AUTOMATED MACHINE LEARNING FOR POSITIVE UNLABELLED LEARNING

A THESIS SUBMITTED TO
THE UNIVERSITY OF KENT
IN THE SUBJECT OF COMPUTER SCIENCE
FOR THE DEGREE
OF PHD.

By
Jack Duke Saunders
August 2023

Abstract

Positive-Unlabelled (PU) learning is a field of machine learning that involves learning classifiers from data consisting of positive class and unlabelled instances. That is, instances that may be either positive or negative, but the label is unknown. PU learning differs from standard binary classification due to the absence of negative instances. This difference is non-trivial and requires differing classification frameworks and evaluation metrics. This thesis looks to address gaps in the PU learning literature and make PU learning more accessible to non-experts by introducing Automated Machine Learning (Auto-ML) systems specific to PU learning. Three such systems have been developed, GA-Auto-PU, a Genetic Algorithm (GA)-based Auto-ML system, BO-Auto-PU, a Bayesian Optimisation (BO)-based Auto-ML system, and EBO-Auto-PU, an Evolutionary/Bayesian Optimisation (EBO) hybrid-based Auto-ML system.

These three Auto-ML systems are three primary contributions of this work. EBO, the optimiser component of EBO-Auto-PU, is by itself a novel optimisation method developed in this work that has proved effective for the task of Auto-ML and represents another contribution. EBO was developed with the aim of acting as a trade-off between GA, which achieved high predictive performance but at high computational expense, and BO, which, when utilised by the Auto-PU system, did not perform as well as the GA-based system but did execute much faster. EBO achieved this aim, providing high predictive performance with a computational runtime much faster than the GA-based system, and not substantially slower than the BO-based system.

The proposed Auto-ML systems for PU learning were evaluated on three versions of 40 datasets, thus evaluated on 120 learning tasks in total. The 40 datasets consist of 20 real-world biomedical

datasets and 20 synthetic datasets. The main evaluation measure was the F-measure, a popular measure in PU learning. Based on the F-measure results, the three proposed systems outperformed in general two baseline PU learning methods, usually with statistically significant results. Among the three proposed systems, there was no statistically significance difference between their results in general, whilst a version of the EBO-Auto-PU system performed overall slightly better than the other systems, in terms of F-measure.

The two other main contributions of this work relate specifically to the field of PU learning. Firstly, in this work we present and utilise a robust evaluation approach. Evaluating PU learning classifiers is non-trivial and little guidance has been provided in the literature on how to do so. In this work, we present a clear framework for evaluation and use this framework to evaluate the proposed systems. Secondly, when evaluating the proposed systems, an analysis of the most frequently selected components of the optimised PU learning algorithm is presented. That is, the components that constitute the PU learning algorithms produced by the optimisers (for example, the choice of classifiers used in the algorithm, the number of iterations, etc.). This analysis is used to provide guidance on the construction of PU learning algorithms for specific dataset characteristics.

Acknowledgements

First and foremost, I would like to give thanks to Professor Alex Freitas. Attending university was not something I had ever considered an option for myself, let alone the pursuit of a PhD. The patience and time given by Professor Freitas has been unwavering and invaluable, and I am profoundly grateful for the opportunity that he has given me.

I would also like to give thanks to the University of Kent computing department, in particular the members of my supervisory panel and the Director of Graduate Studies, Daniel Soria, Fernando Otero, and Rogério de Lemos. The feedback and guidance given throughout the previous years of study have been instrumental to the formation of this work.

My research has been funded by a studentship granted by the Engineering & Physical Sciences Research Council (EPSRC), and as such I would like to give thanks to EPSRC for this opportunity.

I am grateful to the anonymous reviewers who have given time to evaluating my publication submissions and providing feedback and insight on this work.

There are also several key individuals in my personal life to whom I would like to give thanks. Firstly, to Ryan, for encouraging me to undertake the PhD and for constantly being a source of support and inspiration. Secondly, to Jess, for her time, patience, and for keeping me sane over the last few years. And, finally, to Annabel - without you, none of this would have been possible.

Contents

List of Tables

List of Figures

List of Procedures' Pseudocodes

Glossary

AUC-ROC	Area under the receiver operating characteristic curve
Auto-ML	Automated machine learning
BO	Bayesian optimisation
CASH	Combined algorithm selection and hyperparameter optimisation
CWA	Closed world assumption
DF-PU	Deep Forest for Positive-Unlabelled learning
EA	Evolutionary algorithm
EBO	Evolutionary Bayesian Optimisation
EDA	Estimation of distribution algorithm
EM	Expectation maximisation
EI	Expected improvement
FN	False negative
FP	False positive
FPGA	Field-programmable gate array
GA	Genetic algorithm
GGP	Grammar-based genetic programming
GP	Genetic programming
GPC	Gaussian process classifier
GPU	Graphics processing unit
LDA	Linear discriminant analysis
kNN	K-nearest neighbour
MAE	Mean absolute error
ML	Machine learning

