of 0.85 on Bal.

4.7.2 Programs with Average AUC

The third program that is analysed, shown in Figure 4.8, has an AUC of 0.85. This is only slightly higher than the average AUC achieved by this fitness function over 50 GP runs (0.84) but much lower than the AUC of the two previous programs. Unlike the two previous programs, this program does not have an overall structure with an *if* function in the higher levels of the tree. Instead, this program uses a series of nested `if` functions embedded deep within the right side of the tree (as shown in Figure 4.8)

The fourth program, shown in Figure 4.8, has an AUC of 0.84 which represents the average AUC by this fitness function (over 50 GP runs). This classifier is also the smallest program evolved over 50 GP runs using Auc_E. Even though this program does not achieve a very good AUC, its small size makes further analysis easier to try to understand how GP has learnt to solve this problem. Similar to the above program, this program uses no high-level conditional logic operators in the tree as the only `if` function is embedded deep within the tree. The small program size may also suggest that this classifier lacked the representation for good AUC performance compared to the larger (and more complex) programs in Figures 4.6 and 4.7.

4.7.3 Trends

This analysis focuses on one difficult classification task, Bal, and shows that solutions with similar AUC performances tend to have similar overall structures.

```
(%
  (-  (* (+ 0.80 f3) (% f1 f2))   f0)
  (+
     f3)
     (-
        0.11
        (if<0
          (+ (- f0 f1) (- (- f2 f3) (- -0.60 -0.18)))
          (* (- (+ f3 f2) -0.41) (% (- -0.33 f0) (- f2 f0)))
           f0)))
```

Figure 4.9: Smallest evolved GP classifier with an AUC of 0.84 on Bal.

One might expect that analyses of the evolved programs for other tasks (and fitness functions) might also reveal a similar pattern, that is, the *best* evolved programs share a similar overall structure but this structure is different from other solutions. It can be expected that when these evolved programs are grouped together based on their performances, the overall structures of programs within each group are relatively similar to each other, but different from programs in other groups. For example, just as Figures 4.6 and 4.7 are similar in their performance and structures, these are different to Figures 4.8 and 4.9.

This is because solutions in different groups will have different building blocks. For example, the high level `if` function in Figures 4.6 and 4.7 may constitute good building blocks as these are common in well-performing solutions and may represent a successful strategy to achieve good AUC performances.

4.8 Summary

The goal of this chapter was to develop several new fitness functions for classification with unbalanced data to find solutions with good classification ability on the minority and majority class (high AUC). These new fitness functions perform cost adjustment between the two classes during the learning process, allowing the unbalanced learning data to be used "as is" in the learning process. This means that no prior knowledge is required about the input data, nor is any sampling algorithm needed to first re-balance the training data before fitness evaluation.

4.8.1 AUC of Fitness Functions

The AUC performances of the evolved GP solutions and the GP training times using the new fitness functions are compared to several existing approaches in the fitness function for classification. Overall, the three AUC-based fitness

functions find solutions with the best (highest) AUC on the tasks, but also incur the longest training times. The WMW statistic in the fitness function, Wmw, finds solutions with very similar performances to the full AUC in the fitness function Auc_F, but both methods have similarly long training times. In contrast, the reduced-precision AUC in the fitness function, Auc_E, offers a significant reduction in GP training time while still ensuring that the evolved solutions have comparatively high AUC.

A new fitness function measuring the distance between class distributions, $Dist$, finds solutions that perform as well as the AUC-based measure Auc_E, but with training times that are twice as fast as Auc_E on the tasks. Two new fitness functions based on the mean-squared-error for each class, $Amse$, and the correlation ratio, $Corr$, also find solutions with similar performances to Auc_E, and with slightly better training times than Auc_E. Of these, $Amse$ significantly outperforms its counterpart, the traditional measure Ave (which uses the average accuracy of the two classes) in nearly all tasks. A new fitness function which incrementally rewards correct predictions further away from the class boundary ($Incr$), is particular useful in tasks with very few minority class examples, achieving AUC results that are as good as the AUC-based fitness functions (and significantly better than the traditional Ave) in these tasks. Similarly, the new measure $Bands$ which promotes solutions with equally high accuracy on both classes, performs significantly better than the traditional measures Ave and $AveM$ in three out of six tasks.

These GP methods also outperform Naive Bayes (NB) and Support Vector Machines (SVM) on the tasks, particularly when the level of class imbalance in a task is very large. In these cases, both NB and SVM show biased classification results.

4.8.2 AUC of $Wave$ Frontier

Varying the relative importance of the minority and majority class accuracy in a fitness function which uses a weighted-average of these two objectives, $Wave$, does not significantly improve AUC compared to an equal weighting of the two objectives on these tasks. However, different weighting configurations in $Wave$ produces a frontier along the minority and majority class trade-off surface, but generating this frontier is a lengthy process requiring multiple GP runs for the different weighting configurations. This is because each configuration must be specified *a priori* in the fitness function.

4.8.3 Multi-Objective GP

The GP methods described in this chapter focus on the single fittest individual found in the evolutionary search. These methods use either the ROC curve of this individual to capture the performance trade-off between the minority and majority class accuracy, or the frontier of solutions along this trade-off surface produced by the fitness function $Wave$. An alternative approach to approximate the trade-off between these two objectives is to use evolutionary multi-objective optimisation (EMO) to simultaneously evolve a set of the best trade-off solutions (*Pareto front*) along the objectives in a single optimisation run. An EMO approach allows decision-makers to choose a preferred classifier *a posteriori* from the evolved Pareto front without requiring the objective preference to be specified *a priori*. The next chapter develops a multi-objective GP (MOGP) approach where the accuracy of the minority and majority class is trade-off against each other for cost-adjustment in the learning process.

Chapter 5

Multi-objective GP Approach

This chapter is organised as follows. The first section provides an introduction of the main concepts and the chapter goals. The second section outlines the multi-objective GP approach. The third and fourth sections present the experimental results on the tasks. The fifth section provides a summary of this chapter.

5.1 Introduction

In classification with unbalanced data, the accuracy on the minority and majority class is in conflict where increasing the accuracy on one class usually results in a performance trade-off on the other. Two approaches are developed in the previous chapter to represent this performance trade-off in the evolved GP classifiers. The first constructs an ROC curve for an evolved GP classifier where different class thresholds produce different true positive and true negative rates (i.e. different minority and majority class accuracies, respectively). The second generates a frontier of GP classifiers using a weighted-average of the minority and majority class in the fitness function, as shown below, where the weighting coefficient W specifies the relative importance of the objectives.

$$Wave = W \times \text{Minority Accuracy} + (1 - W) \times \text{Majority Accuracy}$$

However, a major limitation of $Wave$ is that the objective preference W must be specified *a priori*. Weighting coefficients are difficult to predict *a priori*, particularly in real-world problems with unbalanced data. These weights can be task-specific and require a lengthy trial and error process to configure as multiple optimisation runs are needed with different weighting coefficients (as shown in the previous chapter).

Evolutionary multi-objective optimisation (EMO) is a fast-growing area of research which offers a promising alternative to learning with multiple conflicting objectives [95][188][53][42][36][40]. Unlike canonical single-objective ("single-predictor") classifier learning techniques where the single fittest individual is returned from the training process, in EMO a set (or *Pareto front*) of the best trade-off solutions is simultaneously evolved along the objectives in a single optimisation run. This allows decision-makers the freedom to choose a preferred classifier (with the desired trade-off) *a posteriori* from the evolved Pareto front, without requiring the objective preference to be specified *a priori*. EMO accomplishes this by treating the objectives independently in the learning process using the notion of Pareto Dominance in fitness.

EMO has shown great success in three main problem domains in classification: model regularisation [96][26][70][92][50], ROC optimisation [162][108][65], and ensemble learning [159][123][170][168]. The first two problem domains assume that the class distributions in the classification tasks are balanced, while EMO for ensemble learning typically uses sampling techniques to first re-balance the training data during fitness evaluation when data is unbalanced. A multi-objective GP (MOGP) approach where the accuracy of the minority and majority class is trade-off against each other for cost adjustment in the learning algorithm when data sets are unbalanced, has not previously been explored.

This chapter addresses this by developing a MOGP approach using the accuracy on the minority and majority class as the two learning objectives. This is different to the canonical single-objective/single-predictor GP from the previous chapter (hereinafter referred to as SGP) where a single evolved genetic program classifier is required to capture the performance trade-off between these two objectives (using an ROC curve). In contrast, the MOGP approach delegates this requirement to the *set* of genetic program classifiers evolved along the Pareto frontier of the minority-majority class trade-off surface. As a result, the MOGP approach does not require that the individual classifiers are highly accurate on the two objectives, but rather that the Pareto front contains a good set of trade-off solutions.

5.1.1 Fitness in MOGP

As mentioned above, EMO treats the objectives as separate in the learning algorithm using Pareto Dominance in the fitness function. In Pareto Dominance, a solution's performance is ranked on all the objectives relative to all other

solutions in the population. This ranking is important as it affects the way selection is performed if the objectives are to be treated independently in the evolution. Coello Coello et al. [42] categorise three main types of Pareto Dominance measures: dominance rank, dominance count and dominance depth. Dominance rank is the number of other individuals that a given individual is dominated by. Dominance count is the number of other individuals that a given individual dominates. Dominance depth sorts the individuals in the population into fronts by depth. Each dominance measure has a different bias towards different regions of the Pareto frontier. Dominance rank and dominance depth (used in the well-known NSGAII [53] algorithm) tend to reward exploration at the edges of the frontier while dominance count (used in the well-known SPEA2 [188] algorithm) tends to reward exploitation in the middle of frontier [42].

In the MOGP approach, it is not clear which Pareto Dominance measure in the fitness function will find better-performing Pareto frontier solutions on these classification tasks with unbalanced data. To address this, this chapter compares two Pareto Dominance measures in the fitness function in MOGP . The first uses dominance rank (from the NSGAII [53] algorithm), and the second uses dominance count (from the SPEA2 [188] algorithm).

5.1.2 Chapter Goals

This chapter has two mains goals. The first is to develop a MOGP approach where the accuracy of the minority and majority class is trade-off against each other, with particular emphasis on how to represent the Pareto Dominance measure in fitness. Two Pareto Dominance-based measures are compared in MOGP to investigate which measure finds better-performing frontier solutions on these tasks. Using the AUC to measure the individual classification ability of a Pareto front solution, the second goal is to investigate how the AUC changes in different regions of the frontier in objective-space, and compare the AUC of canonical SGP and MOGP solutions.

5.2 Multi-objective GP Approach

In EMO, the evolutionary search is focused on improving the *set* of non-dominated solutions until they are Pareto-optimal [42]. To achieve this in MOGP, two major adaptations to canonical SGP are required. The first is to modify the fitness function to use Pareto Dominance to rank the solutions in the population

on the learning objectives. This MOGP approach uses the accuracy on the minority and majority class as the two competing learning objectives. The second is to modify the evolutionary search algorithm to simultaneously evolve a set (or Pareto front) of genetic program solutions along the learning objectives. These two adaptations are discussed below.

5.2.1 MOGP Fitness

An important aspect in EMO is the notion of Pareto dominance in fitness [188][53]. This allows the solutions to be ranked according to their performance on all the objectives relative to all other solutions in the population. This ranking is important as it affects the way selection is performed if the objectives are to be treated separately in the evolution. In this MOGP approach, the two objectives are the classification accuracy of the minority and majority class. The minority class accuracy is the number of minority class inputs that are correctly predicted over the total number of minority class inputs (in the training set), and likewise for the majority class inputs (i.e. majority class accuracy).

Pareto Dominance

In Pareto dominance, a solution will *dominate* another solution if it is at least as good as the other solution on all the objectives and *better* on at least one. As the two objectives in this MOGP approach are to be maximised, "better" means higher. This concept can be expressed using Eq. (5.1), where the symbol $\succ$ represents the dominance relation between two solutions S_i and S_j, and $(S_i)_m$ denotes the performance of solution S_i on the mth objective.

$$S_i \succ S_j \longleftrightarrow \forall m[(S_i)_m \geq (S_j)_m] \wedge \exists k[(S_i)_k > (S_j)_k] \tag{5.1}$$

In the above equation, $S_i \succ S_j$ means than S_i dominates S_j on the objectives. Solutions are *non-dominated* if they are not dominated by any solution in the population.

Two Pareto-based Dominance Measures: NSGAII and SPEA2

Two well-established Pareto dominance measures are the dominance rank [53] and dominance count [188] of a given solution. Dominance rank is the number of other solutions in the population that dominate a given solution (lower is better). Dominance count is the number of other solutions that a particular solution

dominates (higher is better). Each measure has a different bias towards solutions on the Pareto frontier: dominance rank is known to reward exploration at the edges of the frontier while dominance count tends to reward exploitation in the middle of frontier [53]. Two well-established EMO algorithms which use these measures include SPEA2 [188] and NSGAII [53]. SPEA2 uses both dominance rank and dominance count, while NSGAII uses only dominance rank.

In NSGAII, the fitness value for the solution S_i is its dominance rank, as shown in Eq. (5.2). This is the number of other solutions in the population that dominate S_i. A non-dominated solution will have the best fitness of 0, while high fitness values indicate poor-performing solutions (i.e., solutions that are dominated by many individuals). Fitness values in NSGAII are therefore to be minimised.

$$NSGAII(S_i) = |\{j | j \in Pop \wedge S_j \succ S_i\}| \tag{5.2}$$

This fitness scheme for NSGAII is illustrated in Figure 5.1(a) for several solutions where each point represents the performance of a solution on the two objectives. Assuming that both objectives are to be maximised, the non-filled points show non-dominated solutions (fitness values of 0) while the filled points show the dominated solutions.

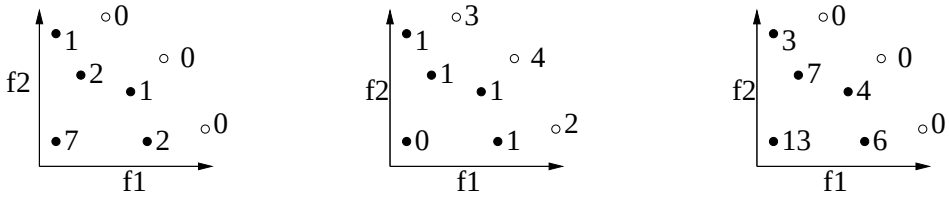
In SPEA2, both dominance rank and dominance count are used in fitness. Firstly, each solution in the population is assigned a *strength* value D. This is the dominance count for solution S_i, i.e., the number of other solutions it dominates in the population, as shown below.

$$D(S_i) = |\{j | j \in Pop \wedge S_i \succ S_j\}|$$

Then, the fitness value for a given solution is determined by the sum of the strengths of all its dominators, shown by Eq. (5.3). In other words, the sum of all dominance counts (strengths) of other solutions in the population that are dominated by S_i. Similar to NSGAII, fitness here is to be minimised where non-dominated solutions have the best fitness of 0.

$$SPEA2(S_i) = \sum_{j \in Pop, S_i \succ S_j} D(S_j) \tag{5.3}$$

This scheme is also illustrated in Figure 5.1(b) and 5.1(c). Figure 5.1(b) shows the strength values $D(S_i)$ for a set of solutions on the two objectives, and Figure 5.1(b) shows the final SPEA2 fitness values. Immediately noticeable in Figure 5.1(c) is that all the dominated solutions for SPEA2 have unique fitness values, whereas some of the dominated solutions in NSGAII have identical fitness values.



(a) NSGAII Pareto Ranking (b) SPEA2 *Strength* Values (c) SPEA2 Pareto Ranking

Figure 5.1: Pareto-based fitness values for NSGAII and SPEA2 where filled points are dominated solutions and non-filled points are non-dominated solutions.

This chapter compares NSGAII and SPEA2 in MOGP as these represent two well-known algorithms from the literature which use the two main Pareto-based dominance measures in fitness (dominance rank and dominance count) in different ways to evolve Pareto fronts.

Secondary Fitness Measure: "Crowding" Distance

In addition to Pareto dominance, both NSGAII and SPEA2 algorithms use a secondary measure in fitness. This corresponds to the "crowding" distance between solutions in *objective-space*, to promote a good spread of solutions along the trade-off frontier. Crowding is the Manhattan distance between solutions in objective-space, where solutions in sparsely populated regions of objective-space are preferred over solutions in densely populated regions. Crowding is needed in fitness to resolve selection when the primary fitness (Pareto dominance measure) is the same between two or more individuals. This means that if two or more individuals have the same Pareto ranking (e.g. non-dominated solutions), the individual with the better crowding distance is preferred in selection.

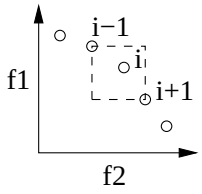


Figure 5.2: The "crowding" distance used in MOGP.

In NSGAII, the crowding distance for a given solution is defined as the average distance to the solution's nearest neighbours along each of the objectives [53], as shown in Figure 5.2. The crowding distance is similar in SPEA2, except that here the distance can be configured to a given solution's k nearest neighbours along each of the objectives [188].

The crowding distance used in MOGP (for both Pareto Dominance measures) is the same as NSGAII (and $k = 1$ in SPEA2), as shown in Figure 5.2. This ensures that both MOGP approaches use the same crowding measure in fitness to make a fair comparison.

5.2.2 MOGP Search Algorithm

In canonical SGP, the single fittest individual in the population is returned as the output of the evolutionary search. In MOGP, the set of non-dominated individuals is simultaneously improved over generations, and this front is returned as the output of the evolutionary search. To achieve this, MOGP combines the parent and offspring populations at every generation, and selects the fittest individuals in this merged parent-child population as the parent population for the next generation (called the archive population). The selection process sorts the individuals in the merged parent-child population based on their primary fitness values (Pareto Dominance ranking according to Eq. 5.2 or Eq. 5.3), and uses the secondary fitness ("crowding" distance) to establish an ordering of individuals with equivalent Pareto Dominance ranks. For example, all non-dominated solutions will have the same Dominance rank of 0. By selecting the fittest individuals in both parent and offspring populations at every generation, elitism is preserved in the population. This ensures that non-dominated solutions are not lost over generations.

At every generation, the offspring population is generated using traditional crossover and mutation operators (similar to canonical SGP).

This evolutionary search algorithm in MOGP is the same in both NSGAII and SPEA2, except for two aspects. Firstly, the size of the archive population in SPEA2 can be different to size of the child population (at each generation). In MOGP, the archive and child population are the same size (500) for consistency between the two MOGP approaches. Secondly, SPEA2 uses an additional truncation operator when the number of non-dominated individuals in the merged parent-child population exceeds the archive population size in a given generation. For example, given a population size P, the size of the merged parent-child population will be $2P$ but only P individuals must be selected to represents the parent population in the next generation. In this case, the truncation operator in SPEA2 iteratively removes non-dominated individuals that are very close to one other in objective-space; one individual is removed per iteration. In other words, very close neighbours are removed until the number of non-dominated

individuals in the merged parent-child population is the same size as the target population. This additional truncation operator is ignored in MOGP as the "crowding" measure in fitness can achieve a similar effect in the selection process.

5.3 Performance of Evolved Pareto Fronts in MOGP

This section presents the experiments results for MOGP, focusing on the performance of the evolved Pareto fronts. This section has five main parts. The first part outlines the MOGP evolutionary parameters. The second part discusses two MOGP evaluation techniques used in the experimental results. The three remaining parts present the MOGP experimental results, each part relating to a different aspect of the performance of the Pareto fronts in MOGP. All three parts address the first goal of this chapter and investigate which of the two Pareto Dominance-based measures in MOGP finds better-performing Pareto fronts on the tasks.

5.3.1 MOGP Setup and Evolutionary Parameters

The same GP framework from the previous two chapters is used to represent the genetic program solutions. This includes the same tree-based representation, function and terminal sets, and classification strategy for the genetic programs.

Where possible, the evolutionary parameters in MOGP are kept the same as the SGP approaches (from the previous two chapters) for consistency and for a fair comparison between the MOGP and SGP approaches. The ramped half-and-half method is also used in MOGP for generating programs in the initial population and for the mutation operator [104]. Likewise, in both MOGP and SGP approaches, the population size is 500, maximum program depth is 8 (to restrict very large programs in the population), and the evolution is allowed to run for a maximum of 50 generations or until a solution with optimal fitness is found. In MOGP, a solution with optimal fitness has 100% accuracy on both objectives (minority and majority class accuracy).

Only two evolutionary parameter settings are different in MOGP. Firstly, NSGAII [53] and SPEA2 [188] do not use the elitism genetic operator. As discussed above, elitism is replaced by the non-dominated sorting procedure applied to the union of the parent and child populations at each generation. As a result, crossover and mutation rates in MOGP are 60% and 40% to balance exploitation and exploration in a similar manner as in SGP (where crossover,

mutation and elitism rates are 60%, 35% and 5%, respectively).

Secondly, NSGAII [53] and SPEA2 [188] both use tournament selection with a tournament size of 2; whereas SGP uses a tournament size of 7 (as recommended in the GP literature). Both NSGAII [53] and SPEA2 [188] recommend using binary tournament selection in conjunction with the non-dominated sorting procedure. For this reason, MOGP also uses binary tournament selection.

5.3.2 Evaluating the Performance of the MOGP Fronts

Two evaluation techniques are used to summarise different aspects of the performance of the evolved Pareto fronts over the series of 50 MOGP runs. These include the *hyperarea* and *attainment summary surfaces* of the MOGP Pareto fronts. A brief description of these two evaluation techniques is given below.

Pareto Front Hyperarea

The *hyperarea* (also known as the hypervolume) is the area under the Pareto-approximated front in *objective-space* [42], similar to the area under the ROC curve (or AUC). However, while the AUC represents the performance of a single classifier at varying classification thresholds, the hyperarea represents the classification performance of the *set* of classifiers along the front. The hyperarea is a useful "single figure" measure to represent the area of objective-space correctly classified by the front, i.e., the convergence of a front.

The hyperarea is calculated in a similar way as the AUC, i.e., by taking the sum of the areas of individual trapezoids fitted under each front solution in objective-space [42]. Hyperarea values range between 0 and 1 where the higher the value, the better the performance of the front.

Summary Attainment Surfaces

Attainment summary surfaces are used to approximate the "average" performance of an evolved Pareto-approximated front for a particular MOGP approach over 50 runs on a task. Recall (from Chapter 2) that attainment summary surfaces are a useful technique in EMO to summarise the outcome of a series of MOGP experiments where a potentially different set of non-dominated solutions can be returned from each MOGP run [102]. Each attainment surface comprises of evolved Pareto front solutions (from all runs) that have identical *attainment* values, where the number of attainment surfaces corresponds to the number

Table 5.1: Average (± standard deviation) hyperarea of evolved Pareto-approximated fronts, Pareto optimal (PO) front, and training times (seconds 's' or minutes 'm') for the MOGP approaches over 50 runs. The (statistically) significantly better average hyperarea is highlighted in bold, and the higher PO front hyperarea is underlined.

Task	NSGAII Fitness			SPEA2 Fitness		
	Hyperarea		Training	Hyperarea		Training
	Average	PO Front	Time	Average	PO Front	Time
Ion	0.793 ± 0.041	0.952	$8.3s \pm 1.3$	$\mathbf{0.848 \pm 0.041}$	<u>0.992</u>	$9.3s \pm 2.4$
Spt	0.733 ± 0.026	0.938	$16.9s \pm 2.1$	0.732 ± 0.032	<u>0.971</u>	$9.7s \pm 2.5$
Ped	0.881 ± 0.013	0.903	$3.5m \pm 52.6$	$\mathbf{0.902 \pm 0.019}$	<u>0.922</u>	$3.9m \pm 1.1$
Yst_1	0.793 ± 0.008	0.917	$23.5s \pm 4.5$	0.793 ± 0.009	<u>0.931</u>	$20.8s \pm 7.1$
Yst_2	0.942 ± 0.008	0.986	$23.5s \pm 4.4$	$\mathbf{0.949 \pm 0.011}$	<u>0.991</u>	$20.1s \pm 8.1$
Bal	0.749 ± 0.049	<u>0.993</u>	$20.1s \pm 2.6$	0.757 ± 0.063	0.985	$15.2s \pm 3.9$

of total MOGP runs (e.g. 50 MOGP runs produce 50 attainment surfaces). A solution's attainment value is the probability that the MOGP system will produce (or evolve) another solution which performs better than or equal to the given solution on all objectives (*weakly dominates*)[102]. The *median* attainment surface, i.e., the set of solutions with attainment values of 0.5, corresponds to those solutions with a 50% probability of attainment with respect to all runs. This set represents the "average" performance of an evolved front over 50 MOGP runs.

5.3.3 MOGP Hyperarea

Table 5.1 reports the average (and standard deviation) hyperarea of the evolved Pareto-approximated fronts on the *test* set, as well as the average training times in second (s) or minutes (m), for the two MOGP approaches over 50 runs. Table 5.1 also includes the hyperarea of the *Pareto-optimal* (PO) front with respect to all MOGP runs. The PO front is the set of non-dominated solutions from the union of all Pareto-approximated fronts evolved over the 50 runs. The PO front with the higher hyperarea is underlined in Table 5.1.

In Table 5.1, the MOGP approach with the significantly better hyperarea is highlighted in bold for each task. The statistical significance test (of the average hyperarea) is calculated using the common random numbers technique at a 95% level of significance. This technique computes the 95% confidence interval of the hyperarea differences between the two MOGP approaches, on a run-by-run basis over 50 independent runs.

According to Table 5.1, SPEA2's average is hyperarea statistically better than NSGAII on the three tasks, and not statistically different to NSGAII on the remaining three tasks. The hyperarea of the Pareto-optimal (PO) front is also better in SPEA2 for all tasks except Bal (where NSGAII is better). Table 5.1 also shows that the two MOGP approaches show similar average training times. This suggests that the MOGP approach using SPEA2 evolves Pareto fronts are as good as NSGAII on half the tasks, and better than NSGAII, on the other tasks.

Further Analysis

The hyperarea in MOGP and the area under the ROC curve (or AUC) in canonical SGP represents the same learnt *concept* in the objective-space, i.e., both capture the different trade-offs between the minority class accuracy and the majority class accuracy. The hyperarea and AUC are also calculated in the same way, by fitting trapezoids under the different trade-off points in the objective-space and summing the areas of the individual trapezoids. Therefore, the hyperarea in MOGP (using NSGAII and SPEA2) and the AUC in SGP (using the fitness function from the previous chapter) can be *roughly* compared to provide an overall *indication* of how these methods perform on the tasks. However, strictly speaking, this is not a direct comparison between these two methods due to the fundamental differences between SGP and MOGP. In canonical SGP, the AUC represents the performance of a *single* classifier (at different decision thresholds). In MOGP, the hyperarea represents the classification performance of a *set* of classifiers (Pareto front).

According to Table 5.1, the average hyperarea for MOGP with SPEA2 is similar to the average AUC using a "good" fitness function in SGP (i.e. with a high s-rank) in Table 4.4 (on page 94 in the previous chapter), on those tasks where SPEA2's average hyperarea is significantly better than NSGAII (Ion, Ped and Yst_2). In contrast, the average hyperarea for MOGP with NSGAII on these three tasks is similar to the average AUC using a "mid-level" fitness function (i.e. with mid-level s-rank values in Table 4.4). In the remaining three tasks, the average hyperarea for MOGP with both SPEA2 and NSGAII are similar to the average AUC using a "mid-level" fitness function in SGP.

Interestingly, the hyperarea of the PO front (over 50 runs) for MOGP with SPEA2 is better than the *best* AUC achieved by any SGP fitness function (over 50 runs) in nearly all tasks (expect Ped). The hyperarea of the PO front for MOGP with NSGAII only accomplishes this in four tasks (Ion and Ped are the two exceptions). In Ped, the PO front hyperarea for SPEA2 is similar to the best

AUC achieved by a "good" SGP fitness function; while the PO front hyperarea for NSGAII is *worse* the best AUC achieved by any SGP fitness function

This suggests that in a typical run, MOGP with SPEA2 produces similar trade-offs between the two objectives to SGP with a "good" fitness function in half the tasks, and SGP with a "mid-level" fitness function in the remaining tasks. However, the "best" set of trade-off solutions found by SPEA2 over all MOGP runs (PO front) are generally better than the "best" ROC curve achieved by any single SGP classifier over all runs. In contrast, a typical run of MOGP with NSGAII is only as good as SGP with a "mid-level" fitness function on the tasks.

5.3.4 MOGP and Canonical SGP

This section examines the *median attainment summary surface* of the MOGP approaches on the tasks. This allows us to investigate why the MOGP approach using SPEA2 outperforms NSGAII on half the tasks, and compare the performances of the evolved Pareto fronts to canonical SGP. The median attainment surface for a given MOGP approach represents an approximation of the "average" performance of an evolved Pareto front over 50 runs on a given task.

Figure 5.3 shows the median attainment surface and the Pareto-optimal front for the two MOGP approaches (on the *test* set) over 50 runs for the tasks. Figure 5.3 also includes the frontier produced by canonical SGP on the tasks. This corresponds to the performance of the fittest evolved SGP solutions on the two classes (also on the test set), using the weighted-average of these two objectives in the fitness function (SGP with $Wave$ from the previous chapter). To produce the SGP frontier, seven different weighting coefficients between 0.2 and 0.8 (at intervals of 0.1) are used in the fitness function (in $Wave$). In SGP, each run using a given weighting coefficient in the fitness function is repeated 50 times, and the average performances are shown in Figure 5.3. This means that for each task, the SGP frontier is generated using 350 different runs (50 runs $\times$ seven weighting coefficients); while the MOGP frontier is generated in 50 runs.

Note that the axis scopes in Figure 5.3 are different for the tasks.

Figure 5.3 shows that the evolved Pareto fronts, particularly the MOGP approach with SPEA2, contain an accurate set of solutions along the minority and majority class trade-off frontier for these tasks. This highlights an important advantage of MOGP over canonical SGP: a *single* run of the MOGP algorithm can trace out good trade-off solutions leaving the final choice for the decision-maker. In contrast, in SGP this trade-off must be determined *a priori* needing multiple

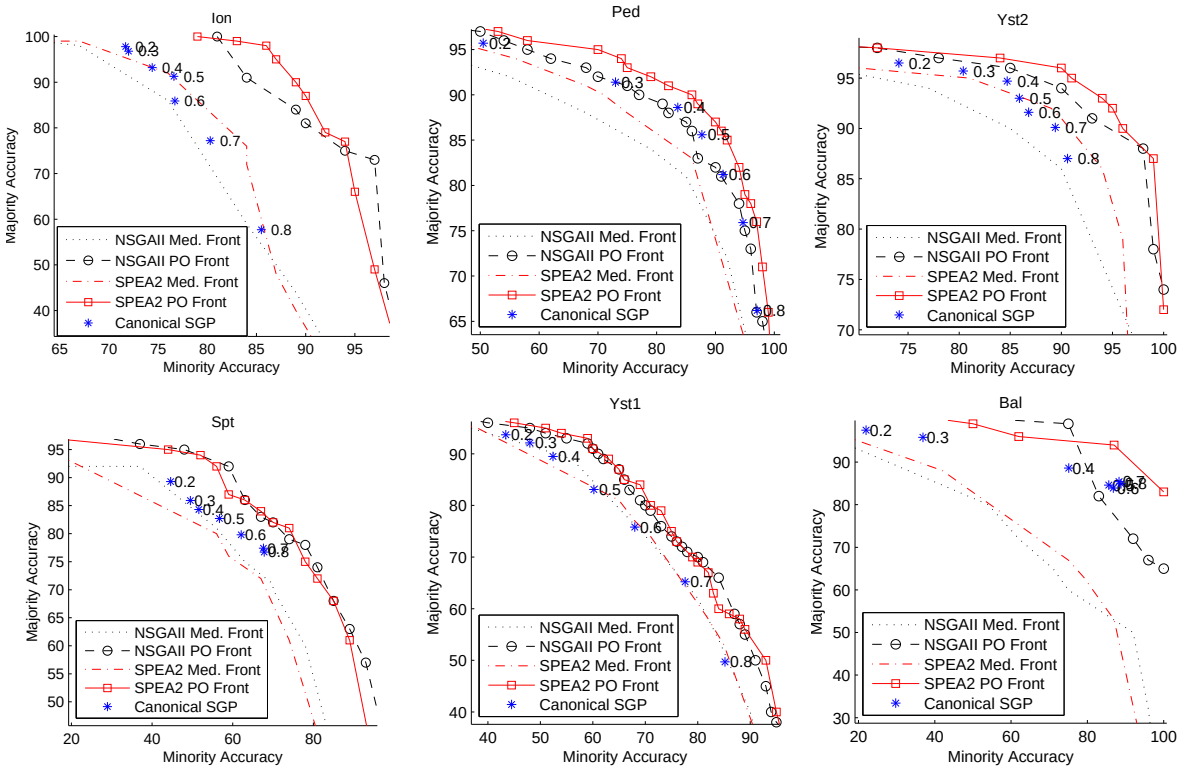


Figure 5.3: Classification performance of evolved solutions using two MOGP approaches (NSGAII and SPEA2), and canonical SGP using fitness functions $Wave$. In Ion, Ped and Yst_2 (top row), the average hyperarea for SPEA2 is statistically better than NSGAII. There is no significant difference in hyperarea for the remaining tasks (bottom row).

SGP runs (one for each weighting coefficient) to generate a frontier.

Although a single run of the SGP method uses less time than the MOGP approaches, the SGP method requires a much longer time to get a reasonable Pareto front. The SGP training times (with a particular weighting coefficient) are discussed in the previous chapter. To recap, this is approximately 2.6-2.8 seconds for Ion and Spt, 4.7 seconds for Bal, 12-13 seconds for the Yst tasks, and 5.0 minutes for Ped (on average over 50 GP runs).

A closer inspection of Figure 5.3 (particularly the median attainment surfaces for SPEA2 and NSGAII) shows that in some tasks (such as Ion, Ped and Yst_2), SPEA2's average front lies along the SGP frontier. In contrast, the average front for NSGAII lies below the SGP frontier in these tasks. The explains why the average hyperarea for SPEA2 (from Table 5.1) is statistically better than NSGAII in these three tasks (Ion, Ped and Yst_2). Similarly, the PO front for SPEA2 also clearly dominates the PO front for NSGAII in two of these tasks (Ped and Yst_2).

This suggests that on these tasks, MOGP with SPEA2 can evolve frontier solutions in a single run that perform better than, or at least as well as, multiple runs of canonical SGP. However, MOGP with NSGAII cannot achieve this to a sufficient level of accuracy, as the solutions along the SGP frontier clearly dominate the NSGAII median attainment front.

As hypothesised, the likely reason for this difference in behaviour is the inherent bias between the two fitness schemes. The SPEA2 algorithm [188] tends to evolve more solutions in the middle region of the frontier, pushing this front outwards toward the *zenith* point (100% accuracy on both objectives). In contrast, the NSGAII algorithm [53] tends to evolve a better "spread" of solutions along the whole of the frontier. For these classification tasks, edge-region solutions are less desirable than middle-region solutions, as these represent biased classifiers (i.e. classifiers with high accuracy rates on one class alone).

The next section explores the differences in the *overall* behaviour of the evolved Pareto fronts over all 50 runs for the two MOGP approaches.

5.3.5 Overall Pareto Front Behaviour

Figure 5.4 shows the performances (on the test set) of *all* Pareto front solutions over 50 runs for the two MOGP approaches (for three tasks). In these figures, each point (or circle) represents the performance of a Pareto front solution in *objective-space*, where the size of each circle is proportional to the *frequency* that a solution with the same performance on the two objectives is found in each of the 50 runs. In other words, the number of times a solution with the same performance is evolved in each of the 50 runs. The larger the circle, the larger the number of Pareto front solutions that are evolved with the same performance on the two objectives over 50 runs. Conversely, smaller circles mean that fewer solutions are evolved over 50 runs which have the same performance on the two objectives. For the purposes of this analysis, two (or more) non-dominated solutions with the same performance on the two objectives in the *same* run are counted only once for that particular run. In Figure 5.4, the vertical and horizontal axis correspond to the majority and minority class accuracies, respectively, and the axis scopes are different for the tasks.

