University of Nottingham, School of Computer Science

Evolutionary Generation of Classifiers using Genetic Programming

Dissertation

Author
Joshua McDonagh

Supervised by
Ender Özcan

I hereby declare that this dissertation is all my own work, except as indicated in the text.

10th May 2021

Abstract

One of the areas of artificial intelligence in which a growing level of research and understanding is being developed is within that of genetic programming (GP), a technique that allows for the solving of potentially complex problems by evolving programs. As our knowledge in GP becomes more comprehensive, the scope for which GP can be applied becomes ever wider, including that of the widely encountered problem of classification.

This dissertation proposes and implements a way in which GP can be applied to solve two classification problem instances: the diagnosis of breast cancer patients and the predicting of whether a climate model will fail when simulated given certain parameters. This paper includes the methodologies that will be necessary to complete this work, the design of the GP system, how it will be implemented and configured (including configurations after tuning), and an evaluation of its performance compared with other relevant alternative machine learning methods. Finally, this paper will provide a concluding review of the project, the objectives and tasks it has involved, how they were performed, and final reflections on the project and the processes involved to complete it.

Acknowledgements

I would like to acknowledge and thank Dr Ender Özcan, my supervisor, for his first-class support and guidance throughout this piece of work, as well as Prof Nelishia Pillay for making available her invaluable insight in genetic programming research.

Additionally, I would like to express my gratitude to my friends and family, particularly my Mum, who have provided unwavering support during the time I have been undertaking this project.

Given the present situation regarding the ongoing COVID-19 pandemic, I would also like to communicate my particular acknowledgment towards university staff who have enabled work and study to continue to take place despite the current crisis.

Finally, I would like to extend my thanks and indebtedness towards all key workers who have kept our society going, and to NHS staff who continue to make huge sacrifices in saving people's lives. They are the best of us.

Table of Contents

1 Introduction

1.1 Background and Motivation

The use of machine learning technologies to advance automated intelligence and decision making has been growing extensively within recent history to meet and solve a diverse array of problems. Whether it is to help tackle climate change [1], or to predict the continued spread of the ongoing COVID-19 pandemic [2], artificially intelligent systems will undoubtably play a pivotal role in how we make decisions and process data to better the world around us, indeed, machines could even be found running whole or large parts of governments in a distant future [3]. Thus, the way in which machines are able to operate intelligently is becoming more important, and making sure the methods that machines use to process information and decisions are as reliable and optimal as possible will be crucial as a result.

One important area within machine learning is that of genetic algorithms and evolutionary learning, where the natural principle of 'survival-of-the-fittest' is applied to candidate solutions of a particular problem. There are multiple forms of evolutionary techniques that can be harnessed for machine learning, one of which is genetic programming (GP), which allows for the automated generation of a program using an evolutionary process from a high-level problem statement. Specifically, it involves the generation of a population of computer programs, and the iterative process of gradually transforming those programs into new programs that are better able to operate on the given problem via a process of natural selection.

A common supervised machine learning instance is that of classification, where the task is to sort and classify given data based on the labels and attributes already given to that data. Classification is a widely applied process, and is being used for a growing number of different purposes across many different areas. One of the more notable and growing applications of classification is within that of medicine and healthcare, such as for the diagnosis of neurodegenerative diseases like Dementia [4]. A high-profile example of when an attempt (although botched) to apply a classification algorithm en masse was when the British Department of Education and Ofqual calculated the grades of A Level students in England after no exams in 2020 could take place due to the COVID-19 pandemic[1]. Not only does this example show how classification algorithms have become so important, it also emphasises how great a negative impact there can be if the algorithm does not work as well as is expected of it.

Using genetic programming to generate classifiers is useful as it can be used to generate innovative ways of performing classification that doesn't require any humans to understand how the generated classification algorithms works (provided that it classifies data correctly and expectedly), as is identified by [5] on page 2. As a result, intricate and potentially complex classification algorithms that would otherwise be hard to generate manually, could be solved automatically via the natural-selection technique that genetic programming harnesses. Additionally, the ability for genetic programming to generate novel solutions could mean that more optimal classification algorithms could be created as opposed to what would otherwise be produced via manual solving of such classification problems.

In terms of the problem instances to be applied to this work, the first instance is that of breast cancer diagnosis. Developing a classification system that is able to classify and diagnose a patient as to whether the instance of breast cancer is malignant or benign can be useful in being able to quickly ascertain the priority to which the cancer should be dealt with. To achieve this, a diagnostic breast cancer dataset provided by the University of Wisconsin (and available via the UCI machine learning repository as the *Breast Cancer Wisconsin (Diagnostic) Data Set*) can be utilised which has 569 instances of breast cancer cases, all of which are labelled with the corresponding diagnosis. This will be a useful piece of work in this case as it will allow us to consider how able GP systems can be when trying to detect and classify patients with breast cancer, and determine its effectiveness when compared with other machine learning systems that are designed to classify and diagnose breast cancer.

The second instance is that of detecting the parameter settings of Intergovernmental Panel on Climate Change (IPCC) models that cause them to fail. A method in which we better understand the state of our climate is via the use of climate models that simulate the Earth's climate under a variety of different parameters. The issue with these simulations though, particularly by the ones used by the IPCC, is that they can often fail or stop working due to various issues that occur in the processing of the simulation [6]. As a result, a data set is available via the UCI machine learning repository (*Climate*

[1] BBC News, "A-levels and GCSEs: How did the exam algorithm work?," Last modified 20 August 2020, Accessed on 16 November 2020, https://www.bbc.co.uk/news/explainers-53807730.

Model Simulation Crashes Data Set) that provides 540 instances of climate model simulation outcomes, each being labelled with the simulation outcome. Using this data set, we can then apply our GP system so that it can generate a classifier which takes a simulation instance as input and determines whether it will fail or not. This work is useful because it potentially provides a way of determining whether certain combinations of simulation parameters cause such models to fail, thus meaning users of IPCC-class models can avoid using certain parameter combinations. The crisis of climate change and its effects on the world are of paramount importance, and so the ability to make the modelling of environmental conditions more reliable could be especially desirable if the GP system is successful.

1.2 Aims and Objectives

The main aim in this project is to design and develop a genetic programming approach to automatically generate classifiers that can accurately and reliably classify sets of data of the set domains (breast cancer diagnosis and climate simulation crashes).

In order to achieve the main aim of this project, the key objectives involved are:

1. To research the main qualities and features of genetic programming, to be familiarised with classification and its different frameworks, and to identify the metrics involved.
2. To finalise the datasets for two domains and analyse the data of which the datasets that will be processed by the classifiers generated will come from.
3. To design and develop a framework using Jenetics to generate the classifiers.
4. To detect the best parameter settings (tuning) for the genetic programming approach and to train and test the genetic programming approach to generate classifiers for the two problem domains.
5. To compare the performance of the automatically generated classifiers to those of standard machine learning algorithms and to write up my findings within my final dissertation document.

1.3 Description of the Work

The work to be undertaken is the development of a program which uses genetic programming (GP) to produce classifiers that can classify datasets of two domains (as introduced in Section 1.1):

- *Breast Cancer Wisconsin (Diagnostic)* (BCD) Dataset: this is a set of data instances where each instance records whether a possibly cancerous cell is malignant or benign as a class label, along with 10 real-value features that represent potentially relevant observations towards the corresponding cell nucleus.
- *Climate Model Simulation Crashes* (CMSC) Dataset: this is a set of data instances where each instance records whether a climate model simulation for a particular piece of software crashes (fails) or succeeds when ran using certain parameters (which are represented as 18 scaled features).

Note that both of these datasets present binary classification problems to be solved, as for the BCD dataset, the GP system needs to determine whether each given instance is a benign or malignant cell nucleus; and for the CMSC dataset, the GP system needs to determine whether each given instance is a simulation that succeeds or fails. In terms of how these data sets will be used in the work, both datasets will be split, with one portion of both datasets being used to train the GP system, and the other portion of both datasets being used to test the GP system. Also note that both datasets are unbalanced (with the CMSC dataset being far more unbalanced than the BCD dataset):

- For the BCD dataset, 34.5% of cancer instances are classified as malignant, as oppose to 65.5% for benign.
- For the CMSC dataset, 8.5% of simulation run instances are classified as failed, as oppose to 91.5% for successful.

An important area within AI and machine learning is that of genetic algorithms and evolutionary learning, where the natural principle of ‘survival-of-the-fittest’ is applied to candidate solutions of a particular problem. There are multiple forms of evolutionary techniques that can be harnessed for machine learning, one of which is GP. In terms of how GP will work to deal with the given problem instances, the GP system will evolve a set of classifier programs, each initially generated with random features and characteristics. Each classifier should utilise the features of the given data instances as well as additional constants and functions in order to classify given instances. The idea here is that, with each generation of the evolution cycle, the classifiers that populate the GP population will progressively get better and improve fitness, until finally a classifier with a maximum fitness is found. To improve the fitness of the classifiers in the population, the GP system uses a naturally inspired trick to evolve the population of classifiers and generate new populations iteratively, with each generation being generally better fitting than the last. As with biological evolution, this process of training the GP system involves mating and mutating classifier individuals between each generation such

as to find and share characteristics of well-fitting classifiers. Once enough generations have taken place, we choose the best fitting classifier that has been found and use it as our classifier for the given problem instances.

Once this GP system has been developed, its performance will be assessed and analysed using the given problem domains, from which it will be known how well this GP system operates. To perform this assessment, the output classifier of the system will be assessed via the use of a performance metric, which will then be compared against the performance of other machine learning techniques to solving classification problems. The information provided from this can then be used to conclude how well the GP system works, its strengths and weaknesses, and how some suggested changes to certain parts of the GP system could be implemented for further improvement.

2 Related Work

2.1 Genetic Programming and Classification using Genetic Programming

Genetic programming is an area of artificial intelligence where a great deal of scope in research is available. We are learning an increasing amount about how GP can be applied to different problems and how we can better adapt GP to produce more optimal results. One of the areas in which GP is finding more utilisation is that of solving classification problems. Classification problems are widespread in machine learning and AI, which reflects the occurrences of real-world classification problems that are encountered universally, for example: there have been studies on classifying news on Twitter [7] as there has work on hyperspectral image classification [8], two areas that are totally different but both involving classification problems. As a result, a greater understanding of how GP could be applied to improve the performance of classification problems could be highly impactful and useful for a myriad of classification instances.

As is observed by different applications of GP, including that of [9] which applies GP to classify fingerprints, the use of GP can result in the discovery of unconventional and unique solutions to classification problems, and can exploit trends or patterns in the given features to produce more optimal results that would otherwise be overlooking via alternative methods. Indeed, [10] finds that GP, when tasked with medical supply classification, can achieve a level of performance similar to that of artificial neural networks, meaning there is a substantial opportunity available to research and find new ways of constructing and building these GP systems such that they can work even more effectively to find better performing solutions for classification.

As a result, this work seeks to explore a way in which GP can be achieved to solve two given problem domains. This GP implementation will utilise various techniques and methodologies to solve the given classification problems, with the performance being assessed finally with a comparison of other machine learning techniques. After this work is completed, the research, development and final assessment of this project will provide knowledge on how GP can perform on the given problem domains given the applied techniques and systems, and analyse where particular strengths and weaknesses of the GP system are once its performance is assessed. As GP and its use in solving classification problems is a still a growing field in terms of research, what is proposed and found in this work could be useful to future researchers to influence their own application of GP systems and the solving of their own classification problems.

2.2 Problem Domains and Benchmarks

2.2.1 The Breast Cancer Wisconsin (Diagnostic) (BCD) Dataset

Classification problems involved in healthcare and medicine are becoming especially important today, with a wide range of research going into making machine learning systems that are able to reliably predict and diagnose patients based on symptoms and other data. Such classification problems can vary from physical health issues, such as skin lesion classification as given by [11], to mental health issues, such as the classification of unipolar and bipolar depression as given by [12]. A particular area where classification techniques are being applied is to the diagnosis of different cancers, such as the classification of breast cancer [13] and Leukaemia [14], which is understandable given approximately one in six deaths (globally) are due to cancer[2], meaning there is clearly a demand for classification systems that can reliably and accurately diagnose patients who may have cancer. One such cancer that is particularly prevalent is breast cancer, which is a leading killer of women across the world. As a result, one of the problem instances the GP system to be developed will operate on is that of breast cancer diagnosis.