MLC	Multilabel classification
MLP	Multilayer perceptron
MSE	Mean squared error
PEBL	Positive example based learning
PI	Probability of improvement
PN	Positive-negative
PU	Positive-unlabelled
RECIPE	Resilient Classification Pipeline Evolution system
ReLU	Rectified linear unit
RN	Reliable negative
SAR	Selected at random
SCAR	Selected completely at random
S-EM	Spy method with Expectation Maximisation
SGD	**S**tochastic gradient descent
SMBO	**S**equential model-based optimisation
SVM	Support vector machine
TGP	Tree-based GP
TN	True negative
TP	True positive
TPOT	Tree-based Pipeline Optimisation Tool

Chapter 1

Introduction

Classification is a type of supervised machine learning task where an algorithm essentially learns, from data, a model (classifier) to categorise objects (instances) based on the characteristics (features) of those objects [1][2]. Standard binary classification methods have proved an invaluable type of machine learning in recent years across a wide variety of application domains.

However, certain learning tasks that do not fit the binary classification paradigm are often treated as though they do. When curating real-world datasets, obtaining fully labelled data may prove challenging. There are many scenarios where labelling data is expensive or impractical. Consider data regarding the classification of genes (instances) into class labels representing gene functions. Genes either have evidence associating them with a certain function (class label), or they do not, characterizing a binary classification problem. If there is evidence that a gene has a certain function (as a result of a biological experiment), the instance representing that gene will be labelled with the positive class for that function. However, it is harder to find reliable instances of the negative class because a lack of evidence associating a gene with a specific function is not evidence for a lack of association. Biological experiments are expensive and time-consuming to conduct; hence, it is likely that a given gene has simply not had any experiments conducted on it to confirm whether it is associated with the particular function or not. If we were to train a standard binary classifier on such data with the given class labels, we would train the classifier to predict whether a gene is labelled as having a certain function, rather than whether a gene actually is associated with that function.

Consider another example where obtaining fully labelled data is impractical. Web scraping is commonly used to collect vast amounts of data from the internet, which is then used for various purposes such as sentiment analysis, market research, or predictive modelling. However, the enormous volume and diversity of this scraped data make it almost impossible to manually label every single instance. Instead, a sample of the instances to be considered may be labelled, whilst the rest of the instances are left unlabelled. In this case, the resulting dataset would consist of a set of labelled positive instances (web pages which belong to the class of interest) and a set of unlabelled instances, which may be positive or negative, but whose class is unknown.

These examples characterise Positive-Unlabelled (PU) learning problems, which differ from standard binary classification due to the absence of a separately defined negative class in the dataset (that is, the concept of a negative class exists, but as the unlabelled instances may be either positive or negative, the negative class is not explicitly observed in the dataset) [3], a common scenario in domains such as bioinformatics [4], text classification [5], pharmacology [6], and others [3]. PU learning has not, however, been widely adopted in the literature. Many studies simply employ the closed world assumption (CWA) [7], assuming that unlabelled instances are negative instances. This work looks to address this issue.

Automated Machine Learning, often referred to as Auto-ML, is a rapidly advancing subfield of machine learning, which aims to automate complex aspects of the machine learning process. It focuses on algorithm selection and hyperparameter tuning, effectively optimizing algorithm configurations to ensure the best performance with minimum manual intervention [8][9]. The goal of Auto-ML is to simplify the machine learning process for non-experts and increase efficiency in model development by automatically searching for the best machine learning algorithm or pipeline (a set of algorithms applied in order) and their best hyperparameter settings for a given task. For a more detailed discussion of Auto-ML, see Sections 1.2 and 2.4.

In this work, we aim to make PU learning methods more accessible and robust with the introduction of new Auto-ML systems specific to PU learning. An Auto-ML system for PU learning will limit the need for expert involvement and make PU learning accessible to those with little knowledge of PU learning. Furthermore, we look to fill gaps in the current PU learning literature to allow for a more reliable and effective classification framework, aiming at producing an Auto-ML

system for PU learning that would achieve higher predictive performance than current PU learning methods. The proposed type of Auto-ML system, called Auto-PU, is developed as three separate Auto-ML systems utilising three types of optimisation methods: two well-known methods, a Genetic Algorithm (GA) and Bayesian optimisation (BO), and a new hybrid optimisation method named as evolutionary Bayesian optimisation (EBO). The three Auto-PU systems are novel contributions to the area of machine learning and particularly PU learning in general since they currently represent the only Auto-ML systems for PU learning in the literature.