The three tasks shown in Figure 5.4, Ion, Ped and Yst_2, correspond to those tasks where the average hyperarea for SPEA2 is significantly better than NSGAII (from Table 5.1), and the median attainment surface for SPEA2 clearly dominates NSGAII (from Figure 5.3). In these figures, the NSGAII fronts are shown in the

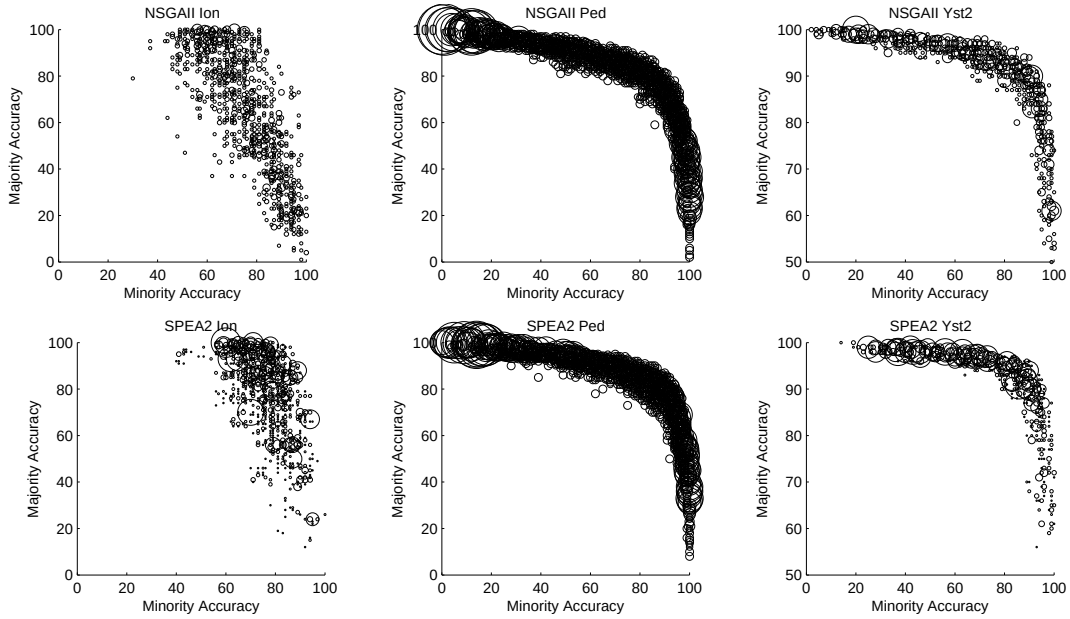
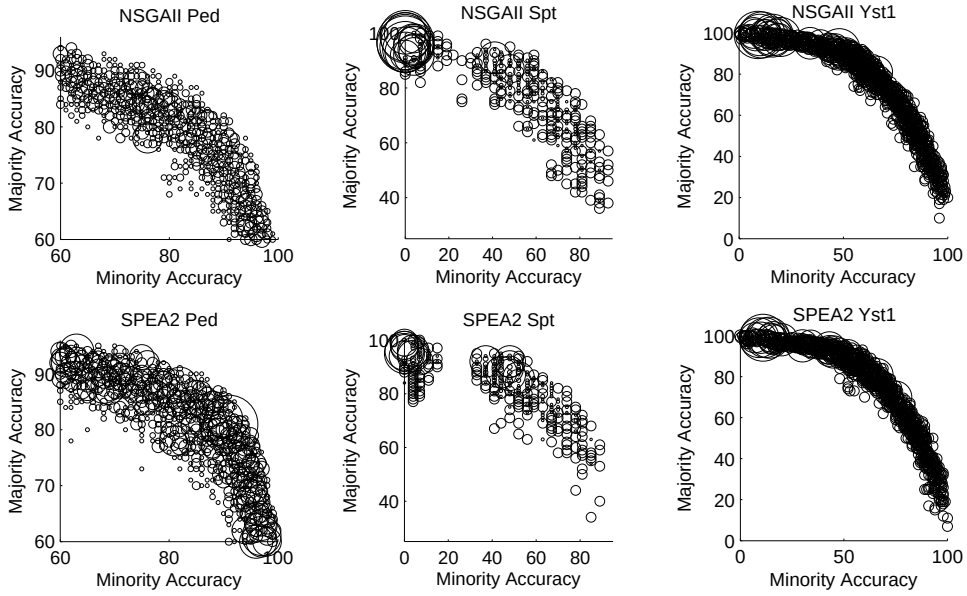


Figure 5.4: Accuracy of all Pareto front solutions evolved over 50 runs for the MOGP approaches on three tasks (Ion, Ped and Yst_2). Circle size is proportional to frequency.

top row and the SPEA2 fronts in the bottom row.

Figure 5.4 shows that over all 50 MOGP runs, SPEA2 finds more solutions in the middle region of the frontier; while NSGAII fitness finds a better "spread" of solutions along the whole of the frontier. In Ion and Yst_2, there are a greater number of large-sized circles in the middle region of the frontier for SPEA2 (compared to NSGAII); while the NSGAII fronts are more "spread out" along the objectives than SPEA2. In Ped, Figure 5.4 shows that SPEA2 has a slight "bulge" in the middle region of the frontier compared to NSGAII. This bulge can be more clearly seen in Figure 5.5(a) for Ped.

In the remaining three tasks (Spt, Yst_1 and Bal), the overall performances of the Pareto front solutions are relatively similar for both MOGP approaches, as seen in Figure 5.5(b) for Spt and Yst_1. This is because in these three tasks, both MOGP approaches show very similar hyperarea values and median attainment fronts. Note that Bal is omitted in Figure 5.5(b) for space constraints and because the overall Pareto front behaviour for both MOGP approaches in Bal is almost identical.



(a) (b)

Figure 5.5: Accuracy of all Pareto front solutions evolved over 50 runs (Ped task) where circle size is proportional to frequency.

Biased Pareto front Solutions

Figure 5.4 (for Ped) and Figure 5.5(b) for the Spt and Yst_1 also show that the *density* of solutions is highest in the region of objective-space where majority class accuracy is very high (e.g. 100%) and minority class accuracy is very low (e.g. 0%). This is represented by the very large circles in the top-left corner of the figures for these three tasks, seen in both MOGP approaches. This means that in nearly all 50 MOGP runs for these tasks, each evolved Pareto front contains some solutions that are highly *biased* toward the majority class, and with very poor minority class accuracy.

The presence of these highly biased solutions on the Pareto fronts is not a major concern in these three tasks, as Figure 5.5 shows that other good trade-off solutions are still found (e.g. in the middle region of the front). This is clearly visible in each task. Rather, these solutions simply reflect the notion that a biased non-dominated performance on the two objectives is relatively easy to achieve in MOGP. For example, if a trivial genetic program solution is evolved that classifies all the input instances as belonging to the majority class, then this solution will achieve 100% majority class accuracy and 0% minority class accuracy. This solution will be non-dominated on the two objectives unless another solution is evolved which also achieves 100% accuracy on the majority class, but has a greater minority class accuracy than 0% (to dominate the given

solution). Until such a solution is evolved, the trivial (biased) solution will remain as non-dominated in the population.

This effect is not seen in other tasks (such as Ion or Yst_2) as solutions with better performances on one or both of the objectives are evolved to replace these highly biased solutions on the Pareto fronts. For example, in Ion in Figure 5.4, solutions with very high majority class accuracy (e.g. nearly 100%) also have at least 40% accuracy on the minority class.

5.4 AUC Analysis of the Pareto front in MOGP

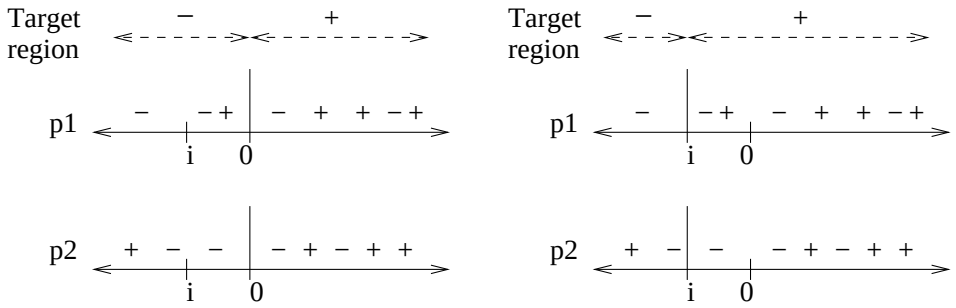
This section presents the AUC results of the MOGP approaches to address the second goal of this chapter, to investigate the AUC of the evolved Pareto front solutions in MOGP and compare this to canonical SGP solutions.

5.4.1 Pareto front Solutions with Different Models

In order to focus on the overall patterns in the evolved Pareto fronts over all 50 runs, Figures 5.4 and 5.5 (in the previous section) ignore those Pareto front solutions that produce the *same* performance on the two objectives in a given MOGP run. However, even though two Pareto front solutions may have the same performances on the two objectives in a given MOGP run, it does not necessarily mean that these two solutions will have the same internal models or structures. Solutions with different internal models/structures will not produce the same set of output values when evaluated on all the input instances. The AUC of a given Pareto front solution can determine whether two or more solutions have different internal models, as the AUC calculation examines every output value of a solution to determine how well the solution separates these output values across different classification thresholds.

For example, Figure 5.6(a) shows the output values of two solutions, p_1 and p_2, when evaluated on eight input instances from two classes. The output values are denoted by $+$ and $-$ for the positive and negative class, respectively. Using zero as the boundary between the two classes, both solutions have the same number of correct predictions for each class, as shown in Figure 5.6(a). This corresponds to a true positive (TP) rate of 0.75 (as $\frac{3}{4}$ positive class inputs are correctly classified), and a true negative (TN) rate of 0.5 (as $\frac{2}{4}$ negative class inputs are correctly classified).

However, these two solutions clearly have different internal models as their



(a) Class threshold is 0 (b) Class threshold is i

Figure 5.6: Output values denoted by $+$ and $-$ for the positive and negative class, respectively, for two solutions (p_1 and p_2). In (a), p_1 and p_2 have the same accuracy on the two classes relative to zero as the class threshold; while in (b), p_1 and p_2 have different accuracy rates on the two classes relative to class threshold i.

output values are different. Therefore, when the output values are evaluated relative to another class threshold, i in Figure 5.6(b), these solutions have a different number of correct predictions for each class. For p_1, the TP rate is 1 and the TN rate is 0.25. For p_2, the TP rate is 0.75 and the TN rate is 0.25. These corresponds to different ROC points for solution p_1 and p_2 and thus, their AUC will also be different.

This section compares the number of Pareto front solutions with different internal classification models (i.e. different AUC) for the two MOGP approaches (for a given run). Table 5.2 (left-hand column) shows the average number of distinct points in *objective-space* (on the test set) produced by the MOGP approaches over 50 runs. This corresponds to the average number of solutions on a given Pareto front which produce different performances on the two objectives (over 50 runs). These figures reflect the average size of an evolved Pareto front *not including* those solutions that produce the *same* performances on the two objectives (in a given run).

The right-hand column in Table 5.2 shows the average sizes of the *full* evolved Pareto fronts over 50 runs for the MOGP approaches. These correspond to the number of all Pareto front solutions evolved in a given run including those which produce the *same* performances on the two objectives, i.e., Pareto front solutions with different AUC values on the test set.

As expected, Table 5.2 shows that the number of solutions with different AUC values (right-hand column) is higher than than the number distinct points in objective-space (left-hand column) for both MOGP approaches in all tasks. This confirms that some Pareto front solutions have different internal

Table 5.2: The average number of Pareto front solutions that produce distinct points in *objective-space* (test set), and the number of Pareto front solutions with different internal models (different AUC) over 50 runs for the MOGP approaches.

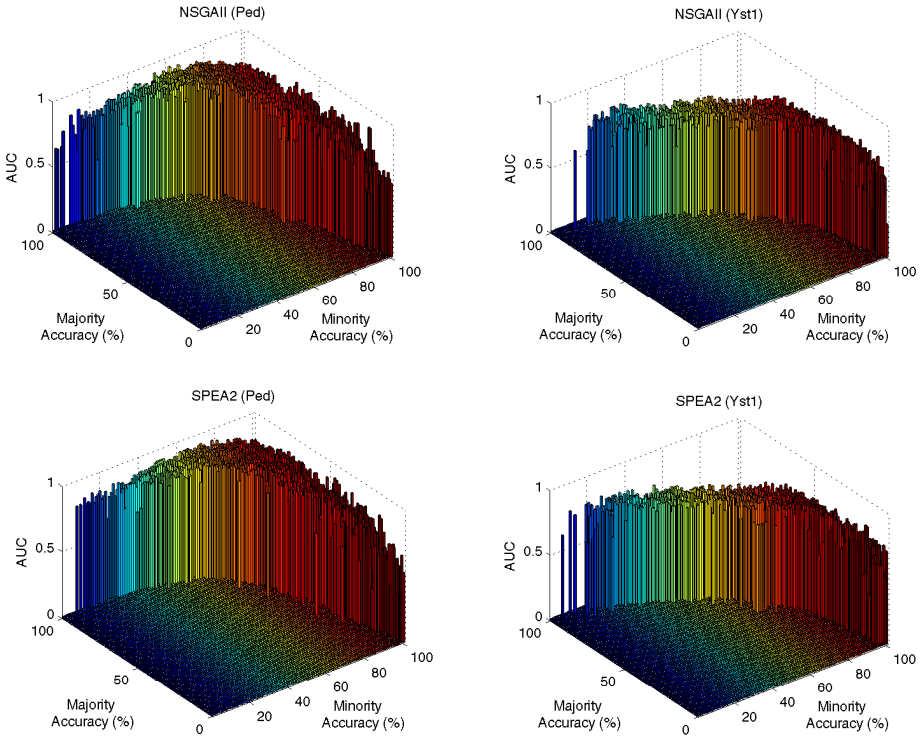
Task	Unique Points in *Objective-space*		Total Pareto Front Size	
	NSGAII	SPEA2	NSGAII	SPEA2
Ion	20.08	22.80	21.22	46.54
Spt	19.36	19.22	39.14	91.60
Ped	112.96	134.74	117.30	162.80
Yst1	63.08	58.20	86.74	105.64
Yst2	21.06	29.06	26.66	87.08
Bal	18.02	14.76	29.70	62.28

models/structures even though they produce the same performance on the two objectives for both MOGP approaches.

Interestingly, the number of solutions with different AUC values is always higher in SPEA2 than NSGAII in all tasks, even though both MOGP approaches use the same population size (of 500). This shows that SPEA2 contains more non-dominated solutions (on average) with different internal models/structures in the final generation than NSGAII in all tasks.

In fact, Table 5.2 shows that the average number of distinct points in objective-space (left-hand column) is only larger in NSGAII (than in SPEA2) in exactly three tasks (Spt, Yst_1 and Bal). These correspond to the same three tasks where the average hyperarea for NSGAII and SPEA2 is not statistically different to one another (Table 5.1), and where both systems show similar median attainment surfaces (Figure 5.3). Intuitively, it follows that the MOGP approach with more distinct points in objective-space will also produce a better hyperarea, as more of the objective-space is covered by these solutions. However, in these three tasks, NSGAII only performs as well as SPEA2 (in terms of their hyperarea and median attainment surfaces) but *not better* than SPEA2. This may be because NSGAII still contains *fewer* non-dominated solutions with different internal models/structures than SPEA2, as shown in Table 5.2 (right-hand column).

It should be mentioned that Pareto fronts sizes are noticeably larger in Ped (compared to the other tasks) for both MOGP approaches. This is expected due to the very large number of training examples in Ped (more than 12000 examples), where slight variations in the objective performances by the evolved solutions produce more non-dominated points in the objective-space.



(a) Ped (b) Yst_1

Figure 5.7: AUC of all Pareto front solutions evolved over 50 MOGP runs for NSGAII (top) and SPEA2 (bottom) on two tasks (Ped and Yst_1). Each vertical bar represents a Pareto front solution (on the two objectives) and the heights of the vertical bars represent the AUC.

5.4.2 Pareto front AUC in Regions of Objective-Space

Table 5.2 shows that SPEA2 contains more non-dominated solutions with different internal models/structures than NSGAII in all tasks, and that this may be a factor in why SPEA2 performs better that NSGAII on some tasks (in terms of their hyperarea and median attainment surfaces). Another contributing factor may be the *quality* of solutions evolved by two MOGP approaches in terms of their AUC. Examining the AUC of a given Pareto front solution provides an indication of the quality of the classification model in the solution where the higher the AUC, the better the quality of the classification model. This section compares the AUC of the Pareto front solutions for the two MOGP approaches (NSGAII and SPEA2). However, as a given Pareto front can contain many different genetic program classifiers, each with their own AUC, careful consideration must be taken when comparing which MOGP approach evolves solutions with better AUC on the tasks in a fair and meaningful way.

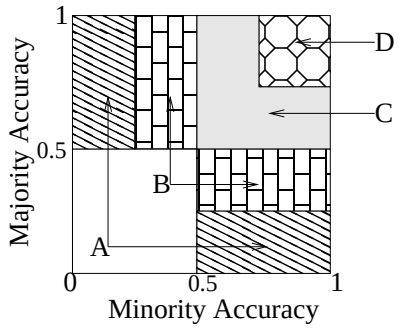


Figure 5.8: The regions of objective-space.

To illustrate this point, consider Figure 5.7 which shows the AUC (on the test set) of *all* evolved Pareto front solutions over 50 runs for NSGAII and SPEA2 (on the Ped and Yst_1 tasks). In these figures, each vertical bar represents the performance of a Pareto front solution (on the two objectives), where the height of the vertical bar represents the AUC of the corresponding solution. If two or more solutions have the same performance on the two objectives and their AUC is different, the average AUC is shown in Figure 5.7. The top row in Figure 5.7 shows the NSGAII fronts, while the bottom row shows the SPEA2 fronts.

Figure 5.7(a) shows that for Ped task, the vertical bars are longer in the middle-region of the frontier than the end-region for both NSGAII and SPEA2. This means that the AUC is higher in the middle-region solutions (i.e. solutions with high accuracy on both classes) than in the end-regions solutions (i.e. biased solutions). This is expected as not all the frontier solutions represent solutions with good classification ability, particularly not the highly-biased solutions on the end-regions of the frontier. However, this observation is not very clear in Figure 5.7(b) for the Yst_1 task. Here the heights of the vertical bars (AUC of the Pareto front solutions) for both NSGAII and SPEA2 are very similar along the entire region of the frontier. This means that it is also not very clear from Figure 5.7(b) whether the AUC in the different regions of the frontier is better or worse in NSGAII or SPEA2.

To try to quantify this AUC analysis of the Pareto front solutions evolved using two MOGP approaches, the objective-space is divided into four regions, as shown by Figure 5.8, and only the AUC of the solutions in a particular region is compared to one another. The first two regions, A and B in Figure 5.8, correspond to biased solutions on the two objectives. Region A represents highly-biased solutions which have less than 25% accuracy on either class; while region B represents solutions with between 25-50% accuracy on either class. The other two

Table 5.3: Average AUC ($\pm$ standard deviation) of the Pareto front solutions in four regions of objective-space (from Figure 5.8), and the percentage of solutions in a given region (over all Pareto front of solutions from 50 runs) for NSGAII and SPEA2 on the tasks. Significantly better AUC between NSGAII and SPEA2 is highlighted in bold.

Task	Region		NSGAII		SPEA2	
		Obj. Perf.	AUC	% Solutions	AUC	% Solutions
Ion	A	< 25	0.56 ± 0.06	8.58%	0.56 ± 0.02	2.06%
	B	25–49	0.59 ± 0.09	21.96%	0.59 ± 0.08	5.80%
	C	50–74	0.72 ± 0.10	53.91%	$\mathbf{0.75 \pm 0.08}$	41.43%
	D	≥ 75	0.82 ± 0.06	15.55%	$\mathbf{0.84 \pm 0.06}$	50.71%
Spt	A	< 25	0.65 ± 0.09	55.75%	$\mathbf{0.68 \pm 0.09}$	47.27%
	B	25–49	0.72 ± 0.07	19.83%	$\mathbf{0.74 \pm 0.06}$	33.41%
	C	50–74	0.72 ± 0.06	21.87%	$\mathbf{0.74 \pm 0.06}$	16.99%
	D	≥ 75	0.77 ± 0.04	2.55%	0.78 ± 0.03	2.34%
Ped	A	< 25	0.65 ± 0.13	22.51%	0.65 ± 0.12	24.39%
	B	25–49	0.78 ± 0.10	26.00%	0.79 ± 0.10	23.045%
	C	50–74	0.84 ± 0.06	33.61%	0.85 ± 0.07	29.86%
	D	≥ 75	0.86 ± 0.04	17.89%	$\mathbf{0.89 \pm 0.05}$	22.30%
Yst1	A	< 25	0.68 ± 0.10	24.44%	$\mathbf{0.70 \pm 0.10}$	29.17%
	B	25–49	0.72 ± 0.08	22.98%	$\mathbf{0.74 \pm 0.08}$	31.14%
	C	50–74	0.76 ± 0.04	43.16%	0.76 ± 0.04	36.48%
	D	≥ 75	0.76 ± 0.03	3.41%	$\mathbf{0.80 \pm 0.04}$	3.20%
Yst2	A	< 25	0.79 ± 0.14	13.58%	$\mathbf{0.81 \pm 0.07}$	1.01%
	B	25–49	0.83 ± 0.13	10.73%	0.84 ± 0.12	25.40%
	C	50–74	0.88 ± 0.09	32.11%	0.88 ± 0.08	36.70%
	D	≥ 75	0.90 ± 0.06	43.59%	$\mathbf{0.92 \pm 0.04}$	36.89%
Bal	A	< 25	0.56 ± 0.06	47.35%	0.55 ± 0.04	57.77%
	B	25–49	0.63 ± 0.07	20.67%	$\mathbf{0.69 \pm 0.06}$	24.05%
	C	50–74	0.69 ± 0.07	29.49%	0.69 ± 0.08	16.41%
	D	≥ 75	0.74 ± 0.07	2.49%	$\mathbf{0.78 \pm 0.07}$	1.77%

Table 5.4: Average AUC ($\pm$ standard deviation) of SGP solutions using fitness functions Auc_F, $Corr$, $Dist$, $Amse$ and Ave on the tasks (over 50 runs). These results are repeated from Table 4.4 (in Chapter 4). Fitness functions are ordered according to their average AUC for each task (descending order).

Task	Function	AUC	Task	Function	AUC	Task	Function	AUC
	Corr	0.87 ± 0.04		AucF	0.77 ± 0.04		AucF	0.92 ± 0.01
	Dist	0.86 ± 0.05		Amse	0.75 ± 0.04		Dist	0.90 ± 0.02
Ion	Amse	0.85 ± 0.05	*Spt*	Corr	0.74 ± 0.05	*Ped*	Corr	0.89 ± 0.01
	AucF	0.85 ± 0.04		Dist	0.73 ± 0.05		Amse	0.88 ± 0.02
	Ave	0.80 ± 0.06		Ave	0.71 ± 0.05		Ave	0.87 ± 0.04
	AucF	0.83 ± 0.02		Amse	0.96 ± 0.01		AucF	0.84 ± 0.09
	Dist	0.83 ± 0.03		Corr	0.95 ± 0.02		Amse	0.78 ± 0.10
Yst_1	Amse	0.82 ± 0.02	Yst_2	AucF	0.95 ± 0.03	*Bal*	Dist	0.77 ± 0.13
	Corr	0.81 ± 0.02		Dist	0.94 ± 0.03		Corr	0.75 ± 0.11
	Ave	0.79 ± 0.03		Ave	0.93 ± 0.04		Ave	0.71 ± 0.15

regions, C and D in Figure 5.8, correspond to solutions that are relatively accurate on the objectives. Region C represents solutions with between 50-75% accuracy on either class, while region D represent solutions with at least 75% accuracy on both classes. Note that regions C and D are mutually exclusive.

Table 5.3 shows the average (and standard deviation) AUC on the test set of the Pareto front solutions in these four regions of the objective-space for the two MOGP approaches (NSGAII and SPEA2). This analysis is performed on the union of *all* Pareto front solutions over 50 runs of the two MOGP approaches. Table 5.3 also shows the proportion of Pareto front solutions in each region of the objective-space (relative to the total number of Pareto front evolved over 50 runs) for the two MOGP approaches. The proportion of solutions each region sums to 100% in the tasks.

The *Wilcoxon rank-sum* statistical test of the AUC is used to determine which MOGP approach achieves a statistically significantly better AUC in each region, at a 95% level of significance. The MOGP approach with a statistically significantly better AUC is highlight in bold in Table 5.3. If neither MOGP approach is highlighted in bold in a given region, then the Wilcoxon test shows no statistically significant difference in AUC (i.e. null hypothesis is true). This statistical test is chosen (and not the common random numbers statistical test previously used in in Section 5.3.3 to test hyperarea of the two MOGP approaches), as the Wilcoxon test allows the two samples sizes to have different lengths. In other words, the number of Pareto front solutions in a given region can be different for NSGAII and SPEA2. In contrast, the common random numbers method requires that the two samples sizes are the same length.

Table 5.3 shows that a large variation in AUC performances by the Pareto front solutions can be seen in the different regions. For both MOGP approaches, the AUC values are higher as the objective performances increase in the tasks. This can be seen by the low AUC values for biased Pareto front solutions (e.g. in region A) compared to relatively accurate solutions (e.g. in regions C and D). This affirms that the better the performance of a Pareto front solution on the two objectives, the better the AUC — and that this is true for *all* the tasks. Although this observation was suggested earlier, it was not very clear in Figure 5.7(b). Furthermore, Table 5.3 also shows to what extent this is true for the different tasks. AUC values for solutions in region A are substantially worse than solutions in region D for Ion, Ped and Bal; whereas this difference in AUC between these two regions is not as large in the remaining three tasks for both MOGP approaches.

Interestingly, the AUC for SPEA2 is as least as good as NSGAII, or (statisti-

cally) significantly better than NSGAII, in all four regions and in all tasks in Table 5.3. The AUC for solutions region D is significantly better for SPEA2 in nearly all tasks (except Spt where both MOGP approaches show similar AUC values). This suggests that SPEA2 not only finds more non-dominated solutions in the final generation than NSGAII (as discussed in the previous section in Table 5.2), but that the AUC of the solutions evolved using SPEA2 is as good as, or better than, NSGAII in the important middle-region of the Pareto frontier (where solutions have high AUC) in all tasks. This may be another factor in why SPEA2 evolved better-performing fronts that NSGAII on some tasks.

5.4.3 AUC of MOGP and SGP Solutions

Another advantage to dividing the objective-space into regions is that the AUC of the Pareto front solutions in different regions can readily be compared to the AUC of canonical SGP solutions on the tasks. Without this separation of Pareto front solutions based on their objective performances, a comparison between MOGP and SGP solutions would not be very meaningful as the Pareto front solutions represent varied range of classifiers with different AUC performances. To this end, only Pareto front solutions with relatively good performances on the two objectives (such as those in regions C and D in Figure 5.8) are compared to canonical SGP solutions, to provide a better indication of the kinds of solutions evolved in MOGP and SGP.

This analysis compares the AUC results in Table 5.3 with the AUC results using several different fitness functions in SGP (from the previous chapter). These SGP results are repeated in Table 5.4 for convenience (originally from Table 4.4 in Chapter 4). Table 5.4 shows the average AUC of SGP solutions using fitness functions Auc_F, $Corr$, $Dist$, $Amse$ and Ave on the tasks (over 50 runs). The fitness functions are ordered according to their average AUC for each task (descending order). The motivations for selecting only these five SGP fitness functions in this comparison is made clearer in the discussion below.

When the average of the minority and the majority class accuracy is used in the fitness function in SGP (i.e. SGP with Ave), the AUC of the evolved SGP solutions are better than the Pareto front solutions in region C (i.e. solutions with at least 50% accuracy on the objectives) in nearly all tasks (except Spt). This suggests that the SPEA2 and NSGAII solutions in region C have internal models/structures that are not as good as the SGP solutions with Ave. However, the Pareto front solutions for SPEA2 in region D (i.e. solutions with at least 75%

accuracy on the objectives) have better AUC than SGP with Ave in nearly all tasks (except Yst_2). This suggests that good MOGP solutions for SPEA2 (in region D) have internal models/structures that are generally better than SGP with Ave on the tasks. This is interesting because the fitness in MOGP and SGP (with Ave) both use the minority and the majority class accuracies of a solution (e.g. Pareto dominance based on a solution's accuracy on the objectives), and not a separability-based measure (such as the AUC in fitness). In contrast, the MOGP solutions in region D for NSGAII are only better than SGP with Ave in three tasks (Ion, Spt and Bal); in the remaining tasks, SGP with Ave has a sightly better AUC.

However, the AUC of the SPEA2 solutions in region D are not as good as SGP using the AUC directly in the fitness function (i.e. SGP with Auc_F), or SGP with the new fitness functions (such as $Dist$, $Amse$ and $Corr$) in all tasks (except Spt). The better AUC by these SGP approaches suggests that the individual SGP solutions have better class separability than the MOGP solutions and, by implication, more complex internal models than the MOGP solutions.

This is not unexpected. Due to the nature of the MOGP approach, MOGP solutions are not expected to have very high AUC compared to these SGP solutions, since the full *set* of Pareto front solutions are used to represent the performance trade-offs between the minority and the majority class accuracy. In contrast, individual solutions in SGP are required to capture this performance trade-off using an ROC curve.

5.5 Summary and Discussions

The main goal of this chapter was to develop a multi-objective GP (MOGP) approach to classification where the accuracy of the minority and the majority classes are traded-off against each other in the learning process, with particular emphasis on how to develop an effective Pareto Dominance measure in the fitness function. The second goal was to investigate the kinds of solutions evolved by MOGP along the trade-off frontier in terms of their AUC, and compare how the complexity of these trade-off solutions differs to the SGP solutions.

This chapter shows that the Pareto fronts evolved using MOGP contains an accurate set of solutions along the minority and majority class trade-off frontier for the classification tasks. A *single* run of the MOGP algorithm can trace out good set of trade-off solutions, leaving the final choice for the decision-maker. Canonical SGP requires a much longer time to get a reasonable Pareto front using a weighted-average of these two objectives in the fitness function ($Wave$) as

multiple SGP runs are needed, each using a different weighting coefficient which is specified *a priori*.

5.5.1 Pareto Dominance Measures in MOGP

A Pareto Dominance measure using both dominance rank and dominance count in the fitness function in MOGP (such as the SPEA2 algorithm [188]) finds Pareto front solutions that perform better than, or at least as well as, multiple runs of canonical SGP. However, a Pareto Dominance measure using only dominance rank in the fitness function in MOGP (such as the NSGAII algorithm [53]) cannot achieve this to a sufficient level of accuracy in three out of six tasks. In these three tasks, the Pareto fronts evolved using MOGP with SPEA2 dominate the NSGAII fronts. This chapter shows that this is because the fitness function with SPEA2 evolves more solutions in the middle region of the frontier, pushing this front outwards toward high accuracy rates both the minority and majority class. In contrast, the fitness function with NSGAII tends to evolve a better "spread" of solutions along the whole of the frontier. In these classification tasks, end-region solutions are not very desirable as these typically represent biased classifiers which have high accuracy on one class but poor accuracy on the other. The evolved populations using MOGP with SPEA2 also contain more non-dominated solutions (on average) in the final generation than NSGAII in all tasks.

5.5.2 AUC of Pareto Front Solutions in MOGP

Using the AUC to measure the classification ability of an individual Pareto front solution, AUC values generally improve as the objective performances increase. This means that AUC is better in the middle-region of the frontier (i.e. solutions with high accuracy on both classes) compared to the end-regions of the frontier (i.e. biased solutions). Examining the AUC of the evolved Pareto front solutions in different regions of objective-space shows that MOGP with SPEA2 not only finds more non-dominated solutions in the final generations than NSGAII, but that these solutions for SPEA2 generally also have better AUC than NSGAII in nearly all tasks.

MOGP solutions which have at least 50% accuracy on both classes have similar AUC to SGP solutions found using the weighted-average fitness function ($Wave$). However, these MOGP solutions have poorer AUC than SGP solutions evolved using a good fitness function (from the previous chapter). This shows that in SGP, solutions capture this performance trade-off individually (via an

ROC curve); whereas the MOGP approach delegates this requirement to the set of genetic program classifiers along the Pareto frontier.

5.5.3 MOGP for Ensemble Learning

One of the key advantages of the MOGP approaches is that the evolved Pareto fronts represent highly-accurate classifiers, each with a different performance bias toward either the minority or majority class. The goal of this chapter has been to present these classifiers to the end-user for the final selection. However, as the front of non-dominated classifiers has as much information about how to solve a particular problem as any single individual, the combined knowledge or classification abilities of these non-dominated individual can be used *co-operatively* in an ensemble, to further improve classification performances. In an ensemble of classifiers, a simple strategy to combine together the classification abilities of multiple individuals is to use a *majority vote* of these individuals. Here each individual votes on what class label to assign to a given data instance, and the class label with the most number of votes determines the class of that particular instance.

However, for an ensemble of classifiers to be more accurate than any of its individual members, the individual members must be diverse in their outputs, i.e., make different errors on different inputs. This allows for cooperation between individuals in the ensembles. The next chapter develops an ensemble-based MOGP approach to classification with unbalanced data, and focuses on adapting the fitness function in MOGP to evolve a diverse and accurate set of Pareto front solutions which can be successfully combined into an ensemble of classifiers.

Chapter 6

MOGP for Ensemble Learning

This chapter is organised as follows. The first section outlines the chapter introduction and goals. The second section discusses the ensemble learning approaches. The third section outlines the ensemble combination and selection strategies used in the experiments. The fourth section examines the evolved Pareto fronts in MOGP. The fifth section presents the ensemble classification results on the tasks. The sixth section examines the ensemble "wins" on a run-by-run basis for the tasks. The seventh section compares the MOGP ensembles to canonical SGP, Naive Bayes and Support Vector Machines. The eighth section analyses several evolved MOGP classifiers. The last section provides a summary of this chapter.

6.1 Introduction

The previous chapter shows that a single run of the MOGP algorithm can simultaneously evolve an accurate set of genetic program classifiers along the minority and majority class trade-off frontier for the classification tasks. The MOGP algorithm accomplishes this by treating these two objectives independently in the learning process using Pareto dominance in fitness. MOGP, in particular with SPEA2 [188], evolves Pareto front solutions in a single run that perform better than, or at least as well as, multiple runs of canonical single-objective GP (SGP) using a weighted-average of these two objectives in the fitness function. As the evolved Pareto front is a set of genetic program classifiers, each with a different performance bias toward either the minority or the majority class, the final choice (i.e. the classifier with the desired trade-off) is left for the decision-maker.

However, one of the key advantages of the MOGP approach is that the full evolved Pareto front contains at least as much information about how to

solve a particular classification task as any single individual. This means that the combined knowledge or classification abilities of these individuals can be used *co-operatively* in an ensemble of classifiers to further improve classification performances. In an ensemble of classifiers, each individual votes on the class label, and the class label with the most number of votes is taken as the class of a particular input instance. An ensemble of classifiers can be more accurate than its individual members provided that the individual members are both accurate and *diverse*. Diverse ensemble members should not make the same errors on the same inputs; otherwise, the ensemble members risk misclassifying all the same inputs together. In other words, in a good ensemble, if one individual generates an error for a given input, i.e., votes for the incorrect class label, the other members should not also make the same error.

This chapter develops an approach which combines the genetic program classifiers along the Pareto front in MOGP into an ensemble, to further improve the classification performances on the minority and the majority class on the unbalanced tasks.

6.1.1 Diversity Between Individuals

Constructing accurate and diverse ensembles is a difficult problem. Dietterich [54] discusses two main techniques (among others) to generate diversity among individuals in the ensembles. The first uses bagging and boosting techniques where the input-space is sampled into smaller subsets (called bootstrap samples) which are used to train the individual ensemble members [159][164][137]. The second uses the inherently stochastic and population-based nature of evolutionary algorithms (EAs) to generate diverse individuals. Some EAs have been combined with bagging and boosting techniques for ensemble diversity [123][118][37]. Some others use an additional penalty term in the fitness function such as Negative Correlation Learning (NCL) [36][40][35] to encourage diversity between individuals, or use a cooperative co-evolutionary method (such as "teaming" in GP [29][165]) to evolve teams of weak individuals that cooperate strongly together. Approaches which uses diversity measures in fitness, or "teaming"-based methods are different from typical bagging and boosting techniques as these works utilise the *full* training set to learn the individual ensemble members, to promote better interaction and cooperation between individuals. In bagging, sampling techniques are used to divide the training data into smaller subsets for learning.

This chapter focuses on how to incorporate ensemble diversity measures in the fitness function in MOGP to promote/encourage diversity among the evolved solutions. Two diversity measures are compared in the fitness function in MOGP, namely, NCL [119][40][47] and pairwise failure crediting (PFC) [36].

While existing ensemble learning approaches show good results on a wide range of classification tasks with balanced classes, some approaches have limitations when data sets are unbalanced, which this MOGP approach tries to address. When data sets are unbalanced, most existing works rely on sampling techniques to either create balanced bootstrap samples when training bagging approaches [123][118][124][37], or first re-balance the training data when diversity measures (such as NCL) are used in fitness evaluation [168][169]. This means that the quality of the individual members are dependent on the sampling algorithm used in the learning process. This MOGP approach uses the original unbalanced training data "as is" in the GP learning process, using Pareto dominance in fitness in MOGP to perform cost adjustment between the minority and the majority class. This allows us to concentrate on the evolutionary search and diversity measures in the MOGP algorithm, and remove any dependence on a sampling algorithm to artificially re-balance the class distributions during the learning process.

As a result, there has been little previous work which investigates how to adapt the ensemble diversity measures in fitness to account for the skewed class distributions in these tasks [168]. Most works calculate diversity on all examples irrespective of class (as the classes are assumed to be balanced) [35][36][119][40], or first re-balance the training data *prior* to the diversity calculation when data is unbalanced [168]. In [168], the diversity is only calculated on the minority class instances, and diversity on the majority class is ignored in fitness. In contrast, no previous work using genetic program classifiers (as the base learners) in the ensembles measures diversity separately for each class using the original unbalanced data. To address these limitations, in this MOGP approach the diversity on both the minority and the majority class contributes *equally* in fitness.