[2] World Health Organisation, "Cancer," Last modified 12 September 2018, Accessed on 8 December 2020, https://www.who.int/news-room/fact-sheets/detail/cancer.

2.2.2 The Climate Model Simulation Crashes (CMSC) Dataset

Besides healthcare, classifiers can be useful in other fields that are important to modern society. Given the continued effects of global warming and the alarmingly increased global temperature[3], the tackling of climate change and the affects it has on civilisation is becoming more and more a necessity. As a result, machine learning is being utilised increasingly in different ways to help tackle climate change, such as for generating future weather files under climate change scenarios [15], and with that, classification problems have also been discovered and tackled via various methods to help mitigate against some of the impacts of climate change, such as from forecasting floods [16] to identifying tropical cyclone intensity using satellite infrared imagery [17]. As a result, it is plain to see that machine learning has a role to play in mitigating the effects of climate change, and classification problems constitute a good proportion of that. For this piece of work, we'll be utilising a dataset that allows us to train the GP system towards producing a classifier which can detect the parameters necessary to cause a given IPCC model to fault (as introduced in Section 1.1). Such climate models have been utilised for a range of applications, and are used to consider and assess the impact of various parts of the environment, such as living marine resources [18].

3 Genetic Programming

As introduced in Section 1 and expanded in Section 2, genetic programming allows for the automated generation of a program that solves a given problem using an evolutionary process [19]. Specifically, it involves the generation of a population of computer programs (individuals), and the iterative process of gradually transforming those individuals into new programs that are better able to operate on the given problem via a process of natural selection. The way in which computer programs are compared and selected should be based on some fitness metric so that the better performing computer programs are more likely to continue through the evolution process, whilst also maintaining good diversity amongst the generated populations so that high-quality individuals do not completely dominate generated populations. Specifically, for our classifiers to evolve, we use a selection operator to choose a classifier from the current population in some way, generally preferring classifiers that are better fitting. In addition, there are two other main operations to consider when evolving the population of computer programs:

- Crossover: an operator which selects generally two classifiers using the selection operator, and swaps certain elements of both classifiers between each other, resulting in at least one new classifier which is added to the next generation. This operator simulates the 'mating' involved in the natural world that encourages evolution, and how only the 'strongest' have the opportunity to mate and pass on their DNA.
- Mutation: an operator which selects a classifier using the selection operator and randomly changes a part of the classifier, resulting in a new child classifier that is added to the next generation. This operator simulates the random mutation that can occur in DNA within the natural world, which is an essential part of evolution.

As mentioned in Section 1.1, GP is able to produce unique and inventive solutions to given problems via its evolutionary process, as explained by [5] on page 2. This is useful for solving classification problems, particularly if there is a large number of different possible combinations of features in the given data set (thus requiring a more complex and intricate solution), because GP's evolutionary and innovative nature adapts it better for exploring a greater area of the solution space without being exhaustive in its methods [20].

3.1 Representation of Classifier Individuals

When using classifiers in a GP environment, we should consider each classifier D to be a mapping of $D: R^p \rightarrow N_{hc}$, where R^p is the p-dimensional real space and N_{hc} is the set of label vectors for a c-class problem [21]. To put this simply, 'a classifier is a function which takes a feature vector in p dimension as input and assigns a class label to it'.

The main method considered for representing classifiers in GP has been that of numeric expression-based tree structures. Representing programs as trees is especially common in GP and can take different forms, though most provide numerical values when processed. For classification, the processing of a tree provides the numeric class prediction for a given data instance. For multi-class classification, trees are often constructed for each class per individual, as is done for [21], in the case of this project though, we are performing binary classification so we only need to represent classifiers using a single tree for each.

[3] R. Lindsay and L. Dahlman, "Climate Change: Global Temperature," Last modified 14 August 2020, Accessed on 8 December 2020, https://www.climate.gov/news-features/understanding-climate/climate-change-global-temperature.

In terms of the specific type of tree we should utilise, we can make use of parse trees to represent programs [22]. Parse trees are made up of three main types of node:

- The root node: first element of the tree.
- Interior nodes: the functions of the program.
- Leaf nodes: the terminals of the program (constants and/or variables).

Parse trees can be an effective form of representing programs, so long as the set of functions available to the system is sufficient for the problem at hand. For example, the expression $zy(y + 0.639z)$ can be represented as a parse tree as is demonstrated by Figure 1.

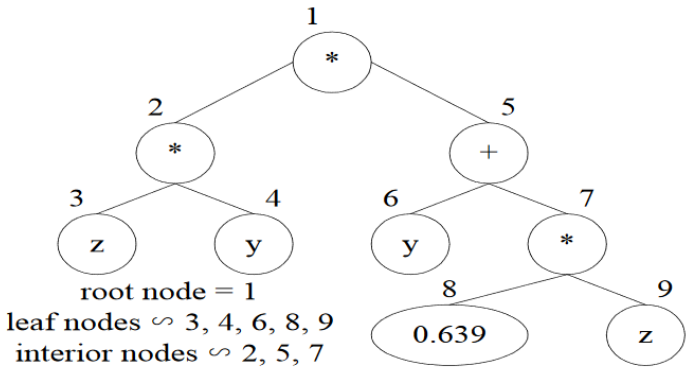

Figure 1: example visualisation from [22] of how a parse tree can represent an expression.

Use of tree-based structures for GP is popular because they're a simple representation that can be applied well when concerning how individuals are used and manipulated by the GP process. For example: it is relatively straightforward to apply crossover systems using the tree representation because you are able to seamlessly swap over nodes and whole branches during the mating process.

There are various alternatives to tree-based structures that are also useful for certain GP problems, one of which being linear GP, which is a specific subset of genetic programming where programs are represented linearly as a sequence of imperative-style instructions [23]. Another alternative is that of stack-based representation where GP involves the use of post-fix notation equations to represent each individual, which are processed using stack operations [24]. Both of these alternatives have their own advantages, but the main reason in which tree-based structures are more popular, and why this project will harness tree-based structures, is because it is more straightforward and less complex to apply crossover and mutation operations on tree-based structures. With tree-based representation, it's trivial when performing crossover and mutation, as you can simply swap, insert and modify subtrees in the individuals without any additional effort required. With linear GP and stack-based representation, crossover and mutation involves more work to make sure changes in the representations are legal and don't break the individual, this is because such representations are less rigid and more liberal compared with tree-based structures, and so are more prone to the possibility of improper handling (such as the use of an invalid number of parameters in certain cases).

3.1.1 Maximum Tree Depth and Bloating

For the classification problems that are involved, it is important that individuals follow a framework that encourages generalisation and deters issues such as overfitting from occurring. A particular problem here is that of bloating, where the tree representations developed are highly complex and large, and are highly likely to make use of redundant functionality and tree features (i.e. equivalent functionality can be achieved by a simpler tree). The main way of preventing bloating is by limiting trees to a maximum depth (as is suggested on pages 26 and 27 of [5]), meaning programs would be forced to produce better fitting solutions within a limited number of possible nodes and reduce redundancy. Importantly, the maximum depth must be chosen carefully, a depth too shallow could mean individuals don't have enough flexibility to generate a good fitness, conversely, a depth too deep could mean bloating and the potential for overfitting and redundancy can still occur.

3.2 Terminal and Function Sets

Each program representation will consist of values and operations (functions) that, when run, will return a numerical value. As a result, we need to develop a set of terminal values and set of functions that the GP system can utilise in the creation of each individual. The contents of these sets depend hugely on the problem domains that the GP system is being applied to (which in this case is classification).

3.2.1 Terminal Set

The terminal set consists of values and information that requires no parameters or arguments. Pages 19 and 20 of [5] suggests that the terminal set may consist of:

- *The program's external inputs*: values that are input and are known at the time of the program's execution, and usually take the form of named variables.
- *Functions with no parameters (0-arity)*: these are typically used because they might returning varying values each time they're used, such as a random function that returns a random value. Another reason these might be used is because they might provide side effect functionality (like printing a value to a console).
- *Constant values*: values that a pre-set, generated randomly during the construction of the individual, or spawned through a process of mutation.

Given the use of the GP system in this project to generate non-stochastic classifier algorithms, there is no use of including functions with no arguments in the terminal set for this project; functions that provide side effects are not needed and random functions would incorporate a level of stochastic functioning that would be undesirable in a classification system. For the program's external inputs, this would be needed in the project's terminal set such as to make sure that data instance features are involved the classification of data instances. Additionally, constant values will be required too in order to supplement the classifier's ability to turn given data instance features into a classification.

3.2.2 Function Set

The terminal set consists of functions and operations that require parameters in order to run and return a value. An example of GP classification system is that of [25] which has a function set of four simple operations: addition, subtraction, multiplication and protected division (provides evaluation safety as it returns zero if a division by zero occurs). We can additionally make use of two extra simple operations, exponentiation and nth root, as arithmetic shortcuts in the classifiers produced. It's useful to make use a small number of simple operations, as it gives the evolutionary system more freedom to explore the solution space and find a well-fitting solution, which is possible without the restrictions posed by complex and specialised functions.

A GP classification system given in [26] makes use of an additional three relation operators:

- *GT_J* represents the greater than operator ($>$).
- *LT_J* represents the less than operator ($<$).
- *Eql_J* represents the equals operator ($==$).

Note these three operators take in four parameters: the first two parameters are the comparison values, the third parameter is the value returned by the function if the comparison yields true, and the fourth parameters if the value returned by the function if the comparison yields false.

The concern with regard to using these additional functions is that the increase in the resources (and thus, the solution space) that the learning process has to use may introduce more opportunities for the system to fall into local minima. However, relational operations can be useful, particularly for classification where other machine learning techniques (particularly decision trees) can successfully harness relational comparisons to classify data sets, and so applying similar relational techniques in the genetic programming approach to classification could be useful and result in better fitting classifiers.

It is important also to consider the function set's adherence to type consistency so that nodes and subtrees can be interchanged and joined arbitrarily during crossover and mutation, as is pointed out on pages 21 and 22 of [5]. Fortunately, the functions within this function set comply with type consistency, as the parameters of all the functions and the return value type of call the functions are of the same real-valued type, meaning nodes and subtrees can easily be mixed and manipulated arbitrarily without any issues being caused. The result of this means that the mutation and crossover operators can be designed without need of considering or working around type information, meaning mutation and crossover operations are less constrained and more liberalised in terms of what they can do with given classifiers.

3.3 Fitness Function

In order for the GP system to be able to determine which candidate classifiers are of a better quality for binary classification, a fitness function is required. The calculation of basic fitness functions are not especially complex, but the application of certain fitness functions can be more useful than others, particularly when it comes to how balanced the classification data set is when it is processed by the GP system.

One of the most widely used fitness function is that of accuracy, which is the ratio of the total number of correct classifications to the sample size [27]. This is a useful metric to use if the given dataset to perform binary classification on is balanced, as accuracy processes whether all predictions are correct. The issue with unbalanced data here comes from the fact that, if only a small proportion of the data instances are of a particular class, then the classification system could mis-classify all of those data instances and still receive a high accuracy (given most or all other data instances are correctly predicted). The reason this is a relevant issue is because the two datasets considered for this project are not balanced, as explained in Section 1.3.

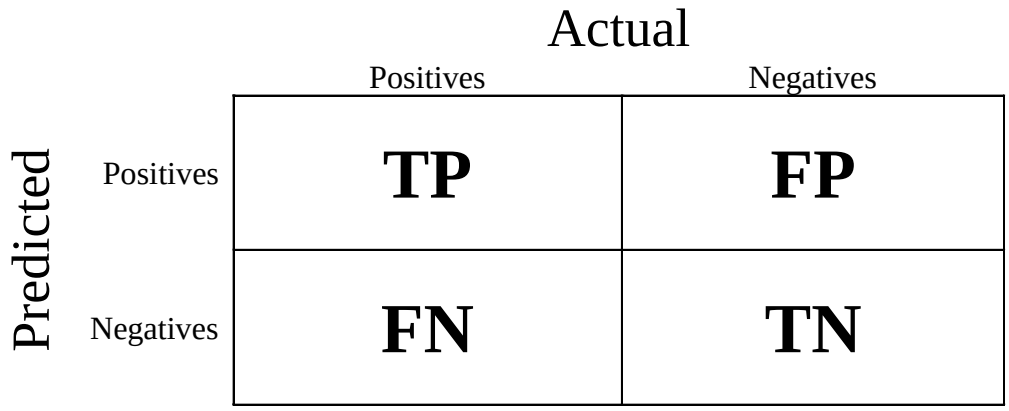

Figure 2: binary classification confusion matrix [28].

As a result, it would be better to consider the fitness function of f-measure, which combines recall and precision metrics (where recall is weighted by a value β), as is utilised for example in [29]. The reason that this fitness function is more suitable than alternatives (such as accuracy) is because it prevents the classes in the considered datasets that have a minority representation from being overwhelmed by better classification performance for the alternative classes. Thus, it would be beneficial to apply the fitness function of f-measure to this project.

Using Figure 2's TP, FP, FN and TN, precision can be calculated

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall can be calculated

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Thus, f-measure can be calculated

$$\text{f-measure} = (1 + \beta^2) \times \frac{\text{precision} \times \text{recall}}{(\beta^2 \times \text{precision}) + \text{recall}}$$

Like in [29], the β value will be set to 1 as is traditionally the case. As a result, our f-measure fitness function becomes

$$\text{f-measure} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

3.4 Selection Operator

Selection of individuals to be reproduced, mated, or mutated for the next generation is important as it determines largely how the GP system can progress from one generation to another. When considering selection operators, two main qualities are assessed, the ability for each selection operator to select better fitting individuals, and the ability for each selection operator to encourage diversity in the next generation. Finding a balance between these two qualities is important in order to produce generally well-performing populations with enough diversity to try and avoid problems such as overfitting.

Several selection methods have been considered, including tournament selection [24] and lexicase selection [30], but the selection operator that has been found to be most suitable for this project is that of roulette-wheel selection. Roulette-wheel selection operates by associating each individual with a probability of being selected: a probability that is proportional to the individual's fitness, i.e. the better fitting the individual is, the higher its probability of being selected

[31]. Given N individuals of a population with each having an associated fitness $x_i > 0$ where $i = 1,2,3,\dots,N$, the probability p of selection for individual i can be calculated

$$p_i = \frac{x_i}{\sum_{i=1}^{N} x_i}$$

The idea here is that the individuals with a higher fitness will be more likely to be selected by the algorithm, but does not eliminate the opportunity for less well-fit individuals to also be selected, meaning the resulting generation is less likely to be dominated by better-fitting parents. The only limitation is that if any members of the population are hugely better-fitted than other members, then the opportunity for worse-fitting individuals to be selected is far less likely.

When comparing roulette wheel with alternatives, some selection methods put a more significant amount of weight on selecting better-fitting individuals over individuals to promote diversity. This can sometimes be problematic, particularly for classification where diversity is key to making sure overfitting is less likely to occur. One considered selection method is that of tournament selection, which is less able to accommodate diversity as wholly as roulette wheel selection because after it has collected a small set of individuals from the current population, it will always select the individuals with the best fitness as stated in pages 14 and 15 of [5], meaning some of the least well-fitting individuals have almost no chance of ever being selected. A similar issue is present with another considered selection operator: lexicase selection. In lexicase selection [30], individuals are discarded from the opportunity to be selected if they do not fit especially well in most test cases, which is a barrier for diversity because it means (like tournament selection) the individuals that are generally worse fitting or are not amongst the best fitting for most test cases are unlikely to get a chance in being selected. Making sure even the worst-fitting individuals have some chance of selection means that a wider diversity can be maintained and issues that a lack of diversity could potentially cause in classification problems (such as overfitting or becoming stuck a local minimum) can be prevented.

3.5 Crossover Operator

Crossover operators are used to ‘mate’ two different parent classifiers to produce a child or children with features that are inherited from both parents. The general idea is that better-fitting parents (depending on the selection operator) are selected to mate, so that functionality can be shared and mixed to produce children that will then be inserted into the next generation. This is a highly beneficial way of populating the next generation as it allows the opportunity for the training of the GP system to find and consider new classifier compositions, with the better performing being utilised further and the poorer performing either being discarded or improved heavily. How this is specifically implemented depends on what crossover operator is used.

Note that pages 15 to 17 of [5] points out explicitly that when performing crossover, the crossover operator should be applied to copies of the parent classifiers, so that the original classifiers from the current population aren’t manipulated themselves and can be used again if selected another time.

The simplest and standard way if performing crossover in GP is by selecting two individuals and then ‘mating’ them together by randomly choosing subtrees in both individual tree-representations and swapping those chosen subtrees with one another, generating two child individuals for the next generation [32], as is visualised by Figure 3. The idea behind this method is that it tries to combine and share features of selected individuals to produce a more diverse and potentially better fitting set of children for the next generation. The main issue with this method though is that the completely random element of this crossover operator means that mating may result in worse performing children, and could possibly create generations that contain generally less well-fitting individuals.

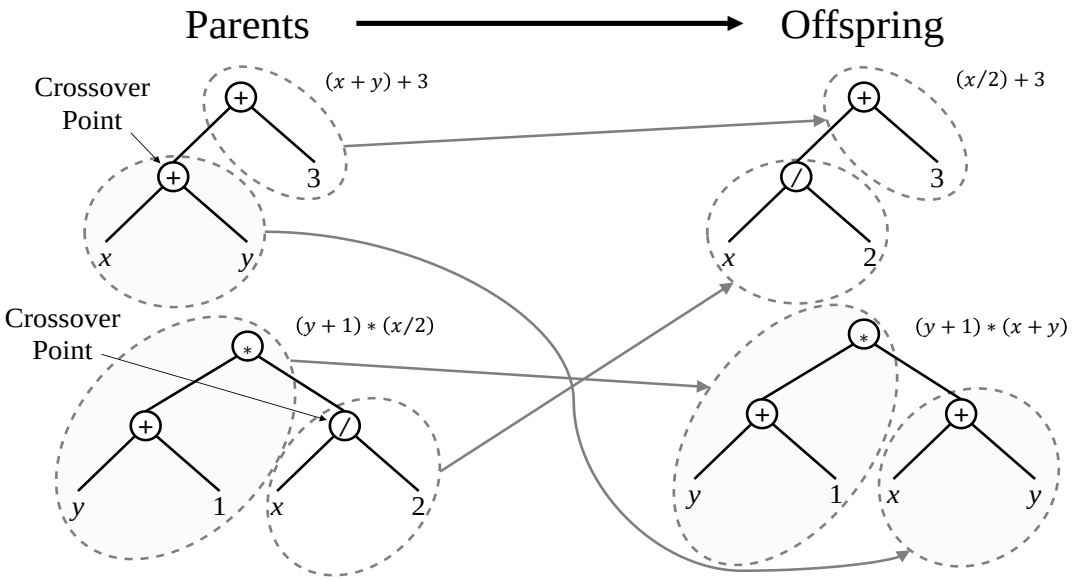

Figure 3: example of standard GP subtree crossover based on the example given by [5] in pages 15 to 17.

There are other variations of this standard crossover operation, such as the operator provided in [33] which combines two individuals in a similar way, but produces only a single child and puts more weight on direct reproduction and mutation to generate the rest of each new generation. The issue though is that in most of these variations, the completely random element remains meaning we still have the issue with children being generated that can easily be worse performing then their parents.

Instead, we could look at alternatives that do not rely solely on random methods to mate individuals. One such instance is that given in pages 15 to 17 of [5] called the headless chicken crossover (HCC) operator which exploits hill-climbing to produce better-fitting children. In this operator, only a single individual A within the current population is selected to be a parent, and, instead of selecting another individual, a new individual B_1 is randomly generated entirely to be the second parent. Parent A is then mated with parent B_1 using the same random subtree method as is provided in the standard method to produce a child individual C_1. This child C_1 is kept and inserted into the new generation if it is better fitting than its parent A. If this is not the case, C_1 is rejected, and we generate a new random individual B_2 which is mated with A like before. This process is repeated iteratively until a child C_n is produced which is better fitting than A, from which C_n is then kept and inserted into the new generation.

The advantage of the HCC operator is that its use of hill-climbing means that all children generated as a result of crossover will be better fitting than its parent from the previous generation. The problem though is that the resulting population is less diverse, as the individuals of the population will largely be generated after being found to be better fitting than individuals of the previous generation. The reason this is a problem is because a lack of a diverse generation and a greater reliance on hill climbing can mean there is greater chance that the GP process could get stuck in a local minimum of the search space. Additionally, there is also the chance of overfitting where the hill-climbing method forces the resulting solution to closely fit with the training dataset, but causes the solution to generalise poorly with non-training data as the populations generated during the GP process are not sufficiently diverse.

As a result, it would be more beneficial to utilise the standard subtree crossover as given by [32], as it is able to provide a better level of diversity than the HCC operator in the newly generated generation, which could be essential for making sure the training of GP has a wide enough scope in terms of the solution space to find better fitting solutions. In addition, the use of the selection operator to provide generally better-fitting solutions for crossover to mate means that the crossover process is still guided towards the production of better-fitting results. The advantage of using the standard crossover operator here is reflected by various classification problems that are solved via GP, such as that of the well-performing proposal set out in [33].

3.6 Mutation Operator

For mutation, we take a single individual from the population, and modify this parent individual in some way to produce a new child. The most common form of mutation used in GP is that of subtree mutation as given in pages 15 to 17 of [5], where a point is randomly chosen within the given classifier tree and is replaced by a new randomly generated subtree, as is shown as an example in Figure 4. The advantage of applying mutation is that it introduces a further amount

of diversity into the population, and thus widening the scope of resources for which the GP process can utilise in producing better fitting individuals.

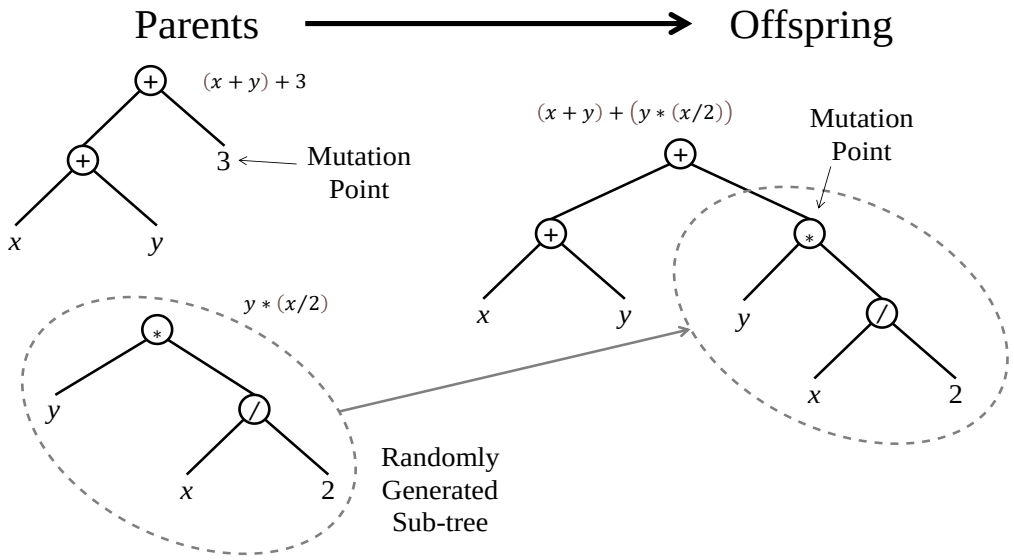

Figure 4: example of subtree mutation given by [5] on pages 15 to 17.

Note that we can identify a potential issue here with the use of subtree mutation, as it has the potential to produce children that are worse fitting than the selected parents. A way in which this issue may be tackled has been suggested by [34] which is to apply point mutation. With point mutation, the same process is applied as the standard subtree mutation, but unlike subtree mutation, point mutation applies the mutation to the parent twice to return two children, the worse fitting of which is dropped. If the remaining child is better fitting than the parent, then the child is transferred to the new generation, otherwise, the parent has a 50% chance of being transferred to the new generation.