6.1.2 Ensemble Combination and Selection Strategies

Much work in this area explores good ensemble combination and selection strategies [29][180][37] [39][76]. Ensemble combination strategies investigate how to combine together the outputs of individual members in the final ensemble, and ensemble selection strategies investigate how to choose which learned base classifiers to include in the final ensemble.

This chapter uses a majority vote (or average) of the predictions of the individual Pareto front solutions in MOGP to obtain the ensemble output, as this represents a common ensemble combination strategy. However, as the evolved Pareto fronts in MOGP can also contain *biased* solutions with high accuracy on one class and poor accuracy on the other (as shown in the previous chapter), these biased ensemble members can influence the final ensemble vote. This chapter evaluates three other strategies to potentially limit the influence of biased ensemble members on the ensemble vote. The first assigns weights (based on the fitness of an individual) to control the relative importance of each member's contribution in the ensemble. A similar weighted-voting strategy has been shown to outperform the traditional majority voting strategy on a range of classification tasks in ensemble learning [29][180][37] [39]. The second uses a simple ensemble selection strategy which disallows biased Pareto front solutions (with less than 50% accuracy on both classes) from voting in the ensembles, as these biased individuals are no better than random guessing on the tasks. A similar approach is shown to successfully prune unwanted members from the ensembles [137]. The third strategy also uses an ensemble selection strategy, based on the off-EEL (offline evolutionary ensemble learning) [76] algorithm. This algorithm employs a greedy search to choose which individuals in the ensembles produce the best ensemble results. These four strategies for ensemble combination and selection are evaluated in MOGP to investigate which strategy produces the best classification results on the unbalanced tasks.

6.1.3 Goals

This chapter has two main goals. The first goal is to develop an ensemble learning approach to combine Pareto front classifiers using the fitness function to promote diversity between individuals. Two ensemble-diversity measures are developed in the fitness function to calculate diversity separately for the minority and the majority classes, and these are compared to an ensemble approach using no explicit ensemble-diversity measure in fitness. The second goal is to compare four ensemble combination and selection strategies in the MOGP ensembles, to investigate which strategy produces better ensembles results on the tasks.

6.2 MOGP Approaches for Ensemble Learning

This section outlines the MOGP approaches for ensemble learning. This includes the underlying MOGP approach to evolving the base learners (genetic program solutions), and the two adaptations to the fitness function to encourage diversity between the base learners.

6.2.1 Underlying MOGP Approach

Recall from the previous chapter that the MOGP approach uses the accuracy on the minority and majority classes as the two learning objectives to evolve a Pareto front of genetic program solutions along this trade-off surface. As the previous chapter shows that MOGP with SPEA2 [188] evolves better-performing solutions on these tasks than MOGP with NSGAII [53], MOGP with SPEA2 is used to represent the underlying MOGP approach to evolving the ensemble members. This means that the evolved Pareto front returned from the MOGP search with SPEA2 (for a given run) forms the ensemble of genetic program solutions. To compare the effectiveness of the two ensemble-diversity measures in fitness, NCL and PFC (which are discussed in subsequent sections), MOGP with SPEA2 is used to represent the ensemble learning approach where no explicit ensemble-diversity objective is used in fitness — this is called the *Baseline* MOGP approach.

The Baseline MOGP approach simply relies only on the stochastic way in which new classifiers are created in the evolutionary search process (e.g., using the genetic operators) for diversity between solutions. A similar approach which uses only the randomness of the initial weights in a neural network-based ensemble [137], is shown to be sufficient for diversity. This chapter extends this idea to genetic program-based ensemble classifiers. By comparing the ensemble performances using the Baseline MOGP approach with the two diversity-based MOGP approaches using NCL and PFC, this chapter investigates whether the Baseline's approach for diversity in the evolved solutions is sufficient compared to MOGP with NCL or PFC on the tasks.

6.2.2 Diversity in MOGP Fitness

As discussed, a key condition for an ensemble of classifiers to be more accurate than any of its individual members is that the ensemble members must be accurate and diverse with respect to their outputs [54]. Diverse ensemble members should not make the same errors on the same inputs, otherwise the

ensemble will risk misclassifying all the same inputs together. One of the main techniques to construct diverse ensembles involves injecting randomness into the learning algorithm [54]. In the Baseline MOGP approach, the stochastic way in which new GP solutions are created (e.g. using the genetic operators) is used to evolve diverse classifiers. However, without an explicit diversity objective in fitness to encourage the evolved solutions to make different errors on different inputs, the ensemble members are not guaranteed to be diverse with respect to their predictions. For this reason, the MOGP approach is adapted to incorporate a diversity objective into the fitness function, aiming to reward solutions which have better diversity with better fitness values. Two measures are investigated to promote the evolution of diverse solutions in the population: Negative Correlation Learning (NCL) and Pairwise Failure Correlation (PFC). These measures have proven effective in evolving diverse ensembles for classification [36][168].

However, most related works which use the NCL or PFC in fitness (such as [36][40][35][126]) calculates diversity relative to all examples in the training set irrespective of class, as these works assume that data sets are *balanced*. When data sets are unbalanced, these measures must first be adapted to calculate the diversity for a solution *separately* on each class, to account for the skewed class distributions in these tasks. Otherwise, these diversity measures also risk being biased toward the majority class. This study adapts these measures to use the average diversity on the minority and majority class as the final diversity estimate in the fitness function, to treat the diversity on both classes as equally important in fitness. As discussed above, only one other approach has also used NCL with unbalanced data [168]. However, in [168], the NCL is only applied to minority class instances (diversity on the majority class instances is ignored).

6.2.3 Negative Correlation Learning (NCL)

The first measure to encourage diversity among the individuals in the population uses NCL as a correlation penalty term in the fitness function [40][36][35]. The NCL measure is based on the principle of minimisation of mutual information between variables [40][36][35]. In MOGP, NCL is used to measure the phenotypic differences between the solutions in the ensemble and the rest of the population. The NCL measure, given by Eq. (6.1), calculates the *average* correlation penalty for each class, for a given solution p in the population.

$$NCL_p = \frac{1}{2}\sum_{c=1}^{K}\left(\frac{1}{MN_c}\sum_{i=1}^{N_c}(G_i^p - E_i)\left[\sum_{j=1,j\neq p}^{M}(G_i^j - E_i)\right]\right) \tag{6.1}$$

where

$$G_i^p = \frac{1}{1 + e^{gp_i^p}}$$

In Eq. (6.1), K is the number of classes, and N_c is the number of training examples in class c. G_i^p is the processed output, and gp_i^p is raw output, of genetic program p when evaluated on the ith example in class c. The processed genetic program output is the raw program output when scaled between the range $[0, 1]$ using a transfer (or sigmoid) function. This is required by the NCL calculation, otherwise genetic programs which produce large output values risk unduly inflating the NCL penalty. The sigmoid function is applied to the value returned from the root node of the genetic program during the fitness evaluation. This ensures that positive raw output values are "spread out" between 0.5 and 1, and negative raw output values are spread out between 0 and 0.5 (when the raw output value is 0, the processed output is 0.5).

E_i is the output of the ensemble on the ith example in class c. This is the predicted class label returned by the ensemble, i.e., 0 or 1 for the majority or minority class, respectively, according to a majority vote of all ensemble members. The ensemble size (the number of non-dominated solutions in the current generation) is given by M.

As the NCL is a penalty term in fitness, the lower the NCL value for a particular class, the better the diversity of a solution.

In typical classification tasks where the class distributions are balanced, the NCL penalty is calculated with respect to all training examples [36][40][35][35]. However, Eq. (6.1) is adapted in this approach to calculate diversity separately for each class, to account for the skewed class distributions in these tasks. The average NCL penalty on the minority and majority then represents the final diversity estimate.

The NCL penalty is incorporated into MOGP by using Eq. (6.1) as the *secondary* fitness measure instead of the "crowding" distance. This means that the NCL term is used to resolve selection (e.g. for crossover/mutation and archive selection) when the primary fitness measure (Pareto ranking using SPEA2) is equal between two or more individuals. NCL is used as the secondary fitness measure because Eq. (6.1) requires the ensemble output (E) in its calculation. This means that the primary fitness measure must be applied to the population first to determine which solutions are non-dominated in the population (i.e. the current Pareto front), as these solutions form the ensemble.

	Inputs				
	1	2	3	4	5
a_1	0.8	0	0.7	0.5	0.2
a_2	0.5	0.6	0.1	0.7	0
a_3	0.1	0.7	0.6	0.1	1
E	1	1	1	1	0

(a)

Calculation	*Inputs*				
1. $a_3 - E$	-0.9	-0.3	-0.4	-0.9	0
2. $a_1 - E$	-0.2	-1	-0.3	-0.5	-0.8
$a_2 - E$	-0.5	-0.4	-0.9	-0.3	-1
(sum)	-0.7	-1.4	-1.2	-0.8	-1.8
3. (1)×(2)	0.63	0.42	0.48	0.72	0

(b)

Figure 6.1: (a) The (processed) outputs for three solutions and the ensemble output (E) on the five inputs (incorrect predictions are underlined assuming that the target class label is 1). (b) The three steps to calculate the NCL for solution a_3 where final NCL value for a_3 is 0.15 $\left(\frac{\sum \text{step 3}}{M \times N_c} = \frac{2.25}{3 \times 5}\right)$.

Example of NCL Calculation

To illustrate how NCL is calculated between solutions, Figure 6.1(a) shows the (processed) outputs of three genetic program solutions, a_1, a_2 and a_3, on five input examples from the minority class. Assuming that the target class label is `1` (minority class), processed outputs that are ≥ 0.5 are considered correct class predictions; otherwise they are incorrect class predictions (these are underlined in Figure 6.1(a)). Assuming that these three solutions represent the ensemble, the ensemble output (i.e. predicted class label) is given by E in Figure 6.1(a). As discussed earlier, the ensemble output is obtained using a majority vote of the class labels of individual members. The ensemble output E matches the target class label on the first four inputs, as exactly two individuals vote for the correct class label on these four inputs. Therefore, the ensemble accuracy in Figure 6.1(a) is 80% (four out of five inputs are correctly labeled), while each solution only achieves an individual accuracy of 60% (three out of five inputs correct).

Figure 6.1(b) shows how the NCL for solution a_3 is calculated on each input instance. Step 1 corresponds to the first term in Eq. (6.1), i.e., $G_i^p - E_i$, which compares the outputs of the given solution with the outputs of the ensemble. Step 2 corresponds to the second term in Eq. (6.1), i.e., $\sum_{j=1,j\neq p}^{M} G_i^j - E_i$, which compares the outputs of the given solution with the other ensemble member's outputs. In step 3, the results from the previous two steps are multiplied (for each input), and these values are then summed over the five inputs. This final value is then normalised (0.15) to represent the diversity for solution s_3 on these five inputs.

6.2.4 Pairwise Failure Crediting (PFC)

The second diversity measure is PFC [36], as given by Eq. (6.2) which calculates the PFC for solution p with respect to class c.

$$PFC_{c,p} = \frac{1}{T-1} \sum_{q=1,q\neq p}^{T} \frac{\sum_{i=1}^{N_c} Diff(gp_i^p, gp_i^q)}{Err_c^p + Err_c^q} \tag{6.2}$$

where

$$Diff(gp^p, gp^q) = \begin{cases} 1 & \text{if } I_{cls}(gp^p) \neq I_{cls}(gp^q) \\ 0 & \text{otherwise} \end{cases}$$

In Eq. (6.2), T is population size, N_c is the number of training examples in class c, and gp_i^p is the raw output of genetic program p when evaluated on the ith example in class c. The function $Diff(\cdot)$ is used to compute the Hamming distance (HD) between the outcomes of two solutions (p and q) on class c. This function returns 1 if the predicted class labels of two genetic program solutions are different for a given input instance, or 0 otherwise. The predicted class label of a solution is determined by indicator function I_{cls} in Eq. (6.2) which simply returns 1 (minority class) if raw output value is zero or positive, or 0 (majority class) otherwise according to the zero-threshold strategy. In Eq. (6.2), Err_c^p is the number of incorrect class predictions by a given solution p on class c. An incorrect prediction occurs when the predicted and actual class labels for a given input are different. Unlike NCL, Eq. (6.2) will return values between 0 and 1 where the higher the PFC value, the better the diversity, i.e., lower overlap of common errors.

Similar to the NCL, Eq. (6.2) calculates the diversity between examples from the two classes separately to account for the skewed class distributions in the tasks. The average PFC on the minority and the majority classes then represents the final diversity.

PFC has two major differences to NCL. Firstly, PFC is a population-level diversity measure. This means that PFC measures the diversity of each solution with respect to *all* other solutions in the population, whereas NCL compares the outputs of a solution to the ensemble and the ensemble members (not other solutions in the population that are *not* in the ensemble). This also means that unlike NCL, PFC does not require the ensemble's output (on a given input) in the PFC equation. Secondly, PFC measures diversity based on the binary-valued outcome of a genetic program solution in terms of 1 or 0 for a correct or incorrect class prediction, respectively, whereas NCL uses the (processed) output values of

the solutions.

Note that the computational overhead required to compute the PFC during fitness evaluation, where each solution is compared to all others in the population, is $T(T-1)$ total comparisons between solutions (where T is the population size). However, the total number of comparisons can be reduced by simultaneously accumulating PFC values between any two solutions in a pairwise manner. For example, the diversity between two solutions a_1 and a_2 in the population will be the same when a_1 is compared to a_2 during a_1's fitness evaluation, and when a_2 is compared to a_1 during a_2's fitness evaluation. By simultaneously accumulating PFC values between solutions in a pairwise manner, $\frac{1}{2}T(T-1)$ comparisons are required to compute the PFC for the entire population.

PFC in MOGP Fitness

As each solution in the population is compared to all others, the PFC aims to make the solutions in the population uncorrelated to all other solutions. This is different to NCL which aims to minimise the correlation between solutions and the ensemble. As discussed above, Eq. (6.2) does not require the ensemble output in the PFC equation. This allows the PFC measure to be used at *any* stage in the fitness evaluation. In contrast, NCL requires that the non-dominated set of solutions in the population be known prior to the NCL calculation, as these solutions represent the ensemble and the ensemble's output is used in the NCL equation.

To take advantage of this flexibility, Eq. (6.2) is incorporated into the objective performance of the evolved solutions (alongside the classification accuracy) on the two classes, and *before* the Pareto ranking (using the SPEA2 algorithm) is applied to the population. This gives equal selection preference to accurate and diverse solutions in the population. This is shown by Eq. (6.3), where $(S_p)_c$ is the objective performance of solution p on objective c.

$$(S_p)_c = Y\left(\frac{1 - Err_c^p}{N_c}\right) + (1-Y)PFC_{c,p} \tag{6.3}$$

In Eq. (6.3), weight factor Y specifies the trade-off between accuracy (first component in the equation) and diversity (second component in the equation) where $0 < Y < 1$. The MOGP approach with PFC uses a Y value of 0.5 to treat accuracy and diversity as equally important in fitness. In both the Baseline MOGP and MOGP with NCL, the objective performance $(S_p)_c$ uses only the accuracy of a solution p on class c. By incorporating the accuracy and diversity

	Input				
Solution	1	2	3	4	5
a_1	1	0	1	1	0
a_2	1	1	0	1	0
HD	0	1	1	0	0

(a) Diversity is $\frac{HD(a_1,a_2)}{err_{a_1}+err_{a_2}} = \frac{2}{2+2} = 0.5$

	Input				
Solution	1	2	3	4	5
a_1	1	0	1	1	0
a_3	0	1	1	0	1
HD	1	1	0	1	1

(b) Diversity is $\frac{HD(a_1,a_3)}{err_{a_1}+err_{a_3}} = \frac{4}{2+2} = 1$

Figure 6.2: Pairwise PFC comparisons between three solutions (a_1, a_2 and a_3) on five inputs (in the same class).

of solutions into the objective performances, the Pareto ranking of the solutions (according to SPEA2) is not solely based on the accuracy of the solutions on the two classes (as is the case for MOGP using NCL). This allows the PFC ensembles to contain more diverse but potentially less accurate solutions.

Other Weighting Coefficients

Other weighting coefficients for Y (e.g. 0.25, 0.75 and 1) have also been explored in Eq. (6.3) to investigate if these can improve ensemble performances compared to an equal weighting between accuracy and diversity (Y of 0.5) on the tasks. Preliminary results find that an equal weighting between accuracy and diversity in PFC shows the best ensemble performances on the two classes compared to Y values of 0.25, 0.75 and 1 on the tasks. As exploring other weighting coefficients for Y is not one of the main goals of this chapter, these preliminary results are omitted from this chapter but can be seen in the Appendix B (in Section B.2.2).

Example of PFC Calculation

Figure 6.2 illustrate how the PFC is computed in a pairwise manner for the same three genetic program solutions (a_1, a_2 and a_3) on the same five inputs (as used in the previous example in Figure 6.1(a) for NCL). However, Figure 6.2 shows the *outcomes* of the solutions on the inputs, where 1 is a correct class prediction and 0 is an incorrect prediction. Figure 6.2(a) compares solutions a_1 and a_2, while Figure 6.2(b) compares solutions a_1 and a_3. Although all three solutions generate the *same* number of errors (two errors on the five inputs), a_1 and a_3 are more diverse in their outputs (than a_1 and a_2) as these solutions make different errors on the same inputs. In Figure 6.2(a), the pairwise PFC contribution between a_1 and a_2 is 0.5; while in Figure 6.2(b), the pairwise PFC contribution between a_1 and a_3 is 1 (better). Assuming that the population consists of only these three

solutions, the final PFC value for a_1 is 0.75 as shown below.

$$PFC_{a_1} = \frac{1}{T-1}\left(\frac{HD(a_1,a_2)}{err_{a_1}+err_{a_2}} + \frac{HD(a_1,a_3)}{err_{a_1}+err_{a_3}}\right) = \frac{1}{2}(0.5+1) = \frac{1}{2}(1.5) = 0.75$$

6.3 Ensemble Combination and Selection

This section outlines four ensemble combination and selection strategies used in the MOGP experiments, and the rationale for choosing these strategies in MOGP.

6.3.1 Majority Voting

A majority vote of the predictions of the individual ensembles members represents a typical strategy to combine the ensemble members [54][29]. For a given input, the class label with the most votes from the ensemble members is taken as the ensemble output. If a *draw* occurs in the voting process, then the minority class is returned as the predicted class.

However, as the previous chapter shows that the evolved Pareto fronts can also contain biased solutions, i.e., solutions with high accuracy on one class and poor accuracy on the other, it can be expected that these biased ensemble members might also influence the final ensemble vote (this is discussed in more detail in Section 6.5.1 in the experimental results). To exclude highly-biased ensemble members from the voting process, Pareto front solutions with less than 5% accuracy on either class are excluded from the majority voting process.

Three other strategies are also compared to potentially limit the influence of biased ensemble members on the final ensemble vote (outlined below).

6.3.2 Fitness-Weighted Majority Voting

This strategy uses a fitness-weighted majority vote of the Pareto front solutions where the weight (or contribution) of a given individual in the voting process is based on the fitness of that individual on the training set. This strategy aims to reduce the contribution of biased solutions in the voting process. Previous work has shown that a weighted majority vote can outperform the traditional majority vote on a range of classification tasks [29][180][37][39]. For the Baseline MOGP, an individual's fitness corresponds to its average accuracy on the minority and the majority classes (on the training set), as the Baseline does not use an explicit ensemble-diversity objective in fitness. This means that biased Pareto front solutions which have poor accuracy on one class will contribute less in the voting

process, while Pareto front solutions with high accuracies on both classes will contribute more.

For NCL and PFC, an individual's fitness is the average of its accuracy and diversity on the two classes (on the training set). For PFC, this is the average of Eq. (6.3) for the minority and majority class as Eq. (6.3) already reflects both the accuracy and diversity of a solution on a particular class. For NCL, this is a combination of the diversity and accuracy on both classes, as shown below (for solution p).

$$\tfrac{1}{2}\left(NCL_p + \tfrac{1}{2}(Acc_{min} + Acc_{maj})\right)$$

where NCL_p is the average diversity on the two classes according to Eq. (6.1), and Acc_{min} and Acc_{maj} are accuracy on the minority and the majority classes, respectively.

6.3.3 Accuracy-based Ensemble Selection

A simple policy to exclude biased solutions from the voting process is to raise the threshold for minimum performance required on the objectives. This ensemble selection strategy simply disallows biased Pareto front solutions with less than 50% accuracy on either the minority or the majority class (on the training set) from voting in the ensemble. This means that only relatively accurate members (with at least 50% accuracy on both classes) will participate in the ensemble voting process. Individuals with at least 50% accuracy on both classes implies that a solution is better than random guessing on the tasks (as discussed in [137] where a similar strategy is used to prune the ensembles).

6.3.4 Off-EEL for Ensemble Selection

The accuracy-based ensemble selection strategy (discussed above) offers a naive yet effective approach to choosing which individuals to use in the final ensemble (from the set of evolved Pareto front classifiers). A more exhaustive and arguably better ensemble selection approach is the off-EEL (offline evolutionary ensemble learning) algorithm [76]. This strategy uses the off-EEL algorithm to find which solutions in the ensembles produce better ensemble performances.

The off-EEL algorithm uses a greedy search to construct the ensembles from a pool of base classifiers (i.e. evolved Pareto front of genetic program solutions). This algorithm sorts the input set of base classifiers according to their fitness

values on the training set (similar to the fitness-weighted majority vote), from the fittest to the least fit classifiers. Then, each classifier is removed from the (sorted) input set and inserted into the ensemble where, at each step, the ensemble is evaluated using a majority vote of the base classifiers in the current ensemble. Once all the base classifiers from the input set are processed, the ensemble with the best performance is taken as the final ensemble. Only odd numbered ensemble sizes are considered as these constitute ensembles where *no draws* can occur in the voting process.

6.4 Evaluation of Diversity Measures in MOGP

This section outlines the experimental setup and MOGP evolutionary parameters, and presents an evaluation of the two diversity measures in the fitness function in MOGP (in terms of hyperarea of the evolved Pareto fronts and MOGP training times).

6.4.1 MOGP Setup and Evolutionary Parameters

The underlying approach to evolving accurate and diverse Pareto front solutions to represent the ensemble members uses MOGP with SPEA2 [188] (from the previous chapter). This means that the same MOGP framework is used to represent the genetic program solutions, and the evolutionary parameters in MOGP are also kept the same as the previous chapter. See the previous chapter for more details.

6.4.2 MOGP Pareto Front Hyperarea

To investigate how the diversity objectives in MOGP affects the performance of the evolved Pareto fronts (on the *test* sets), Table 6.1 reports the average (and standard deviation) hyperarea of the evolved Pareto-approximated fronts, and the hyperarea of the Pareto-optimal (PO) front for the NCL and PFC approaches (over 50 runs). Recall from the previous chapter that the PO front is the set of non-dominated solutions from the union of all Pareto-approximated fronts evolved over 50 runs. For a comparison, Table 6.1 also includes the average hyperarea and PO hyperarea of the Baseline MOGP approach, i.e., when no ensemble diversity objective is used in fitness. Note that the Baseline MOGP results are repeated here for convenience from Table 5.1 (in the previous chapter).

Table 6.1: Average ($\pm$ standard deviation) hyperarea of evolved Pareto-approximated fronts, and hyperarea of the Pareto-optimal (PO) front for the three MOGP approaches (Baseline, NCL and PFC) over 50 runs. The pairs of hyperarea results in bold or italics denote that these two approaches achieve a statistically significantly better hyperarea than the remaining approach (but not each other). The highest PO front hyperarea from all three approaches is underlined.

Task	Basline MOGP		MOGP with NCL		MOGP with PFC	
	Average	PO Front	Average	PO Front	Average	PO Front
Ion	**0.848 $\pm$ 0.041**	<u>0.992</u>	**0.849 $\pm$ 0.039**	0.981	0.828 $\pm$ 0.032	0.982
Spt	0.732 $\pm$ 0.032	<u>0.971</u>	0.733 $\pm$ 0.031	0.964	0.719 $\pm$ 0.025	0.964
Ped	**0.902 $\pm$ 0.019**	0.922	**0.905 $\pm$ 0.011**	<u>0.926</u>	0.883 $\pm$ 0.010	0.921
Yst_1	**0.793 $\pm$ 0.009**	<u>0.931</u>	**0.795 $\pm$ 0.010**	0.922	0.774 $\pm$ 0.009	0.923
Yst_2	**0.949 $\pm$ 0.011**	<u>0.991</u>	**0.949 $\pm$ 0.007**	0.972	0.928 $\pm$ 0.011	0.989
Bal	0.757 $\pm$ 0.063	0.985	*0.810 $\pm$ 0.078*	<u>1.0</u>	*0.800 $\pm$ 0.065*	<u>1.0</u>

Tukey's Honestly Significant Difference (HSD) [166] multiple comparisons test is used to find the statistically significant differences in the average hyperarea for the three MOGP approaches (over 50 runs). Recall that Tukey's multiple comparisons test compares the hyperarea from each system to all others (on a run-by-run basis), and outputs a 95% confidence interval for each pairwise comparison between systems. In four tasks (Ion, Ped, Yst_1 and Yst_2), the average hyperarea for the Baseline and NCL approaches is not statistically significantly different from one another, but both are statistically significantly better than PFC (these are highlighted in bold in Table 6.1). In Bal, the average hyperarea for NCL and PFC are not statistically significantly different from each other, but both are statistically significantly better than the Baseline approach (these are italicised in Table 6.1). In the remaining task (Spt), the average hyperarea for all three MOGP approaches are not statistically significantly different from one another. Note that the highest PO hyperarea achieved by a given MOGP approach over all 50 runs is underlined in Table 6.1.

Table 6.1 shows that in four out of six tasks (in bold in Table 6.1), the PFC approach finds non-dominated solutions with lower classification accuracy on the two classes than both the Baseline and NCL approaches, as the average hyperarea for PFC is the lowest. However, the frontier solutions from the PFC approach may be less accurate but potentially more diverse in their outputs than the Baseline and NCL solutions, due to the selection bias in fitness for PFC (this is explored further in the next section). In Bal, the hyperarea for PFC and NCL is significantly better than the Baseline MOGP. This suggests that the PFC and NCL

Table 6.2: Average training times for the three MOGP approaches in seconds (s) or minutes (m) over 50 runs.

Task	Baseline MOGP	MOGP with NCL	MOGP with PFC
Ion	9.3s $\pm$ 2.4	1.4m $\pm$ 6.8	30.4s $\pm$ 2.6
Spt	9.7s $\pm$ 2.5	1.2m $\pm$ 5.3	29.5s $\pm$ 1.6
Ped	3.9m $\pm$ 1.1	90.2m $\pm$ 1.3	17.7m $\pm$ 24.8
Yst_1	20.8s $\pm$ 7.1	5.6m $\pm$ 14.4	1.2m $\pm$ 4.5
Yst_2	20.1s $\pm$ 8.1	5.3m $\pm$ 18.9	1.1m $\pm$ 5.9
Bal	15.2s $\pm$ 3.9	2.4m $\pm$ 8.5	45.1s $\pm$ 3.2

approaches find frontier solutions that are more accurate on the two classes, and also potentially more diverse in their outputs, than the Baseline MOGP.

In terms of the PO hyperarea for the three MOGP approaches (Baseline, NCL and PFC), the Baseline achieves the the highest PO hyperarea in four tasks (Ion, Spt, Yst_1 and Yst_2). As discussed above, this is due to the selection bias in the Baseline MOGP where the PO solutions for NCL and PFC are less accurate (but potentially more diverse in their outputs) than the Baseline MOGP. In Ped, NCL achieves the best PO hyperarea; while both NLC and PFC achieve the best PO hyperarea in Bal. In Bal in particular, both NCL and PFC find at least one non-dominated solution with 100% accuracy on both the minority and the majority classes (on the test set). This solution represents the best PO hyperarea of 1 in Table 6.1 where the PO frontier consists of this one point alone (in objective-space). It is interesting that neither the Baseline MOGP approach, nor the single-objective GP methods using the different fitness functions (from Chapter 4), is able to accomplish this in any task. This suggests that identifying and promoting solutions with good accuracy *and* diversity on the two classes using the NCL and PFC measures in the fitness function has the potential to achieve perfect solutions on difficult tasks such as Bal.

In Spt, the average hyperarea for all three MOGP approaches are similar to one another (i.e. not statistically different). Further analysis of the results for Spt finds that this is because the Pareto fronts for the three MOGP approaches tend to dominate each other in different regions of the objective-space. This can be seen later in Figure 6.4 which compares the median attainment summary surface for the PFC and Baseline MOGP approaches.

To compare the training times for the different MOGP approaches, Table 6.2 reports the average training times in seconds (s) or minutes (m) over 50 runs. Table 6.2 shows that as expected, both NCL and PFC incur longer average training times than the Baseline approach. This is due to the additional

computational effort required to calculate the corresponding diversity measure in fitness evaluation. However, this increase is not a serious concern in most tasks. PFC also shows faster average training times than NCL in the tasks. For the largest training set, Ped (more than 24000 examples), NCL incurs substantially longer training times than the Baseline and PFC approaches.

6.5 MOGP Ensemble Classification Results

This section presents the classification results of the MOGP approaches and ensemble strategies on the tasks. This section has three main parts. The first part presents the performance of the MOGP ensembles using the majority vote and fitness-weighted majority vote strategies. The second part presents the performance of the MOGP ensembles using the two ensemble selection strategies. The third part analyses how the MOGP ensemble members cooperate to solve the classification tasks.

6.5.1 Voting Accuracy for the Pareto Front Ensemble

Table 6.3 reports the average minority and majority class accuracies (with standard deviations) of the evolved ensembles using the three MOGP approaches (Baseline, NCL and PFC) on the *test* sets over 50 runs. In the column called PF-vote, the ensembles use a majority vote of the full evolved Pareto front (PF) from a given MOGP run. Recall that in this voting process, the class label with the most votes from the ensemble members is taken as the ensemble output. The average number of non-dominated solutions that participate in the ensemble voting process (ensemble sizes) are also shown in Table 6.3.

Immediately noticeable in Table 6.3 for the PF-vote strategy are the strong majority class performances for the three MOGP approaches. In some tasks such as Ion and Yst_2, the corresponding minority class accuracies are still reasonably good, while in the others, particularly Ped and Spt, this is very poor. This shows that in most tasks, the evolved Pareto fronts can contain more solutions biased toward the majority class than the opposite case, i.e., solutions with good minority accuracy or middle-region solutions, as these biased solutions can influence the final ensemble vote, thus biasing the final ensemble prediction.

Analysis of the ensemble performances during the evolution reveals that as the evolution progresses over generations, *more* solutions with strong majority class accuracies achieve non-dominated status than solutions with good minority

Table 6.3: Average accuracy ($\pm$ standard deviation) on the test set and ensembles size for the Pareto Front ensemble using the majority vote (PF-vote) and fitness-weighted vote (PF-Wvote) over 50 runs.

Task	MOGP	Size	Majority Vote (PF-vote)		Weighted Vote (PF-Wvote)	
			Minority %	Majority %	Minority %	Majority %
Ion	Baseline	18.8	84.5 ± 6.2	83.5 ± 9.9	82.5 ± 5.8	89.1 ± 8.3
	NCL	12.7	85.5 ± 5.2	86.8 ± 7.3	80.9 ± 5.9	91.4 ± 5.9
	PFC	28.1	84.9 ± 5.1	92.4 ± 6.4	79.6 ± 6.2	96.3 ± 3.7
Spt	Baseline	7.8	44.5 ± 5.5	88.8 ± 2.7	86.4 ± 13.7	59.9 ± 36.4
	NCL	9.5	48.6 ± 5.6	86.5 ± 2.9	84.1 ± 12.0	63.3 ± 32.7
	PFC	27.3	44.6 ± 5.4	90.8 ± 2.3	75.9 ± 10.4	72.6 ± 22.2
Ped	Baseline	124.9	12.5 ± 27.2	87.1 ± 28.2	89.1 ± 3.6	83.3 ± 3.1
	NCL	52.6	71.8 ± 8.9	91.7 ± 2.7	88.0 ± 3.2	83.5 ± 3.7
	PFC	71.6	82.4 ± 5.6	92.1 ± 2.4	92.5 ± 1.6	84.1 ± 3.1
Yst_1	Baseline	46.7	58.0 ± 4.0	87.1 ± 2.4	70.0 ± 4.2	77.3 ± 4.3
	NCL	25.8	63.6 ± 3.8	83.0 ± 3.3	70.6 ± 3.7	76.8 ± 4.4
	PFC	39.7	64.6 ± 4.8	82.5 ± 4.3	71.8 ± 5.3	75.4 ± 6.5
Yst_2	Baseline	18.5	77.1 ± 4.6	96.2 ± 1.1	82.8 ± 3.6	95.1 ± 1.3
	NCL	16.1	77.6 ± 6.0	95.3 ± 1.7	83.6 ± 4.7	93.7 ± 1.7
	PFC	27.9	81.2 ± 4.9	95.5 ± 1.5	89.6 ± 3.2	92.1 ± 1.9
Bal	Baseline	9.8	53.3 ± 21.4	94.1 ± 4.4	84.2 ± 12.5	71.4 ± 23.6
	NCL	8.4	59.2 ± 16.1	87.8 ± 6.6	86.9 ± 11.8	66.0 ± 29.5
	PFC	20.8	51.7 ± 18.2	95.4 ± 3.5	87.3 ± 9.3	74.1 ± 17.1

accuracies or middle-region solutions. This effect can be seen, to varying degrees, in Figure 6.3 for the Baseline and PFC approaches. These figures show the minority and majority class performances of the MOGP ensembles on the test sets for 50 generations (averaged over 50 runs). MOGP with NCL is omitted in Figure 6.3 as these figures aim to show the general behaviour of the ensembles with and without the diversity objective in fitness, and the NCL performances are relatively similar to PFC on these tasks. The axis scopes in Figure 6.3 are different for the tasks.

Figure 6.3 clearly shows that *more* solutions with stronger majority class accuracy (than solutions with stronger minority accuracy) are included in the ensembles over generations (for the Baseline and PFC approaches), as the ensemble accuracy reflects which class receives the more votes by the different members. This effect is more pronounced in those three tasks in the bottom row of Figure 6.3 (Spt, Ped and Yst_1), particularly for the Baseline MOGP. Here minority class accuracies are declining, while majority class accuracies are improving, over generations. By comparison, minority class accuracies do not decline as sharply

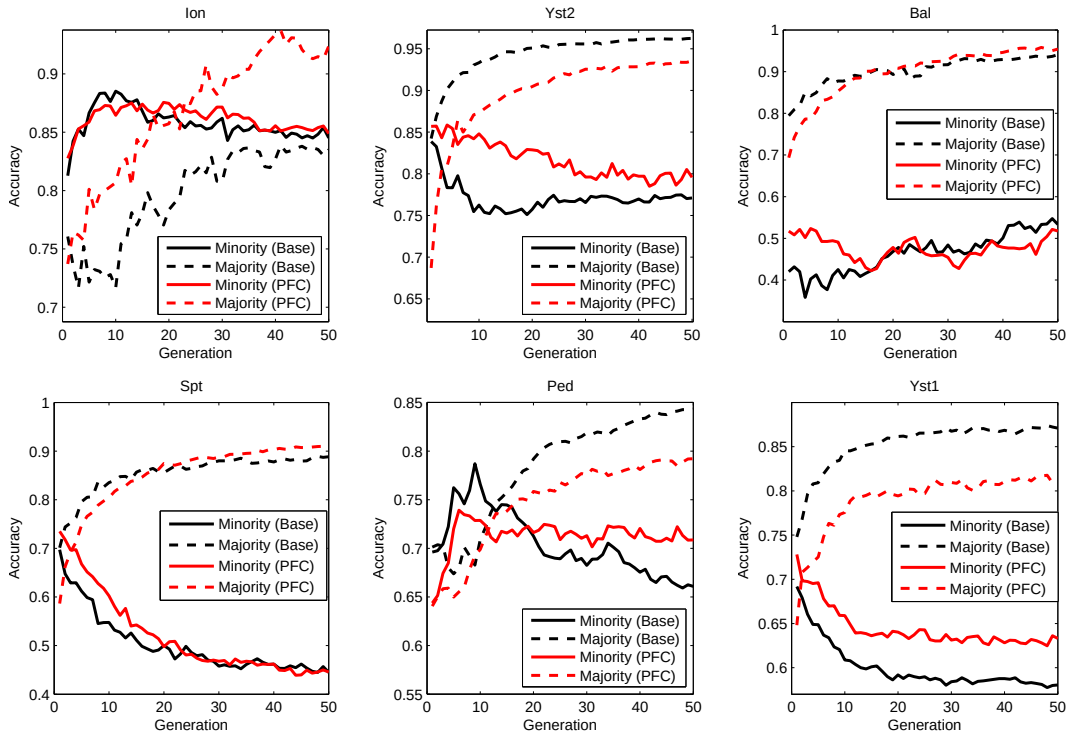


Figure 6.3: MOGP ensemble performances on the minority and majority class (test set) using PF-vote over generations for Baseline and PFC.

in the two tasks in the top row of Figure 6.3 (Ion and Yst_2), even though majority class accuracies are still improving over generations. Only in one task, Bal, do both the minority and the majority class accuracies improve over generations. However, in Bal, minority class accuracies are still substantially lower than majority class accuracies. This may be due to the relatively high level of class imbalance in Bal.

Fitness-weighted Majority Vote

As discussed above, a strategy to limit the influence of biased ensemble members on the final ensemble vote uses a fitness-weighted majority vote of the Pareto front solutions. Here the weight (or contribution) of a given individual in the voting process is based on the fitness of the individual on the training set. The ensemble results using this fitness-weighted majority vote are shown in the column called PF-Wvote in Table 6.3. For presentation convenience, this strategy is called PF-Wvote as the Pareto front (PF) participates in a weighted voting ("Wvote") approach.

According to Table 6.3, all three MOGP approaches with the PF-Wvote strategy show more balanced class performances with better minority class

accuracies compared to the PF-vote strategy in all tasks (except Ion). This is particularly noticeable in Spt, Ped (for the Baseline MOGP), Yst_1 and Yst_2. This suggests that the PF-Wvote strategy succeeded in reducing the influence of biased Pareto front solutions in the ensembles in these tasks. In some tasks (such as Ped and Yst_2), the higher minority class accuracies achieved by the PF-Wvote strategy incur only a relatively small trade-off in the corresponding majority class accuracies; while in some others (such as Spt and Bal), this trade-off is larger. In these four tasks (Ped, Yst_2, Spt and Bal), the PFC ensembles show equally high (and more balanced) class accuracies than the Baseline and NCL ensembles, which have high minority class accuracies but lower majority class accuracies (as mentioned above).

Table 6.3 also shows that in two tasks, Ped and Bal, the PFC ensembles dominate both the Baseline and NCL ensembles. In the four remaining tasks, all three MOGP ensembles are non-dominated with respect to each other. Interestingly, the Baseline MOGP ensembles show similar performances to NCL using this PF-Wvote strategy in some tasks such as Ion, Ped, Yst_1 and Yst_2. This is surprising as the Baseline MOGP uses no ensemble-diversity objective in fitness. This suggests that in these tasks, the Baseline ensembles which use only the stochastic processes within GP for diversity among solutions, can be competitive to the NCL ensembles which uses an explicit diversity measure in fitness. However, further investigation of the differences between the three MOGP approaches is needed; this is explored later in Section 6.6.1 (which compares "wins" for the different MOGP approaches on the tasks).

6.5.2 Ensemble Selection

In addition to the PF-Wvote strategy, two ensemble selection strategies are also compared in MOGP to limit the influence of biased ensemble members in the voting process, to improve ensemble performances. Table 6.4 shows the ensemble accuracies (on the test set) and ensemble sizes using the accuracy-based selection strategy and the off-EEL algorithm for ensemble selection over 50 runs for the MOGP approaches. The accuracy-based selection strategy simply *removes* Pareto front solutions with less than 50% accuracy on either class (on the training set) from the ensemble. This strategy is called the RPF-vote in Table 6.4 as the ensembles are *reduced* compared to the PF-vote and PF-Wvote strategies. The off-EEL algorithm uses a more exhaustive and arguably better ensemble selection approach than RPF-vote to choose individuals from the Pareto front for the final

Table 6.4: Average accuracies (± standard deviation) on the test set and ensembles sizes using RPF-vote and off-EEL [76] ensemble selection strategies (50 runs).

Task	MOGP	RPF-vote			Off-EEL		
		Size	Minority %	Majority %	Size	Minority %	Majority %
Ion	Baseline	7.8	79.9 ± 7.2	87.2 ± 9.1	5.6	83.7 ± 5.8	89.2 ± 8.8
	NCL	10.8	82.6 ± 6.9	91.0 ± 3.2	9.9	82.2 ± 5.5	89.5 ± 8.1
	PFC	22.3	81.7 ± 5.8	95.8 ± 3.8	21.2	83.9 ± 5.2	96.6 ± 2.8
Spt	Baseline	2.8	69.9 ± 11.7	70.1 ± 17.4	3.9	56.0 ± 10.1	83.6 ± 4.8
	NCL	4.1	71.1 ± 9.0	78.4 ± 8.7	5.2	53.9 ± 9.7	83.8 ± 4.8
	PFC	12.1	62.1 ± 8.0	80.5 ± 4.8	10.7	66.3 ± 8.5	79.9 ± 6.6
Ped	Baseline	87.8	81.4 ± 14.1	79.3 ± 28.9	65.3	89.5 ± 1.5	84.6 ± 2.3
	NCL	43.2	87.4 ± 4.5	84.2 ± 6.1	40.4	88.8 ± 3.0	84.4 ± 2.8
	PFC	40.1	91.6 ± 1.9	85.2 ± 3.0	55.2	90.6 ± 1.5	87.9 ± 1.5
Yst_1	Baseline	24.1	68.7 ± 3.2	77.5 ± 3.8	33.6	68.5 ± 5.5	80.4 ± 5.2
	NCL	17.0	69.0 ± 2.6	80.7 ± 1.8	13.5	64.1 ± 5.0	83.4 ± 4.0
	PFC	16.5	71.0 ± 4.4	75.5 ± 5.4	29.2	70.6 ± 5.4	78.8 ± 5.5
Yst_2	Baseline	15.2	80.6 ± 7.8	94.8 ± 2.2	6.5	92.3 ± 2.9	90.7 ± 2.9
	NCL	13.4	89.6 ± 5.2	91.9 ± 2.9	8.6	80.4 ± 7.3	94.5 ± 2.1
	PFC	20.6	89.2 ± 3.2	92.3 ± 1.8	17.2	93.1 ± 2.6	90.8 ± 2.4
Bal	Baseline	4.7	82.6 ± 13.7	59.0 ± 21.4	4.7	71.4 ± 15.9	85.6 ± 9.0
	NCL	4.9	61.8 ± 4.2	94.1 ± 5.9	5.5	62.1 ± 17.5	85.6 ± 7.3
	PFC	10.1	83.6 ± 9.4	79.5 ± 10.3	10.9	81.4 ± 12.1	86.2 ± 9.1

Table 6.5: Average accuracies (± standard deviation) on the test set and ensembles size using the majority vote (PF-vote) and fitness-weighted vote (PF-Wvote) strategies (50 runs). Repeated from Table 6.3.

Task	MOGP	Size	Majority Vote (PF-vote)		Weighted Vote (PF-Wvote)	
			Minority %	Majority %	Minority %	Majority %
Ion	Baseline	18.8	84.5 ± 6.2	83.5 ± 9.9	82.5 ± 5.8	89.1 ± 8.3
	NCL	12.7	85.5 ± 5.2	86.8 ± 7.3	80.9 ± 5.9	91.4 ± 5.9
	PFC	28.1	84.9 ± 5.1	92.4 ± 6.4	79.6 ± 6.2	96.3 ± 3.7
Spt	Baseline	7.8	44.5 ± 5.5	88.8 ± 2.7	86.4 ± 13.7	59.9 ± 36.4
	NCL	9.5	48.6 ± 5.6	86.5 ± 2.9	84.1 ± 12.0	63.3 ± 32.7
	PFC	27.3	44.6 ± 5.4	90.8 ± 2.3	75.9 ± 10.4	72.6 ± 22.2
Ped	Baseline	124.9	12.5 ± 27.2	87.1 ± 28.2	89.1 ± 3.6	83.3 ± 3.1
	NCL	52.6	71.8 ± 8.9	91.7 ± 2.7	88.0 ± 3.2	83.5 ± 3.7
	PFC	71.6	82.4 ± 5.6	92.1 ± 2.4	92.5 ± 1.6	84.1 ± 3.1
Yst_1	Baseline	46.7	58.0 ± 4.0	87.1 ± 2.4	70.0 ± 4.2	77.3 ± 4.3
	NCL	25.8	63.6 ± 3.8	83.0 ± 3.3	70.6 ± 3.7	76.8 ± 4.4
	PFC	39.7	64.6 ± 4.8	82.5 ± 4.3	71.8 ± 5.3	75.4 ± 6.5
Yst_2	Baseline	18.5	77.1 ± 4.6	96.2 ± 1.1	82.8 ± 3.6	95.1 ± 1.3
	NCL	16.1	77.6 ± 6.0	95.3 ± 1.7	83.6 ± 4.7	93.7 ± 1.7
	PFC	27.9	81.2 ± 4.9	95.5 ± 1.5	89.6 ± 3.2	92.1 ± 1.9
Bal	Baseline	9.8	53.3 ± 21.4	94.1 ± 4.4	84.2 ± 12.5	71.4 ± 23.6
	NCL	8.4	59.2 ± 16.1	87.8 ± 6.6	86.9 ± 11.8	66.0 ± 29.5
	PFC	20.8	51.7 ± 18.2	95.4 ± 3.5	87.3 ± 9.3	74.1 ± 17.1

ensemble. Off-EEL iteratively adds individuals to the ensemble and evaluates the ensemble performance (on the training set) at each step. Once all Pareto front solutions are processed, the ensemble with the best performance is taken as the final ensemble.

For convenience, the PF-vote and PF-Wvote ensemble performances (from Table 6.3 in the previous section) are repeated in Table 6.5 (below Table 6.4) to make comparisons between these ensemble approaches easier.

Advantage of off-EEL for Ensemble Selection

Table 6.4 shows that both the RPF-vote and off-EEL strategies show relatively balanced performances on both the minority and majority classes compared to the PF-vote (from Table 6.5). Both strategies achieve this by only allowing relatively accurate Pareto front solutions from voting in the ensembles. This can be seen by the smaller ensemble sizes for RPF-vote and off-EEL in Table 6.4 (compared to Table 6.5), and suggests that both strategies are effective in limiting the influence of biased Pareto front solutions in the ensemble voting process.

According to Table 6.4, the off-EEL ensembles either dominate, or are non-dominated relative to, the RPF-vote results in the tasks. This is important as it shows that the off-EEL ensembles are at least as good as, or in some cases better than, the RPF-vote ensemble in the tasks. In some cases where the off-EEL ensembles dominate the RPF-vote ensembles such as Ion (Baseline and PFC) and Ped (Baseline and NCL), the off-EEL ensemble sizes are even smaller than the RPF-vote ensembles. This suggests that the RPF-vote ensembles still contain some individuals that do not positively contribute to the ensemble performance. This is not unexpected as the off-EEL algorithm uses an additional search to choose good individuals for the ensembles; whereas in the naive RPF-vote strategy, the performance threshold to exclude individuals from the ensembles is determined *a priori*.

A similar observation can be seen when the off-EEL strategy is compared to the PF-Wvote strategy from Table 6.5. The off-EEL ensembles are at least as good as, or better than, the PF-Wvote ensemble in all tasks.

Tables 6.4 and 6.5 also show that the PFC ensembles using both off-EEL and PF-Wvote dominate both the Baseline and NCL ensembles in two tasks (Ped and Bal). In three other tasks (Ion, Yst_1 and Yst_2), PFC also dominates the Baseline but only using the off-EEL strategy (PFC and the Baseline are non-dominated to each other using the PF-Wvote strategy in these three tasks). This may be because the Baseline MOGP uses no explicit ensemble-diversity objective in fitness, and the

PFC approach can find individuals that are more diverse in their outputs. When individuals with good diversity are combined together in the ensembles created using off-EEL, the PFC ensembles (with off-EEL) improve to a greater extent than the Baseline ensembles (with off-EEL). The NCL ensembles are not able to achieve this as NCL does not dominate the Baseline or PFC in any task using either the off-EEL and PF-Wvote strategies. This suggests that the PFC ensembles may have the best diversity from the three MOGP approaches (this is also explored further in the next section which compares "wins" for the different MOGP approaches on the tasks).

Baseline Better with Off-EEL

In the Baseline MOGP approach, the PF-Wvote results (from Table 6.5) dominate the RPF-vote results in Table 6.4 in nearly all tasks. The only exception is Spt where RPF-vote shows equally high class accuracies, while PF-Wvote has a slight bias toward high minority class accuracy only. As mentioned above, the poorer performance by the RPF-vote strategy may be due to the main limitation of this strategy, i.e., the criterion for ensemble selection is determined *a priori* (and remains fixed for all tasks). This means that these ensembles still contain individuals that do not positively contribute in the voting process. In contrast, by reducing the contributions of individuals with poorer fitness in the voting process (relative to other fitter solutions) using the fitness-weighed vote in PF-Wvote, the PF-Wvote strategy can generally outperform RPF-vote strategy on the tasks.

For the NCL and PFC approaches, the PF-Wvote and RPF-vote strategies produce similar (non-dominating) ensemble performances in nearly all tasks (except Bal). This suggests that both methods are similarly effective in keeping the ensemble performances relatively well-balanced on the two classes (compared to PF-vote). In Bal, the NCL results for PF-Wvote and RPF-vote vary in their minority to majority class bias, e.g., the PF-Wvote is stronger on the minority class, while the RPF-vote is stronger on the majority class. This may be due to noise or the comparatively high level of class imbalance in Bal.

Due to these limitations for the RPF-vote strategy, the remainder of this chapter will focus on the PF-Wvote and off-EEL strategies.

6.5.3 Cooperation of Ensemble Members

The advantage of evolving an ensemble of accurate and diverse classifiers is that the ensemble can perform better than all of its individual members due to better

generalisation in the voting process. However, the ensemble results in Tables 6.4 and 6.5 using the different ensemble combination strategies, do not show to what extent the diversity objectives in fitness contribute to the ensemble performances, e.g., by encouraging better cooperation between members. To try to answer this question, the ensemble performances are contrasted to the performances of the individual ensemble members themselves on a *run-by-run* basis, with and without the diversity objective in fitness. The median attainment surface (from the previous chapter) is used to approximate the performance of an "average" evolved Pareto front along the two objectives over 50 MOGP runs. In other words, the median attainment surface is used to represent the performance, on average, of the individual ensemble members over 50 runs of a particular MOGP approach.

Figure 6.4 shows the median attainment surfaces for the Baseline and PFC approaches, and their ensemble performances with off-EEL for each of the 50 runs (on a run-by-run basis) on the tasks. MOGP with NCL is omitted in Figure 6.4 as the main aim of these figures is to contrast the ensemble behaviour with and without a good diversity objective in fitness (and not the differences between the two diversity objectives in fitness). In fact, Tables 6.4 and 6.5 also suggests that PFC may even achieve better diversity between individuals than NCL, as the PFC ensembles are as good as, or better than, the NCL ensembles on these tasks (however, this aspect is explored in more detail in the next section). Note that the axis scopes in Figure 6.4 are different for the tasks.

Figure 6.4 shows that both the Baseline and PFC ensembles dominate their corresponding median attainment surface in all tasks, as the ensemble performances lie *above* the median attainment surface. This shows the cooperation between solutions as the ensemble performances are *better* than an average set of individual ensemble members. In some tasks (Ion, Ped, Bal and Yst_2), *more runs* of the PFC ensembles dominate the Baseline ensembles, even though the Baseline median attainment surface clearly dominates the PFC median attainment surface in these tasks. This shows that in these tasks, the PFC ensembles show better diversity/cooperation between individuals than the Baseline ensembles, as better diversity/cooperation leads to better ensemble performances.

In some tasks such as Yst_1 and Yst_2 in Figure 6.4, the PFC and Baseline ensembles show relatively similar performances on the two classes. However, in these two tasks, the PFC ensembles still lie "further above" their median attainment surface than the Baseline ensembles, which lie "closer to" their attainment surface. This suggests that while the PFC and Baseline ensembles

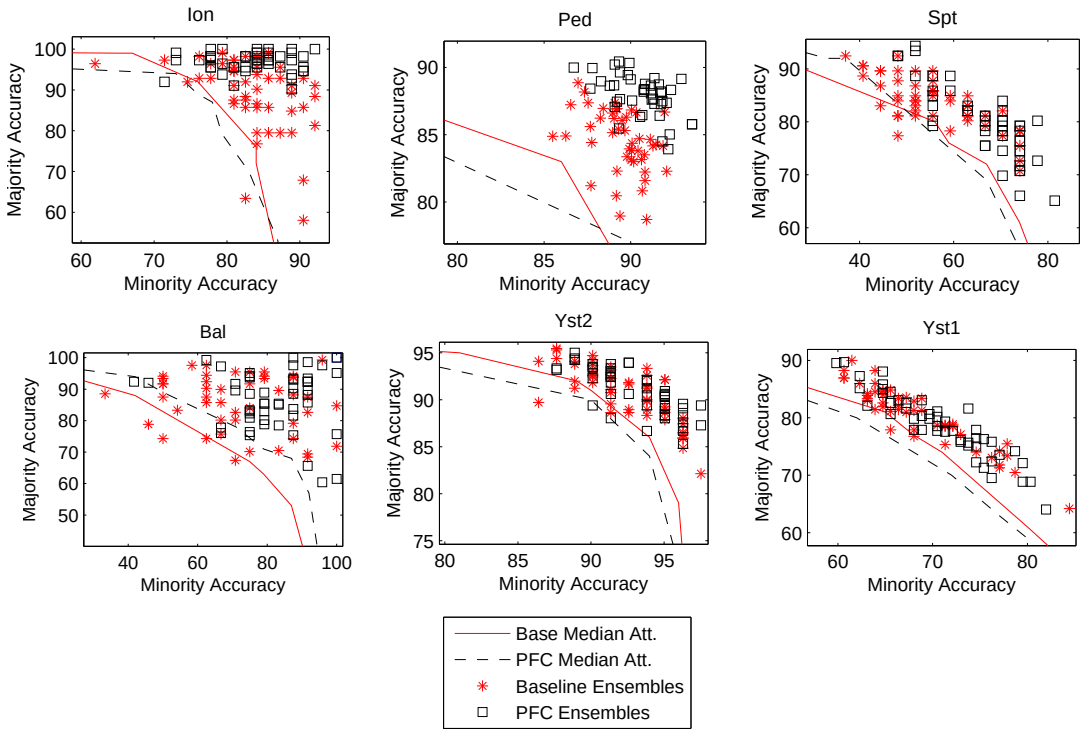


Figure 6.4: MOGP ensemble accuracies (on a run-by-run basis) and median attainment surface ("average" front performance) for Baseline and PFC approaches with off-EEL for 50 runs.

performance may be similar to one another in these tasks, the PFC individuals are more diverse (than the Baseline individuals) as the PFC ensembles show better cooperation relative to their median attainment surface, than the Baseline ensembles relative to their median attainment surface.

As previously discussed (in Section 6.4.2), the Baseline median attainment surface dominates the PFC median attainment surface in most tasks (Bal and Spt and the two exceptions), due to the selection bias in fitness for these MOGP approaches. Solutions with high accuracy rates on the two classes (but which are potentially less diverse) are favoured in the Baseline MOGP, while solutions that are both accurate *and* diverse are favoured in MOGP with PFC. In Bal, the PFC median attainment surface dominates the Baseline median attainment surface, suggesting that PFC in the fitness function finds solutions that are also more accurate on the two classes than the Baseline MOGP. This may be why one particular run of the PFC ensembles achieves 100% on both classes in Bal (on the test set).

In Spt, the median attainment surfaces for PFC and the Baseline dominate each other in different regions of the objective-space.

6.6 Counting Ensemble "Wins"

This section tries to quantify the run-by-run analysis from the previous section (e.g. Figure 6.4) to further investigate the differences between the MOGP approaches and the ensemble voting/selection strategies. By comparing the classification result of each MOGP approach on a run-by-run basis over the 50 runs, this section investigates which MOGP approach (Baseline, NCL or PFC) and ensemble combination strategy (PF-Wvote or off-EEL) produces better overall performances across all MOGP runs and tasks. These questions are difficult to answer using only the average ensemble performances reported in Tables 6.4 and 6.5 (in the previous section). For clarity, note that in the experimental setup, a single run of the three MOGP approaches (Baseline, NCL and PFC) all use the same random starting seed and initial population.

A run-by-run analysis of the MOGP approaches has a two-dimensional aspect, as both the majority and the minority class accuracies of the ensembles must be taken into account when determining if one approach is better than another (compared to a "single figure" measure such as the overall accuracy). The Pareto dominance relation is used to summarise a single classification result between any two MOGP approaches on a run-by-run basis basis. This allows us to determine if one approach is better than the other, in terms of a "win", "lose" or "draw" outcome. For two MOGP approaches, gp_1 and gp_2, the three outcomes for a particular run can be defined as follows.

- Win for gp_1 if gp_1 dominates gp_2 (loss for gp_2).
- Win for gp_2 if gp_1 is dominated by gp_2 (loss for gp_1).
- Draw otherwise.

These three outcomes (i.e. win, lose or draw) represent a *multinomial* distribution over N independent runs. This means that the proportion of wins for one approach (call this p_1), the proportion of wins for the other approach (call this p_2), and the proportion of draws between them (call this p_3), sums to 1 over N runs. In a multinomial distribution, a 95% confidence interval of the *difference* in the proportion of wins between two approaches ($p_1 - p_2$) can be calculated for a particular task. This can be used to determine if one MOGP ensemble *significantly dominates* another over all runs. The 95% confidence interval of this difference between any two MOGP approaches is calculated using Eq. (6.4), where $var(p_i)$ is the variance of p_i for the i_{th} MOGP approach in $N = 50$ runs.

$$(p_1 - p_2) \pm 1.96\sqrt{var(p_1 - p_2)} \tag{6.4}$$

where

$$\begin{aligned} var(p_1 - p_2) &= var(p_1) + var(p_2) - \\ &\quad -(var(p_1 + p_2) - var(p_1) - var(p_2)) \\ &= 2var(p_1) + 2var(p_2) - var(p_1 + p_2) \end{aligned}$$

$$\begin{aligned} var(p_i) &= \frac{p_i(1 - p_i)}{N} \\ var(p_1 + p_2) &= \frac{(p_1 + p_2)(1 - p_1 - p_2)}{N} \end{aligned}$$

In the subsequent sections, the ensemble wins between the MOGP approaches are used to explore two main aspects of the ensemble behaviour. The first compares which of the three MOGP approaches (Baseline, NCL or PFC) statistically dominates each other on the tasks, and which achieves more overall ensemble wins over all runs and tasks. The second compares which ensemble combination strategy between PF-Wvote and off-EEL statistically dominates each other on the tasks, and which achieves more overall ensemble wins over all runs and tasks.

6.6.1 Wins for Diversity Measure in MOGP

To compare which of the three MOGP approaches (Baseline, NCL or PFC) statistically dominates each other on the tasks, Table 6.5 shows the pairs of ensemble *wins* between two MOGP approaches, where each is compared with every other (on a run-by-run basis) for 50 runs. Each win-pair in Table 6.5 corresponds to the three pairwise comparisons between the Baseline, NCL and PFC approaches using a particular ensemble combination strategy. Two ensemble combination/selection strategies are examined: fitness-weighted majority vote (PF-Wvote) and off-EEL.

For example, the first win-pair in Table 6.5 for Ion with PF-Wvote is "8,8". This means that when the Baseline MOGP is compared to the NCL approach on a run-by-run basis over 50 runs, both the Baseline and NCL score 8 wins each (i.e. each approach dominates the other exactly 8 times). In the remaining 34 runs, these approaches are non-dominated with respect to each other (i.e. 34 "draws"). Similarly, the next win-pair in Table 6.5 for Ion with PF-Wvote is "7,14".

Table 6.5: "Win" pairs between two MOGP approaches (on a run-by-run basis) over 50 runs for two ensemble combination strategies (PF-Wvote and off-EEL). Total wins (and draws) is the sum of wins (and draws) over all runs and tasks (50 runs $\times$ 6 tasks). Bold results indicate a statistically significantly better ensemble performance (95% significance level).

Task	PF-Wvote Strategy			Off-EEL Ensemble Selection		
	Baseline *vs* NCL	Baseline *vs* PFC	NCL *vs* PFC	Baseline *vs* NCL	Baseline *vs* PFC	NCL *vs* PFC
Ion	8,8	7,14	8,**19**	11,6	6,**22**	3,**20**
Spt	3,5	1,5	1,3	10,7	3,6	1,**9**
Ped	11,3	1,**25**	0,**19**	15,7	0,**20**	0,**26**
Yst_1	7,8	4,2	5,7	5,3	6,3	1,3
Yst_2	7 / 5	5,2	3,**12**	5,1	8,7	0,**4**
Bal	10,7	11,14	10,9	14,8	12,16	4,**21**
Wins	46,36	29,62	27,69	60,32	35,74	9,83
Draws	218	209	204	208	191	208

This means that when the Baseline is compared to PFC on a run-by-run basis over 50 runs, PFC wins against (dominates) the Baseline in 14 runs, while the Baseline wins against PFC in 7 runs. These approaches are non-dominated in the remaining 29 runs (i.e. 29 "draws").

The last two rows in Table 6.5 reports the total number of wins for each pair, and the total number of draws (non-dominated performances), summed over *all* runs and tasks. The total number of wins and draws in each column in Table 6.5 sums to 300 (50 runs $\times$ 6 tasks).

The results of the 95% confidence intervals of the wins between two MOGP approaches are also shown in Table 6.5. The statistically significantly better ensemble performance, denoted by a higher number of wins, is highlighted in bold. It is important to note that as three separate confidence intervals are constructed for each of the three pairwise comparisons, the statistical relationship only applies to a specific pair.

PFC better than NCL over all tasks

According to Table 6.5, the total number of wins (over all tasks) when NCL is compared to PFC is higher in PFC for both ensemble combination strategies. For the PF-Wvote strategy, PFC has 69 total wins while NCL only has 27; for the off-EEL strategy, PFC has 83 total wins while NCL only has 9. This means that in both ensemble combination strategies, the PFC ensembles dominate the NCL

ensembles *more often* than the opposite case over all runs and tasks. The very large difference in total wins between PFC and NCL for off-EEL is due to the PFC ensembles achieving statistically significantly better performances than NCL in nearly all tasks. As the PFC approach, particular with off-EEL, produces better ensemble results than NCL in these tasks, this suggests that the PFC ensembles may be more diverse that the NCL ensembles.

A similar conclusion can be drawn when the PFC ensembles are compared to the Baseline MOGP for the two ensemble combination strategies. PFC always scores more total wins (over all tasks) than the Baseline when these two approaches are compared against each for both strategies. For the PF-Wvote strategy, PFC has 62 total wins while the Baseline only has 29; for the off-EEL strategy, PFC has 74 total wins while the Baseline only has 35. As discussed in the previous section (and shown in Figure 6.4), the better PFC performances are due to better cooperation between members than the Baseline MOGP on these tasks.

In contrast, NCL scores fewer total wins (over all tasks) than the Baseline MOGP for the two ensemble combination strategies. For the PF-Wvote strategy, NCL has 36 total wins while the Baseline has 46; for the off-EEL strategy, NCL has 32 total wins while the Baseline has 60. In fact, Table 6.5 shows that for both PF-Wvote and off-EEL strategies, there is no statistically significant difference in wins between the Baseline and NCL in any of the six tasks. This shows that both NCL and Baseline ensembles achieve very similar performances on the tasks.

Further Discussions

The above results show that the PFC ensembles performed better than NCL on these tasks, particularly for the off-EEL selection algorithm. This may be due to two reasons. The first is the different ways NCL and PFC create "spread" (or diversity) in the population (see [126][125] for theoretical insights into how NCL creates spread in a population). The second is the different ways in which NCL and PFC are used in MOGP: NCL is calculated after the population is ranked on the objectives (using SPEA2), while PFC is calculated before Pareto ranking is done.

Developing an approach which incorporates NCL into the objective performance before Pareto ranking is done (similar to PFC) may improve ensemble performances for NCL. Likewise, new diversity measures (such as the root quartic NCL proposed in [126][125]) may also improve ensemble performances for NCL, due to different ways in which these measures create "spread" in

Table 6.6: "Win" pairs between the two ensemble combination strategies (PF-Wvote and off-EEL) for the MOGP approaches (on a run-by-run basis) over 50 runs. Bold results indicate a statistically significantly better ensemble performance (95% significance level) over 50 runs.

Task	PF-Wvote vs off-EEL		
	Baseline	NCL	PFC
Ion	4,9	4,**15**	1,**25**
Spt	2,2	1,1	10,3
Ped	0,3	1,3	0,**5**
Yst_1	1,**12**	2,4	3,**10**
Yst_2	0,0	1,4	1,6
Bal	3,5	6,3	3,8
Wins	10,31	15,30	18,57
Draws	259	255	225

a population. However, this is outside the scope of this work which focuses on evaluating the use of the traditional NCL and PFC measures in the fitness function in MOGP for diversity (and not adaptations of these measures such as [126][125]). This will be an interesting exercise for future work.

6.6.2 Wins for Ensemble Combination Strategies in MOGP

To compare which of the two ensemble combination strategies statistically dominates each other on the tasks, Table 6.6 shows the pairs of ensemble wins when PF-Wvote is compared to off-EEL (on a run-by-run basis) over 50 runs for the three MOGP approaches (Baseline, NCL and PFC approaches). The statistically significantly better ensemble strategy, denoted by a higher number of wins, is highlighted in bold. For example, the first win-pair in Table 6.6 for Ion is "4,9". This means that when PF-Wvote is compared to off-EEL for the Baseline MOGP, PF-Wvote wins against (dominates) off-EEL in 4 runs, while off-EEL dominates PF-Wvote in 9 runs. In the remaining 37 runs, these strategies are non-dominated to each other.

The last two rows in Table 6.5 reports the total number of wins for each pair, and the total number of draws (non-dominated performances), summed over all 50 runs and for the six tasks (300 total runs).

Table 6.6 shows that the total number of draws (over all tasks) between these two ensemble combination strategies (for the three MOGP approaches) is higher than the total number of draws in all columns in Table 6.5 (from the previous section). This shows that the two ensemble combination strategies are

very closely matched (in terms of performance) as both have relatively few wins against each other on the tasks, particularly for the Baseline and NCL approaches, which have a higher total number of draws (over all tasks) than PFC. For these two MOGP approaches, both ensemble combination strategies are typically non-dominated in nearly all tasks. The only exceptions are Yst_1 for the Baseline, and Ion for NCL (in these cases off-EEL significantly dominates PF-Wvote).

However, the total number of wins (over all tasks) for off-EEL is higher in all three MOGP approaches than PF-Wvote. In PFC, off-EEL significantly dominates PF-Wvote in three tasks (Ion, Ped and Yst_1) and, as a result, PFC with off-EEL shows more total wins (over all tasks) than PFC with PF-Wvote. As the total wins for PFC with off-EEL is substantially larger than both the Baseline and NCL with off-EEL, this suggests that the PFC approach is particularly successful with off-EEL. As the previous section (in Table 6.5) suggests that the PFC ensembles are more diverse than the Baseline and NCL ensembles on these tasks (due to better diversity between individuals), these results imply that the more diverse the ensembles, the more effective the off-EEL algorithm in improving ensemble performance (compared to the PF-Wvote strategy).

6.7 Comparison with SGP, NB and SVM

This section compares the MOGP ensemble classification results to canonical single-objective GP (SGP), Naive Bayes (NB) and Support Vector Machine (SVM) on the tasks. An outline of the experimental setup for SGP, NB and SVM is first presented, followed by the classification results using these methods on the tasks. Note that this comparison is not the primary goal of this thesis, but we would like to have an overall indication on how well these new GP approaches can solve these classification tasks compared with the well-known/common methods (including common GP methods) in classification.

6.7.1 Experimental Setup for SGP, NB and SVM

The classification accuracy of the fittest evolved solutions using SGP with three different fitness functions (over 50 runs) is compared to the MOGP ensembles on the tasks. The three SGP fitness functions correspond to Acc, Ave and Auc. Recall from Chapter 4 that Acc uses the overall classification accuracy in fitness, Ave uses the average classification accuracy of the minority and majority class in fitness, and Auc uses the (full) area under the ROC curve (AUC) in fitness. These fitness functions are chosen to represent the SGP approach for two main reasons. Firstly,

Acc represents the traditional fitness measure in classification, and Ave and Auc are two major current approaches for cost adjustment in fitness (to account for the unbalanced classes). Secondly, these fitness functions provide a good indication of the range of SGP performances on the tasks, where Acc represents very poor SGP performances while Auc represents good SGP performances. For details on these fitness functions, please refer to Chapter 4.

It is important to reiterate that, where possible, the evolutionary parameters in MOGP and SGP are kept the same for a fair comparison between these methods. Both MOGP and SGP use a population size of 500 and a maximum number of 50 generations. Likewise, both methods restrict the maximum program depth of the evolved solutions to 8. This means that the same complexity constraints are placed on both the SGP classifiers and the MOGP Pareto front classifiers (base classifiers in the ensembles). As discussed in the previous Chapter (in Section 5.3.1), only the tournament size and mutation (and elitism) rates are different in MOGP and SGP. In MOGP, a tournament size of 2 is used and the mutation rate is 40% (as elitism is not used). In SGP, a tournament size of 7 is used, and mutation and elitism rates are 35% and 5%, respectively. These parameter values were chosen with a good reason as stated in the previous chapters.

Similar to the previous experimental results using NB and SVM (in Chapter 4), a single run for NB and SVM is generated using the WEKA package [82]. The SVM uses a sequential minimal optimisation algorithm with an RBF kernel and Gamma value of 10.

6.7.2 Classification Results

Table 6.7 shows the (average) minority and majority class accuracies for SGP with the three fitness functions over 50 runs, and a single run of NB and SVM, on the tasks. These results correspond to the accuracy rates when the SGP, NB and SVM classifiers are evaluated using *zero* as the class threshold (on the test set). For convenience, the MOGP ensemble performances using PF-Wvote (fitness-weighted majority vote) and off-EEL [76] ensemble selection strategy (from Tables 6.3 and 6.4 in previous sections) are repeated in Table 6.8 to make comparisons with Table 6.7 easier.

Table 6.8 show that all three MOGP ensembles using the PF-Wvote and off-EEL strategies achieve much more balanced (and better) results than SGP using Acc, NB and SVM in all tasks (except Ion). In those tasks with high levels of class imbalance (such as Spt, Ped, Yst_1 and Bal), these *single-predictor* methods

Table 6.7: Average accuracies ($\pm$ standard deviation) using canonical single-objective GP (SGP) on the test set with three fitness functions (Acc, Ave and Auc) over 50 SGP runs, and a single run of NB and SVM on the tasks.

Task	SGP Acc		SGP Ave ($W = 0.5$)		SGP Auc	
	Minority	Majority	Minority	Majority	Minority	Majority
Ion	73.8 ± 7.7	95.3 ± 3.9	76.6 ± 6.3	91.3 ± 6.1	81.1 ± 5.2	81.3 ± 6.5
Spt	47.4 ± 4.6	88.6 ± 2.5	56.7 ± 8.3	82.7 ± 3.6	70.2 ± 6.7	70.0 ± 5.8
Ped	43.3 ± 14.5	96.6 ± 1.6	87.7 ± 2.3	85.6 ± 2.8	86.2 ± 1.5	86.1 ± 1.6
Yst_1	40.8 ± 4.2	94.6 ± 1.4	60.2 ± 4.6	83.1 ± 3.8	73.0 ± 1.4	72.8 ± 1.5
Yst_2	64.0 ± 8.1	97.4 ± 0.6	85.9 ± 4.0	93.0 ± 2.1	86.8 ± 2.7	88.2 ± 4.1
Bal	9.0 ± 17.5	98.9 ± 1.1	85.6 ± 11.4	84.6 ± 11.7	82.8 ± 8.3	87.1 ± 11.0

Task	NB		SVM	
	Minority	Majority	Minority	Majority
Ion	63.4	88.9	87.5	99.1
Spt	66.7	83.0	37.0	94.3
Ped	83.7	81.4	53.8	92.4
Yst_1	43.4	96.4	32.8	97.4
Yst_2	66.7	98.0	58.0	97.9
Bal	0.0	100.0	0.0	100.0

Table 6.8: Average accuracies ($\pm$ standard deviation) on the test set using PF-Wvote (fitness-weighted majority vote) and off-EEL [76] ensemble selection strategy for the three MOGP approaches (50 runs). These are repeated from Tables 6.3 and 6.4.