As is suggested, the idea with point mutation is that it tries to strengthen diversity within the population whilst also making sure only better fitting individuals make it to the next generation. The issue with point mutation is that its requirement for mutations to occur only if the resulting child is better performing than the parent means that the diversity generated is inevitably limited to some degree, and could potentially allow for better generalising individuals to be discarded if not better fitting than their parents. Additionally, the proportion of the population of which mutation is applied is minimal (usually about 1%) as stated in pages 15 to 17 of [5], meaning, if its subtree mutation that's applied, only a relatively small number of worse-fitting individuals can be generated at most in a single generation and will not likely be selected anyway, arguably making point mutation largely redundant here. As a result of this, the use of the simpler subtree mutation operator would be more suitable.

3.7 Population Size and the Number of Generations

We must determine a population size, i.e. the number of classifiers within each generated population, and a number of generations, i.e. the number of evolutionary iterations that occur of the GP system, which best accommodates the GP training process. Typically, the setting of one of these configurations will be in relation to the other, so if you want fewer generations, you will likely choose to have a greater population size to account for the reduced number of crossover and mutation operations that can occur; conversely, if you want a greater number of generations, you will likely choose to have a lesser population size to account for the increased number of crossover and mutation operations that can occur, and vice versa.

As a point of information, it has been found in [35] that an optimal population size is difficult to determine given problems that are complex to solve. Based on the fact that this project intends to develop a trainable GP system that can produce a classifier to classify instances of data from certain domains, there is a great potential here for the complexity of the problem to make it extremely difficult for the optimal population size to be calculated, and thus a strict process of trying to find an optimal population size may be wasteful of effort. As a result, determining the population size and the number of generations is difficult. Instead, this work will set these parameters based on the applications of previous GP systems.

The GP system for classification proposed by [26] utilises a relatively small population size (80) and a high number of generations (500). This system proposed has been tested on four different datasets: *WBCD*, *Pima*, *IRIS* and *WINE*, the characteristics of which are shown in Table 1.

Table 1: characteristics of the data sets from [26].

Dataset Name	Number of Samples	Number of Features	Number of Classes
WBCD	683	10	2
Pima	768	8	2
IRIS	150	4	3
WINE	178	13	3

The performance of this GP system using the proposed datasets has been recorded using 10-fold cross-validation and is shown in Table 2.

Table 2: 10-fold cross-validation testing results from [26].

Dataset Name	Worst (%)	Best (%)	Mean (%)
WBCD	92.65	100.00	96.10
Pima	70.12	80.26	76.56
IRIS	86.67	100.00	95.33
WINE	88.89	100.00	97.19

As the results given by Table 2 show, the GP system given in [26] performs well in most cases, with the mean for each data sets sitting generally around 96%. The one exception here is for that of the Pima dataset, where the performance in the worst, best and mean cases are all much poorer than for the other datasets, though it's important to note that the performance for Pima is still as good as or better than non-GP alternatives as mentioned by [26], meaning it's poorer-performance is potentially just as a result of a particularly hard to solve dataset.

In addition, there are other examples of GP classification systems that use many generations and (relatively) small populations, such as [36] (which uses a population size of 2000 and a maximum generation of 500,000). As a result, this piece of work will utilise a higher number of generations and a lower population.

3.8 Termination and Designating the Result

In order for the GP process to know when to stop evolving classifiers, we need to set a termination criterion. Typically, when designing the GP system, a predefined maximum number of generations M is set so that once the GP system hits that number of generated generations, the evolutionary process ends, as stated on page 27 of [5]. The limiting of the number of generations that can be generated and evolved before termination means that the GP system, if M is set wisely, will end as the classifiers being generated can no longer be bettered (in terms of fitness and performance) by further evolution, thus preventing many more useless generations being generated. Choosing the number of generations is often in conjunction with choosing the size of the population, which is discussed and determined in Section 3.7.

Additionally, both [21] and [24] advance this by also including a second criterion where the GP process will also end if the fitness of a classifier reaches 1, which is a beneficial criterion as it prevents the GP system from continuing to operate even after a perfect-fitting classifier has been discovered. In terms of designating the result, if a perfect-fitting classifier has been found, then automatically, that classifier will be designating the result of the GP process, otherwise, if M generations have been produced, then the best fitting of the classifiers in the most recent generation is designated the result.

4 Design and Implementation

4.1 Java Framework

The project has been implemented in Java: a programming language that is extensively used and harnesses object-oriented techniques to enable the creation of robust functionality with concise and compact code. Java is an especially secure language, and can prevent any implementations using memory dangerously. It doesn't make use of pointers and handles garbage collection automatically, meaning the resulting implementation will have no risk of memory leaks or other associated memory issues. Its platform-independence is also of great advantage as it allows the capacity to execute and run the Java implementations on different platforms without need of much consideration towards how those different platforms work. This is possible via the layer of abstraction which the Java virtual machine (JVM) provides. The only necessity to run the Java implementation, therefore, is the installation of the appropriate libraries and the JVM.

Additionally, a Java library called Jenetics (https://jenetics.io/) has been harnessed as a base framework for the operation and running of the GP system. Jenetics is a library that can be used to implement a variety of different evolutionary systems, notably including GP, as well as additional functionality to support evolutionary systems, such as parallel processing capabilities and no runtime dependencies on additional libraries. Another framework that was considered was that of the Java Evolutionary Computation Toolkit (ECJ) which is similar to Jenetics in the way that it is also useful in implementing various evolutionary systems and functionalities. ECJ is a reliable and comprehensive Java framework that would have also been useful in this project, however, due to ECJ's old age, it doesn't harness full use of newer features that Java supports, meaning it's application may not be as efficient as that of Jenetics which makes full use of the facilities that modern Java provides.

4.2 Design and Implementation of Methodologies

A number of different methodologies that are required for this work have been discussed in Section 3. Importantly, this section considers the design and implementation of these methodologies to produce the GP solution.

Note that parameter tuning will be performed to finalise some hyperparameters, thus meaning final parameter settings cannot yet be determined until parameter tuning is completed.

4.2.1 Representation of Classifier Individuals and Maximum Tree Depth

As considered in Section 3.1, each individual program will be represented as a parse tree, with each tree containing internal function nodes and terminal leaf nodes. The data that can be represented by terminals can be that of input feature values or constant values (there is also the option of functions with no arguments, but no such functions have been found to be necessary for classification). The internal nodes use the associated function to process the child node values (with the number of child nodes equal to the number of function parameters) and return a value for that node. The set of functions that are available to each parse tree is pre-set. As discussed in Section 3.1, tree-based representations provide a rigid framework for which data flows throughout the program, a framework that is highly applicable to GP processes including mutation and crossover, where branches can be manipulated and exchanged between programs with little computational complexity. Both data sets that are being considered for this project are binary classification problems, thus meaning there is no need for any additional frameworks to examine multi-class data sets, i.e. we can make use of a single parse tree per individual.

In terms of how this has been implemented specifically within the Java code, the Jenetics library's GP module provides a pre-made tree representation system that largely works in the required way specifically for GP, which has thus been utilised. The only modification that has been necessary here is to implement a mechanism for classification which determines how the evaluation of the tree is translated into a predicted class: for this, it has been implemented such that if the parse tree evaluates to a negative value, the negative class is predicted, otherwise the positive class is predicted.

Additionally, as considered in Section 3.1.1, a maximum tree depth must be implemented for classifiers to reduce bloating. Based on the depth limits of [37], which uses a depth limit of 4, and [38], which experiments using a depth limit of 3-5, this project will harness the use of a depth limit between 1 and 5, from which a final depth limit will be deduced and identified after experimentation and parameter tuning. As is displayed in the good performance of the systems presented by [37] and [38], a depth limit of no greater than 5 prevents extensive overfitting and improves the generalisation of the GP trained classifiers.

4.2.2 Terminal and Function Sets

As stated in Section 3.2.1, the terminal set that will be harnessed by the parse trees will consist of:

- *Feature values*: the value for a particular feature in the given dataset instance. This is important for the GP process so that the classifiers generated can utilise the feature values to predict a class.
- *Constant values*: a value that is pre-set, randomly generated during the creation of the tree, or generated during mutation. This is useful for the GP system as it provides the ability for additional values to be utilised to generate classifiers in conjunction with feature values, which can result in classifiers that harness weights and biases (for example).

Additionally, as is discussed in Section 3.2.2, the function set will (naturally) be far more comprehensive than the terminal set. Thus, the function set will consist of:

- *Addition* $(+)$: takes two parameters and returns the summation of those parameter values.
- *Subtraction* $(-)$: takes two parameters and returns the first parameter value minus the second parameter value.
- *Multiplication* $(\times)$: takes two parameters and returns the product of those parameter values.
- *Protected Division* $(\div)$: takes two parameters and returns the first parameter value divided by the second parameter value, or returns zero if the second parameter value is zero (hence, protected).
- *Exponentiation* $(\exp)$: takes two parameters and returns the first parameter value to the power of the second parameter value.
- *nth Root* $(\sqrt{})$: takes two parameters and returns the second parameter root the first parameter.
- *Greater Than* $(>)$: takes four parameters and returns the third parameter if parameter one is greater than parameter two, otherwise the fourth parameter is returned.
- *Less Than* $(<)$: takes four parameters and returns the third parameter if parameter one is less than parameter two, otherwise the fourth parameter is returned.
- *Equal To* $(==)$: takes four parameters and returns the third parameter if parameter one and parameter two are equal, otherwise the fourth parameter is returned.

4.2.3 Fitness Function

As discussed in Section 3.3, the performance metric that is to be harnessed is that of f-measure. F-measure has been found to be a useful fitness function for this project as the datasets being used are unbalanced, and so alternative fitness functions, such as the popular accuracy fitness function, would not best appreciate how unbalance within the given datasets might affect the results from training the GP system.

4.2.4 Selection Operator

As considered in Section 3.4, the selection operator that has been used for this project is that of roulette-wheel selection, which works by associating each classifier with a probability of being selected where the probability is proportional to the classifier's fitness.

The Jenetics library provides a pre-made implementation of the roulette-wheel selection operator, and operates as is necessary for this work. As a result, this library feature has been used in the project implementation.

4.2.5 Crossover and Mutation Operators

The operations that will be utilised for crossover and mutation have been discussed and determined within Sections 3.6 and 3.7 respectively. In terms of the specific operators that will be applied for:

- Crossover: a standard subtree crossover operator will be utilised using parents selected via the selection operator. This operator will take the form of a standard subtree crossover operator.
- Mutation: a subtree mutation operator will be utilised using a parent selected via the selection operator. This operator will take the form of a subtree mutation operator.

Note that both operators have been implemented already in the Jenetics library, which is what will be utilised in this project implementation.

In terms of the crossover and mutation probabilities that we'll use, most GP systems for classification have a high crossover probability and a low mutation probability, such as is found for the GP systems proposed by [33] and [39]. As a result, the crossover and mutation probabilities for this project will be similar, with crossover probability being between 0.7 (70%) and 1.0 (100%), and mutation probability being between 0.0 (0%) and 0.3 (30%). Note that the final settings for the crossover and mutation probabilities will be determined through parameter tuning.

4.2.6 Population Size and the Number of Generations

As found in Section 3.7, the utilisation of a relatively high number of generations and a relatively small population size would be beneficial for this project, as has been utilised in example GP systems such as [26] and [36]. Thus, the specific population size and number of generations that will be set and used in this work will be 1000 and 7000 respectively; parameters which follow the findings of Section 3.7 where the number of generations should be higher than the population size.

4.2.7 Termination and Designating the Result

As is discussed in Section 3.8, one of the termination criterions is that of reaching the maximum number of generations, the reasoning for which is obvious, if you've reached the maximum number of generations which has been set (in this case is 7000) and justified (see Section 3.7 for the justification of the number of generations for this project), you cannot continue with the evolutionary process. Another termination criterion discovered as part of the discussion in Section 3.8 is that the training of the GP system will end if a classifier generated during the GP training process has a fitness of 1 (i.e. has a 'perfect' fitness). Again, this is also a natural termination criterion here as, if a classifier has been developed that works perfectly, then there's no need to continue the evolutionary process as the best possible fitting solution has been identified.

As a result, the two termination criterions for this project are:

- Reach 7000 generations.
- Classifier achieves a perfect fitness.

4.3 Cross-Validation

When applying this system to the given datasets, we need to organise a strategy for how the data is split into training and testing datasets. Simply splitting the given dataset into two is a possible way of achieving this, but the problem is that of predictor bias, where overfitting can occur because the training set utilised is not as representative of all possible solutions as it could otherwise be. An alternative to this is the use of cross-validation, where the given dataset is segmented into n folds (none of which overlap). Cross-validation occurs iteratively, and thus GP is trained (in this case) n times, where each fold has an opportunity to be the testing dataset, and the rest of the folds are the training dataset. As a result of all the data having the opportunity to train the GP system means that the process has a greater appreciation of generalisation and avoiding as much overfitting as possible.