Task	MOGP	PF-Wvote		Off-EEL	
		Minority %	Majority %	Minority %	Majority %
Ion	Baseline	82.5 ± 5.8	89.1 ± 8.3	83.7 ± 5.8	89.2 ± 8.8
	NCL	80.9 ± 5.9	91.4 ± 5.9	82.2 ± 5.5	89.5 ± 8.1
	PFC	79.6 ± 6.2	96.3 ± 3.7	83.9 ± 5.2	96.6 ± 2.8
Spt	Baseline	86.4 ± 13.7	59.9 ± 36.4	56.0 ± 10.1	83.6 ± 4.8
	NCL	84.1 ± 12.0	63.3 ± 32.7	53.9 ± 9.7	83.8 ± 4.8
	PFC	75.9 ± 10.4	72.6 ± 22.2	66.3 ± 8.5	79.9 ± 6.6
Ped	Baseline	89.1 ± 3.6	83.3 ± 3.1	89.5 ± 1.5	84.6 ± 2.3
	NCL	88.0 ± 3.2	83.5 ± 3.7	88.8 ± 3.0	84.4 ± 2.8
	PFC	92.5 ± 1.6	84.1 ± 3.1	90.6 ± 1.5	87.9 ± 1.5
Yst_1	Baseline	70.0 ± 4.2	77.3 ± 4.3	68.5 ± 5.5	80.4 ± 5.2
	NCL	70.6 ± 3.7	76.8 ± 4.4	64.1 ± 5.0	83.4 ± 4.0
	PFC	71.8 ± 5.3	75.4 ± 6.5	70.6 ± 5.4	78.8 ± 5.5
Yst_2	Baseline	82.8 ± 3.6	95.1 ± 1.3	92.3 ± 2.9	90.7 ± 2.9
	NCL	83.6 ± 4.7	93.7 ± 1.7	80.4 ± 7.3	94.5 ± 2.1
	PFC	89.6 ± 3.2	92.1 ± 1.9	93.1 ± 2.6	90.8 ± 2.4
Bal	Baseline	84.2 ± 12.5	71.4 ± 23.6	71.4 ± 15.9	85.6 ± 9.0
	NCL	86.9 ± 11.8	66.0 ± 29.5	62.1 ± 17.5	85.6 ± 7.3
	PFC	87.3 ± 9.3	74.1 ± 17.1	81.4 ± 12.1	86.2 ± 9.1

show biased classification results. In Bal in particular, none of these methods achieve more than 10% accuracy on the minority class (Bal has highest level of class imbalance). In Ion, SVM achieves the best results (87% and 99% on the minority and majority class, respectively). The MOGP ensembles cannot, on average, match the SVM results. However, closer examination of the PFC results with off-EEL on a run-by-run basis finds that the three *best* PFC runs score a better accuracy on both classes than SVM. These three runs achieve 88.9/99.1%, 88.9/100%, and 92.1/100% on the minority/majority class, respectively.

On average, the PFC ensembles with off-EEL dominate SGP using Auc in three tasks (Ion, Ped and Yst_2). In Bal, SGP with Auc and PFC (with off-EEL) achieve very similar results (within 1% accuracy for each class). As the model complexity of the evolved genetic program classifiers are the same in both canonical GP and MOGP, the PFC ensembles (and, to a lesser extent, the Baseline and NCL ensembles) are better than SGP on some of these tasks for two main reasons. Firstly, this is due to more support for two learning objectives (minority and majority accuracy) in MOGP. In other words, in SGP with Auc, each classifier tried to achieve the best trade-off between the two objectives *individually* (by maximising their AUC); whereas in MOGP, each classifier is one point (of many) along the Pareto front. Secondly, combining these Pareto front classifiers into an ensemble where individuals work together (by voting) further improves performances, as the ensemble performs *at least as well* as its individual members.

When a diversity objective such as PFC is introduced in the fitness function during evolution, the ensemble performs better than most of its individual members, as this performance *dominates* the performance of the individual members. This is due to the cooperation between the individual members, as discussed in Section 6.5.3 (and shown in Figures 6.4).

Even in those tasks where the MOGP ensemble results are similar to, or dominated by, SGP using Ave or Auc (such as Yst_1), the MOGP ensembles still perform better than most of its individual members. In these tasks a likely reason for the not very good MOGP ensemble performance is the relatively poor performance of the Pareto-approximated fronts compared to the frontier generated by SGP using fitness function Ave. Recall (from the previous chapter) that when the median attainment surface for MOGP with SPEA2 is compared to the SGP frontier with Ave (in Section 5.3.4), the MOGP front is *dominated by* the SGP front in some tasks (such as Ped and Bal). This shows that very high accuracy cannot be expected from the ensemble if the individual ensemble members (i.e. Pareto front solutions) themselves are not sufficiently accurate (e.g. relative to the

```
if<0
 (*                                                                          \\arg1
    (*
      (- -0.69 (- (% (+ f2 0.47)(% -0.66 f1)) f2))
      (+ (+ f2 0.47) (+  (* (if<0 -0.60 f3 0.88) f2) 0.60)))
    (+ (if<0 f0 f2 f0) (% -0.34 0.96)))
 (* (* (- f1 0.29) (* 0.58 -0.32)) (* (- -0.96 -0.51)(% f2 0.20)))          \\arg2
 (%                                                                         \\arg3
   (+ 0.58 (if<0 (- (* -0.29 (* 0.08 f3)) (+ 0.64 0.05)) (* -0.55 (% -0.01 f1)) -0.41))
   (- 0.96 (- (* f3 f2) (* f0 f1)))))
```

Figure 6.5: Evolved MOGP classifier with 100% accuracy on training and test set for Bal.

SGP frontier). This highlights the importance of developing a good underlying multi-objective algorithm to trace out an accurate and diverse set of ensemble members across all the tasks.

6.8 Evolved MOGP Programs

This section examines four evolved MOGP classifiers on the Bal task since the high level of class imbalance in Bal makes this a difficult classification problem for canonical GP, NB and SVM to solve (as demonstrated by the biased results for these methods). Four evolved MOGP classifiers are analysed below. These correspond to the evolved MOGP program which achieved 100% accuracy on both the training and test sets for Bal (as discussed in Section 6.4.2), and three other programs which cooperate well together when combined in the ensemble.

6.8.1 Evolved Program with Perfect Accuracy

The evolved program which achieved 100% accuracy on both the training and test sets for Bal (using the PFC-based MOGP approach) is shown in Figure 6.5. Similar to the previous program analysis of SGP classifiers in Chapter 4 (in Section 4.7), $f0$—$f3$ in this program correspond to the four input features in Bal. For convenience, this program has been indented for easier interpretability.

Interestingly, this program shares the same overall structure as the evolved program from Chapter 4 (shown in Figure 4.6 on page 107) which also performs very well on Bal (very high AUC). Both programs contain three distinct parts which form the input arguments to the outer-most `if` function in the root node of the tree. The previous analysis from Chapter 4 suggests that this high-level conditional logic operator may be a successful strategy to classify the data inputs.

This hypothesis is affirmed by the program in Figure 6.5 as this classifier has even better performance than the program from Chapter 4 (Figure 4.6).

The program in Figure 6.5 is also smaller that the SGP program from Chapter 4 (Figure 4.6). As a result, the first argument of the (outer-most) `if` node in Figure 6.5 is much smaller (and simpler) than in the SGP program. These two factors may have allowed the MOGP solution to better generalise on the training and test sets for Bal.

6.8.2 Good Programs for the Ensemble

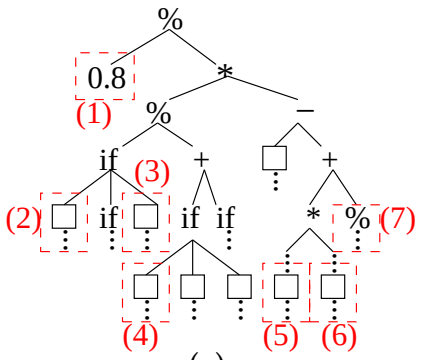
This analysis examines three evolved programs using the PFC-based MOGP approach which cooperate well together when combined in the ensemble. The first program, shown in Figure 6.6, represents a non-dominated MOGP solution which achieves 83% and 91% accuracy on the minority and majority class, respectively, on the test set. The second program (another non-dominated solution from the same run), scores 90% and 80% accuracy on the minority and majority class, respectively, and is identical to Figure 6.6 except for seven major differences (these are underlined in Figure 6.6). These seven differences, shown in Figure 6.7(b), are responsible for the variation in performance between the two solutions. The overall tree structure shared by both non-dominated solutions are shown in Figure 6.7(a), where the (red dashed) squares around a particular sub-tree show where in the tree these seven differences occur. The symbol □ in Figure 6.7(a) represents a sub-tree that is omitted. Note that Figure 6.6 is not indented in a similar way to Figure 6.5 above, as the overall structure of this program can already be seen in Figure 6.7(a).

Figure 6.7(a) shows that these two non-dominated solutions both use a series of nested `if` functions deep within the tree, in combination with the other functions ($+$,$-$,$\times$ and %). As these nested `if` functions occur in the same positions in both trees, the variation in performance for the two solutions must be due to other differences between these trees (such as Figure 6.7(b)).

The third non-dominated solution from the same run that is analysed is shown in Figure 6.8. This solution achieves lower accuracies than the previous solutions, 72% and 67% on the minority and majority class, respectively, and is also noticeably smaller. All three MOGP program trees are "unbalanced", i.e., they all have exactly one leaf node on the left side of the tree compared a larger subtree on the right side. However, the right side of the tree in this program is completely different from the right side of these trees in the previous two

```
(% 0.8 (* (% (if<0 (* (- (% 0.6 f0) (% f0 f3)) (if<0 (- 0.5 f2) f2 (- f1 -0.7))) (if<0
(if<0 (% -0.7 f3) (+ f1 f3) (+ f0 f2)) (% (% f2 0.6) (- f1 f2)) (+ (- f2 f1) (- f3 f0)))
(* (- f2 (* f0 -0.5)) (% (+ -0.6 f2) (+ -0.1 f2)))) (+ (if<0 (% (- f2 f1) (% 0.5 f0))
(- (f3 (- f3 f1)) (+ (if<0 0.2 f1 0.2) (if<0 f1 f3 f3))) (if<0 (+ (- -0.6 0.2) (* 0.2
-0.1)) (* (* f1 0.4) f0) (- (% f0 f2) -0.8)))) (- (% (if<0 (* (* -0.4 f2) (- 0.5 f2)) (*
1.0 (* -0.1 f1)) (if<0 f3 (% f0 0.1) 0.9)) (% (+ (% f0 -1.0) 0.8) (- (+ -0.7 -0.1) (* f3
f2)))) (+ (* (- (+ f1 f0) (if<0 f3 f2 f0)) (% -0.1 (- f2 -0.5))) (% f1 0.7)))))
```

Figure 6.6: An evolved MOGP ensemble program for Bal.



(a)

1. `f0`
2. `0.2`
3. `(* (% 0.3 f1) 1.3)`
4. `(if<0 f0 0.9 f0)`
5. `0.7`
6. `(if<0 f0 -0.1 f3)`
7. `(if<0 (% 0.5 (+ -0.7 f2)) -0.8 -1.7)`

(b)

Figure 6.7: (a) Overall structure of two GP trees (for Bal) where □ represents a sub-tree (omitted) and the dashed rectangles (around a given sub-tree) show where in the overall structure the seven differences occur; and (b) sub-trees in the second GP tree that are different from Figure 6.6.

programs (Figure 6.7(a)). Another difference between these three programs is that Figure 6.8 only has two simple `if` functions deep within the tree. In fact, one of these `if` conditions (`if<0 0.4 f0 (- f2 f1)`) does not represent a true conditional expression since it will never branch to execute the first argument (as 0.4 is not less that 0). These factors may be why this program performs much poorer than the previous two programs.

6.8.3 Trends

Inspection of the other evolved programs from other MOGP runs (with PFC) reveals a similar pattern, i.e., good programs evolved by MOGP with PFC share a similar overall structure but this structure is different in other non-dominated solutions. A similar observation is also discussed in the previous program analysis of SGP classifiers in Chapter 4 (Section 4.7.3). However, program analysis in MOGP can be more difficult than in canonical SGP since multiple programs are evolved in a single run, and there are multiple runs to consider.

```
(-
  0.9                                                                    \\left-side
  (+                                                                     \\right-side
    (+
      (* (* (if<0 0.4 f0 (- f2 f1)) (- f1 f2) (- f1 (% (* 0.6 f3) (- -0.7 f0)))) f0)
      (- f3 (* (% (- (if<0 f2 -0.2 f1) (- f2 f0)) (- (- f0 f3) 0.03)) -0.06))))
```

Figure 6.8: A smaller evolved GP tree (for the Bal task).

Also similar to the previous analysis in Chapter 4, high-level `if` conditions (as seen in Figure 6.5) may also represent a successful strategy to achieve very good performances on the two classes, as this useful building block was discovered in both canonical SGP and MOGP approaches.

This analysis also shows that different groups of programs will have different building blocks. For example, the common nested `if` functions in Figure 6.7(a) may constitute good building blocks as these are common in other well-performing solutions in the same run. Likewise, the solution shown by Figure 6.8 may use different building blocks which allow this particular program (and other similar-performing programs) to specialise on certain parts of the input-space. This diverse nature of evolved programs allows the ensembles to improve system performances. However, analysing the evolved programs is not the main goal of this thesis and a more detailed analysis is out of the scope of this thesis.

6.9 Summary

The goal of this chapter was to adapt the fitness function in MOGP to promote diversity between individuals and combine the evolved genetic program classifiers along the Pareto front into an ensemble where members vote on class membership. Two ensemble-diversity measures are incorporated in the fitness function in MOGP to estimate diversity between solutions *separately* for the minority and the majority classes. This accounts for the unbalanced classes; otherwise, these diversity measures risk being biased toward the majority class. These diversity measures include Negative Correlation Learning (NCL) and Pairwise Failure Crediting (PFC). The ensemble performance using these two diversity-based MOGP approaches are compared to each other, to a (Baseline) MOGP ensemble approach which uses no explicit ensemble-diversity measure in fitness, and to canonical SGP, NB and SVM on the tasks.

The second goal of this chapter was to evaluate and compare two ensemble combination and selection strategies in the MOGP ensembles, to investigate

which strategy produces the best ensemble performances on the tasks. The important conclusions from these two research goals are outlined below.

6.9.1 Ensemble Combination and Selection

This chapter shows that when the full Pareto front of solutions forms the ensemble (for a given MOGP run), the MOGP ensembles show biased classification results toward the majority class in nearly all tasks, due to the influence of biased individuals in the voting process. This occurs because more solutions with stronger majority class accuracies (than minority class accuracies) achieve a non-dominated status in the population as the evolution progresses. Two strategies are shown to successfully reduce the influence of these biased individuals in the ensembles, to improve ensemble performances on both classes. The first uses a fitness-weighted majority vote of the Pareto front (PW-Wvote), and the second uses the off-EEL (offline evolutionary ensemble learning) algorithm [76]. While off-EEL shows the best ensemble performances on the two classes for all the tasks, both strategies outperform a simple accuracy-based ensemble selection approach (called RPF-vote).

6.9.2 Ensemble Diversity in MOGP Fitness

The PFC-based MOGP, particularly with off-EEL, is found to evolve better-performing ensembles than both the NCL-based and Baseline MOGP approaches, due to better cooperation and diversity between individuals. The increased diversity in the PFC-based approach is due to the selection bias in fitness where individuals with equally high accuracy and diversity rates on the two classes are favoured; whereas in the Baseline and, to a lesser extent, the NCL approaches, individuals with high accuracies on the two classes (but which are potentially less diverse) are favoured in fitness. When the individuals evolved from the PFC-based approach are combined using off-EEL, ensembles performances improve as the more diverse the individuals, the more effective the off-EEL strategy for improving ensemble performance (compared to the other ensemble combination strategies).

However, the Baseline approach for evolving ensembles shows competitive results (relative to the PFC and NCL approaches) on some tasks when combined with the fitness-weighted majority vote strategy (PW-Wvote). This suggests that while the Baseline MOGP uses no ensemble-diversity objective in fitness, the stochastic way in which new classifiers are created in the evolution in GP (e.g.

using the genetic operators) can provide sufficient diversity between individuals to achieve good ensemble performances with PF-Wvote on some tasks.

6.9.3 Comparison with SGP, SVM and NB

Interestly, the NCL and PFC approaches are both able to find one solution with 100% accuracy on both classes on the training and test sets, on the task with the greatest level of class imbalance (Bal). The best runs of canonical SGP, NB and SVM could not accomplish this on any task. This shows that promoting better diversity between individuals in the population using NCL and PFC in fitness can also help evolve better-performing genetic program solutions on some tasks. On at least three out of six tasks, the MOGP approaches, in particular PFC, outperforms canonical SGP, NB and SVM. This is due to two important reasons in MOGP. Firstly, MOGP provides more support for the learning objectives (minority and majority class accuracy) where a *set* of Pareto front genetic program classifiers is evolved to capture the trade-off between the objectives. In contrast, this is accomplished by individual genetic program classifiers in SGP (via an ROC curve). Secondly, by combining these Pareto front classifiers into an ensemble where individuals cooperate (by voting on class membership), good performances can be achieved on the objectives as the ensembles perform better than its individual members.

6.9.4 Ensemble Optimisation

Finding the best combination of individuals (from the set of Pareto front solutions) to form the ensembles can be thought of as a separate combinatorial optimisation problem. After the initial training phase to evolve the Pareto front in MOGP, a secondary optimisation/search process can be invoked to find the best combination of individuals which produce the best ensemble results on the tasks. This ensemble optimisation approach is advantageous over off-EEL as off-EEL selects individuals for the ensembles based on a linear ordering of their fitness values, and does not consider diversity between different subsets of individuals. The next chapter develops an new evolutionary-based approach which treats ensemble selection as a combinatorial optimisation problem, to find small but highly diverse subsets individuals that cooperate well together in the ensemble. This approach evolves composite solutions where multiple Pareto front individuals are combined into a single (composite) genetic program to represent the (optimised) ensembles.

Chapter 7

Composite Solutions for Ensemble Selection

This chapter is organised as follows. The first section outlines the chapter introduction and goals. The second section discusses the GP approach to composite solutions. The third section outlines the experimental setup for evolving composite solutions. The fourth section presents the experimental results on the tasks. The fifth section provides a summary of this chapter.

7.1 Introduction

The experimental results in the previous chapter show that combining the evolved Pareto front classifiers into an ensemble whose members vote on the class label can produce good generalisation on unseen instances from both the minority and majority class on the unbalanced data sets. This is because the ensembles perform better than its individual members on the tasks, particularly when an ensemble diversity measure such as pairwise failure crediting (PFC) [36] is used in the fitness function to encourage diversity between individuals. Diversity is important as it ensures that the individual members make different errors on the same inputs; otherwise, the ensemble members risk misclassifying all the same inputs together. However, the previous chapter shows that when the full set of Pareto front solutions is used in the voting process, the ensemble can be influenced by biased Pareto front solutions and thus, exhibit strong performances on the majority class but weak performances on the minority class. Two useful strategies to limit the influence of biased Pareto front solutions in the ensembles are evaluated to improve ensemble performances on both classes. These include a fitness-weighted majority vote strategy (called PF-Wvote) and a post-training

ensemble selection algorithm, off-EEL (offline evolutionary ensemble learning) [76]. Off-EEL is found to be particularly effective on the tasks, showing the best ensemble performances on the two classes.

7.1.1 Ensemble Optimisation

These strategies try to address a difficult problem in ensemble learning, that is, how to choose good individuals from the pool of base classifiers to form the ensembles. As mentioned earlier, the PF-Wvote and off-EEL strategies (from the previous chapter) use the fitness values of the individuals (on the training set) as the criterion for ensemble selection. While these strategies represent a useful starting point for choosing good individuals for the final ensembles, a major limitation of fitness-based ensemble selection is that selection is based on a *linear ordering* of the base classifiers by their fitness values. Recent work [29][180] has shown that different combinations of individuals can show better diversity and cooperation in the ensembles compared to a linear ordering of individuals. This is because a linear ordering of individuals (by fitness) does not guarantee that a subset of the fittest N individuals will necessarily show very good diversity (or cooperation) relative to each other.

This chapter develops a new GP-based approach to ensemble selection which treats ensemble selection as a combinatorial optimisation problem, to quickly find highly diverse combinations (or subsets) of Pareto front solutions which cooperate well together in the ensemble.

Previous work on ensemble selection from the literature typically assign a weight value to each individual in the pool of base classifiers, where weights over a certain threshold mean that a particular individual is included in the final ensemble [37][180][29][39]. These approaches typically learn this weight vector in two ways. The first is to co-evolve the base classifiers and ensemble in parallel [29][39]. While these approaches show good results on some tasks, some research suggests that this co-evolutionary learning approach can be prone to noise due to the iterative and cooperative way in which the ensembles are constructed [76][180]. For example, poor base classifiers in the early stages of the evolution can also bias the way the ensemble is optimised in the early stages of the evolution.

The second method invokes a secondary training phase to optimise the ensemble weights (typically via a genetic algorithm), after the initial training phase to learn the base classifiers [37][180]. This method is typically favoured

over co-evolutionary approaches as it decouples (or separates) the ensemble optimisation process and the initial training phase to learn the base classifiers, allowing researchers to focus on one aspect at a time. Some approaches also use an extra validation set for the secondary training phase [37][180]. However, a major limitation of weight-based approaches for ensemble selection is that fine-tuning the individual weight values can be difficult and time consuming, particularly when the pool of base classifiers is large (i.e. there are many weights to configure) [180]. Fine-tuning this weight vector must account for the different relationships (or correlations) between individuals where good individuals must be assigned high weight values, while poor-performing (or non-contributing) individuals must also be assigned low weight values; otherwise, the weight vector risks also including non-contributing members in the ensembles. Even in simpler bit-string representations (where each bit specifies whether a member is included or not in the ensemble), the (optimised) ensemble may not necessarily be much smaller than the original ensemble, unless sparcity of the weight-vector or bit-string is explicitly encouraged in the optimisation process [39].

7.1.2 Composite Genetic Program Solutions

To address these limitations, this chapter develops an ensemble selection approach using GP where the optimised ensemble is represented as a *composite solution* of base classifiers (i.e. Pareto front solutions). A composite solution is a single genetic program comprising of a subset of diverse Pareto front solutions which cooperate well together in the ensemble. This genetic program representation for combining Pareto front individuals into composite solutions has two advantages over traditional weight-based and bit-string ensemble selection approaches. Firstly, by limiting the sizes of the composite solutions in the evolution to small GP trees, small but highly diverse subsets of Pareto front individuals are selected in the (pruned) ensembles, due to selection pressure for the limited positions within the composite solutions. Secondly, configuring different function sets for the composite solutions allows the outputs of the individual solutions within a composite tree to be manipulated in different ways to control what the composite tree computes and thus, the output of the ensemble.

This chapter develops two types of composite solutions to represent the (optimised) ensembles. The first uses composite voting solutions (CSVote) to represent traditional voting-based ensembles. The second uses logical operates to combine and manipulate the outputs of the individuals within the composite

solutions. These composite logical solutions (CSLogic) allow the (optimised) ensembles more "decision making" abilities when classifying the input instances compared to the traditional voting-based ensembles (CSVote). The performances of the evolved composite solutions are also compared to the off-EEL algorithm [76] for ensemble selection on the tasks.

7.1.3 Chapter Goals

The main goal of this chapter is to develop a GP approach to ensemble selection using composite solutions to find diverse combinations (or subsets) of Pareto front individuals that cooperate well together in the ensemble. Two sub-goals are investigated. The first sub-goal is to compare which of the two types of composite solutions (CSVote and CSLogic) shows better generalisation on the tasks. The second sub-goal is to investigate whether the same training data is sufficient to learn/evolve both the base classifiers (in MOGP) and composite solutions, or whether an additional validation set (to evolve the composite solutions) can improve ensemble performances on the tasks.

7.2 Composite Solutions

This section first discusses why ensemble selection should be treated as a combinatorial optimisation problem, and then outlines the new approach using composite solutions for ensemble selection. The latter part includes an overview of the process for evolving a composite solution for ensemble selection, and the structure of the composite solutions in terms of their terminal and function sets.

7.2.1 Ensemble Selection as a Combinatorial Optimisation Problem

As discussed, the major limitation of fitness-based ensemble selection approaches is the linear way in which the ensembles are constructed using the fitness of the base classifiers. To illustrate this point using an example, consider the off-EEL algorithm [76] for ensemble selection (from the previous chapter). This algorithm first sorts the Pareto front solutions according to their fitness values on the training set, to establish an ordering of individuals based on their accuracy and diversity on the two classes. Assuming that there are T Pareto front solutions, this algorithm constructs T intermediate ensembles by iteratively copying each

individual from the ordered Pareto front into the ensemble. At each step, the intermediate ensemble (which contains one more individual than the ensemble from previous step) is evaluated on the training set. Once all Pareto front solutions are processed, the ensemble with the highest accuracy is taken as the final (optimised) ensemble.

This linear ordering of Pareto front solutions by fitness does not guarantee that the fittest N solutions (in this ordered list) are diverse relative to *each other*. In the PFC-based MOGP approach, an individual's fitness represents its accuracy and diversity relative to *all* other solutions in the population (as the PFC measure is a population-based diversity estimate). This means that each individual's diversity estimate is relative to every other in the population, dominated and non-dominated solutions alike. To establish which Pareto front solutions have the best diversity relative to other Pareto front solutions, the PFC measure must be re-applied to different *subsets* of solutions.

As the space of all possible subsets of solutions on a given Pareto front is large ($2^T - 1$ subsets[1] for a Pareto front of T solutions), finding highly diverse subsets of individuals that cooperate very well together is a difficult combinatorial problem. To describe this idea more clearly, let $X = \{p_1, p_2, p_3, ..., p_T\}$ be a set of T non-dominated individuals in the population, and let the function $div(Y)$ calculate the diversity (i.e. overlap of common errors) on the training set between only those individuals in subset $Y \subseteq X$. To find the subset Y with the best diversity on the training set (to represent the pruned ensemble), $div(Y)$ must be evaluated for all possible subsets of Y, e.g., $\{p_1\}$, $\{p_1, p_2\}$,$\{p_1, p_2, p_3\}$, $\{p_1, p_2, p_4\}$, etc. An exhaustive search to explore all possible subsets of Y is impractical as each $div(Y)$ estimate uses one pass through the training set. This represents a computationally expensive and time-consuming task, particularly for large ensembles and data sets.

This chapter addresses this issue by developing a GP-based evolutionary search to quickly explore different combinations of highly cooperative subsets of Pareto front solutions to form the ensembles. This evolutionary search takes, as input, the evolved set of Pareto front classifiers returned from a MOGP run, and evolves a composite solution which represents an amalgamation of highly diverse and accurate individuals.

[1]This number includes single-member ensembles, i.e., ensembles using a winner-takes-all voting strategy (where the output of the ensemble corresponds to the output of the single member). Naturally this number will be smaller if a minimum cardinality of three is assumed for the ensembles (three members mean *no draws* can occur in the voting process.

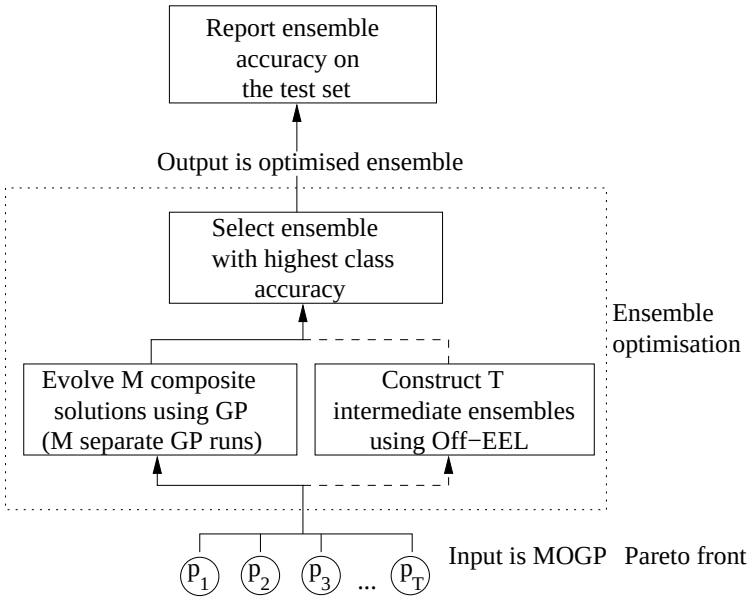
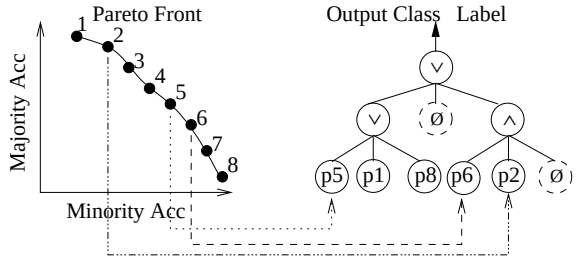


Figure 7.1: Overview of the process for ensemble selection using composite solutions and off-EEL [76] for a given set of base classifiers (evolved Pareto front from a MOGP run).

7.2.2 Composite Trees for Ensemble Selection

An overview of the process for combining Pareto front solutions into composite solutions for ensemble selection is shown in Figure 7.1 (for a given MOGP run). Figure 7.1 also shows how off-EEL [76] is used for ensemble selection, to highlight the main difference between these two approaches (as both methods are used in the experimental results). In Figure 7.1, the composite solutions and off-EEL take, as input, an evolved set of Pareto front classifiers from a given MOGP run. The evolutionary search to evolve composite solutions is repeated M times, each with a different random starting seed. This means that M separate GP runs are executed for the same input set of Pareto front solutions (M is set to 30 in the experiments), where each GP run returns a single evolved composite solution. After M GP runs there will be a total of M evolved composite solutions, i.e., M (optimised) ensembles. The composite solution with the highest (average) accuracy on the minority and majority class (on the training set) is taken as the final optimised ensemble (for that particular MOGP run). The optimised ensemble is then evaluated on the test set.

This overall process is similar for off-EEL except for two major differences. Firstly, off-EEL constructs T intermediate ensembles where T is the number of solutions in the given Pareto front. This means that T is variable and depends



(a) Pareto front (b) Composite solution

Figure 7.2: Combining a subset of Pareto front solutions (from a given MOGP run) into a single composite solution.

on the current input set (i.e. Pareto front from a given MOGP run), whereas M for the composite solutions remains fixed in all GP experiments. Secondly, off-EEL constructs each intermediate ensemble in a deterministic process (by iteratively copying an individual from the Pareto front into the ensemble until all individuals are processed). In contrast, each composite solution is *evolved* using GP, which is a stochastic process.

Although the composite solutions are described in the context of the MOGP approach in Figure 7.1, they are not restricted to genetic program classifiers and can also be used in conjunction with any underlying ensemble learning algorithm.

7.2.3 Structure of Composite Solutions

A tree-based structure is used to represent the composite solutions. This structure is decomposed into the terminal and function sets.

Terminal Set

The terminal set used to represent composite solution trees is

$$\{\emptyset, p_1, p_2, ..., p_T\}$$

where the terminal symbol p_i represents a link to the ith base classifier from the set of T Pareto front solutions (from a particular MOGP run) as shown in Figure 7.2. Figure 7.2(a) represents the evolved Pareto front (base classifiers) returned from an MOGP run (along the minority and majority class axis), while Figure 7.2(b) shows a composite solution which combines together many different base classifiers. In this composite solution, the leaf-nodes link to the corresponding base classifiers on the Pareto front. The same base classifier can

Base classifier	p_1	p_2	p_5	p_6	p_8
Raw output	1.6	-12.3	-0.5	0.5	1.0
Class Label	`1`	`0`	`0`	`1`	`1`

Figure 7.3: Raw (real-valued) output values and predicted class labels for five Pareto front solutions p_i (when evaluated on a given input). Raw outputs are mapped to class labels using zero as the class threshold.

also be represented by different leaf-nodes in the composite solutions. In other words, two or more distinct leaf-nodes in a composite solution can link to the same Pareto front solution. This particular scenario is discussed in more detail in the next section.

The terminal set symbol $\emptyset$ represents a $null$-valued terminal, used as a "blank" input argument to a particular function node. Allowing $null$-valued terminals in the composite solutions varies the number of base classifiers within any given composite solution, rather than insisting that every leaf-node in a composite solution maps to a base classifier. For example, the composite solution in Figure 7.2 uses five base classifiers (to represent the optimised ensemble) whereas there are exactly seven leaf-nodes; the remaining two leaf-nodes are $null$-valued terminals.

The meaning of the two function nodes in Figure 7.2(b) ($\wedge$ and $\vee$) is discussed in the subsequent sections.

When a composite solution is executed (i.e. evaluated on a given input instance), the ith base classifier representing terminal node p_i is first executed, and the *predicted class label* of this base classifier is taken as the return value of the terminal node. As there are exactly two classes in these data sets, binary values `0` or `1` are used to represent the two predicted class labels. As the raw output of a base classifier (when evaluated on a given input instance) is a real number, this number is mapped to the two class labels using zero as the class threshold, i.e., `1` (minority class) if the base classifier's raw output is zero or positive, or `0` (majority class) otherwise.

For example, let Figure 7.3 show the raw (real-valued) outputs when the five base classifiers (p_1, p_2, p_5, p_6 and p_8) from Figure 7.2 are evaluated on a given input instance from the minority class. Figure 7.3 also shows the predicted class labels for these base classifiers, which forms the leaf node values of the composite solution in Figure 7.2 (when evaluated on this input instance).

The terminal nodes in the composite solutions use binary values (and not the real-valued raw outputs of the base classifiers) so that the composite solutions

act on the predicted class labels of the base classifiers. This rationale is made clearer in the next section when the function sets for the composite solutions are described.

7.2.4 Functions in Composite Trees

The structure (discussed earlier) to represent the composite solutions is chosen for two important reasons. Firstly, the leaf-nodes in the terminal set provides a mechanism to link together multiple base classifiers into a single composite solution. Secondly, the function set (discussed below) provides a flexible mechanism to control how the outputs of the base classifiers are processed within a composite solution and thus, how the final output of the composite solution is determined. By configuring different types of function nodes, the outputs of the base classifiers within a composite solution can be manipulated to change what the composite solution computes.

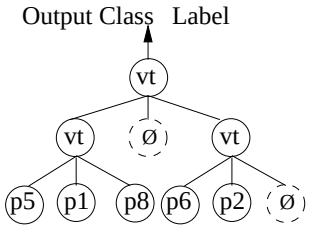
To address the second goal of this chapter, two types of function sets are compared to combine the outputs of the base classifiers within the composite solutions. The first function set uses a majority-vote of each base classifier within a given composite solution. Using this function set, the composite voting solutions (CSVote) represent subsets of highly diverse base classifiers that cooperate well together when combined in the ensemble voting process. As the return types of the terminal nodes (i.e. base classifiers) in the composite solutions are (binary) class labels, The CSVote trees output a class label.

The second function set uses logical operators in the composite solutions to transform the composite solutions into logical expressions. By manipulating the (binary-valued) class predictions of the base classifiers within a composite logic solution (CSLogic), the CSLogic trees are allowed stronger "decision making" ability when classifying the input instances (compared to the CSVote approach). The CSLogic trees also output a class label.

The CSVote and CSLogic approaches are described below.

Composite Voting Solutions (CSVote)

To transform a composite solution into an ensemble where each member votes on class membership, the CSVote approach uses a function set consisting of a singleton function $\{vt\}$. This function is used in two ways. Firstly, when vt is the *root* node in a CSVote tree, the function computes the majority vote of each base classifier within the tree. Secondly, when vt corresponds to an internal (non-leaf)



$$\begin{aligned} &= vt\left(vt(p_5, p_1, p_8), \emptyset, vt(p_6, p_2, \emptyset)\right) \\ \Longrightarrow \quad &= vt(p_5, p_1, p_8, p_6, p_2) \\ &= vt(\texttt{0,1,1,1,0}) \\ &= 1 \end{aligned}$$

(a) CSVote Tree (b) Evaluating CSVote Tree

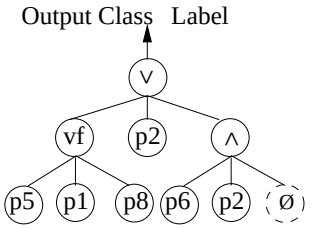
Figure 7.4: Composite voting solution (CSVote) and evaluation of this CSVote tree using terminal node values from Figure 7.3 (tree output is the class label 1 denoting the minority class).

node in a CSVote tree, this function serves no purpose other than to join terminal (leaf) nodes or other vt nodes to the root node. In this case, these internal vt nodes simply pass each of its input arguments up the tree to the root node. This function takes exactly 3 input arguments, which can be other function nodes or terminals nodes.

Only allowing the root to process the majority vote of all the base classifiers in the CSVote tree treats each vote as equally important in the voting process. It is important to note that the output of CSVote tree can be different if each internal (non-leaf) vt node computes the majority vote of its input arguments alone. This particular scenario is outlined in more detail in the next section (which discusses the function set for CSLogic).

Figure 7.4(a) shows a CSVote tree comprising of the five base classifiers . whose predicted class labels (on a given input instance) are shown in Figure 7.3. Using these terminal node values for the base classifiers, Figure 7.4(b) shows how this CSVote tree is evaluated to obtain its output. When this CSVote tree is executed, the (binary) class labels returned from the base classifiers are taken as the terminal node values, and the two internal function nodes pass each of its input arguments up to the root node of the tree. When the $\emptyset$ terminal nodes are encountered, no value is passed up to the root node. The root node then computes a majority vote of these five class labels, and outputs a class label of `1` (denoting the minority class).