The number of folds (n) that should be utilised can vary. For this work, whilst each iteration of the cross-validation process will use the majority of the dataset for training, a sizeable portion will still be utilised for testing to make sure a greater variety of instances within the unbalanced datasets utilised are represented in the testing folds. As a result, the number of folds for this work will be 5, the same as that which is applied in [40] and [41] for example. This means that for each iteration of the cross-validation process, 80% of the dataset will be used for training (4 folds) and the remaining 20% is used for testing (1 fold).

4.4 High-Level Pseudocode

To provide a general visualisation of how GP is to be trained for the given classification problem instances, Algorithm 1 shows an extremely high-level pseudocode implementation.

Algorithm 1 GP Implementation

```
Start
    // GP training process
    Get the training and testing sets
    for number of generations do
        (1) Assign each classifier with a probability of being selected (for use by the roulette-
        wheel selection operator
        (2) Generate a new generation of individuals using the crossover and mutation
        operators via utilisation of the selection operator
        if a classifier with perfect fitness is found then
            return the perfectly fitting classifier found
            End
        end if
    end for
    return the best fitting classifier found
End
```

Algorithm 2 shows how this GP implementation shown in Algorithm 1 will be utilised with 5-fold cross-validation (as set out in Section 4.3) for the purpose of experimentation and evaluation of this GP approach.

Algorithm 2 GP Implementation Fitness Measuring using 5-Fold Cross-Validation

```
Start
    // 5-Fold Cross-Validation
    Split the dataset into 5 folds
    for each fold do
        Set the current fold to the testing set and combine the other folds to make the training
        set
        // GP training process
        for number of generations do
            (1) Assign each classifier with a probability of being selected (for use by the
            roulette-wheel selection operator
            (2) Generate a new generation of individuals using the set crossover and
            mutation operators via utilisation of the selection operator
            if a classifier with perfect fitness is found then
                continue to next fold
            end if
        end for
    end for
    return average fitness
End
```

5 Computational Experiments

5.1 Parameter Tuning Experiments

In order to determine the parameter settings that allow for better fitting classifiers to be generated, parameter tuning has been performed. As previously discussed, the parameters that will be tuned will be crossover probability, mutation probability, and maximum tree depth. The reason for this is because the effect of the setting of these hyperparameters can potentially be profound on the performance of the GP system, such as is found in [38].

As a result, parameter tuning has been performed on the GP system returning 245 GP run instances. As Figure 5 shows, the fitness for the GP system running on the BCD dataset remains relatively stable and high at roughly 0.91 for all the tested maximum tree depths (except for the maximum tree depth of 1 where both datasets are trained on relatively poorly). Since the GP training on the CMSC dataset generates more variability and poorer results, most of our attention must be spent on determining what works best for the CMSC dataset (since the BCD works well under most settings anyway).

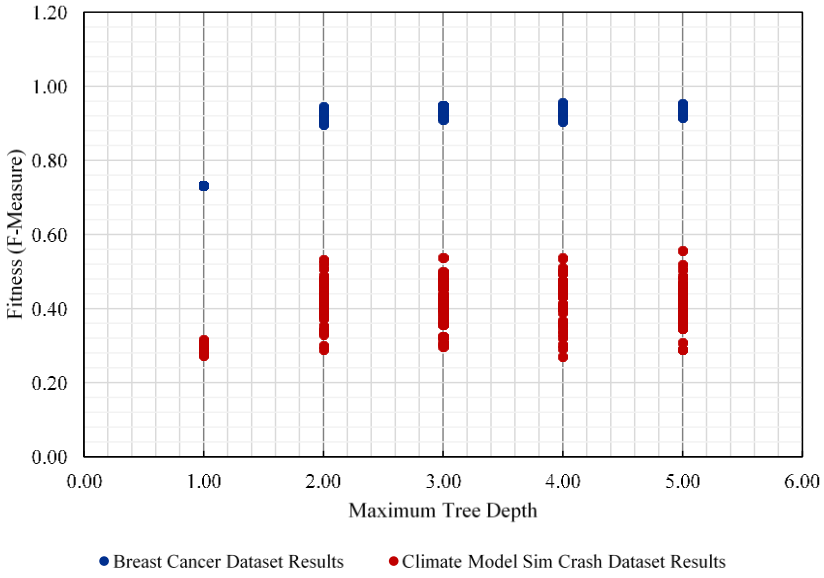

Figure 5: a scatter plot which shows the parameter tuning (f-measure) fitness results for both datasets depending on the maximum tree depth parameter.

5.1.1 Maximum Tree Depth

As justified in Section 4.2.1, the values that have been tested for the maximum tree depth are from 1 to 5. We must judge which maximum tree depth should be used. Table 3 shows the worst and best f-measure values found for each maximum tree depth during parameter tuning for the CMSC dataset, as well as the f-measure mean for each maximum tree depth too. Based on the information given by Table 3, the maximum tree depth that will be utilised is that of 3 as it has, for the CMSC dataset, the highest worst-case f-measure and joint-highest f-measure mean. In addition, whilst it does not have the highest best-case f-measure, it still has a decent f-measure of 0.54 which is only 0.02 less than the highest.

Table 3: the worst, best and mean f-measure values of the CMSC tuning instances for each maximum tree depth after each maximum tree depth was trialled 49 times.

Maximum Tree Depth	CMSC Dataset F-Measure Worst	CMSC Dataset F-Measure Best	CMSC Dataset F-Measure Mean
1	0.27	0.32	0.30
2	0.29	0.53	0.41
3	0.30	0.54	0.42
4	0.27	0.54	0.42
5	0.29	0.56	0.41

5.1.2 Crossover and Mutation Probabilities

The crossover probability has been tested on the values from 0.7 to 1.0 and the mutation probability has been tested on the values from 0.0 to 0.3 (as justified in Section 4.2.5). For both the crossover probability and the mutation probability, there is a 0.05 step between values tested, so:

- For crossover probability, the values of 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, and 1.0 have been tested.
- For mutation probability, the values of 0.0, 0.05, 0.1, 0.15, 0.2, 0.25, and 0.3 have been tested.

To determine the best parameters for the crossover and mutation probabilities, we will observe the tuning instances that were found to be the top 10 best performing in terms of f-measure fitness for the CMSC dataset, as is shown in Table 4.

Table 4: top 10 best performing (in terms of the CMSC dataset's f-measure) tuning instances.

Maximum Tree Depth	Crossover Probability	Mutation Probability	BCD Dataset F-Measure	CMSC Dataset F-Measure
5	0.85	0.30	0.95	0.56
3	1.00	0.10	0.92	0.54
4	0.90	0.15	0.95	0.54
2	0.75	0.15	0.93	0.53
2	0.90	0.05	0.91	0.52
5	0.80	0.25	0.95	0.52
4	0.90	0.20	0.92	0.51
2	0.95	0.15	0.92	0.51
5	0.90	0.00	0.94	0.50
4	0.70	0.30	0.94	0.50

We can't automatically choose the parameter settings of the best experimental instance in Table 4 because we need to take into account natural variabilities that can occur for each tuning instance, as GP systems are stochastic and can return perform differently even with the same parameter settings. In addition, it is important to note that the best performing parameter settings in Table 4 utilises a maximum tree depth of 5, which needs to be considered carefully as a higher maximum tree depth can potentially result in a greater level of bloating and, thus, overfitting, which we want to avoid.

As a result, an observation we can make from Table 4 is that the most common crossover and mutation probabilities here are 0.90 and 0.15 respectively, thus indicating that these are the parameters that are most closely associated with well-performing GP instances. Based on this, we will use these values for our crossover and mutation probabilities.

5.2 Final Parameter Settings

Table 5 shows the final parameter settings for this project, with the tuned parameter settings being set as justified in Section 5.1.

Table 5: the final GP parameter settings for this project.

Parameter	Setting
Terminal Set	Feature Values, Constant Values
Function Set	+, −, ×, ÷, exp, √, >, <, ==
Maximum Generation	7000
Population Size	1000
Selection Operator	Roulette-Wheel Selection
Crossover Operator	Standard Subtree Crossover
Mutation Operator	Standard Subtree Mutation
Crossover Probability	90%
Mutation Probability	15%
Termination Criteria	Classifier achieves perfect fitness or reach 100 generations
Maximum Tree Depth	3

5.3 Performance Evaluation

In order to evaluate the proposed GP system, we will compare its performance with that of a set of alternative machine learning approaches for applying our datasets. All of the alternative machine learning approaches proposed in this section have been implemented and will be ran in MATLAB. As is the same as what has been proposed for the GP system, each alternative approach will be operated on the BCD and CMSC datasets, and each will be executed using 5-fold cross-validation.

The use of linear discriminant analysis (LDA) for classification of the BCD dataset was suggested by [42], which is a method of classifying data by splitting the dataset linearly into discrete groups, thus producing a linear classifier. The class prediction y for a given datapoint is calculated

$$y = \boldsymbol{w}^T \boldsymbol{x}$$

where $\boldsymbol{x}$ is our feature vector and $\boldsymbol{w}$ is our weight vector. Note that LDA is a generalisation of Fisher LDA [43], thus meaning it reduces overlapping of the classification groups as much as possible by adjusting the values of $\boldsymbol{w}$.

1-nearest neighbour (1NN) is a method of classification proposed and utilised for the BCD dataset by [44], and is a variant of k-nearest neighbours (KNN). KNN provides non-linear classification and works by assigning a class to a given datapoint depending on the class of the closest other datapoints, i.e. each instance is assigned the same class as that of another datapoint we already know which minimises some distance function. Note then that KNN is non-parametric and is memory based as, for KNN to work, we need to keep all training data in memory so this data can be compared with incoming data instances to be classified. k determines how many nearby datapoints are considered during classification, such as k being 3 would mean only the three nearest other datapoints are considered. In the case of 1NN, k will be 1, meaning the process of classifying a given data instance will utilise only the single nearest other datapoint stored in the KNN model.

The support vector machine (SVM) method of classification is proposed and utilised for the CMSC dataset by [45]. SVMs involve the identifying of a linear hyperplane that best separates the dataset depending on the class. The mechanism which SVMs use to better divide the dataset is by maximising margins, which is the distance between the hyperplane and the nearest datapoint on either side, as a result, the idea is that the SVM will find and choose the hyperplane with the largest margins. Classification can then occur by determining which side of the hyperplane the given data point sits on. In cases where the dataset is non-separable and the different classes of the dataset are 'mixed' somewhat, a soft margin can be employed where a hyperplane can be found which allows some datapoints sitting on

the wrong side. For this SVM mechanism, not only is the aim to maximise the margins of the hyperplane, but also to penalise the datapoints on the wrong side of the hyperplane.

The proposed GP system as well as the additional machine learning approaches introduced here have all been performed on both datasets (BCD and CMSC) with the fitness results consisting of both accuracy and f-measure for each dataset presented in Table 6.

Table 6: fitness of the different machine learning implementations for classification using the BCD and CMSC datasets.

Machine Learning Implementation	BCD Dataset Accuracy	BCD Dataset F-Measure	CMSC Dataset Accuracy	CMSC Dataset F-Measure
GP	0.95	0.93	0.90	0.51
LDA	0.93	0.91	0.94	0.50
1NN	0.93	0.90	0.89	0.25
SVM	0.96	0.95	0.95	0.62

Note that each alternative machine learning implementation has been applied to both datasets, not just to the dataset that they were explicitly recommended for. This is to widen the scope of performance evaluation.

5.3.1 Performance Evaluation for the BCD Dataset

LDA and 1NN, two approaches suggested for the BCD dataset, perform worse than the proposed GP system for both fitness measures of accuracy and f-measure. This means that the GP system is not only able to generate a classifier for the BCD dataset when classifying data instances as a whole, but, as is indicated by the f-measure fitness, is also capable of making sure under-represented classes (as the BCD dataset is unbalanced) are not too highly misclassified. This fact is bolstered by the confusion matrix of Figure 6 which shows that a classifier generated by the GP system is able to correctly classify the vast majority of cases for both classes of the dataset. We can thus evaluate that the proposed GP system works well against the approaches suggested for the particular problem instance. The SVM approach has also been applied to the BCD dataset, despite its recommendation for the CMSC dataset by [45], and performs marginally better than the GP system in both fitness metrics considered. As a result, we can see that the SVM method is the best performing option considered here for the BCD dataset.