When two or more distinct leaf-nodes in a composite solution link to the *same* base classifier, each distinct leaf-node is counted as a separate vote in the voting process. For example, if one of the $\emptyset$ terminal nodes in Figure 7.4(a) is replaced with the node value p_2, then the predicted class label for this classifier will be counted *twice* in the voting process. This is permitted to provide some base



$$\begin{aligned} &= vf(p_5, p_1, p_8) \vee p_2 \vee (p_2 \wedge p_6 \wedge \emptyset) \\ \Longrightarrow \quad &= vf(0, 1, 1) \vee 0 \vee (0 \wedge 1) \\ &= 1 \vee 0 \vee 0 \\ &= 1 \end{aligned}$$

(a) CSLogic Tree (b) Evaluating CSLogic Tree

Figure 7.5: Composite logic solution (CSLogic) and evaluation of this CSLogic tree using terminal node values from Figure 7.3 (tree output is the class label 1 denoting the minority class).

classifiers with a *stronger* influence in the voting process; selection pressure in the evolution determines which base classifiers are assigned this privilege.

In the case of a tie when the two classes have the same number of votes, the minority class label (i.e. 1) is chosen.

Composite Logical Solutions (CSLogic)

To transform a composite solution into a logical expression, the composite logical solutions (CSLogic) use a function set consisting of three functions $\{\wedge, \vee, vf\}$. These functions take 3 input arguments. The function $\vee$ represents a logical disjunction and returns 1 whenever one or more of its input arguments is 1 (0 otherwise), while the function $\wedge$ represents a logical conjunction and returns 1 only if all of its input arguments are 1 (0 otherwise).

The function vf represents a majority vote of all its input arguments. This function returns 1 if two or more of its input arguments are 1, or returns 0 if two or more of its input arguments are 0. In the case of a tie (e.g. if the three input argument are 0, 1 and the $\emptyset$ symbol, respectively), the minority class label is returned (i.e. 1). Unlike in the CSVote trees where only the root node computes the majority vote of all base classifiers, each internal node in this configuration computes a new value (based on its input arguments) to pass up the tree. This provides the internal function nodes in the CSLogic trees some "decision making" ability when processing the inputs.

Figure 7.5(a) shows an example CSLogic tree comprising of the five base classifiers whose outputs (on a given input instance) are shown in Figures 7.3. Using the same terminal node values for the base classifiers from Figure 7.3, Figure 7.5(b) shows how the corresponding logical expression is evaluated (for this input instance) to obtain its output. Note that the $\emptyset$ symbol nodes are

ignored when the CSLogic tree is interpreted. The final class label returned by this CSLogic tree is also 1.

7.3 Experimental Setup for Composite Solutions

This section outlines the experimental setup to evolve composite solutions. This includes a discussion on which MOGP approach (from the previous chapter) is used to train the base classifiers, the GP evolutionary parameters used to evolve the composite solutions, and the two configurations used to train/test the composite solutions.

7.3.1 Underlying MOGP Base Classifiers

The PFC-based MOGP approach (from the previous chapter) is used to generate the pool of base classifiers (i.e. evolved Pareto front solutions). These base classifiers are used as the input to the evolutionary process to learn the composite solutions where the i^{th} run to evolve a given CSVote or CSLogic tree (to represent the pruned ensemble), uses the Pareto front returned from the i^{th} MOGP run.

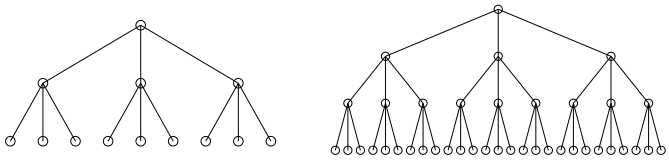
The PFC-based MOGP approach is chosen (and not the NCL-based MOGP or the Baseline MOGP from the previous chapter), as this approach is shown to find individuals with better diversity than the other two MOGPs. The previous chapter shows that the evolved Pareto front solutions using the PFC-based MOGP exhibit better cooperation between ensemble members, particularly with off-EEL, than the other two MOGP approaches on the tasks.

It must be noted that although the composite solutions are evaluated in the context of the PFC-based MOGP approach (i.e. MOGP used to learn the base classifiers), the CSVote and CSLogic approaches are not restricted to genetic program classifiers. The composite solutions approach can be used in conjunction with any underlying learning algorithm to generate the base classifiers. However, evaluating the CSVote and CSLogic approaches with base classifiers generated from other learning algorithms is outside the scope of this work.

7.3.2 Evolutionary Parameters

Sizes of Composite Solution

As the goal of the CSVote and CSLogic approaches are to discover *small* and highly cooperative subsets of base classifiers, a restriction is imposed on the



(a) Tree depth is 2 (b) Tree depth is 3

Figure 7.6: Fully formed composite trees of depth 2 and 3.

maximum tree depth allowed in the evolution. This forces the evolution of smaller composite solutions, i.e., trees comprising of *fewer* base classifiers, compared to trees of larger depths. This reduces the risk of *non-contributing* base classifiers being included in the CSVote and CSLogic trees (i.e. individuals that do not positively contribute to the ensemble accuracy), due to more selection pressure for the limited positions within the composite solutions.

To achieve this, this chapter compares two maximum tree depth settings of 2 and 3 for the composite solutions. The *depth* of a tree corresponds to the number of edges in the longest path from the root node to a given leaf node. When the tree depth is limited to 2, a composite solution can include, at most, nine base classifiers as the function nodes take exactly three input arguments, as shown in Figure 7.6(a). Likewise, when the tree depth is restricted to 3, a composite solution can use, at most, 27 base classifiers, as shown in Figure 7.6(b). It is important to reiterate that the CSVote and CSLogic trees of depths 2 and 3 can also contain *fewer* than 9 and 27 base classifiers, respectively, due to the $null$-valued terminal set symbol $\emptyset$. This symbol represents a "blank" input argument to a given function node.

Single-member composite trees, i.e., CSVote or CSLogic trees with only one terminal node that is not the $\emptyset$ symbol, are permitted in the evolution. A single-member ensembles is akin to a winner-takes-all ensemble combination strategy where the output of the ensemble corresponds to the output of a single member.

Other GP Parameters

A single GP run to evolve a single CSVote or CSLogic composite solution (for a given MOGP Pareto front) is akin to the canonical (single-objective) GP (SGP) approach from Chapters 3 and 4. The only major difference in these composite solutions is that the base classifiers returned from a MOGP run are required as input, in addition to the training set used to evaluate the fitness of the evolved composite solutions.

Similar to the SGP configuration from Chapters 3 and 4, the ramped half and half method is used to generate an initial population of composite trees. Crossover, mutation and elitism rates are also 60%, 35% and 5%, respectively, and the tournament selection size is 7.

The evolution is limited to 30 generations unless a composite solution with 100% accuracy on both classes on the training set is evolved, at which point the evolution is stopped. A population size of 300 is used. These three parameters (i.e. number of generations, population size and the maximum tree depth from above) are the only parameters that are different to the SGP configuration from Chapters 3 and 4, to ensure the training phase to evolve composite solutions is relatively fast.

Fitness Function

The output of a composite solution (when evaluated on a given input instance) is a class label. To evolve composite solutions with good classification accuracy on both the minority and the majority classes, the fitness function uses the average classification accuracy on the minority and majority class (on the training set) to represent the fitness of a given composite solution.

7.3.3 Training Sets for Composite Solutions

To address the second sub-goal of this chapter, two configurations are used to train and test the composite solutions on the tasks. These are outlined below.

Training Configuration 1

The main approach to train the CSVote and CSLogic composite solutions use the same training set that is also used in MOGP to learn/evolve the ensemble members (i.e. Pareto front solutions). Recall from the previous chapter that in MOGP, half the examples in each class from the full (original) data sets are randomly split into the *training* and the *test* sets. Both the training and the test sets preserve the same class imbalance ratio as the original data set. For convenience, these training and test sets are referred to as TRAIN50 and TEST50 in the experimental results.

Training Configuration 2

A limitation of the above training configuration is that the same training set is used to learn/evolve both the individual base classifiers and the composite

solutions. This can potentially lead to overfitting. To avoid this, a second configuration is also used to train the CSVote composite solutions. In this approach, the original data sets are randomly split into three non-overlapping subsets: a *training* set containing 40% of the data instances, a *test* set containing 40% of the data instances, and a "*validation*" set containing the remaining 20% of the data instances. All three sets preserve the same class imbalance ratio as the original data set.

In this experimental setup, the training set is kept aside to learn/evolve the ensemble members (i.e. Pareto front solutions) in MOGP; while the "validation" set is used to learn/evolve the CSVote composite solutions (i.e. pruned ensembles). Strictly speaking, this is not a "validation" set, since a validation set is primarily used to monitor the training process for the same algorithm rather than in a two-stage training process. However, evolving composite solutions is a further refinement process rather than a fully independent training process from the original MOGP training process, and the goal is to avoid overfitting. We call it here a "validation" set for convenience.

Similar to the previous training configuration (discussed above), the test set is used to evaluate the ensemble on the unseen input instances.

For convenience (and to distinguish these learning sets from the previous configuration) the training, validation and test sets in this configuration are referred to as TRAIN40, VALIDATION20 and TEST40, respectively, in the experimental results. The data sets are partitioned in this way for two reasons. Firstly, to ensure that the ensemble members and the composite solutions are trained using different (non-overlapping) learning instances. Secondly, to ensure that the learning data used in the combined (two-stage) training process (TRAIN40 and VALIDATION20) is not substantially larger than the unseen test set (TEST40).

7.4 Experimental Results for Composite Solutions

This section presents the experimental results on the tasks, and has four main parts. The first three parts use the first training configuration to evolve the composite solutions and compare these results to the Off-EEL algorithm [76]. The fourth part uses the second training configuration and and focuses on how these evolved composite solutions compare to those evolved using the first training configuration.

Table 7.1: Ensemble accuracy ($\pm$ standard deviation) on the test set, and average ensembles size (minimum ensemble size in parenthesis), for the CSVote and CSLogic approaches to ensemble selection (over 50 runs) when the maximum composite solution tree depth is 2 and 3.

	Max.	CSVote Results			CSLogic Results		
	Dep.	Num.	Minority	Majority	Num.	Minority	Majority
Ion	2	8.9 (7)	84.8 ± 4.6	95.8 ± 2.4	8.3 (6)	80.6 ± 5.7	92.4 ± 4.5
	3	21.6 (9)	81.9 ± 5.3	91.9 ± 6.2	26.5 (13)	67.6 ± 30.0	70.7 ± 32.2
Spt	2	8.8 (7)	63.6 ± 6.2	81.7 ± 4.5	9.0 (9)	54.9 ± 5.9	82.9 ± 4.0
	3	17.7 (7)	55.7 ± 7.3	85.7 ± 3.9	23.3 (7)	51.1 ± 31.9	81.2 ± 23.1
Ped	2	8.7 (7)	88.1 ± 2.3	90.4 ± 2.4	9.0 (9)	85.0 ± 3.0	86.9 ± 3.0
	3	25.7 (9)	90.7 ± 2.2	88.1 ± 3.0	24.0 (11)	87.8 ± 2.1	88.8 ± 2.0
Yst_1	2	9.0 (9)	71.3 ± 5.9	77.7 ± 6.0	8.9 (7)	64.6 ± 3.9	81.4 ± 3.1
	3	17.5 (7)	67.8 ± 5.1	77.9 ± 4.8	26.9 (10)	56.5 ± 36.5	65.9 ± 34.0
Yst_2	2	9.0 (9)	93.1 ± 2.4	90.7 ± 2.3	8.7 (3)	86.1 ± 4.2	93.4 ± 1.9
	3	16.4 (9)	88.8 ± 7.8	90.2 ± 3.8	19.2 (4)	76.0 ± 27.9	78.7 ± 23.4
Bal	2	8.7 (1)	80.7 ± 8.3	89.4 ± 6.1	8.8 (1)	71.1 ± 14.7	91.7 ± 4.7
	3	12.6 (7)	74.3 ± 13.0	86.8 ± 7.6	17.1 (9)	54.5 ± 36.4	79.4 ± 23.3

7.4.1 Ensemble Accuracy for Composite Solutions

Table 7.1 shows the ensemble performances and the ensemble sizes for the CSVote and CSLogic approaches (over 50 runs) on the test set (TEST50) for maximum tree depths of 2 and 3. These composite solutions are evolved using the first training configuration (i.e. using TRAIN50). Recall that when the maximum tree depth is 2, the CSVote and CSLogic ensembles can contain, at most, 9 base classifiers. When the tree depth is 3, the ensembles can contain, at most, 27 MOGP base classifiers. Table 7.1 also includes the minimum ensemble size (over 50 runs) in parenthesis alongside the average ensemble size.

According to Table 7.1, the smaller ensembles (maximum tree depth is 2) generally show better performances than the larger ensembles (maximum tree depth is 3) for both the CSVote and CSLogic approaches on the tasks. The smaller ensembles dominate the larger ensembles in four tasks for CSVote (all except Spt and Ped), and in five tasks for CSLogic (all except Ped). This suggests that the smaller composite solutions contain a more diverse set of base classifiers that cooperate better together than the larger composite solutions, as the larger composite solutions include some members that negatively affect the generalisation ability of these ensembles on the test set.

For the CSLogic approach in particular, the larger ensembles show substan-

tially poorer minority and majority class accuracies than both the smaller CSLogic ensembles and the CSVote ensembles in these tasks. This suggests that the logical expressions represented by the larger CSLogic solutions may be overly sensitive to the training data, as this approach shows poor generalisation on the test sets. The CSVote approaches do not suffer from this problem to the same extent as the CSLogic approaches. This suggests that the voting-based approach to combine the predictions of the individual members in CSVote may be more robust than the logical expressions, particularly as the ensemble sizes increase.

Further Analysis: Composite Solution Sizes

The minimum ensemble sizes (over 50 runs) are shown in Table 7.1 to verify whether any of the evolved CSVote and CSLogic composite solutions represent single-member ensembles. Table 7.1 shows that a single-member composite solution is found for both the CSVote and CSLogic approaches when the maximum tree depth is 2 in one task, Bal. In the remaining tasks, the minimum ensemble sizes are all greater than 1.

A closer examination of these results for Bal reveals that both the CSVote and the CSLogic solutions with an ensemble size of 1 occurs in exactly one GP run, and this is the *same* run for both CSVote and CSLogic. In this particular run, both CSVote and CSLogic solutions also selected the *same* base classifier to represent the single-member ensemble. This base classifier corresponds to the only Pareto front solution to achieve 100% accuracy on the minority and the majority classes on both the training and the test sets (in all 50 MOGP runs). This particular base classifier was mentioned in the previous chapter (in Section 6.4.2 on page 154) since this solution represents a perfect Pareto Optimal hyperarea of 1 for this MOGP approach (i.e. MOGP with PFC). This demonstrates the usefulness of the GP approach to explore different subsets of base classifiers for pruning the ensemble in one instance.

7.4.2 Comparison with Off-EEL for Ensemble Selection

The CSVote and CSLogic approaches are also compared to off-EEL [76] to measure their effectiveness on the tasks as all three approaches aim to optimise which individuals to choose in the ensembles. For convenience, the off-EEL ensemble results from the previous chapter are repeated in Table 7.2; these correspond to the ensemble accuracies on the test sets (TEST50) using TRAIN50 in the training process in MOGP. As the smaller CSVote and CSLogic composite

Table 7.2: Ensemble accuracy ($\pm$ standard deviation) on the test set and average ensembles size using off-EEL [76], CSVote and CSLogic (maximum tree depth of 2) for ensemble selection (over 50 runs).

Tasks	off-EEL			CSVote			CSLogic		
	Size	Minority	Majority	Size	Minority	Majority	Size	Minority	Majority
Ion	21.2	83.9 ± 5.2	96.6 ± 2.8	8.9	84.8 ± 4.6	95.8 ± 2.4	8.3	80.6 ± 5.7	92.4 ± 4.5
Spt	10.7	66.3 ± 8.5	79.9 ± 6.6	8.9	84.8 ± 4.6	95.8 ± 2.4	8.3	80.6 ± 5.7	92.4 ± 4.5
Ped	55.2	90.6 ± 1.5	87.9 ± 1.5	8.7	88.1 ± 2.3	90.4 ± 2.4	9.0	85.0 ± 3.0	86.9 ± 3.0
Yst_1	29.2	70.6 ± 5.4	78.8 ± 5.5	9.0	71.3 ± 5.9	77.7 ± 6.0	8.9	64.6 ± 3.9	81.4 ± 3.1
Yst_2	17.2	93.1 ± 2.6	90.8 ± 2.4	9.0	93.1 ± 2.4	90.7 ± 2.3	8.7	86.1 ± 4.2	93.4 ± 1.9
Bal	10.9	81.4 ± 12.1	86.2 ± 9.1	8.7	80.7 ± 8.3	89.4 ± 6.1	8.8	71.1 ± 14.7	91.7 ± 4.7

solutions generally show better results than the larger composite solutions on these tasks (as discussed earlier), this comparison focuses on the evolved composite solutions with a maximum tree depth of 2 (i.e. pruned ensembles that use, at most, 9 members). For convenience, Table 7.2 also includes these CSVote and CSLogic results (repeated from Table 7.1 in the previous section) when the maximum tree depth is 2.

According to Table 7.2, the CSVote approach achieves non-dominated ensemble performance compared to off-EEL in all tasks. This is not surprising as both methods use the majority vote to combine the outputs of their individual members, and suggests that these two methods are similarly effective in finding good individuals for the ensemble. The CSLogic approach is dominated by off-EEL in one task (Ion), and is non-dominated relative to off-EEL in the remaining five tasks.

Notice that the CSVote approach achieves similar (non-dominating) performances to off-EEL using *fewer* individuals in the ensemble compared to off-EEL. This demonstrates the usefulness of the secondary evolutionary search in the CSVote approach to finding small groups of individuals that cooperate well together. The evolutionary search to evolve CSVote solutions is reasonably fast, taking between 0.2 and 5 seconds on the tasks (this is approximately 2–3% of the training time to evolve a MOGP front).

Comparing Ensemble Wins

To compare which of the three ensemble selection approaches (CSVote, CSLogic and off-EEL [76]) statistically dominates each other on the tasks, and which achieves a higher total number of wins over all runs and tasks, Table 7.3 shows the pairs of ensemble *wins* between two approaches when each is compared with every other (on a run-by-run basis) for 50 runs. Recall (from the previous chapter)

Table 7.3: "Win" pairs between two MOGP approaches (on a run-by-run basis) over 50 runs for three ensemble selection strategies (CSVote, CSLogic and off-EEL [76]). A "win" is when one approach dominates the other on a given run. Total wins (and draws) is the sum of wins (and draws) over all runs and tasks (50 runs $\times$ 6 tasks). Bold results indicate a statistically significantly better ensemble performance (95% significance level).

Task Task	CSVote *vs* CSLogic	CSVote *vs* off-EEL	CSLogic *vs* off-EEL
Ion	**35**,1	8,12	5,**33**
Spt	**17**,2	7,9	2,**9**
Ped	**22**,2	2,**9**	2,**12**
Yst1	**6**,1	5,4	1,**11**
Yst2	**5**,0	7,9	0,**4**
Bal	8,5	11,6	7,9
Total Wins	95,11	40,49	17,78
Total Draws	194	211	205

that a "win" is when one approach dominates the other, and a "draw" is when both approaches are non-dominated, for a given run. The CSVote and CSLogic ensembles in Table 7.3 use composite solutions with a maximum tree depth of 2 (i.e. ensemble sizes of, at most, 9 members). Table 7.3 corresponds to the ensemble wins on the test set (TEST50) when both the composite solutions and the base classifiers are training using TRAIN50 in MOGP. The statistically significantly better ensemble strategy, denoted by a higher number of wins, is highlighted in bold (at a 95% confidence level). The last two rows in Table 7.3 show the total number of wins for each pair, and the total number of draws (non-dominated performances), summed over all 50 runs for the six tasks (300 total runs).

Table 7.3 shows that that over all runs and tasks, the CSLogic ensembles achieve a relatively low number of total wins when compared to both the CSVote and off-EEL approaches. When compared to CSVote, CSLogic has 11 total wins while CSVote has 95; when compared to off-EEL, CSLogic has 17 total wins while off-EEL has 78. This is because both the CSVote and off-EEL achieve (statistically) significantly better ensemble performances than CSLogic in four and five tasks, respectively. As previously mentioned, this suggests that the voting strategy used in CSVote (and off-EEL) to combine the outputs of the individual members produces better results than the logical function represented by the CSLogic solutions. This may be because the voting strategy is better able to generalise

between training and test data, or is more robust to noise in the test set.

Table 7.3 shows that when the CSVote and off-EEL approaches are compared to each other, the total number of wins (over all tasks) is very similar for both approaches. The off-EEL ensembles achieve a (statistically) significantly better performances than CSVote in only one task (Ped). This shows that the secondary evolutionary search to prune the ensembles in CSVote produces similar results compared to the greedy search in off-EEL [76] on these tasks. However, the computational effort required to evolve CSVote solutions is substantially higher than the *Off*-EEL algorithm. A possible reason why the ensemble performances for CSVote are not better than off-EEL is that the evolved composite solutions may be over-trained; this possibility is explored further in the next section below.

7.4.3 Training Performances for Composite Solutions

As mentioned above, the evolved CSVote and CSLogic solutions may suffer from overfitting as both the composite solutions and the base classifiers are evolved using the same training set. To investigate if overfitting has occurred, Table 7.4 shows the ensemble performances on the *training* set (TRAIN50) for the three approaches (CSVote, CSLogic and off-EEL) over 50 runs. These results for the CSVote and CSLogic approaches use, at most, 9 members in the pruned ensembles (i.e. maximum tree depth is 2).

Table 7.4 show that both the CSVote and CSLogic approaches achieve very high accuracies on both classes for the training sets. In some tasks (such as Ion, Spt and Bal), this is nearly 100% accuracy on both classes. The composite solutions, particularly CSVote, also have higher minority and majority class accuracies than off-EEL on the training set in nearly all tasks (except Ped where these methods are non-dominated to each other). The results in Table 7.1 (on the test sets) for both CSVote and CSLogic are substantially poorer than some of these training performances, in particular Ion, Spt and Bal. This suggests that the CSVote and CSLogic approaches are *over-fitted* to the training set in these tasks.

On the other hand, the training performances for off-EEL are not as good as the CSVote and CSLogic approaches in Table 7.4, and only slightly better than the off-EEL results on the test set (from Table 7.2). This suggests that the off-EEL ensembles suffer less overfitting (if any) compared to the CSVote and CSLogic approaches in nearly all tasks.

In the Ped task, all three approaches (CSVote, CSLogic and off-EEL) show relatively similar accuracies on both the training and test sets, suggesting that

Table 7.4: Ensemble performances on the training set (TRAIN50) for the ensemble selection approaches (CSVote, CSLogic and off-EEL [76]) over 50 runs.

	Off-EEL		CSVote		CSLogic	
	Minority	Majority	Minority	Majority	Minority	Majority
Ion	97.0 ± 2.2	98.0 ± 1.5	99.2 ± 1.1	99.0 ± 1.0	99.6 ± 0.8	99.4 ± 0.7
Spt	92.4 ± 5.8	91.4 ± 1.8	100.0 ± 0.0	92.8 ± 0.7	99.9 ± 0.5	90.9 ± 1.2
Ped	92.6 ± 1.9	90.1 ± 2.6	91.7 ± 1.1	92.0 ± 1.5	91.2 ± 1.3	85.1 ± 1.3
Yst_1	83.0 ± 3.3	84.8 ± 3.7	88.1 ± 3.1	86.3 ± 2.5	88.5 ± 2.3	87.2 ± 2.2
Yst_2	91.6 ± 2.9	95.5 ± 1.4	97.4 ± 1.7	94.2 ± 1.4	97.5 ± 1.7	95.0 ± 1.3
Bal	93.8 ± 6.4	90.3 ± 5.5	99.7 ± 1.0	96.2 ± 4.4	98.7 ± 2.2	94.7 ± 3.2

no overfitting is occurring in this task. This is most likely due to the very large number of training examples in this data set (Ped is considerably larger than the other tasks).

However, Table 7.4 nevertheless illustrates the effectiveness of the composite solutions for ensemble selection, as the training performances of the CSVote and CSLogic ensembles dominate the training performance of the off-EEL ensembles. The CSVote and CSLogic ensembles achieve near-perfect classification accuracies on both classes during the training process. Neither the individual ensemble members nor off-EEL can accomplish this during the training process. The CSVote and CSLogic approaches can be particularly useful in optimisation problems or *online learning* which does not need an unseen test set.

To try to address to this overfitting issue, an extra "validation" set is used in the secondary evolutionary search to learn/evolve the composite solutions. This in explored further in the next section.

7.4.4 "Validation" Set in Composite Solution Training

To address the second subgoal of this chapter and the overfitting issue described above, a separate "validation" set is used to learn/evolve the CSVote composite solutions representing the pruned ensembles. Recall (from Section 7.3.3) that in this experimental setup, TRAIN40 is used to learn/evolve the ensemble members (i.e. Pareto front solutions) in MOGP; while VALIDATION20 is used to learn/evolve the CSVote composite solutions (i.e. optimised ensembles). This experimental setup investigates whether this extra "validation" set can improve the generalisation ability of the CSVote composite solutions on the unseen test data (TEST40).

Table 7.5 shows the performances of the evolved CSVote trees on these three

Table 7.5: Average performances of the CSVote approach trained using VALIDATION20, and evaluated on TRAIN40 and TEST40 (over 50 runs).

	On VALIDATION20		On TRAIN40		On TEST40	
	Minority	Majority	Minority	Majority	Minority	Majority
Ion	99.4 ± 1.5	99.9 ± 0.4	96.6 ± 3.0	95.2 ± 3.7	85.8 ± 4.0	95.4 ± 2.4
Spt	98.0 ± 1.2	97.3 ± 1.1	90.2 ± 7.9	90.2 ± 4.5	63.5 ± 7.9	84.3 ± 4.6
Yst_1	72.8 ± 7.9	84.8 ± 7.1	82.5 ± 8.5	75.4 ± 10.1	78.7 ± 6.9	78.2 ± 7.4
Yst_2	97.9 ± 2.3	89.8 ± 2.3	94.4 ± 3.9	89.4 ± 7.1	95.6 ± 1.5	91.4 ± 2.1
Bal	100.0 ± 0.0	98.5 ± 1.9	94.9 ± 5.7	93.8 ± 5.7	81.9 ± 8.6	92.9 ± 4.9

data set partitions (VALIDATION20, TRAIN40 and TEST40) for all tasks except Ped. Ped is omitted as no serious overfitting issues are seen in this task for the CSVote approach (as discussed in the previous section). Furthermore, this analysis focuses on the CSVote approach (and not the CSLogic approach) since this type of composite solution generally shows better results on the tasks (as discusses earlier), particularly when the maximum tree depth is limited to 2.

While the CSVote trees still show very high accuracies on the "validation" sets in Table 7.5, their performance on the training and the test sets (TRAIN40 and TEST40, respectively) is more consistent relative to each other. In other words, the difference in performance between TRAIN40 and TEST40 for CSVote is not as large as the difference between TRAIN50 and TEST50 (from Tables 7.4 and 7.1, respectively, in previous sections). In one task in particular (Yst_1), the CSVote performance on TEST40 even dominates the performance on TRAIN40. This suggests that the CSVote ensembles show less over-fitting on the training sets.

Table 7.5 also shows that the CSVote performances on the TEST40 are slightly better than the CSVote performances on the TEST50 (from Table 7.1) on all tasks. However, this difference in performance is not substantial in nearly all tasks (except Yst_1). Here the CSVote performances on TEST40 dominate the CSVote performances on TEST50 by roughly 1–3% on the minority and the majority classes. This suggests that the use of an extra "validation" set in the CSVote training process can slightly improve ensemble performances on both classes compared to using the same training set to learn/evolve both the ensemble members and the CSVote solutions.

When the CSVote performances on TEST40 are compared to the off-EEL ensembles on TEST50 (from Table 7.2), CSVote dominates Off-EEL on the three tasks which have the highest level of class imbalance (Yst_1, Yst_2 and Bal). In two of these tasks (Yst_1 and Bal), CSVote on TEST40 shows substantially better results than off-EEL on TEST50, improving either the minority or majority class

Table 7.6: Off-EEL performances using TRAIN40 to train the base classifiers and VALIDATION20 to select the best ensemble members, and final performance on the unseen test sets TEST40 (over 50 runs).

	On VALIDATION20		On TRAIN40		On TEST40	
	Minority	Majority	Minority	Majority	Minority	Majority
Ion	93.3 ± 4.8	98.9 ± 1.6	91.8 ± 7.0	98.8 ± 1.7	72.7 ± 7.4	95.0 ± 4.0
Spt	86.4 ± 8.4	87.1 ± 6.8	87.3 ± 9.4	86.5 ± 4.1	55.7 ± 10.9	84.5 ± 5.3
Yst_1	67.4 ± 7.9	84.7 ± 7.4	80.5 ± 7.7	81.4 ± 8.0	74.5 ± 6.3	79.8 ± 8.3
Yst_2	96.1 ± 3.9	87.5 ± 3.7	96.9 ± 2.1	88.9 ± 3.4	96.2 ± 1.5	88.5 ± 2.8
Bal	93.1 ± 11.3	89.1 ± 7.9	94.8 ± 9.6	90.4 ± 6.9	76.0 ± 14.4	87.0 ± 8.3

accuracy by approx 7–8% (with no trade-off in the accuracy of the other class). For example, in Yst_1, CSVote achieves roughly 78% accuracy on both the minority and the majority classes; while off-EEL shows 70% and 78% accuracy on the minority and majority class, respectively. This suggests that the CSVote approach trained with the extra "validation" set can improve ensemble performances over off-EEL on tasks with very high levels of class imbalance.

Off-EEL using "Validation" Set

Off-EEL is also evaluated using the second training configuration — these results are shown in Table 7.6. In this scenario, the Pareto front is sorted according to their fitness on the training set (TRAIN40), and the Off-EEL algorithm is evaluated using the VALIDATION20 set. This means that the best ensemble performance on VALIDATION20 is selected as the final off-EEL ensemble (for given MOGP run). The final ensemble is then evaluated on the training set (TRAIN40) and the test set (TEST40) to produce the results shown in Table 7.6 (over 50 runs).

According to Table 7.6, it not clear whether the test performances for off-EEL with "validation" (on TEST40) are better or worse than the test performances for off-EEL without "validation" from Table 7.2 (on TEST50). Off-EEL with "validation" is dominated by off-EEL without validation in one task (Ion), and dominates off-EEL without validation in one other task (Yst_1). In the remaining three tasks, both methods are non-dominated to each other. This means that while the extra "validation" set slightly improves the generalisation ability of the CSVote ensembles on most tasks, this is not the case for off-EEL.

As off-EEL shows no major improvement on the test set using the "validation" set on most tasks (except Yst_1), this suggests that choosing the individual members based on the ensemble's performance on the training set can be sufficient for good performance on the test set (compared to using the "validation" set). This

may be because unlike CSVote, off-EEL does not use a secondary training phase to optimise the ensembles.

However, the training performances for off-EEL with "validation" in Table 7.6 (on TRAIN40) are not as good as the training performances for off-EEL without "validation" on Table 7.2 (on TRAIN50). In three tasks, the results for TRAIN50 dominate TRAIN40 (Ion, Spt and Yst_1). In the remaining two tasks, these results are not dominated to each other. This suggests that the "validation" set does succeed in slightly improving the generalisation of the off-EEL ensembles since the difference between TRAIN40 and TEST40 (in Table 7.6) is not as large as the difference between TRAIN50 and TEST50 (in Table 7.2) where some overfitting has occurred.

Not unexpectedly, Table 7.6 also shows that off-EEL on TEST40 is dominated by CSVote on TEST40 (in Table 7.5) in two tasks (Ion and Bal). Here the minority and majority class accuracies for CSVote are much better than off-EEL. While these methods are non-dominated to each other in the remaining three tasks, in two of these tasks (Yst_1 and Yst_2), CSVote also shows a much better balance in the minority and majority class accuracies than off-EEL. As mentioned earlier, this is because the "validation" set can generally improve ensemble performances in CSVote but not in off-EEL for these tasks.

7.5 Summary

The main goal of this chapter was to develop a new GP approach to ensemble selection to quickly find highly-cooperative subsets of individuals from the set of Pareto front solutions evolved in MOGP. This goal was achieved by evolving two types of composite solutions for ensemble selection. Composite solutions represent multiple Pareto front individuals that are amalgamated together into a single genetic program. The first type of composite solution, CSVote, represents individuals whose outputs are combined using the traditional majority vote strategy in the (pruned) ensemble. The second type of composite solution, CSLogic, manipulates the outputs of individual members using logical operators, to allow the ensembles more "decision-making" abilities.

Two sub-goals are considered in this chapter. The first compared which type of composite solutions (CSVote and CSLogic) shows better generalisation on the unseen test sets. The second investigated whether an additional "validation" set, used in the evolution of the composite solutions, improves ensemble performances on the unseen test sets compared to without. These are summarised

below.

7.5.1 Composite Voting and Logic Solutions

This chapter shows that composite solutions which represent small ensembles (consisting of, at most, 9 individuals) outperform larger ensembles (consisting of, at most, 27 individuals) on the tasks. This is because these smaller ensembles comprise of highly diverse individuals that cooperate well together to predict unseen input instances; whereas the larger ensembles contain some individuals that negatively affect the generalisation ability of these final ensemble on the test set. For the CSLogic approach in particular, the larger ensembles show substantially poorer minority and majority class accuracies than the smaller ensembles on the tasks. This suggests that the complex logical expressions represented by the larger CSLogic solutions, may be overly sensitive to the training data.

Comparing the CSVote and CSLogic approaches to the off-EEL algorithm [76] for ensemble selection, off-EEL and CSVote ensembles achieve similar results; whereas off-EEL outperforms the CSLogic approach on the tasks. This suggests that the majority vote strategy used in both CSVote and off-EEL may be more robust than the logical expressions in the CSLogic solutions, as it is better able to generalise on the unseen data, particularly as the ensemble sizes increase. Furthermore, CSVote is able to achieve similar ensemble performances to off-EEL using fewer individuals in the ensemble, suggesting that the CSVote ensembles have better diversity between individual members than off-EEL. Smaller ensembles also reduce the computation time to obtain the ensemble output as there are fewer individual members to evaluate on the input data.

7.5.2 Evolving Composite Solutions

The composite solutions are evolved in a secondary training phase using the same training data also used to evolve the Pareto fronts in MOGP (i.e. half of all input instances from the original data sets). After this secondary training phase, the evolved composite solutions show very high accuracies on the training set. Neither the individual members in MOGP, nor the off-EEL algorithm, can accomplish this during the (primary) training process. This suggests that the CSVote and CSLogic approaches can be particularly useful in optimisation problems or online learning which does not need an unseen test set. However, the evolved composite solutions, in particular CSVote, are only able to perform as

well as, but not better than, off-EEL on the unseen test sets, although the number of individuals is the CSVote ensembles is much smaller.

To try to address this overfitting issue, an extra "validation" set (containing 20% of the total input instances) is used to evolve the CSVote solutions. While the extra "validation" set slightly improved the generalisation of the CSVote ensembles on the unseen test set in most tasks (compared to without), this was not the case for off-EEL, since off-EEL does not use a secondary training phase to optimise the ensembles. On some tasks CSVote with the validation set showed substantially better performances on both classes compared to off-EEL using the validation set.

Chapter 8

Conclusions

The overall goal of this thesis was to develop new internal cost-adjustment techniques in GP for binary classification with unbalanced data. The focus of this thesis was to enhance the GP learning algorithm to use the unbalanced input data directly in the learning process, thereby removing any dependence on a sampling algorithm to first artificially re-balance the learning data prior to the evolutionary learning process. This goal was achieved by developing a number of new methods in GP to evolve genetic program classifiers with good classification ability on both the minority and the majority classes. These new methods were evaluated by applying GP to a range of binary benchmark classification tasks with unbalanced data.

The rest of this chapter summarises the research objectives achieved by this thesis and the main conclusions from the individual chapters, and provides further discussions on more general topics covered in the whole thesis and key areas for future work.

8.1 Achieved Objectives

This thesis has achieved the following research objectives

- The thesis proposes a GP approach to binary classification with unbalanced data focusing on cost-adjustment within GP rather than the traditional data-balancing techniques. This thesis shows that unlike tasks with multiple balanced classes where some dynamic classification strategies perform significantly better than the traditional static classification strategy, either a static or dynamic strategy in the evolution shows no major difference in the performance of evolved GP classifiers on these binary tasks.

- This thesis proposes several new fitness functions in GP to perform cost adjustment between the minority and the majority classes. These new fitness functions evolve GP classifiers with high AUC, and with relatively fast GP training times, using the unbalanced data sets directly in the learning process without first re-balancing the data (via sampling).

- This thesis proposes a multi-objective GP (MOGP) approach which treats the accuracies of the minority and the majority classes separately during the learning process. The MOGP approach evolves a good set of trade-off solutions in a single run that perform as well as, and in some cases better than, multiple runs of canonical single-objective GP (SGP).

- This thesis proposes an ensemble-based approach to classification where multiple MOGP classifiers vote on the predicted class label. Two fitness functions are developed to treat the diversity on both the minority and the majority classes as equally important in the fitness function. The evolved ensembles outperform their individual members on the tasks due to good cooperation between members.

- This thesis proposes a GP approach to ensemble selection to quickly find small groups of individuals that cooperate well together in the ensemble. The pruned ensembles use much fewer individuals to achieve performances that are as good as an existing approach to ensemble selection, particularly on tasks with high levels of class imbalance, thereby reducing the total time to evaluate the ensemble.

8.2 Main Conclusions

The section discusses the main conclusions for the five research objectives (from Chapter 1).

8.2.1 GP for AUC Optimisation

Research goals 1(a) and 1(b) propose a GP approach to binary classification with unbalanced data focusing on the classification strategy and fitness function in GP. In this GP approach, the fittest evolved genetic program from the evolutionary search process represents the learned concept, and the area under an ROC curve (AUC) measures how well a classifier performs on the minority and majority

classes. Based on Chapters 3 and 4 which address these research goals, the following conclusions can be drawn.

Classification Strategies in GP

The classification strategy in GP specifies how the raw (real-valued) output for a genetic program classifier is translated to a class label. While a non-static classification strategy performs better than the traditional static classification strategy in tasks with multiple (balanced) classes, this research finds that this is not the case for these binary class imbalance tasks. Rather, there is no major differences in the AUC of the evolved GP classifiers using the two strategies on these tasks. This shows that GP can sufficiently tweak the mathematical expressions representing the classifiers to "shift" its outputs relative to the fixed class boundary. An advantage of the traditional static strategy is that more uniformity is introduced in the population. Using the non-static strategy, the evolved solutions can lack this uniformity as different class boundaries are defined for the solutions.

Fitness Functions in GP

The fitness function in GP measures the overall performance of a solution by comparing the solution's predicted class label to the target (or actual) class label for all input instances in the training set. This thesis shows that the traditional fitness function in GP based on the overall classification accuracy (or error rate), evolves biased classifiers with high majority class accuracies but often very poor minority class accuracies (and thus, poor AUC) on the unbalanced tasks. Several new fitness functions are proposed to evolve solutions with high AUC when data sets are unbalanced, and with faster training times than using the AUC directly in the fitness function. The three new fitness functions with the best classification results on the tasks include the following. The first finds the distance between the output values of a solution on the two classes by modelling these outputs as two separate class distributions. The second uses the average mean square error (MSE) for each class to "calibrate" a given solution's outputs to (pre-defined) target class values. The third uses the statistical measure, the correlation ratio, to measure the separability between the output values of a solution on the two classes. Two new measures are also developed to improve a well-known measure based on the average accuracy of the two classes in the fitness function.

The GP methods are found to outperform two popular learning algorithms,

namely, Naive Bayes (NB) and Support Vector Machines (SVM) on the tasks, particularly when the level of class imbalance is large. Both NB and SVM show biased classification results in this case.

8.2.2 MOGP for Evolving Pareto Fronts

Research goal 2(a) proposes a multi-objective GP approach where the accuracies of the minority and the majority classes are traded-off against each other in the learning process. The novelty of this approach is that a Pareto-based fitness function is used to the treat the unbalanced classes *independently* (as separate objectives) in the learning process (i.e. for cost adjustment when the unbalanced data is used directly in training). This allows multiple trade-off solutions to be evolved in a single optimisation run, leaving the final choice for the decision-maker. Canonical SGP requires a much longer time to get a reasonable front as the objective preference is specified *a priori* (multiple SGP runs are needed each with a different objective preference).

Pareto Dominance Measures in MOGP

A Pareto dominance measure using both dominance rank and dominance count in the fitness function in MOGP (SPEA2 [188]) finds Pareto-frontier solutions that perform better than, or at least as well as, multiple runs of canonical SGP. A Pareto dominance measure using only dominance rank (NSGAII [53]) cannot achieve this to a sufficient level of accuracy in half of the tasks. The fitness function using SPEA2 finds more solutions in the middle region of the frontier, pushing this front outwards toward high accuracy rates on both classes; while the fitness function using NSGAII finds a better "spread" of solutions along the whole of the frontier. In MOGP, end-region solutions represent biased classifiers. The evolved populations using SPEA2 also contain more non-dominated solutions in the final generations than NSGAII on the tasks.

AUC of Pareto Front Solutions in MOGP

Analysis of the AUC of the evolved Pareto front solutions in different regions of the objective-space confirms that, as expected, the AUC is better in the middle-region of the frontier (i.e. solutions with high accuracy on both classes) compared to the end-regions. More interestingly, SPEA2 not only finds more non-dominated solutions in the final generations than NSGAII, but these solutions for

SPEA2 also have better AUC than NSGAII in nearly all tasks. This also explains why the evolved Pareto fronts are better using MOGP with SPEA2 on most of the tasks. Some Pareto front solutions are also found to have poorer individual AUC performances than good SGP solutions. This is because individual genetic program solutions in SGP capture the performance trade-off between the two objectives, minority and majority class accuracy, using an ROC curve; whereas in MOGP, this requirement is delegated to the *set* of genetic program solutions on the Pareto front.

8.2.3 MOGP for Ensemble Learning

Research goal 2(b) proposes an MOGP approach to evolving ensembles of Pareto front solutions where members vote on the class label, using the fitness function to promote diversity between Pareto front solutions in the evolution. Unlike traditional ensemble learning approaches (where the unbalanced data is first re-balanced via sampling), two established diversity measures are adapted to calculate the diversity separately for the two classes (to account for the unequal classes), and these are incorporated into the fitness function in MOGP. These ensemble approaches are evaluated using several ensemble combination and selection strategies from the literature.

Ensemble Combination and Selection Strategies.

When the full evolved Pareto front is used in the ensemble (using a majority vote strategy), the ensemble performances are biased toward the majority class in nearly all tasks. This is due to the influence of more Pareto front solutions with a stronger majority class bias (than minority class bias) in the voting process, as more of these solutions achieve a non-dominated status in the population during the evolution. Two effective strategies to choose only good Pareto front individuals for the ensemble are shown to improve ensemble performances on both classes. These include a fitness-weighted majority vote of the individual members and a post-training ensemble selection approach: the offline evolutionary ensemble learning (off-EEL) algorithm [76]. While the off-EEL algorithm shows the best ensemble performances on the two classes for the tasks, both strategies outperform a naive accuracy-based ensemble selection method which simply removes individuals with less than 50% accuracy on the objectives from the ensemble. The off-EEL algorithm is particularly successful on the tasks as it uses a greedy search to find which members contribute to good ensemble

performances, where the more diverse the individual members, the better the ensemble performances using off-EEL.

Ensemble Diversity in Fitness

MOGP using pairwise failure crediting (PFC) for diversity in fitness is found to outperform both MOGP using negative correlation learning (NCL) and a baseline MOGP approach (using no diversity measure in fitness), particularly when combined with off-EEL algorithm. This is due to better cooperation between individuals using PFC. The MOGP approaches show that the stochastic way in which new GP classifiers are created in the evolution (e.g. using the genetic operators), and a good fitness function to encourage diversity between members in the population, provides sufficient diversity between individuals for good ensemble performances.

Ensembles vs Canonical Single-Predictor Learners

Both the NCL and PFC approaches evolve at least one solution with 100% accuracy on both the training and test set on the task with the highest level of class imbalance (Bal). The best runs of canonical SGP, NB and SVM could not accomplish this on any task. The MOGP ensembles, in particular with PFC, also outperform canonical SGP, NB and SVM in half the tasks. This is due to the additional support provided in MOGP for the two objectives as the Pareto front captures the trade-off between the objectives; whereas in SGP, this is accomplished by individual genetic program classifiers (via an ROC curve). Combining these Pareto front classifiers into an ensemble where individuals cooperate by voting further improves performances on the objectives as the ensembles perform better than their individual members.

8.2.4 Composite Solutions for Ensemble Selection

Research goal 2(c) proposes a new GP approach to ensemble selection to efficiently find small groups of diverse Pareto front solutions for the ensemble. To avoid "fine tuning" a large weight vector (as used in traditional methods), the new approach evolves composite GP solutions to represent the (optimised) ensemble by combining multiple Pareto front individuals into a single composite solution. The new approach has two main novelties. Firstly, by imposing a size constraint on the composite solutions, selection pressure can be used to automatically find small groups of diverse members for the ensemble. Secondly, using

different function sets in the evolution provides a mechanism to manipulate the outputs of the individual members, to control how the ensemble determines its final classification decision.

Representing Composite Solutions

The composite voting solutions (CSVote) where individuals are combined using the traditional majority vote strategy, are found to achieve similar ensemble performances to the off-EEL algorithm for ensemble selection on the tasks, but with *fewer* individuals in the ensemble. The smaller CSVote ensembles comprise of more diverse individuals that cooperate well together, and also require less computational time to obtain the ensemble output since there are fewer individual members to evaluate. Both CSVote and off-EEL outperform composite logical solutions (CSLogic), where the outputs of the individuals are combined using logical operators. The majority vote strategy used in both CSVote and off-EEL may be more robust than the logical expressions in CSLogic, as overly complex logical expressions in the CSLogic solutions can overfit the training data.

Evolving Composite Solutions

When the composite solutions are evolved using the same training data also used to learn the Pareto fronts in MOGP, these show near-perfect accuracy on the training set. Neither the individual members in MOGP, nor the off-EEL algorithm, can accomplish this during the (primary) training process. However, these evolved composite solutions do not perform as well on the unseen test set. One approach to address this overfitting uses an extra validation set to evolve the CSVote solutions. While this validation set only slightly improves ensemble performances on the test sets (compared to without), the CSVote ensembles using the validation set show substantially better ensemble performances on both classes than off-EEL using the validation set.

8.3 Discussions

The previous sections have provided detailed discussions on the chapter related topics. This section provides further discussions on more general topics not specific to a particular chapter but general to the whole thesis and the GP and data mining/machine learning communities.

8.3.1 No "Best" Fitness Function in GP

While this thesis provides a thorough evaluation of the AUC performances and the training times using several different fitness functions in GP on the tasks, no one fitness function is recommended as the "best" on the tasks since each has different strengths and limitations. The choice of which fitness function to use for a given task depends on the goals/requirements of the end-user. As one might expect, longer training times for a particular fitness function typically produced better AUC performances on the tasks, e.g., as seen for the AUC-based fitness functions which produced the best AUC performances but also incurred the longest training times. In contrast, the new distance-based fitness function is much simpler to implement, and has significantly faster training times, than the AUC-based fitness functions; this is not longer than 2.5 minutes (on average) on even the largest data set. While the AUC for this fitness function is generally good, they are not better than AUC-based fitness function.

8.3.2 AUC in GP

The performance the evolved GP classifiers is evaluated using the AUC. While this is a useful measure in class imbalance scenarios, the traditional AUC measure makes no assumptions about which ROC points are "better" than others on a given ROC curve (in terms of their true positive and false positive rates). In other words, in the AUC, each point on an ROC curve is as good as every other point. This can be seen as a limitation as the AUC makes no attempt to relate different ROC points to one another e.g., in a similar way that the analysis in MOGP (in Chapter 5) considered solutions in the middle-region of the Pareto front better than end-region solutions (as the latter represents biased classification performances on the two classes). This means that it can be difficult to choose the "best" classification threshold to represent the minority and the majority class accuracies for a given classifier, e.g., to compare performances to other learning approaches such as the MOGP ensembles (as seen in Section 6.7.2 in Chapter 6). In Chapter 6, the ensemble performances, measured in terms of their minority and the majority class accuracies, are compared to the evolved SGP classifiers (from Chapter 4) when the classifiers are evaluated using their "default" classification threshold (zero). These conclusions may be different using another classification threshold to represent the minority and the majority class accuracies of a given classifier.

However, the use of another good, inclusive measure for evaluating classifier

performance can depend on the problem domain and/or the end-goal of the research; this is an open issue in machine learning.

8.3.3 MOGP vs Canonical GP

In binary classification, an ROC curve for a single classifier in canonical GP (SGP) and the set of classifiers on the Pareto front in MOGP represent the same *concept* in the objective-space, i.e., both capture the performance trade-offs between the minority and the majority classes in the objective-space (as discussed in Section 5.3.3). However, MOGP and SGP accomplish this in fundamentally different ways. In SGP, the AUC represents the performance of a *single* classifier (at different decision thresholds); whereas in MOGP, the Pareto front represents a set of classifiers (each using the same decision threshold). Due to this fundamental difference, the MOGP approach can have four advantages over SGP.

The first is that the MOGP Pareto front can provide more trade-off options than the ROC curves in SGP. This is because the fitness function in MOGP has a dual purpose in the evolution: pushing the Pareto fronts outwards towards the zenith point (using Pareto dominance), and encouraging diversity on the Pareto front (via the "crowding" measure) to ensure that fronts are well-populated and contain a good "spread" of solutions. In contrast, fitness functions in SGP which aim to maximise the AUC of the evolved classifiers do not explicitly enforce this dual aspect in the evolution. This means that high AUC in an evolved SGP classifier does not guarantee that the ROC curve will have many different points that are "spread out" along the curve, since high AUC can be achieved using only a few good ROC points. However, as the diversity of the Pareto fronts in MOGP is not explicitly compared to the ROC curves in SGP in the thesis, this aspect is not fully explored but represents an important direction for future work.

The second advantage for MOGP is the uncomplicated way that the evolved Pareto fronts can be used in practice. In MOGP, the end-user can readily select a classifier from the evolved Pareto front with the desired performance trade-off on the two classes (as all classifiers are evaluated using the same "default" class threshold). In contrast, in SGP a classifier must be evaluated using the corresponding class threshold that is used to bias the final classification decision, to obtain the desired performance trade-off on the two classes.

The third advantage is that the Pareto front of solutions can be combined into an ensemble to further improve classification performances on the two classes (as shown in Chapter 6 of this thesis).

The fourth advantage is that additional learning objectives can easily be incorporated into MOGP since these learning objectives are treated independently in fitness (via Pareto dominance). While this can also be accomplished in SGP, some *a priori* knowledge is required about how to combine the multiple objectives together into a scalar (fitness) value, e.g., using an aggregation function such as Eq. (2.3) discussed in Chapter 2 (on page 27 in Section 2.4.1).

For these reasons, this thesis recommends MOGP over canonical SGP.

8.3.4 Data Mining, Machine Learning and GP

This section highlights the relationships between the work in this thesis, and data mining/machine learning and GP. Classification with unbalanced data is emerging as a hot topic in data mining/machine learning due to the large number of real-world domains affected by class imbalance. In this thesis we enhance the GP learning technique to solve classification problems with unbalanced data reasonably well. Previously, canonical GP produced biased performances on classification tasks with unbalanced data. Using the new GP techniques in this thesis, GP shows competitive results (on both the minority and the majority classes) on tasks with varying levels of class imbalance and even outperforms two popular machine learning algorithms, NB and SVM, on tasks with particularly high levels of class imbalance. By enhancing the capabilities of the GP learning system itself to account for the naturally-occurring class imbalance inherent in a particular problem, machine learning practitioners are freed to focus on other improvements (e.g., parameter tuning) to further boost classification results and system performances (as apposed to spending time to choose/apply a good sampling policy to artificially re-balance the learning data before the training process).

8.4 Future Work

The section highlights key areas of future work.

8.4.1 Classification with Multiple-classes.

The scope of this research includes only binary classification tasks. While GP can deal with binary classification tasks very well, GP for multiple-class tasks is more difficult. In multiple-class tasks with unbalanced data, there may be one

majority class but more than one minority class, one minority class but more than one majority class, or a combination of these (i.e. multiple minority and majority classes). Unlike other popular classification algorithms like decision trees or neural networks, in traditional tree-based GP, careful consideration must be taken to determine how to map a GP classifier's numeric outputs to the set of class labels. In other words, determining a suitable classification strategy for multiple-class classification in non-trivial. This typically involves defining static class boundaries on the number line for each class label *a priori* [183][185], or dynamically assigning these class boundaries on a solution-by-solution basis [120][185]. However, finding appropriate class boundary regions (to separate the classes) can be difficult, particularly in tasks with a large number of classes.

An alternative strategy uses binary decomposition to transform a multiple-class task with k classes into $k - 1$ binary tasks, where a single class is selected and all other classes are collapsed/combined to form the other class [155][123]. A binary classifier is then trained using this split, and this process is repeated for each class. However, aggregating these binary classifiers together in the final step requires careful consideration. Another alternative uses different GP representations better suited to multiple-class tasks (than tree-based GP), such as linear GP where each class is represented by its own output register [28][58].

Assuming that a suitable multiple-class classification strategy is in place (e.g. using one of the above strategies), the GP methods proposed by this thesis can easily be adapted for multiple-class tasks, since these methods all work at the *fitness level* in the optimisation process, by endeavouring to treat each class *separately* using the fitness fitness. In SGP, the new measures in the fitness function strive to measure the level of separability between the different classes, irrespective of the number of classes, and aim to evolve classifiers with equally good accuracy rates on all classes.

Similarly, the Pareto-based fitness function in MOGP also ensures that each class in multiple-class tasks is treated *independently* in the evolution, since the accuracy of each class is a seperate EMO objective. However, the performance of some EMO algorithms may deteriorate when the number of learning objectives increases (e.g. beyond the typical case of three of four objectives), due to large number of potential trade-off solutions (i.e. many non-dominated solutions along the objectives) [42][101][171]. In [101], scalability in EMO is discussed in more detail, and several useful techniques are proposed to address this.

The information theoretic measures in the fitness function for evolving ensembles in MOGP (i.e. NCL and PFC) can also be easily adapted for multiple-

class tasks, since the diversity of each class is measured *separately*. However, this approach also suffers from a similar scalability constraint beyond the typical case of three or four objectives (as discussed above).

Assuming that a suitable classification strategy is in place, the CSVote composite solutions can also be easily adapted for ensemble selection with multiple classes. This is because this method combines the predicted *class labels* of base learners using the majority vote strategy, and the fitness function to evolve composite solutions treats the classification accuracy of each class as equally important. However, the terminal set in the CSLogic composite solutions must first be adapted for multiple classes, in particular, the $\wedge$ and $\vee$ terminal symbols since these assume binary class labels.

Evaluating the new methods proposed in this thesis on multiple-class tasks with unbalanced data represents an important direction for future work.

8.4.2 Canonical SGP

Performance Measures.

The AUC is a useful measure in class imbalance scenarios. The conclusions about the GP fitness functions are relative to the AUC. However, as discussed earlier, the AUC also has some limitations. A weighted AUC [174] measure provides a useful starting point to address some of these limitations, as certain regions of the ROC space (e.g. the middle region of ROC curve) can be given a greater importance than other regions (e.g. end regions). Future work would evaluate the fitness functions in GP (from Chapter 4) using this weighted AUC measure, and compare how this measure ranks the different fitness functions compared to the traditional AUC. Another limitation of the AUC is that it must be adapted for multi-class classification [83][146]. Alternatively, a variety of other performance measures can also be used to evaluate and compare the different GP fitness functions (and SVM and NB) [69][34]. As mentioned earlier, the use of another good, inclusive measure for evaluating classifier performance can depend on the end-goal of the research and is an open issue in machine learning.

8.4.3 MOGP

Other Components in EMO Search.

This study focuses on one main aspect in the EMO learning algorithm, i.e., Pareto dominance in the fitness function. An interesting direction for future work would

be to investigate whether the performance of the evolved Pareto fronts in MOGP can be improved using other EMO components, such as the following.

A *strong* Pareto dominance relation provides an alternative to the *weak* Pareto dominance relation used in MOGP. In strong Pareto dominance, exactly one solution maps to a given point in the objective-space; whereas in weak Pareto dominance, many solutions can map to the same point in the objective-space.

The crowding measure in MOGP uses the distance to a solution's nearest neighbours. Other crowding measures in fitness (such as [26], [75] and [171]) can affect the selection process and the diversity of the Pareto fronts [51][113]. What difference (if any) will these crowding measures might have on the evolved Pareto fronts in MOGP (compared to the current approach)?

In MOGP, a fixed-size archive population is used. Some EMO approaches allow variable-sized archive populations [43] or use archive populations that are persistent across different replications [156], to ensure that no non-dominated individuals are lost over all generations.

Additional Learning Objectives in MOGP.

In a similar As discussed earlier, MOGP can also be used for classification tasks with more than two unbalanced classes, as the objectives are treated independently in the learning algorithm, provided that a suitable multiple-class classification strategy is also used.

A useful new direction of future work includes incorporating other objectives such as a model regularization term (e.g. program size of genetic program solutions) into MOGP, e.g., for parsimony pressure in the evolution. Inclusion of a a regularization terms would investigate whether smaller, less-complex solutions have better generalisation ability on the Pareto front on these and other tasks. However, as discussed above, the performance of some EMO algorithms may deteriorate when the number of objectives increases beyond the three of four objectives, due to large number of candidate (trade-off) solutions along the objectives.

8.4.4 Ensemble Learning in GP

Ensemble Diversity.

Chapter 6 suggests that the PFC measure promotes better diversity between individuals than NCL on the tasks. Incorporating NCL into the objective

values before Pareto ranking is done (similar to the way PFC is calculated) using a two-phase Pareto ranking approach (as NCL requires the output of the non-dominated front in its calculation), or new improvements to the NCL measure (such as the root quartic NCL [126][125]), may also improve ensemble performances. Improving the NCL, and developing new measures for diversity, is left for future work.

This thesis uses the ensemble performances on the test set to compare the diversity measures in MOGP. More theoretical analysis using measures that estimate the level of diversity in the populations during the evolution (such as the Q-statistic or Q-value [169][168]) may also provide useful insights into how these measures create diversity in MOGP. This would be an interesting avenue for future work.

Composite Solutions with Real-valued Outputs.

As the evolved composite solutions (in Chapter 7) act on the class decisions of the individual members, the ensemble output is also a class label. This restricts the type of fitness function that can be used to evolve composite solutions. Adapting the composite solutions to output a real-valued number (for a given input instance) can provide a finer-grain fitness landscape and may improve the generalisation ability of the pruned ensembles. This can be implemented by using the "confidence" of the ensemble (on a given input) as its output where the "confidence" is the ratio of correct to incorrect votes by the individual members within a composite solution. High confidence values imply that more individuals agree on the predicted class compared to low confidence values. A good fitness function can then be developed to maximise the ensemble *confidence* on the training set. Using real-valued outputs for the composite solutions also allows a classification threshold to be introduced to bias the ensemble's decision (on a given input), where different classification thresholds can be used to produce an ROC curve.

Comparison with Bagging and Boosting Approaches.

As bagging and boosting with balanced bootstrap sampling is a common approach in class imbalance, future work would compare the MOGP approach to bagging and boosting on these (and other) tasks.

8.4.5 GP in General

More Detailed Program Analysis.

Analysing several evolved genetic programs in SGP and MOGP revealed some interesting insights into how GP learns to solve a particular problem. However, this analysis only focused on a few evolved genetic programs on one particular task. An interesting exercise for future work would conduct a more detailed program analysis of the SGP and MOGP solutions, e.g., by collecting more detailed information/statistics about the evolved programs and examining these trends across multiple runs and tasks. However, this represents a difficult challenge, particularly in MOGP since multiple programs are evolved in a single run, and there are multiple runs to consider. Another interesting aspect of program analysis for future work would be to examine the behaviour of different subtrees within a given program using actual output values (e.g. when the program is evaluated on the data instances). This can help to identify good building blocks and also provide useful insights into which subtrees do not contribute to the overall performance of the program.

Evolutionary Parameters.

Chapter 4 shows that tweaking the evolutionary parameters in GP (e.g. increasing the population size and maximum program depth) to increase the search space explored by GP can improve system performances. However, the improved GP parameters are only evaluated on one data set. Future work would investigate if these (and other) evolutionary parameters can also improve classification performances on other tasks. Similarly, increasing the search space in MOGP using different evolutionary parameters might also improve the Pareto front on the tasks. This, in turn, might then improve ensemble performances since the ensemble is at least as good is its individual members.

More Unbalanced Data Sets

In future work, these GP methods would be evaluated on more classification tasks with unbalanced data.

Bibliography

[1] Pacific-asia knowledge discovery and data mining conference, 2009. "http://www.kdnuggets.com/datasets/competitions.html".

[2] ABBASS, H. Pareto neuro-ensembles. In *AI 2003: Advances in Artificial Intelligence*, vol. 2903 of *LNCS*. 2003, pp. 554–566.

[3] ABBASS, H. Pareto neuro-evolution: constructing ensemble of neural networks using multi-objective optimization. In *IEEE Congress on Evolutionary Computation* (2003), vol. 3, pp. 2074–2080.

[4] ABBASS, H. Pareto-optimal approaches to neuro-ensemble learning. In *Multi-Objective Machine Learning*, Y. Jin, Ed., vol. 16 of *Studies in Computational Intelligence*. 2006, pp. 407–427.

[5] ALEJO, R., GARCIA, V., SOTOCA, J. M., MOLLINEDA, R., AND SANCHEZ, J. Improving the classification accuracy of RBF and MLP neural networks trained with imbalanced samples. In *Intelligent Data Engineering and Automated Learning (IDEAL): 7th International Conference* (September 2006), Springer-Verlag, pp. 467–471.

[6] ALFARO-CID, E., SHARMAN, K., AND ESPARCIA-ALCAZAR, A. A genetic programming approach for bankruptcy prediction using a highly unbalanced database. In *Applications of Evolutionary Computing*, M. Giacobini, Ed., vol. 4448 of *LNCS*. Springer, 2007, pp. 169–178.

[7] ALPAYDIN, E. *Introduction to Machine Learning*. MIT Press, 2004.

[8] ASUNCION, A., AND NEWMAN, D. UCI Machine Learning Repository, 2007. University of California, Irvine, School of Information and Computer Sciences. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[9] BADRAN, K. M., AND ROCKETT, P. I. The roles of diversity preservation and mutation in preventing population collapse in multiobjective genetic programming. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation* (2007), GECCO '07, ACM, pp. 1551–1558.

[10] BANZHAF, W., NORDIN, P., KELLER, R. E., AND FRANCONE, F. D. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, 1998.

[11] BARANDELA, R., SANCHEZ, J., GARCIA, V., AND RANGEL, E. Strategies for learning in class imbalance problems. *Pattern Recognition 36*, 3 (2003), 849–851.

[12] BATISTA, G., PRATI, R. C., AND MONARD, M. C. Balancing strategies and class overlapping. In *Advances in Intelligent Data Analysis VI, 6th International Smposium on Intelligent Data Analysis, IDA 2005* (2005), A. F. Famili, J. N. Kok, J. M. Peña, A. Siebes, and A. J. Feelders, Eds., vol. 3646 of *LNCS*, Springer, pp. 24–35.

[13] BHOWAN, U., JOHNSTON, M., AND ZHANG, M. Differentiating between individual class performance in genetic programming fitness for classification with unbalanced data. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2009), IEEE Press, pp. 2802–2809.

[14] BHOWAN, U., JOHNSTON, M., AND ZHANG, M. Genetic programming for image classification with unbalanced data. In *Proceedings of 24th International Conference on Image and Vision Computing New Zealand* (2009), IEEE Press, pp. 316–321.

[15] BHOWAN, U., JOHNSTON, M., AND ZHANG, M. Multi-objective genetic programming for classification with unbalanced data. In *Proceedings of the 22nd Australasian Joint Conference on Artificial Intelligence* (2009), vol. 5866 of *LNCS*, Springer, pp. 370–380.

[16] BHOWAN, U., JOHNSTON, M., AND ZHANG, M. A comparison of classification strategies in genetic programming with unbalanced data. In *Proceedings of the 23rd Australasian Joint Conference on Artificial Intelligence* (2010), J. Li, Ed., vol. 6464 of *LNCS*, Springer, pp. 243–252.

[17] 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 42, 2 (2011), 406 –421.

[18] BHOWAN, U., JOHNSTON, M., AND ZHANG, M. Ensemble learning and pruning in multi-objective genetic programming for classification with unbalanced data. In *Proceedings of the 24th Australasian Joint Conference on Artificial Intelligence* (2011), D. Wang and M. Reynolds, Eds., vol. 7106 of *LNCS*, Springer, pp. 192–202.

[19] 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*. (Accepted, April 2012).

[20] BHOWAN, U., ZHANG, M., AND JOHNSTON, M. AUC analysis of the pareto-front using multi-objective GP for classification with unbalanced data. In *Proceedings of 2010 Genetic and Evolutionary Computation Conference* (2010), ACM, pp. 845–852.

[21] BHOWAN, U., ZHANG, M., AND JOHNSTON, M. Genetic programming for classification with unbalanced data. In *Proceedings of the 13th European Conference on Genetic Programming* (2010), vol. 6021 of *LNCS*, Springer, pp. 1–13.

[22] BHOWAN, U., ZHANG, M., AND JOHNSTON, M. Evolving ensembles in multi-objective genetic programming for classification with unbalanced data. In *Proceedings of Genetic and Evolutionary Computation Conference* (2011), ACM, pp. 1331–1339.

[23] BISHOP, C. M. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[24] BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2006.

[25] BONABEAU, E., DORIGO, M., AND THERAULAZ, G. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[26] BOT, M. C. J., BOELELAAN, D., AND LANGDON, W. B. Improving induction of linear classification trees with genetic programming. In *Genetic and Evolutionary Computation Conference* (2000), Morgan Kaufmann, pp. 403–410.

[27] BRADLEY, A. P. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition 30* (1997), 1145–1159.

[28] BRAMEIER, M., AND BANZHAF, W. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation 5*, 1 (2001), 17 –26.

[29] BRAMEIER, M., AND BANZHAF, W. Evolving teams of predictors with linear genetic programming. *Genetic Programming and Evolvable Machines 2*, 4 (2001), 381–407.

[30] BRAMEIER, M., AND BANZHAF, W. *Linear Genetic Programming*. Springer, New York, 2007.

[31] BREIMAN, L. Bagging predictors. *Machine Learning 24, issue 2* (1996), 123–140.

[32] BREIMAN, L., FRIEDMAN, J., OLSHEN, R., AND STONE, C. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

[33] BROWNLEE, J. *Clever Algorithms: Nature-Inspired Programming Recipes*. Lulu, 2011.

[34] CARUANA, R. Data mining in metric space: An empirical analysis of supervised learning performance criteria. In *Proceedings of ROC Analysis in AI Workshop (ECAI)* (2004), ACM Press, pp. 69–78.

[35] CHANDRA, A., AND YAO, X. Divace: Diverse and accurate ensemble learning algorithm. In *Intelligent Data Engineering and Automated Learning*, vol. 3177 of *LNCS*. Springer, 2004, pp. 619–625.

[36] CHANDRA, A., AND YAO, X. Ensemble learning using multi-objective evolutionary algorithms. *Journal of Mathematical Modelling and Algorithms 5* (2006), 417–445.

[37] CHAWLA, N., AND SYLVESTER, J. Exploiting diversity in ensembles: improving the performance on unbalanced datasets. In *Proceedings of the 7th International Conference on Multiple Classifier Systems* (2007), MCS, Springer-Verlag, pp. 397–406.

[38] CHAWLA, N. V., JAPKOWICZ, N., AND KOLCZ, A. Editorial: Special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations Newsletter 6* (June 2004), 1–6.

[39] CHEN, H., TINO, P., AND YAO, X. Predictive ensemble pruning by expectation propagation. *IEEE Transactions on Knowledge and Data Engineering 21* (2009), 999–1013.

[40] CHEN, H., AND YAO, X. Multiobjective neural network ensembles based on regularized negative correlation learning. *IEEE Transactions on Knowledge and Data Engineering 22*, 12 (2010), 1738–1751.

[41] CHEN, J.-X., CHENG, T.-H., CHAN, A. L. F., AND WANG, H.-Y. An application of classification analysis for skewed class distribution in therapeutic drug monitoring – the case of vancomycin. In *In IDEAS Workshop on Medical Information Systems: The Digital Hospital* (2004), IEEE, pp. 35–39.

[42] COELLO COELLO, C., LAMONT, G., AND VELDHUIZEN, D. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic & Evolutionary Computation Series)*. Springer, 2007.

[43] CORNE, D. W., JERRAM, N. R., KNOWLES, J. D., AND OATES, M. J. PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)* (2001), L. Spector, E. D. Goodman, A. Wu, W. Langdon, H. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds., Morgan Kaufmann Publishers, pp. 283–290.

[44] CORTES, C., AND VAPNIK, V. Support-vector networks. *Machine Learning 20* (1995), 273–297.

[45] COVER, T., AND HART, P. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory 13*, 1 (1967), 21–27.

[46] CURRY, R., LICHODZIJEWSKI, P., AND HEYWOOD, M. 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.

[47] DAM, H., ABBASS, H., LOKAN, C., AND YAO, X. Neural-based learning classifier systems. *IEEE Transactions on Knowledge and Data Engineering 20*, 1 (2008), 26–39.

[48] DE JONG, E. D., AND POLLACK, J. B. Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines 4* (2003), 211–233.

[49] DE JONG, E. D., WATSON, R. A., AND POLLACK, J. B. Reducing bloat and promoting diversity using multi-objective methods. In *In Proceedings of the Genetic and Evolutionary Computation Conference GECCO2001* (2001), Morgan Kaufmann, pp. 11–18.

[50] DE LA IGLESIA, B., REYNOLDS, A., AND RAYWARD-SMITH, V. J. Developments on a multi-objective metaheuristic (MOMH) algorithm for finding interesting sets of classification rules. In *Third International Conference on Evolutionary Multi-Criterion Optimization, Mexico*, C. Coello Coello, A. H. Aguirre, and E. Zitzler, Eds., vol. 3410 of *LNCS*. pp. 826–840.

[51] DEB, K., AND JAIN, S. Running performance metrics for evolutionary multi-objective optimization. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02), Singapore.* (2002), vol. 1, pp. 13–20.

[52] DEB, K., MOHAN, M., AND MISHRA, S. Evaluating the ϵ-domination based multi-objective evolutionary algorithm for a quick computation of pareto-optimal solutions. *Evolutionary Computation 13* (2005), 501–525.

[53] DEB, K., PRATAP, A., AGARWAL, S., AND MEYARIVAN, T. A fast elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation 6* (2000), 182–197.

[54] DIETTERICH, T. G. Ensemble methods in machine learning. In *Multiple Classifier Systems, LNCS* (2000), vol. 1857, Springer-Verlag, pp. 1–15.

[55] DORIGO, M., AND GAMBARDELLA, L. M. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation 1*, 1 (1997), 53–66.

[56] DORIGO, M., AND STTZLE, T. *Ant Colony Optimization*. MIT Press, 2004.

[57] DOUCETTE, J., AND HEYWOOD, M. I. GP classification under imbalanced data sets: Active sub-sampling and AUC approximation. In *Proceedings of 11th European Conference in Genetic Programming (EuroGP 08)* (2008), pp. 266–277.

[58] DOWNEY, C., ZHANG, M., AND LIU, J. Parallel linear genetic programming for multi-class classification. *Genetic Programming and Evolvable Machines*, (To appear) (2012).

[59] DUDA, R. O., HART, P. E., AND STORK, D. G. *Pattern Recognition*, 2nd ed. Wiley, New York, 2001.

[60] EGGERMONT, J., EIBEN, A., AND VAN HEMERT, J. Adapting the fitness function in GP for data mining. In *Genetic Programming, 2nd European Workshop* (1999), vol. 1598 of *LNCS*, pp. 193–202.

[61] EGGERMONT, J., EIBEN, A., AND VAN HEMERT, J. A comparison of genetic programming variants for data classification. *IDA 99, LNCS 1642* (1999), 281–290.

[62] EGGERMONT, J., KOK, J. N., AND KOSTERS, W. A. Genetic programming for data classification: Partitioning the search space. In *Proceedings of the 2004 Symposium on Applied Computing (ACM'SAC 04)* (2004), ACM, pp. 1001–1005.

[63] EKÁRT, A., AND NÉMETH, S. Z. Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines 2* (2001), 61–73.

[64] 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.

[65] EVERSON, R., AND FIELDSEND, J. Multi-objective optimisation for receiver operating characteristic analysis. In *Multi-Objective Machine Learning*, Y. Jin, Ed. Springer, 2006, ch. 23, pp. 532–556.

[66] EVERSON, R. M., AND FIELDSEND, J. E. Multiobjective optimization of safety related systems: An application to short-term conflict alert. *IEEE Transactions on Evolutionary Computation 10*, 2 (2006), 187–198.

[67] FAWCETT, T., AND PROVOST, F. Adaptive fraud detection. *Data Mining and Knowledge Discovery 1* (1997), 291–316.

[68] FAYYAD, U. M., PIATETSKY-SHAPIRO, G., AND SMYTH, P. From data mining to knowledge discovery: an overview. *Advances in knowledge discovery and data mining* (1996), 1–34.

[69] FERRI, C., HERNÁNDEZ-ORALLO, J., AND MODROIU, R. An experimental comparison of performance measures for classification. *Pattern Recognition Letters. 30*, 1 (2009), 27–38.

[70] FIELDSEND, J., AND EVERSEN, R. Multiobjective supervised learning. In *Multiobjective Problem Solving from Nature: From Concepts to Applications (Natural Computing Series)*, J. Knowles, D. Corne, and K. Deb, Eds., 1 ed. Springer, 2008, ch. 3, pp. 155–176.

[71] FISHER, R. A. *Statistical Methods for Research Workers*, 14th ed. Oliver and Boyd, 1970.