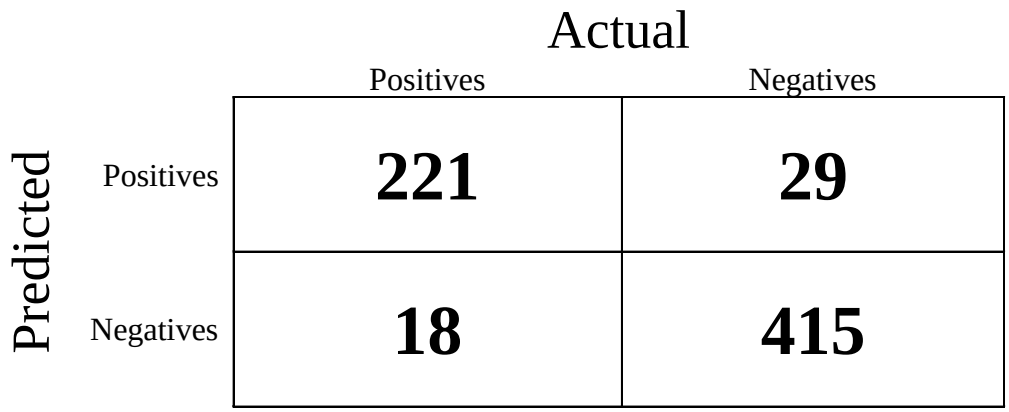

Figure 6: confusion matrix for the classifiers generated by the proposed GP system via the 5-fold cross-validation trained on the BCD dataset.

In terms of the evolution of the GP training process on the BCD dataset, Figure 7 shows how fitness for the classifiers generated improves as the training process goes along. As can be seen by Figure 7, the GP process very quickly finds a well-fitting classifier for the BCD dataset, with the maximum fitness not improving much after approximately 300 generations where a classifier with an f-measure fitness of 0.92 is identified. Furthermore, the f-measure fitness found at the end of the GP-training process is 0.94, only 0.02 greater than what is found after 300 generations. From this, we can thus tell that the maximum number of generations for the GP training process is not too low as the fitness evolution levels out relatively quickly, in fact, it could be said that the maximum number of generations for this dataset is needlessly high as the training process spends the vast majority of its time not finding any better fitting classifiers.

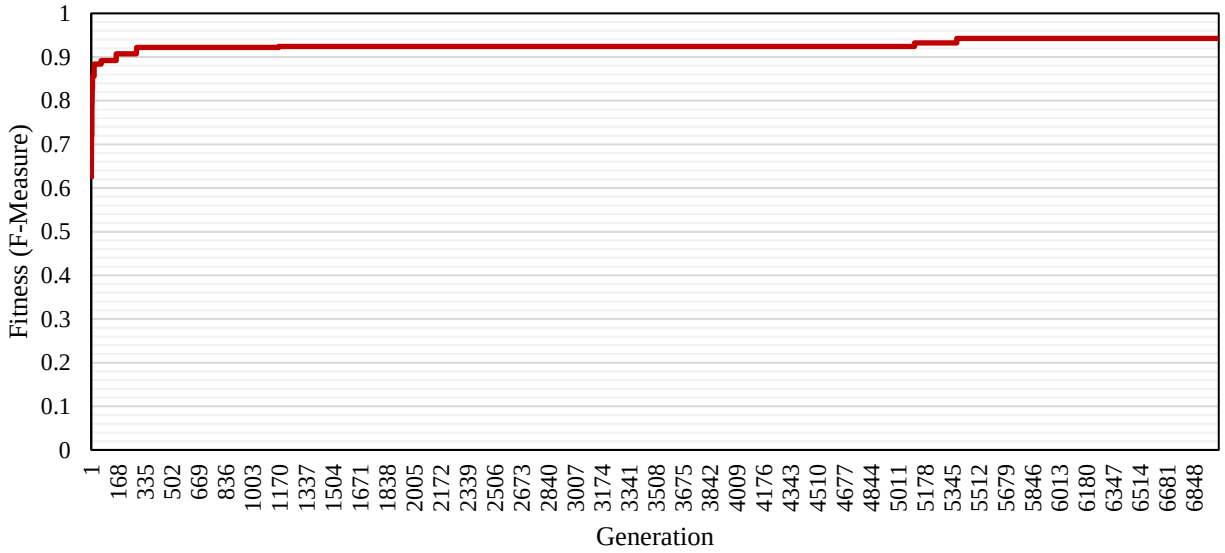

Figure 7: line chart showing the evolution of the maximum fitness measured as the GP system is trained on the BCD dataset.

The parse tree representation for a particular classifier produced for the BCD dataset by the GP system is visualised in Figure 8. In this case, the classifier has an accuracy fitness of 0.97 and an f-measure fitness of 0.96 when tested on a single fold of the cross-validation process. As Figure 8 shows, the parse tree classifier produced by the GP system after training has no redundancy, i.e. there are no branches of the tree that are needlessly verbose (such as a branch which multiplies a feature value by 1). As a result, we can see that the GP system can produce good parse trees with each branch being useful and not overly complex, thus in keeping with the principle of Occam's Razor.

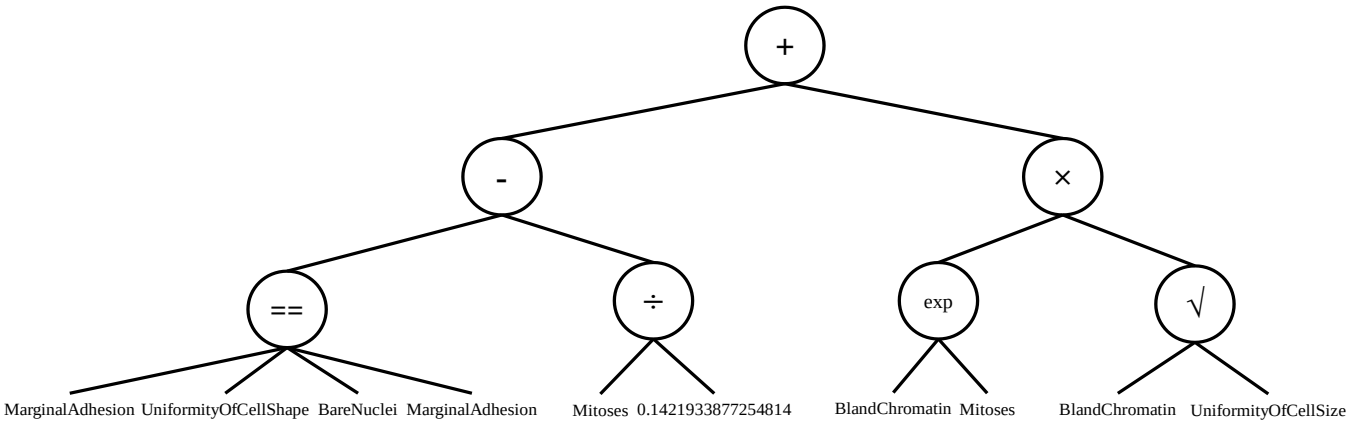

Figure 8: a visualisation of a parse tree classifier produced by the GP system for the BCD dataset.

5.3.2 Performance Evaluation for the CMSC Dataset

For the CMSC dataset, the machine learning method suggested by [45] is that of SVM. Like for the BCD dataset, it also works well for the CMSC dataset. When compared with our proposed GP system, we can observe that the SVM performs better than the GP system for both performance metrics. In terms of accuracy, the GP system does well but is 5% lesser than the SVM implementation's accuracy, meaning that SVM is better at classifying instances for the CMSC dataset generally. In terms of the f-measure statistic, the performance achieved by SVM dwarfs that of the GP system by 11%, meaning that the SVM is also much better at correctly classifying data instances of the under-represented classes specifically than the GP system. The proposed GP system's relatively poorer f-measure performance is reflected also by the confusion matrix of Figure 9, where the number of correctly classified positive cases is dwarfed by the number of incorrectly classified negative cases for a classifier generated by the GP system for this dataset.

The 1NN approach recommended for the BCD dataset works relatively poorly here for the CMSC dataset, as both the accuracy and f-measure statistics for the machine learning method are the poorest measured. The GP system does a better job at producing a classifier that fits well, particularly when it comes to correctly classifying data instances of the under-represented classes as the f-measure recorded for the GP system is 26% greater than that for 1NN. The other approach recommended for the BCD dataset, LDA, demonstrates an interesting quirk when applying such a method of classification to a dataset as unbalanced as the CMSC dataset: its accuracy is better than that of the proposed GP system,

but its f-measure is worse (albeit only slightly). The CMSC dataset is a heavily unbalanced dataset (far more so than the BCD dataset), meaning an excellent accuracy measure can obscure the fact that instances of the under-represented class are being poorly classified, as seems to be apparent here with the LDA implementation. As a result, we can say that LDA here does a better job of generally classifying data instances correctly, but the GP system can more successfully classify under-represented class instances.

		Actual: Positives	Actual: Negatives
Predicted	Positives	**26**	**49**
Predicted	Negatives	**20**	**445**

Figure 9: confusion matrix for the classifiers generated by the proposed GP system via the 5-fold cross-validation trained on the CMSC dataset.

With regards to the evolution of the GP training process on the CMSC dataset, Figure 10 shows how fitness for the classifiers generated improves as the training process progresses. As Figure 10 shows, the training process finds it's best fitting classifier for the CMSC dataset relatively quickly after approximately 2050 generations. In this case, the best fitting classifier found is one that has an f-measure fitness of just over 0.50, which is identified via GP training far before the maximum generation is encountered. Given the GP system's inability to perform as well on CMSC dataset compared to the BCD dataset, a particular concern was as to whether the maximum generation of 7000 was 'cutting off' the GP system prematurely, and that a greater maximum generation would result in a much better fitness being discovered by the GP system, but given what has been found and displayed in Figure 10, this is not the case. Like the BCD dataset in Section 5.3.1, the maximum generation appears needlessly high as the best fitting classifier is found long before the maximum generation is met. Instead, what seems to be the issue is that the evolution of the classifiers can get stuck quite substantially, as is shown in Figure 10 by some of the elongated periods in the evolution process where no improved fitness is observed. This may, again, be down to the unbalanced nature of the dataset having a profound effect, but it may also mean that the parameter settings used are not optimal, and could be further tuned if more time was available to do so.

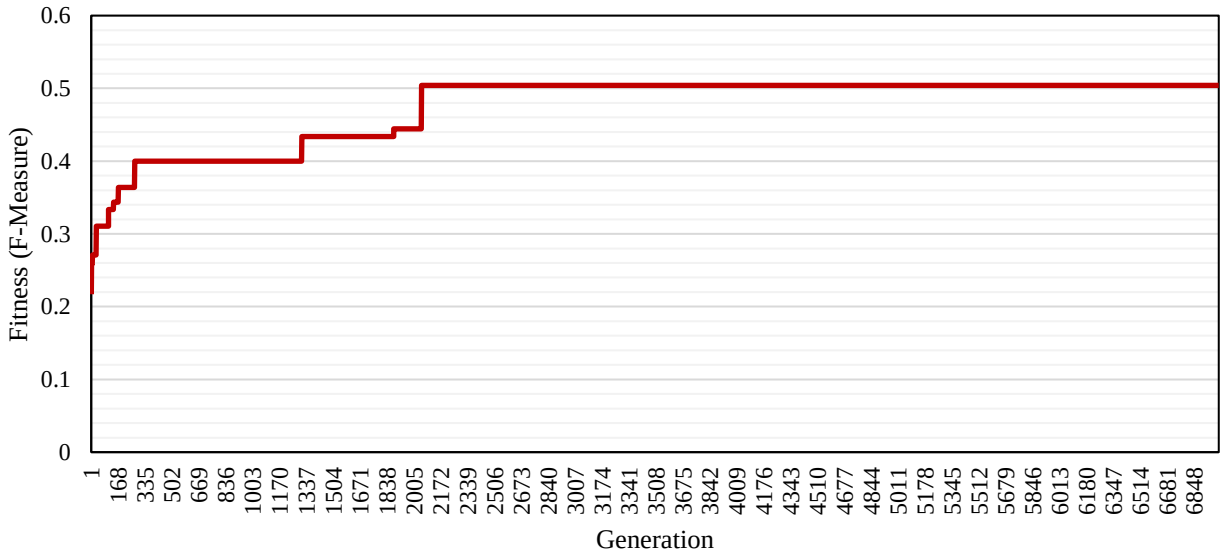

Figure 10: line chart showing the evolution of the maximum fitness measured as the GP system is trained on the CMSC dataset.