[72] FOGEL, L. J., OWENS, A. J., AND WALSH, M. J. *Artificial Intelligence through Simulated Evolution*. John Wiley, 1966.

[73] FONSECA, C., AND FLEMING, P. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms* (1993), pp. 416–423.

[74] FONSECA, C., AND FLEMING, P. An overview of evolutionary algorithms in multiobjective optimization. Tech. rep., July 1994. Dept of Automatic Control and Systems Engineering, University of Sheffield.

[75] FRIEDRICH, T., KROEGER, T., AND NEUMANN, F. Weighted preferences in evolutionary multi-objective optimization. In *Proceedings of the 24rd Australasian Joint Conference on Artificial Intelligence* (2011), vol. 7106 of *LNCS*, Springer, pp. 192–202.

[76] GAGNÉ, C., SEBAG, M., SCHOENAUER, M., AND TOMASSINI, M. Ensemble learning for free with evolutionary algorithms? In *Proceedings of Genetic and Evolutionary Computation Conference* (2007), ACM Press, pp. 1782–1789.

[77] GATHERCOLE, C., AND ROSS, P. Dynamic training subset selection for supervised learning in genetic programming. In *Parallel Problem Solving from Nature (PPSN III)*, Y. Davidor, H.-P. Schwefel, and R. Manner, Eds., vol. 866 of *LNCS*. Springer, 1994, pp. 312–321.

[78] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.

[79] GOLDBERG, D. E., AND HOLLAND, J. H. Genetic algorithms and machine learning. *Machine Learning 3*, 2 (1988), 95–99.

[80] GRAY, H., AND MAXWELL, R. Genetic programming for classification of brain tumours from nuclear magnetic resonance biopsy spectra. *Genetic Programming 1996: Proceedings of the First Annual Conference* (1996), 424–430.

[81] GUSTAFSON, S., BURKE, E. K., AND KRASNOGOR, N. On improving genetic programming for symbolic regression. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (2005), vol. 1, IEEE Press, pp. 912–919.

[82] HALL, M., FRANK, E., HOLMES, G., PFAHRINGER, B., REUTEMANN, P., AND WITTEN, I. H. The WEKA data mining software: An update. *SIGKDD Explorations 11 (1)* (2009).

[83] HAND, D., AND TILL, R. A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine Learning 45* (2001), 171–186.

[84] HANLEY, J. A., AND MCNEIL, B. J. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology 143*, 1 (1982), 29–36.

[85] HERNANDEZ, S., SAEZ, D., AND MERY, D. Neuro-fuzzy method for automated defect detection in aluminium castings. *Image Analysis and Recognition, LNCS 3212* (2004), 826–833.

[86] HOLLAND, J. *Adaptation in natural and artificial systems*. MIT Press, 1992.

[87] HOLLAND, J. H. Adaptation. In *Progress in Theoretical Biology IV*. New York Academic Press, 1976, pp. 263–293.

[88] HOLMES, J. H. Differential negative reinforcement improves classifier system learning rate in two-class problems with unequal base rates. In *Proceedings of the Third Annual Conference on Genetic Programming* (1998), pp. 635–644.

[89] HOWARD, D., ROBERTS, S., AND BRANKIN, R. Target detection of SAR imagery by genetic programming. *Advances in Engineering Software 30* (1999), 303–311.

[90] HUANG, J., AND LING, C. X. Constructing new and better evaluation measures for machine learning. In *Proceedings of the 20th international joint conference on Artifical intelligence* (2007), Morgan Kaufmann, pp. 859–864.

[91] HULSE, J. V., KHOSHGOFTAAR, T. M., AND NAPOLITANO, A. Experiment perspectives in learning from imbalanced data. *Proceedings of the 24th International Conference on Machine Learning* (2007), 435–492.

[92] I MANSILLA, E. B., AND I GUIU, J. G. MOLeCS: Using multiobjective evolutionary algorithms for learning. In *Evolutionary Multi-Criterion Optimization*, E. Zitzler, L. Thiele, K. Deb, C. Coello Coello, and D. Corne, Eds., vol. 1993 of *LNCS*. Springer, 2001, pp. 696–710.

[93] JAPCOWICZ, N., AND STEPHEN, S. The class imbalance problem: A systematic study. *Intelligent Data Analysis 6*, 5 (2002), 429–450.

[94] JENSEN, F. V. *Introduction to Bayesian Networks*. Springer-Verlag New York, 1996.

[95] JIN, Y., AND SENDHOFF, B. Pareto-based multiobjective machine learning: An overview and case studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 38*, 3 (2008), 397–415.

[96] JONG, E. D., AND POLLACK, J. B. Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines 4*, 3 (2003), 211–233.

[97] JONG, K. A. D. *Evolutionary computation - a unified approach*. MIT Press, 2006.

[98] KENNEDY, J., AND EBERHART, R. C. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks VI* (1995), Morgan Kaufmann, pp. 1942–1948.

[99] KENNEDY, J., AND EBERHART, R. C. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[100] KENNETH, S., STORN, R., AND LAMPINEN, J. *Differential Evolution A Practical Approach to Global Optimization*. Springer, 2005.

[101] KNOWLES, J., AND CORNE, D. Quantifying the effects of objective space dimension in evolutionary multiobjective optimization. In *Evolutionary Multi-Criterion Optimization*, vol. 4403 of *Lecture Notes in Computer Science*. 2007, pp. 757–771.

[102] KNOWLES, J., THIELE, L., AND ZITZLER, E. A tutorial on the performance assessment of stochastic multiobjective optimizers. Tech. rep., February 2006. No. 214, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich.

[103] KNOWLES, J. D., AND CORNE, D. W. Approximating the nondominated front using the pareto archived evolution strategy. *IEEE Transactions on Evolutionary Computation 8*, 2 (2000), 149–172.

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

[105] KOZA, J. R., ANDRE, D., BENNETT III, F. H., AND KEANE, M. *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman, 1999.

[106] KOZA, J. R., KEANE, M. A., STREETER, M. J., MYDLOWEC, W., YU, J., AND LANZA, G. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.

[107] KUBAT, M., AND MATWIN, S. Addressing the curse of imbalanced training sets: one-sided selection. In *Proceedings of the Fourteenth International Conference on Machine Learning* (1997), pp. 179–186.

[108] KUPINSKI, M., AND ANASTASIO, M. Multiobjective genetic optimization of diagnostic classifiers with implications for generating receiver operating characteristic curves. *IEEE Transactions on Medical Imaging 18* (1999), 675–685.

[109] LANGDON, W. B. Size fair and homologous tree crossovers for tree genetic programming. *Genetic Programming and Evolvable Machines 1* (2000), 95–119.

[110] LANGDON, W. B., AND BUXTON, B. Genetic programming for combining classifers. In *Proceedings of 2001 Genetic and Evolutionary Computation Conference* (2001), Morgan Kaufmann, pp. 97–96.

[111] LANGDON, W. B., AND BUXTON, B. Genetic programming for improved receiver operating characteristics. In *Multiple Classifier Systems*, J. Kittler and F. Roli, Eds., vol. 2096 of *LNCS*. 2001, pp. 68–77.

[112] LANGDON, W. B., AND BUXTON, B. F. Genetic programming for mining dna chip data from cancer patients. *Genetic Programming and Evolvable Machines 5* (2004), 251–257.

[113] LAUMANNS, M., THIELE, L., DEB, K., AND ZITZLER, E. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary Compututation 10*, 3 (2002), 263–282.

[114] LAVRA, N., AND FAWCETT, T. Editorial: Data mining lessons learned. In *Machine Learning* (2004).

[115] LAW, A. M., AND KELTON, W. D. *Simulation Modeling and Analysis*, 3rd ed. McGraw-Hill Higher Education, 2000.

[116] LAWRENCE, S., BURNS, I., BACK, A. D., TSOI, A. C., AND GILES, L. C. Neural network classification and prior class probabilities. In *Neural Networks: Tricks of the Trade* (1998), Springer-Verlag, pp. 299–313.

[117] LING, C., , LING, C. X., AND LI, C. Data mining for direct marketing: Problems and solutions. In *In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)* (1998), AAAI Press, pp. 73–79.

[118] 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 (2009), 539–550.

[119] LIU, Y., AND YAO, X. Negatively correlated neural networks can produce best ensembles. *Australian Journal of Intelligent Information Processing Systems 4* (1997), 176–185.

[120] LOVEARD, T., AND CIESIELSKI, V. Representing classification problems in genetic programming. In *Proceedings of the 2001 Congress on Evolutionary Computation* (2001), vol. 12, IEEE Press, pp. 1070–1077.

[121] LOWRY, R. Statistical table entries. VassarStats Website for Statistical Computation, Vassar College, New York, 2012. "`http://faculty.vassar.edu/lowry/tabs.html`".

[122] MCCARTHY, K., ZABAR, B., AND WEISS, G. Does cost-sensitive learning beat sampling for classifying rare classes. *Proceedings of the 1st international workshop on Utility-based data mining* (2005), 69–77.

[123] MCINTYRE, A., AND HEYWOOD, M. Multi-objective competitive coevolution for efficient GP classifier problem decomposition. In *IEEE International Conference on Systems, Man and Cybernetics* (2007), pp. 1930 –1937.

[124] MCINTYRE, A., AND HEYWOOD, M. Classification as clustering: A pareto cooperative-competitive GP approach. *Evolutionary Computation 19*, 1 (2011), 137–166.

[125] MCKAY, R., AND ABBASS, H. Anti-correlation: A diversity promoting mechanisms in ensemble learning. *The Australian Journal of Intelligent Information Processing Systems 7*, 3/4 (2001), 139–149.

[126] MCKAY, R., AND ABBASS, H. Anticorrelation measures in genetic programming. In *Australasia-Japan Workshop on Intelligent and Evolutionary Systems* (2001), pp. 45–51.

[127] MILLER, J. F. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference* (1999), pp. 14–17.

[128] MILLER, J. F. *Cartesian Genetic Programming*. Springer Berlin, Heidelberg, 2011.

[129] MITCHELL, T. *Machine Learning*. McGraw Hill, 1997.

[130] MONARD, M. C., AND BATISTA, G. Learning with skewed class distributions. *Advances in Logic, Artificial Intelligence and Robotics* (2002), 173–180.

[131] MUNDER, S., AND GAVRILA, D. An experimental study on pedestrian classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 11 (2006), 1863–1868.

[132] MUTTER, S., PFAHRINGER, B., AND HOLMES, G. The positive effects of negative information: Extending one-class classification models in binary proteomic sequence classification. In *Proceedings of the 22nd Australasian Joint Conference on Artificial Intelligence (AI 09)* (2009), vol. LNAI 5866, Springer, pp. 260–269.

[133] 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), GECCO '09, ACM, pp. 1027–1034.

[134] NESHATIAN, K., ZHANG, M., AND ANDREAE, P. Genetic programming for feature ranking in classification problems. In *Proceedings of the 7th International Conference on Simulated Evolution and Learning* (2008), SEAL '08, Springer-Verlag, pp. 544–554.

[135] NIKULIN, V., MCLACHLAN, G., AND NG, S. K. Ensemble approach for the classification of imbalanced data. In *Proceedings of the 22nd Australasian Joint*

Conference on Artificial Intelligence (AI 09) (2009), vol. 5866 of *LNAI*, Springer, pp. 260–269.

[136] O'NEILL, M., AND RYAN, C. *Grammatical Evolution*. Springer, 2003.

[137] OPITZ, D., AND MACLIN, R. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research 11* (1999), 169–198.

[138] OPITZ, D. W., AND SHAVLIK, J. W. Generating accurate and diverse members of a neural-network ensemble. In *Advances in Neural Information Processing Systems* (1996), MIT Press, pp. 535–541.

[139] ORRIOLS, A., AND BERNADO-MANSILLA, E. Class imbalance problem in UCS classifier system: Fitness adaptation. In *IEEE Congress on Evolutionary Computation* (2005), pp. 604–611.

[140] PARROT, D., LI, X., AND CIESIELSKI, V. Multi-objective techniques in genetic programming for evolving classifiers. *In Proceedings of the 2005 Congress on Evolutionary Computation (CEC2005)* (September 2005), 1141–1148.

[141] PATTERSON, G., AND ZHANG, M. Fitness functions in genetic programming for classification with unbalanced data. In *Proceedings of the 20th Australasian Joint Conference on Artificial Intelligence* (2007), vol. 4830 of *LNCS*, pp. 769–775.

[142] PAZZANI, M., MERZ, C., MURPHY, P., ALI, K., HUME, T., AND BRUNK, C. Reducing misclassification costs. In *Proceedings of the 11th International Conference of Machine Learning* (1994), Morgan Kaufmann, pp. 217–225.

[143] PEDNAULT, E., ROSEN, B., AND APTE, C. Handling imbalanced data sets in insurance risk modeling. In *Workshops at the Seventeenth National Conference on Artificial Intelligence, Learning from Imbalanced Data Sets* (2005), AAAI Press, pp. 58–63.

[144] POLI, R., LANGDON, W. B., AND MCPHEE, N. F. *A field guide to genetic programming*. Published via `http://lulu.com`, 2008.

[145] PRATI, R. C., BATISTA, G., AND MONARD, M. C. Class imbalances versus class overlapping: An analysis of a learning system behavior. In *Advances in Artificial Intelligence, Third Mexican International Conference on Artificial Intelligence* (2004), vol. 2972 of *LNCS*, pp. 312–321.

[146] PROVOST, F., AND DOMINGOS, P. Tree induction for probability-based rankings. *Machine Learning 52*, 3 (2003).

[147] QUINLAN, R. Induction of decision trees. *Machine Learning 1* (1986), 81–106.

[148] QUINLAN, R. *Programs for Machine Learning*. Morgan Kaufmann, 1993.

[149] ROSSET, S. Model selection via the AUC. In *Proceedings of the Twenty-First International Conference on Machine Learning.* (2004), ACM Press, pp. 89–97.

[150] RUSSELL, S., AND NORVIG, P. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, 2003.

[151] RYAN, C., COLLINS, J. J., AND O'NEILL, M. Grammatical evolution: Evolving programs for an arbitrary language. In *EuroGP* (1998), pp. 83–96.

[152] SCHAFFER, J. D. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms* (Hillsdale, NJ, USA, 1985), pp. 93–100.

[153] SCHWEFEL, H.-P. *Evolution and Optimum Seeking*. Wiley, New York, 1981.

[154] SMART, W., AND ZHANG, M. Classification strategies for image classification in genetic programming. In *Proceeding of Image and Vision Computing Conference New Zealand* (2003), pp. 402–407.

[155] SMART, W., AND ZHANG, M. Using genetic programming for multiclass classification by simultaneously solving component binary classification problems. In *Proceedings of the 8th European conference on Genetic Programming* (2005), vol. 3447, pp. 227–239.

[156] SMITS, G. F., AND KOTANCHEK, M. Pareto front exploitation in symbolic regression. In *Genetic Programming Theory and Practice II*, U.-M. O'Reilly, T. Yu, R. Riolo, and B. W. (Eds.), Eds., vol. 8 of *Genetic Programming*. 2005, ch. 17, pp. 283–299.

[157] SONG, D., HEYWOOD, M., AND ZINCIR-HEYWOOD, A. Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE Transactions on Evolutionary Computation 9*, 3 (2005), 225–239.

[158] SONG, D., HEYWOOD, M. I., AND ZINCIR-HEYWOOD, A. N. A linear genetic programming approach to intrusion detection. In *In Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO'03)* (2003), vol. 2724 of *LNCS*, pp. 2325–2336.

[159] STOLFO, S. J., FAN, D. W., LEE, W., PRODROMIDIS, A. L., AND CHAN, P. K. Credit card fraud detection using meta-learning: Issues and initial results. In *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management* (1997), pp. 83–90.

[160] STORN, R., AND PRICE, K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization 11* (December 1997), 341–359.

[161] SUNG, K.-K. *Learning and Example Selection for Object and Pattern Recognition*. PhD thesis, AI Laboratory and Center for Biological and Computational Learning, MIT, 1996.

[162] SUTTORP, T., AND IGEL, C. Multi objective support vector machines. In *Multi-Objective Machine Learning*, Y. Jin, Ed. Springer, 2006, ch. 9, pp. 199–220.

[163] TACKETT, W. A., AND CHAR, K. G. Genetic programming applied to image discrimination. In *Handbook of Evolutionary Computation*, T. Baeck, D. B. Fogel, and Z. Michalewicz, Eds. Oxford University Press, 1997.

[164] TANG, Y., ZHANG, Y.-Q., CHAWLA, N., AND KRASSER, S. SVM modeling for highly imbalanced classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 39*, 1 (2009), 281–288.

[165] THOMASON, R., AND SOULE, T. Novel ways of improving cooperation and performance in ensemble classifiers. In *Proceedings of Genetic and Evolutionary Computation Conference* (2007), ACM, pp. 1708–1715.

[166] TUKEY, J. W. Components in regression. *Biometrics 7* (1951), 33–69.

[167] VAPNIK, V. *The Nature of Statistical Learning Theory*. Springer, 1995.

[168] WANG, S., TANG, K., AND YAO, X. Diversity exploration and negative correlation learning on imbalanced data sets. In *International Joint Conference on Neural Networks* (2009), pp. 3259–3266.

[169] WANG, S., AND YAO, X. Diversity analysis on imbalanced data sets by using ensemble models. In *IEEE Symposium on Computational Intelligence and Data Mining* (2009), pp. 324–331.

[170] WANG, S., AND YAO, X. Theoretical study of the relationship between diversity and single-class measures for class imbalance learning. In *Proceedings of the IEEE International Conference on Data Mining Workshops* (2009), ICDMW, pp. 76–81.

[171] WANG, Z., TANG, K., AND YAO, X. Multi-objective approaches to optimal testing resource allocation in modular software systems. *IEEE Transactions on Reliability 59*, 3 (2010), 563–575.

[172] WEISS, G., AND PROVOST, F. The effect of class distribution on classifier learning: An empirical study. Tech. rep., Department of Computer Science, Rutgers University., 2001.

[173] WEISS, G. M., AND PROVOST, F. Learning when training data are costly: The effect of class distribution on tree induction. *Journal of Artificial Intelligence Research 19* (2003), 315–354.

[174] WENG, C. G., AND POON, J. A new evaluation measure for imbalanced datasets. In *Proceedings of the Seventh Australasian Data Mining Conference (AusDM)* (2008), pp. 27–32.

[175] WHIGHAM, P. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications* (1995), Morgan Kaufmann, pp. 33–41.

[176] WINKLER, S., AFFENZELLER, M., AND WAGNER, S. Advanced genetic programming based machine learning. *Journal of Mathematical Modelling and Algorithms 6 (3)* (2007), 455–480.

[177] WINKLER, S. M., AFFENZELLER, M., AND WAGNER, S. Using enhanced genetic programming techniques for evolving classifiers in the context of medical diagnosis. *Genetic Programming and Evolvable Machines 10* (2009), 111–140.

[178] WITTEN, I. H., AND FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.

[179] YAN, L., DODIER, R., MOZER, M. C., AND WOLNIEWICZ, R. Optimizing classifier performance via the Wilcoxon-Mann-Whitney statistic. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML 03)* (2003), pp. 848–855.

[180] YAO, X., AND LIU, Y. Making use of population information in evolutionary artificial neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 28*, 3 (1998), 417–425.

[181] ZHANG, M. *A Domain Independent Approach to 2D Object Detection Based on the Neural and Genetic Paradigms*. PhD thesis, 2000. Departmentment of Computer Science, RMIT University, Melbourne.

[182] ZHANG, M., AND BHOWAN, U. Program size and pixel statistics in genetic programming for object detection. *Evolutionary Computation in Image Analysis and Signal Processing 3005* (2004), 377–386.

[183] ZHANG, M., CIESIELSKI, V., AND ANDREAE, P. A domain-independent window approach to multiclass object detection using genetic programming. *EURASIP Journal on Advances in Signal Processing 2003*, 8 (2003), 841–859.

[184] ZHANG, M., AND LETT, M. Genetic programming for object detection: Improving fitness functions and optimising training data. *IEEE Intelligent Informatics Bulletin 7*, 1 (2006), 12–21.

[185] ZHANG, M., AND SMART, W. Multiclass object classification using genetic programming. In *Applications of Evolutionary Computing*, vol. 3005 of *LNCS*. Springer Berlin / Heidelberg, 2004, pp. 369–378.

[186] ZHANG, M., AND SMART, W. Using Gaussian distribution to construct fitness functions in genetic programming for multiclass object classification. *Pattern Recognition Letters 27*, 11 (2006), 1266–1274.

[187] ZHAO, H. A multi-objective genetic programming approach to developing pareto optimal decision trees. *Decision Support Systems. 43*, 3 (2007), 809–826.

[188] ZITZLER, E., LAUMANNS, M., AND THIELE, L. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. Tech. rep., 2001. TIK-Report 103, Department of Electrical Engineering, Swiss Federal Institute of Technology.

Appendix A

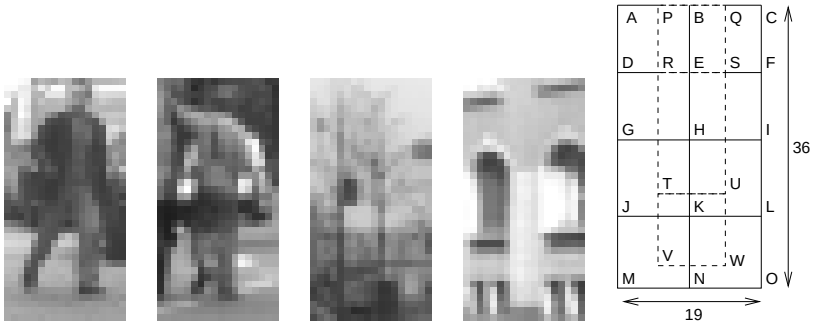
Benchmark Classification Data Sets

The following benchmark tasks with unbalanced data have been used in the experimental results throughout the thesis to evaluate the proposed GP methods. These are available at the UCI Repository of Machine Learning Databases [8] and the Intelligent Systems Lab at the University of Amsterdam [131].

Ionosphere (Ion). This data set contains 351 recorded radar signals collected using high-frequency antennas targeting free electrons in the ionosphere. There are 126 "good" signals (35.8%) and 225 "bad" signals (64.2%), a class imbalance ratio of approximately 1:3. Signals were processed using an autocorrelation function returning two attributes per pulse, giving 34 (real) features (F_1–F_{34}) [8].

Spect Heart (Spt). This data set contains 267 patient records derived from cardiac Single Proton Emmision Computed Tomography (Spect) images. There are 55 "abnormal" records (20.6%) and 212 "normal" records (79.4%), an imbalance ratio of approximately 1:4. Each SPECT image was processed to extract 44 continuous features, these were further pre-processed into 22 binary features (F_1–F_{22}) [8].

Pedestrian images (Ped). This data set contains 24,800 portable gray map (PGM) image cut-outs that are 19×36 pixels in size. These cut-outs include 4,800 pedestrian images (19.4%) and 20,000 (80.6%) background images, an imbalance ratio of approximately 1:4. Example pedestrian and background images are shown in Figure A.1(a). For image features, 22 low-level pixel statistics corresponding to the mean and variance of pixel values around 11 local regions within each image cut-out are extracted. The local regions correspond to generic equally-sized quadrants spread evenly around each image, and rectilinear regions placed around distinguishing regions in the image. These 22 features, F_1–F_{22}, represent the overall pixel brightness/intensity and the contrast of a given region.



(a) (b)

Figure A.1: (a) Example pedestrian (left two) and non-pedestrian image (right two), and (b) local image regions for extracting pixel statistical features.

The first eight pairs of mean and standard deviation pixel statistics, F_1–F_{16}, correspond to the eight equally sized quadrants, A-B-E-D, B-C-F-E, D-E-H-G, E-F-I-H, D-H-K-J, H-I-L-K, J-K-N-M, and K-L-O-N, in Figure A.1(b). The last three pairs of mean and standard deviation pixel statistics, F_{17}–F_{22}, correspond to the specialised rectilinear regions located around the head, body and leg regions, P-Q-S-R, R-S-U-T, and T-U-W-V, respectively, in Figure A.1(b).

Yeast (Yst_1 and Yst_2). This data set contains 1482 instances of protein localisation sites in yeast cells, with eight amino-acid sequences as numeric features (F_1–F_8) [8]. The problem has nine classes, each with a different degree of class imbalance. This data set is decomposed into *many* binary classification tasks using only one "main" (minority) class and *everything else* as the majority class. Two "main" classes are selected: Yst_1 has 244 examples from the mit class (16%), an imbalance ratio of 1:6. and Yst_2 has 163 examples from the *me3* class (11%), an imbalance ratio of 1:9. The mit class has the most minority class instances in this data set, while the *me3* class has one of the least.

Balance Scale (Bal). This data set contains 625 records generated to model psychological experiments in children. Each example is classified into three classes: the balance scale tipped to the right, left, or balanced. Two of these classes, *left* (46%) and *right* (46%), make up the vast majority of instances and are combined into a single (majority) class called "unbalanced" (92%). The remaining class, "balanced", is used as the minority class with 49 examples (8%). This corresponds to an imbalance ratio of approximately 1:12. There are four integer-based attributes corresponding to the left and right weights, and the left and right distances (F_1–F_4) [8].

Appendix B

Additional Material

B.1 Attainment Function in Attainment Surfaces

In the multi-objective GP (MOGP) approach, the outcome of a single MOGP run is a set of evolved solutions along the Pareto-approximated front; this is known as an approximation set. Each solution in an approximation set can be represented by a performance vector $\vec{x}$ where each element in $\vec{x}$ corresponds to the performance of the given Pareto front solution on each objective. As there are two objectives in the MOGP approach (minority and majority class accuracy), the cardinality of $\vec{x}$ is two. Therefore, the output of a single MOGP run can be represented as the set X of all individual performance vectors $\vec{x}$ in the approximation set, where n is the number of solutions in the approximation set, as shown below.

$$X = \{\vec{x_1}, \vec{x_2}, \vec{x_3}, ..., \vec{x_n}\} \tag{B.1}$$

As MOGP is a stochastic algorithm, a given run is repeated a number of times to assess the overall performance of the algorithm, and each replicate uses a different (random) starting seed. The means that a different approximation set X_i is returned from each replicate. Attainment surfaces summarises multiple approximation sets (returned from a series of EMO runs) as a single approximation set [102]. In attainment surfaces, each solution in the approximation sets is assigned an *attainment value* using an *attainment function*. This attainment value corresponds to the probability that a given solution is *attained* by the EMO system with respect to all runs; attainment values range between 1 and 0. The attainment value for a solution represents the probability that an EMO system will produce (or evolve) another solution that is better than, or equal to, the given solution on

all objectives (i.e. *weakly dominates* the given solution).

B.1.1 Attainment Function

An attainment function is used to calculate the attainment value (probability of being attained) of every solution in the approximation sets (generated over multiple runs). The attainment function uses a *weak dominance* Pareto relation to determine if a given solution is attained with respect to every other solution in the approximation sets. A solution p weakly dominates another solution q, denoted $p \preceq q$, if p is equal to or better than q on all objectives. Weak dominance can also apply to two *sets* of solutions, e.g., set Y weakly dominates set Z if every solution in set Z is weakly dominated by at least one solution in Y [102].

Formally, an attainment value for a single solution (performance vector $\vec{y}$) can be estimated from r independent runs by the attainment function as shown below.

$$Att(\vec{y}) = \frac{1}{r}\sum_{i=1}^{r} I(X_i, \vec{y}) \tag{B.2}$$

Where r is the total number of approximation sets (number of runs), X_i is the ith approximation set for a run, and $I(.)$ is the indicator function that determines if the solution represented by performance vector $\vec{y}$ is attained with respect to approximation set X_i. Indicator function $I(.)$ will evaluate to 1 if solution $\vec{y}$ is weakly dominated by set X_i, or zero otherwise, as shown below.

$$I(X_i, \vec{y}) = \begin{cases} 1 & \text{if } (X_i \preceq \{\vec{y}\}). \text{ i.e., } (\vec{x_1} \preceq \vec{y}) \vee (\vec{x_2} \preceq \vec{y}) \ldots \vee (\vec{x_i} \preceq \vec{y}) \\ 0 & \text{otherwise.} \end{cases}$$

In equation B.2, $I(.)$ corresponds to the probability that a given approximation set X_i weakly dominates the set made up of the single performance vector $\vec{y}$. Recall that set X_i will dominate set $\{\vec{y}\}$ if there exists one element in X_i that weakly dominates $\vec{y}$. Therefore indicator function $I(.)$ will return 1 if there exists a solution in X_i that is equal to or better than $\vec{y}$ on both objectives (weakly dominates), or zero otherwise. If 1 is returned, this means that $\vec{y}$ is attained with respect to approximation set X_i.

Table B.1: Average AUC ($\pm$ standard deviation) for fitness functions $Corr$ and $Dist$ using Ave-based approach for class ordering ($W = 2$) over 50 GP runs

Task	$Corr$	$Dist$
Ion	0.87 ± 0.03	0.86 ± 0.05
Spt	0.71 ± 0.07	0.73 ± 0.04
Ped	0.87 ± 0.03	0.87 ± 0.03
Yst_1	0.78 ± 0.03	0.79 ± 0.02
Yst_2	0.94 ± 0.03	0.93 ± 0.09
Bal	0.75 ± 0.10	0.76 ± 0.10

B.1.2 Attainment sets

Solutions with equivalent attainment values k are then be grouped into the corresponding *k%-approximation set*. For example, solutions in the 50%-approximation set (median attainment surface where k is 0.5) represent solutions that have been attained in 50% of all runs. Solutions in the 100%-approximation set (first attainment surface) represent solutions that have been attained in 100% of all runs; these correspond to poor-performing solutions as they are easy to attain. Solutions in the lowest k%-approximation set (last attainment surface) represent solutions that have only been attained once; these represent high-performing solutions as these are difficult to attain.

B.2 Additional Experimental Results

B.2.1 Configuration of $Corr$ and $Dist$ (Chapter 4)

This section shows the experimental GP results when Ave is used in combination with the new measures $Corr$ and $Dist$ in the fitness function in GP, to measure if the output values of a given genetic program solution lie within the target class regions (i.e. majority and minority class outputs should be negative and non-negative, respectively). Ave represents the average classification accuracy of an genetic program solution on the minority and majority class in fitness evaluation. Table B.1 shows the average AUC for fitness functions $Corr$ and $Dist$ using Ave for the tasks (over 50 GP runs). According to Table B.1, the AUC for both $Corr$ and $Dist$ with Ave is similar to, or lower than, the AUC for $Corr$ and $Dist$ with indicator function I_{zt} (from Section 4.3.2 in Chapter 4). This suggests that the AUC for fitness functions $Corr$ and $Dist$ when combined with indicator function I_{zt} is good as, or better than, the AUC when these two fitness functions are combined with Ave. For this reason, indicator function I_{zt} is the preferred method for enforcing the zero-threshold class boundary in fitness functions $Corr$

Table B.2: Ensemble accuracy ($\pm$ standard deviation) on the test set using Y values of 0.25, 0.75 and 1 in the fitness function for the PFC approach (with *Off*-EEL) over 50 runs.

Task	$Y = 0.25$		$Y = 0.75$		$Y = 1$	
	Minority	Majority	Minority	Majority	Minority	Majority
Ion	83.4 ± 4.0	95.9 ± 3.8	83.3 ± 4.0	96.1 ± 3.0	83.7 ± 3.9	94.9 ± 3.0
Spt	68.1 ± 9.2	78.9 ± 5.8	62.4 ± 8.3	82.8 ± 5.5	67.4 ± 11.2	78.8 ± 8.1
Ped	90.1 ± 1.4	86.4 ± 2.1	90.9 ± 1.7	88.2 ± 1.6	87.3 ± 3.1	89.2 ± 2.6
Yst_1	68.7 ± 4.6	78.0 ± 4.7	69.9 ± 6.0	79.3 ± 6.4	68.7 ± 5.1	78.7 ± 5.1
Yst_2	93.8 ± 2.9	90.1 ± 2.5	92.6 ± 2.6	90.5 ± 2.8	91.5 ± 3.5	90.4 ± 2.4
Bal	84.8 ± 7.3	84.1 ± 11.2	83.4 ± 9.2	85.8 ± 8.1	81.2 ± 9.8	83.4 ± 9.6

and $Dist$.

B.2.2 Weighting Coefficients in PFC Fitness (Chapter 6)

This section provides the experimental results comparing different weighting coefficients in the fitness function for the MOGP approach with PFC. Recall that the MOGP approach with PFC defines a trade-off between the accuracy and diversity of a given solution in the fitness function, controlled by weighting coefficient Y in Eq. (6.3) where $0 < Y < 1$. The primary PFC approach (used in Chapter 6) uses Y of 0.5 to treat accuracy and diversity as equally important in fitness. These results compare whether three other weighting coefficients for Y (0.25, 0.75 and 1) in Eq. (6.3) in the fitness function for PFC improves ensemble performances compared to the default (primary) approach (where Y is 0.5) . When $Y < 0.5$, the accuracy on the two classes is treated as more important (than diversity) in fitness, while when $Y > 0.5$, the diversity is treated as more important in fitness. These results use *Off*-EEL [76] as the ensemble selection strategy as this approach shows very good ensemble performances for PFC on the tasks (see Chapter 6) .

The first set of experimental results show the ensemble accuracies on the minority and majority class using the weighting coefficients 0.25, 0.75 and 1 on the tasks. The second set of experimental results analyse the ensemble "wins" when each of these weighting coefficients is compared to the default weight (0.5) in PFC.

Classification Accuracy of PFC Ensembles

Table B.2 shows the average minority and majority class accuracies (on the test set) for the PFC ensembles (with *Off*-EEL) for the three Y configurations (0.25,

Table B.3: "Win" pairs when the current PFC approach ($PFC_{0.5}$) is compared to other Y values in the fitness function (PFC_Y) with *Off*-EEL (on a run-by-run basis) over 50 runs. Bold results indicate a statistically significantly better ensemble performance (95% significance level) over 50 runs.

Task	0.5 *vs* 0	0.5 *vs* 0.25	0.5 *vs* 0.75	0.5 *vs* 1.0
Ion	**22**,6	14,14	18,11	**18**,4
Spt	6,3	6,9	3,6	7,3
Ped	**20**,0	**20**,2	7,12	**12**,2
Yst_1	3,6	3,4	4,4	**5**,0
Yst_2	7,8	6,6	11,4	**12**,1
Bal	16,12	9,13	15,11	13,6
Wins	74,35	58,48	58,48	67,16
Draws	191	194	194	217

0.75 and 1) over 50 MOGP runs. For convenience, a given Y configuration in PFC is referred to as PFC_Y in the discussion below. According to Table B.2, these Y configurations do not show any major difference in performance on the tasks. No Y configuration shows substantially better classification results than the default approach $PFC_{0.5}$ (from in Table 6.4 in Chapter 6). In three tasks (Spt, Yst_1 and Bal), each Y configuration seems to have a slight performance bias toward either the the minority and majority class. Less diversity ($PFC_{0.25}$) seems to have slightly stronger minority class accuracies (than majority class accuracies); while more diversity ($PFC_{0.75}$) seems to have slightly stronger majority class accuracies (than minority class accuracies). In the remaining task, the $PFC_{0.75}$ results dominates $PFC_{0.25}$, suggesting that more diversity in PFC can slightly improve ensemble results (than less diversity) on these tasks.

Ensembles Wins for PFC

Table B.3 shows the pairs of ensemble wins when the default PFC approach ($PFC_{0.5}$) is compared to the four other Y configurations (0, 0.25, 0.75 and 1) on a run-by-run basis over 50 MOGP runs. Table B.3 also includes PFC_0 (i.e. Baseline MOGP), for a more complete picture of how these Y configurations compare to $PFC_{0.5}$. This ensemble wins method of comparing two approaches (on a run-by-run basis), provides a good overall indication of ensemble behaviours over all runs and tasks and thus, a better idea of how the different Y configurations compare to the default PFC approach. The statistically significantly better ensemble strategy, denoted by a higher number of wins, is highlighted in bold in Table B.3.

According to Table B.3, the total number of wins (over all tasks) is always

higher in $PFC_{0.5}$ than in the other PFC_Y approaches. $PFC_{0.5}$ is clearly better than PFC_0 and PFC_1, as $PFC_{0.5}$ achieves more total wins than PFC_0 and PFC_1. ($PFC_{0.5}$ is statistically significantly better than the PFC_1 ensembles in four tasks and PFC_0 in two tasks). However, this difference is not as clear when $PFC_{0.5}$ is compared to $PFC_{0.25}$ and $PFC_{0.75}$. Even though $PFC_{0.5}$ achieves more total wins than $PFC_{0.25}$ and $PFC_{0.75}$, the margin of total wins for $PFC_{0.5}$ is not very large. Interestingly, $PFC_{0.5}$ achieves 58 total wins over all tasks when compared to both $PFC_{0.25}$ and $PFC_{0.75}$; while both these approaches achieve 48 total wins over all tasks compared to $PFC_{0.5}$. This suggests that while none of the other Y configurations can improve ensemble performances compared to the current PFC approach over all tasks, configurations where Y is 0.25 and 0.75 can also produce good ensemble results on these tasks.