The parse tree representation for a particular classifier produced for the CMSC dataset by the GP system is visualised in Figure 11. In this case, the classifier has an accuracy fitness of 0.85 and an f-measure fitness of 0.47 when tested on a single fold of the cross-validation process. Like for the parse tree generated for the BCD dataset in Section 5.3.1 (and

shown in Figure 8), Figure 11 shows that there is no redundancy and no branches of the tree that are needlessly complex, and that the maximum depth of the tree is fully utilised in accordance with the principles of Occam's Razor.

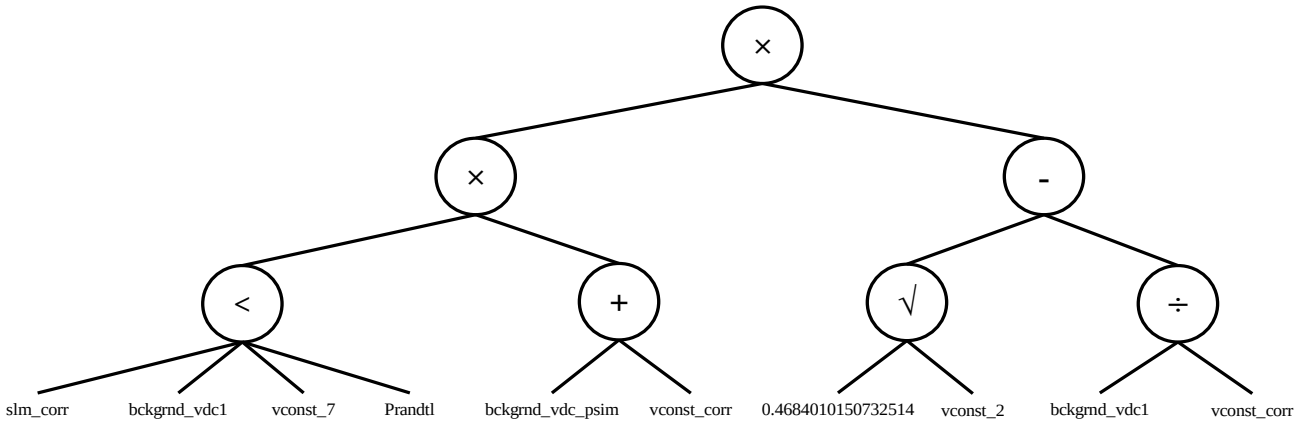

Figure 11: a visualisation of a parse tree classifier produced by the GP system for the CMSC dataset.

5.3.3 Concluding Remarks

The GP system works reasonably well for both the BCD and CMSC datasets. For the BCD dataset, its accuracy and f-measure performance measures indicate that it works exceptionally well in classifying BCD dataset instances, better than the approaches suggested for the BCD dataset, and only being bettered by the SVM method. The GP system also works well for the CMSC dataset, but less so than the BCD dataset. It's accuracy for the CMSC dataset is good but not as good as the LDA and SVM alternatives, and only marginally better then 1NN, but for f-measure, it fares much better and is only bettered by SVM.

The main general observation we can make is that the GP system performs better on the BCD dataset than it does for the CMSC dataset. There could be various reasons for this, including that the relationship between the features and the classes in the CMSC dataset are not as easily recognised by the GP system, but the more pressing notability is that the CMSC dataset is heavily unbalanced (as oppose to the BCD dataset which is less heavily unbalanced), meaning we can infer that the GP system's performance relies substantially on the balancing of the dataset that is used. Moreover, the GP system seems to get regularly stuck when evolving the classifiers for the CMSC dataset, unlike for the BCD dataset which is less prone to such an occurrence, potentially hampering the GP system's performance on the CMSC dataset further.

Additionally, for both datasets, it has been observed that finding the maximum best fitting classifier during GP training happens relatively quickly, and that the final classifier with the highest fitness is generated far before the maximum generation is encountered. From this, we can infer that it is potentially the case that a lower maximum generation parameter could be utilised instead so that the GP system doesn't have to be trained needlessly for too long after the best fitting classifier is found. Furthermore, it has been found for both datasets that the classifier parse trees produced for the best fitting classifiers are good and abide by the principles of Occam's Razor such that no redundant and overly complex branches are present in these trees.

6 Summary and Reflections

6.1 Project Management

Figure 12 shows the work plan involved in the completion of the project as set out in the original project proposal, including an overview of what tasks and objectives have been completed and the sequence in which those tasks have been performed.

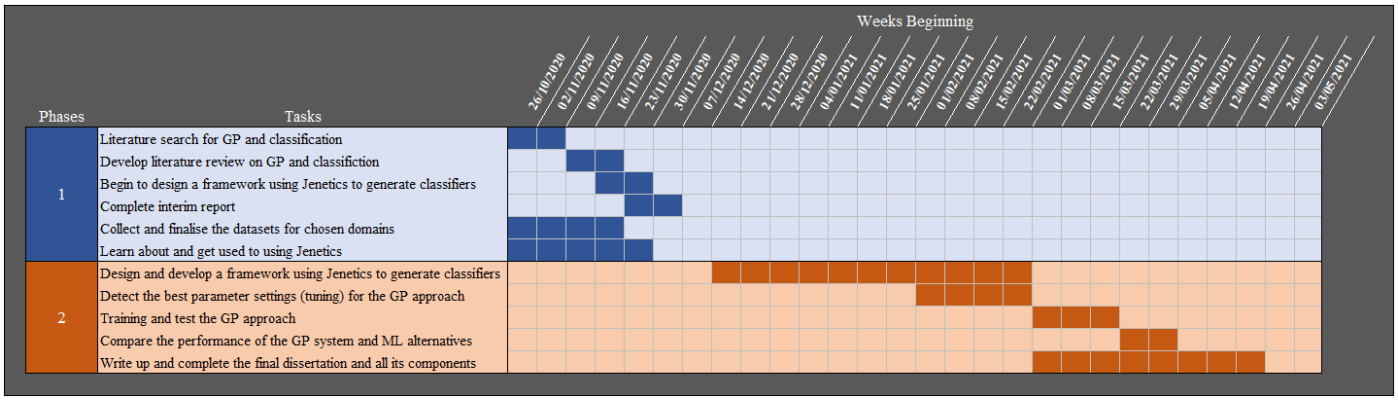

Figure 12: Gantt chart visualising the organisation and management of the project and the tasks involved.

The work plan was divided into two phases, the first being for project research and preparation for development, and the second being for project development and final dissertation completion.

6.1.1 The First Phase

The first phase finished on the week starting 14/12/2020 with the submission of the interim report being the final task. In terms of the individual tasks within phase 1:

- *Literature search for GP and classification*: this took two weeks (in line with the original plan) to complete and took place largely before the literature review began such as to make sure all significant materials were in place for the review. Unlike what was originally planned, a minor level of literature search continued during the literature review, such as to provide additional knowledge and reasoning to support the review process.
- *Develop literature review on GP and classification*: this took two weeks (in line with the original plan). This task was important as it provided a pool of knowledge that has been useful for the planning and undertaking of the project.
- *Begin to design a framework using Jenetics to generate classifiers*: this took two weeks (in line with the original plan) and began when a sufficient amount of knowledge had been gained during the literature review process. What had been set out here was a foundation level of planning for how the project was to be implemented, and how Jenetics would be used to generate classifiers. This has been built upon in phase 2.
- *Complete interim report*: the main bulk of this took two weeks to fully complete (in line with the original plan), although some work had been started on it as early as two weeks before planned. It is the final task for phase 1 and had been completed ready to be submitted by the deadline on 09/12/2020.
- *Collect and finalise the datasets for chosen domains*: this had taken place over the period before it was necessary to begin designing a framework using Jenetics to generate classifiers (in line with the original plan). Here, the task was merely to make sure the datasets needed for the problem domains were collected and properly understood.
- *Learn about and get used to using Jenetics*: this had taken place over the majority of the period set out for phase 1 (in line with the original plan). This task was to use time set aside to learn and get experience using different Jenetics systems and features, ready for the implementation task in phase 2.

6.1.2 The Second Phase

The second phase finished on the week starting 10/04/2021 with the submission of this report being the final task. In terms of the individual tasks within phase 2:

- *Design and develop a framework using Jenetics to generate classifiers*: this task took approximately 11 weeks to complete (in line with the original plan). This task involved using Jenetics to implement a Java program that generates classifiers via the use of genetic programming. A substantial amount of time was allocated to this task as it involved a lot of work and was the source of a great deal of unforeseen issues and irregularities.
- *Detect the best parameter settings (tuning) for the GP approach*: this task took approximately six weeks to complete, and took place towards the end of the framework development process, so that tuning could occur using a Jenetics framework that was implemented already. This task took slightly longer than was originally planned as finding the optimal parameters to tune took longer to finally determine than originally thought.
- *Training and test the GP approach*: this took three weeks to complete (in line with the original plan), and took place once the framework using Jenetics had been implemented and tuned. This task involved the running of the project implementation and the testing of its performance for the set problem domains.

- *Compare the performance of the GP system and ML alternatives*: this took approximately three weeks to consider appropriately. Note that this task involved the research and understanding of performances achieved by other machine learning algorithms and systems, and the comparison of those performances with the performance of the GP system implemented. This look slightly longer than originally planned as the time required to identify recommended machine learning methods and the implementation of such systems in MATLAB was underestimated in the original plan.
- *Write up and complete the final dissertation and all its components*: this task will took approximately five weeks to complete. It should began once the framework of the system had been implemented, so that discussion on how the project had been implemented could be performed initially. It also includes the collation of all important project components, most notably being the analysis and discussion of the system's performance, particularly when compared against machine learning alternatives. This task has taken notably less amount of time than originally planned as a lot of the initial work could be taken directly or in part from the interim report, reducing the amount of time needed to work on the final dissertation specifically.

6.2 Contributions and Reflections

For this project, following appropriate research and preparation in the relevant areas, a GP system has been implemented which, using evolutionary techniques, is able to produce binary classifiers that are characterised by good fitness and an absence of bloating and overfitting. Through a process of tuning some of the hyperparameters involved as explained in Section 5 to maximise performance, the GP system implemented works particularly well when trained on datasets such as BCD which is only unbalanced slightly, achieving levels of fitness greater than that of suggested alternative methods (as found in Section 6). Due to this good performance, I am particularly pleased with the outcome of this work, though I note some disappointment in the poorer performance of the GP system on the CMSC dataset (though performance amongst alternative machine learning methods also seem to suffer for this dataset). Even given the heavily unbalanced nature of the CMSC dataset plainly limits the success of any machine learning method's application, for future work, it would still be interesting to consider how this GP system can be improved to perform better on the CMSC dataset. One such option is to implement more rigid dimensionality reduction techniques to improve the focus of the GP system on the more informative of inputs in the dataset. Such dimensionality reduction could be useful on the CMSC dataset, as it has a higher number of features compared to the BCD dataset, and a focus on the more informative features could potentially improve the performance of the system on the CMSC dataset. Additionally, it may be interesting to consider further parameter tuning in the future to determine settings that may be more able to prevent the GP system getting stuck in the evolution process for the CMSC dataset as is found in Section 5.3.2. However, overall, I am happy with the results of this project and what I have achieved in it, satisfied not only in its particular success when trained upon the BCD dataset, but also the potential room for extended research into modifying the proposed GP system to better perform on heavily unbalanced datasets such as the CMSC dataset.

Reflecting on the decision to utilise the Jenetics Java library for the implementation of the GP system, it was originally planned that a different Java framework would be utilised: ECJ. After spending a number of weeks on the task *Learn about and get used to using ECJ* (now set as *Learn about and get used to using Jenetics*) as covered in Section 7.1, I had a level of trouble in terms of implementing some of the features that the ECJ framework provides that would be useful for the project. In addition, its older nature meant that some of its utilities were non-consistent with the modern advantages of Java, such as generic types and additional performance improving mechanisms. As a result, I researched another set of frameworks and libraries available for Java that could be harnessed to implement GP for classification as set out in this project. After performing this research, I concluded on using Jenetics instead, which meant that some of my experience in getting used to the library that would was to be utilised in the project had been delayed by a small set of weeks. Despite this however, the wider scope of online support for Jenetics meant that adapting to the new library was much quicker and easier than would have been the case with ECJ. Having now implemented the project in Java, Jenetics has worked suitably to provide the necessary functionality for the project, making available a range of facilities used implement the GP system that makes use of modern Java features and implements performance improving mechanisms such as parallelisation.

In terms of following the plan for the project, I have found it mostly straightforward to follow. In most cases, the time allocated for each task has well reflected the actual time that was necessary in practice, and there has been no need to radically reshape or restructure the project plan. This has been encouraged by good time management and organisation skills that I have been able to present in my work so far in the project. In cases where I have struggled to strictly follow to the constraints of the plan, my main weaknesses have mainly derived from an initial underappreciation of how much time would be required to complete certain tasks to a particularly high standard, such as for tuning the GP system and implementing the alternative machine learning systems. I'd also like to note that, not only have I gained a greater insight

into the implementation and performance of GP systems, I have also garnered a more thorough awareness and appreciation of the research techniques that I have had to employ, as well as gaining greater experience in structuring communicating my work via reports such as this, all of which is instrumental in being able to adequately present this work.

7 References

[1] J. S. Vuppalapati, S. Kedari, A. Ilapakurti, C. Vuppalapati, S. Kedari and R. Vuppalapati, "The Development of Machine Learning Infused Outpatient Prognostic Models for tackling Impacts of Climate Change and ensuring Delivery of Effective Population Health Services," *2019 IEEE International Conference on Big Data (Big Data),* pp. 2790-2799, 2019.

[2] S. Tuli, S. Tuli, R. Tuli and S. S. Gill, "Predicting the growth and trend of COVID-19 pandemic using machine learning and cloud computing," *Internet of Things,* vol. 11, p. 100222, 2020.

[3] D. Ronfeldt and D. Varda, "The Prospects for Cyberocracy Revisited," in *Culture and Civilization: Volume 2: Beyond Positivism and Historicism,* New Brunswick, Transaction Publishers, 2010, pp. 130-133.

[4] F. Zhu, X. Li, H. Tang, Z. He, C. Zhang, G.-U. Hung, P.-Y. Chiu and W. Zhou, "Machine Learning for the Preliminary Diagnosis of Dementia," *Scientific Programming,* vol. 2020, pp. 1-10, 2020.

[5] R. Poli, W. B. Langdon and N. F. McPhee, A field guide to genetic programming, Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008.

[6] D. D. Lucas, R. Klein, J. Tannahill, D. Ivanova, S. Brandon, D. Domyancic and Y. Zhang, "Failure analysis of parameter-induced simulation crashes inclimate models," *Geosci. Model Dev.,* vol. 6, pp. 1157-1171, 2013.

[7] I. Dilrukshi and K. D. Zoysa, "Twitter news classification: Theoretical and practical comparison of SVM against Naive Bayes algorithms," *2013 International Conference on Advances in ICT for Emerging Regions (ICTer),* pp. 278-278, 2013.

[8] J. V. Rissati, P. C. Molina and C. S. Anjos, "Hyperspectral Image Classification Using Random Forest and Deep Learning Algorithms," *2020 IEEE Latin American GRSS ISPRS Remote Sensing Conference (LAGIRS),* pp. 132-132, 2020.

[9] J. Hu and M. Xie, "Fingerprint classification based on genetic programming," *2010 2nd International Conference on Computer Engineering and Technology,* vol. 6, pp. V6-193-V6-196, 2010.

[10] M. Brameier and W. Banzhaf, "A Comparison of Genetic Programming and Neural Networks in Medical Data Analysis," *IEEE Transactions on Evolutionary Computation,* vol. 5, no. 1, pp. 7-26, 2001.

[11] N. Hameed, A. M. Shabut, M. K. Ghosh and M. A. Hossain, "Multi-class multi-level classification algorithm for skin lesions classification using machine learning techniques," *Expert Systems with Applications,* vol. 141, p. 112961, 2020.

[12] P. J. C. Suen, S. Goerigk, L. B. Razza, F. Padberg, I. C. Passos and A. R. Brunoni, "Classification of unipolar and bipolar depression using machine learning techniques," *Psychiatry Research,* vol. 295, p. 113624, 2021.

[13] M. Amrane, S. Oukid, I. Gagaoua and T. Ensarİ, "Breast cancer classification using machine learning," *2018 Electric Electronics, Computer Science, Biomedical Engineerings' Meeting (EBBT),* pp. 1-4, 2018.

[14] S.-H. Hsieh, Z. Wang, P.-H. Cheng, I.-S. Lee, S.-L. Hsieh and F. Lai, "Leukemia cancer classification based on Support Vector Machine," *2010 8th IEEE International Conference on Industrial Informatics,* pp. 819-824, 2010.

[15] M. Hosseini, A. Bigtashi and B. Lee, "Generating future weather files under climate change scenarios to support building energy simulation – A machine learning approach," *Energy and Buildings,* vol. 230, p. 110543, 2021.

[16] A. B. Ranit and P. V. Durge, "Flood Forecasting by Using Machine Learning," *2019 International Conference on Communication and Electronics Systems (ICCES),* pp. 166-169, 2019.

[17] A. A. Kurniawan, K. Usman and R. Y. N. Fuadah, "Classification of Tropical Cyclone Intensity on Satellite Infrared Imagery Using SVM Method," *2019 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob),* pp. 69-73, 2019.

[18] C. A. Stock, M. A. Alexander, N. A. Bond, K. M. Brander, W. W. Cheung, E. N. Curchitser, T. L. Delworth, J. P. Dunne, S. M. Griffies, M. A. Haltuch, J. A. Hare, A. B. Hollowed, P. Lehodey, S. A. Levin, J. S. Link, K. A. Rose, R. R. Rykaczewski, J. L. Sarmiento, R. J. Stouffer, F. B. Schwing, G. A. Vecchi and F. E. Werner, "On the use of IPCC-class models to assess the impact of climate on Living Marine Resources," *Progress in Oceanography,* vol. 88, no. 1, pp. 1-27, 2011.

[19] W. Banzhaf, J. R. Koza, C. Ryan, L. Spector and C. Jacob, "Genetic programming," *IEEE Intelligent Systems and their Applications,* vol. 15, no. 3, pp. 74-84, 2000.

[20] I. D. Falco, A. D. Cioppa and E. Tarantino, "Discovering interesting classification rules with genetic programming," *Applied Soft Computing,* vol. 1, no. 4, pp. 257-269, 2002.

[21] N. S. Chaudhari, A. Purohit and A. Tiwari, "A multiclass classifier using Genetic Programming," *2008 10th International Conference on Control, Automation, Robotics and Vision,* pp. 1884-1887, 2008.

[22] T.-S. Wang, L. Chen and M.-M. Wu, "Application of Genetic Programming to Stream-Flow Extension," *2007 International Conference on Machine Learning and Cybernetics,* vol. 2, pp. 974-977, 2007.

[23] A. H. Gandomi, D. Mohammadzadeh S., J. L. Pérez-Ordóñez and A. H. Alavi, "Linear genetic programming for shear strength prediction of reinforced concrete beams without stirrups," *Applied Soft Computing,* vol. 19, pp. 112-120, 2014.

[24] W. La Cava, S. Silva, K. Danai, L. Spector, L. Vanneschi and J. H. Moore, "Multidimensional genetic programming for multiclass classification," *Swarm and Evolutionary Computation,* vol. 44, pp. 260-272, 2019.

[25] J. Ma and G. Teng, "A hybrid multiple feature construction approach for classification using Genetic Programming," *Applied Soft Computing,* vol. 80, pp. 687-699, 2019.

[26] S. Wang, Q. Zhao, Y. Chen and P. Wu, "Function Sequence Genetic Programming for Pattern Classification," *2011 Seventh International Conference on Natural Computation,* vol. 2, pp. 1092-1096, 2011.

[27] G. Canbek, S. Sagiroglu, T. T. Temizel and N. Baykal, "Binary classification performance measures/metrics: A comprehensive visualized roadmap to gain new insights," *2017 International Conference on Computer Science and Engineering (UBMK),* pp. 821-826, 2017.

[28] A. Luque, A. Carrasco, A. Martín and A. d. l. Heras, "The impact of class imbalance in classification performance metrics based on the binary confusion matrix," *Pattern Recognition,* vol. 91, pp. 216-231, 2019.

[29] S. Sharma and S. K. Srivastava, "Feature Based Performance Evaluation of Support Vector Machine on Binary Classification," *2016 Second International Conference on Computational Intelligence Communication Technology (CICT),* pp. 170-173, 2016.

[30] T. Helmuth, L. Spector and J. Matheson, "Solving Uncompromising Problems With Lexicase Selection," *IEEE Transactions on Evolutionary Computation,* vol. 19, no. 5, pp. 630-643, 2015.

[31] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Physica A: Statistical Mechanics and its Applications,* vol. 391, no. 6, pp. 2193-2196, 2012.

[32] M. Zhang, X. Gao and W. Lou, "A New Crossover Operator in Genetic Programming for Object Classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* vol. 37, no. 5, pp. 1332-1343, 2007.

[33] A. Kumar, N. Sinha and A. Bhardwaj, “A novel fitness function in genetic programming for medical data classification,” *Journal of Biomedical Informatics,* vol. 112, p. 103623, 2020.

[34] A. M. Mansuri, D. Kelkar, R. Kumar, P. S. Rathore and A. Jain, “Multiclass classifier designing by Modified Crossover and Point Mutation technique using genetic programming,” *{2012 Ninth International Conference on Wireless and Optical Communications Networks (WOCN),* pp. 1-7, 2012.

[35] A. Piszcz and T. Soule, “Genetic programming: optimal population sizes for varying complexity problems,” *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation,* pp. 953-954, 2006.

[36] S. M. Cheang, K. H. Lee and K. S. Leung, “Evolving data classification programs using genetic parallel programming,” *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.,* vol. 1, pp. 248-255, 2003.

[37] H. Jabeen and A. R. Baig, “Two layered Genetic Programming for mixed-attribute data classification,” *Applied Soft Computing,* vol. 12, no. 1, pp. 416-422, 2012.

[38] J.-H. Hong and S.-B. Cho, “The classification of cancer based on DNA microarray data that uses diverse ensemble genetic programming,” *Artificial Intelligence in Medicine,* vol. 36, no. 1, pp. 43-58, 2006.

[39] Y.-M. Li, M. Wang, L.-J. Cui and D.-M. Huang, “A New Classification Arithmetic for Multi-Image Classification in Genetic Programming,” *2007 International Conference on Machine Learning and Cybernetics,* vol. 3, pp. 1683-1687, 2007.

[40] Y. Bi, B. Xue and M. Zhang, “Multi-objective genetic programming for feature learning in face recognition,” *Applied Soft Computing,* vol. 103, p. 107152, 2021.

[41] A. B. Junior, N. F. F. d. Silva, T. C. Rosa and C. G. Junior, “Sentiment analysis with genetic programming,” *Information Sciences,* vol. 562, pp. 116-135, 2021.

[42] M. C. Ferris and O. L. Mangasarian, “Breast Cancer Diagnosis via Linear Programming,” *IEEE Computational Science and Engineering,* vol. 2, no. 3, pp. 70-71, 1995.

[43] R. A. Fisher, “The Use of Multiple Measurements in Taxonomic Problems,” *Annals of Eugenics,* vol. 7, no. 2, pp. 179-188, 1936.

[44] J. Zhang, “Selecting Typical Instances in Instance-Based Learning,” *Machine Learning Proceedings 1992,* pp. 470-479, 1992.

[45] D. D. Lucas, R. Klein, J. Tannahill, D. Ivanova, S. Brandon, D. Domyancic and Y. Zhang, “Failure analysis of parameter-induced simulation crashes in climate models,” *Geoscientific Model Development,* vol. 6, no. 4, pp. 1157-1171, 2013.