The rest of this chapter is structured as follows: Sections 1.1 and 1.2 give an overview of PU learning and Auto-ML, highlighting the motivation for this work. Section 1.3 outlines the objectives to be achieved. Section 1.4 outlines the contributions of this thesis. Section 1.5 gives the structure of this thesis. Finally, Section 1.6 details the three publications derived from this work.

1.1 Positive-Unlabelled Learning (PU Learning)

PU learning is a classification paradigm that involves learning a machine learning classifier (model) that can distinguish between positive and negative classes, given only positive and unlabelled data [3]. PU learning is discussed in detail in Section 2.5 but is briefly outlined here to explain the motivation of this work.

Over the previous two decades, many PU learning algorithms have been developed with the aim of learning classifiers from positive and unlabelled data [3]. The need for these systems has grown with the vast amount of data that has become available in recent years. Labelling enough data to effectively learn machine learning models is a challenging and expensive task, making it impractical for many researchers. However, as a field, PU learning has not received as much attention as it is arguably warranted, given its applicability. There are many possible reasons for the oversight, including a lack of guidelines in the literature, and a lack of widely applicable PU learning tools. Most PU learning algorithms are developed for specific application domains, such as [10-19]. Therefore, the field could benefit from general purpose PU learning tools that can be easily applied to any application domain. Regarding the lack of guidelines, before this work there was relatively little guidance regarding evaluation of PU learning models. This is a non-trivial issue, as discussed

in Section 2.5, therefore established practices are essential to ensure that proposed PU learning methods can be effectively compared to those that already exist.

This work looks to address these issues by, firstly, establishing guidelines for the evaluation of PU learning methods as published in [20], and, secondly, proposing novel Auto-ML systems specific to PU learning. The need for Auto-ML systems is discussed next.

1.2 Automated Machine Learning (Auto-ML)

Auto-ML is a growing area of machine learning that involves optimising a classification algorithm or pipeline for each specific input dataset [8][9]. The primary goals of Auto-ML are to increase the effectiveness of classification algorithms or pipelines for given learning tasks through optimisation of the components of the pipeline and their respective hyperparameter settings, as well as making machine learning more accessible for those without extensive domain knowledge [8][9].

There are several approaches to Auto-ML, such as evolutionary computation and Bayesian optimisation, both of which are addressed in this work (see Sections 2.2-2.4). By using such optimisation methods, the aim is to remove the need for a trial-and-error approach to algorithm or pipeline optimisation¸ which is a time-consuming and laborious task that may not find an optimal or a near-optimal solution. Furthermore, machine learning algorithms are complex and often have many hyperparameters, each of which impacting the output of the classification model. Conducting a thorough and informed search of the available algorithms and their respective hyperparameters traditionally requires expert-level knowledge of, not just machine learning, but the classifiers themselves. This presents a barrier that Auto-ML looks to remove.

Given the issues discussed in relation to the PU learning literature and the aims of Auto-ML, it follows that PU learning could benefit from Auto-ML systems specifically applied to the area. Auto-ML systems for binary classification are not good in this area as applying the standard binary classification paradigm to PU learning datasets is sub-optimal, as discussed in Section 2.5. Therefore, Auto-ML systems that construct algorithms specifically designed for PU learning are required.

1.3 Objectives

There are three primary objectives of this work. The first is to establish a framework for evaluation of PU learning algorithms. This objective is addressed in Section 2.5 and Chapter 3.

The second objective is to investigate the use of Auto-ML systems and compare their performance to baseline PU learning methods. This objective involves contributing a new proposed Auto-PU learning framework (an Auto-ML framework specifically for PU learning), described in Chapter 3, as well as three new Auto-PU systems that were developed with the aims of high predictive performance and computational efficiency in mind, as follows.

The first proposed Auto-ML system, GA-Auto-PU (described in Chapter 4), based on a Genetic Algorithm as the optimiser (see Section 2.2), was successful in outperforming some baseline PU learning methods with statistical significance, but its good predictive performance came at a large computational expense.

The second system developed, BO-Auto-PU (described in Chapter 5), addressed this issue through the use of Bayesian optimisation (see Section 2.3). However, whilst addressing the large runtime issue (i.e., it was much faster than GA-Auto-PU), BO-Auto-PU achieved overall a predictive performance somewhat lower than the performance achieved by GA-Auto-PU.

Finally, EBO-Auto-PU was developed (as described in Chapter 6), proposing a new hybrid optimisation approach between an evolutionary algorithm and Bayesian optimisation, and successfully acted as a trade-off between the two systems. That is, EBO-Auto-PU achieved overall somewhat better predictive performance than both GA-Auto-PU and BO-Auto-PU, whilst EBO-Auto-PU was also much faster than GA-Auto-PU.

The final objective is to identify guidelines for designing PU learning algorithms, in regard to recommending specific algorithmic components that should be used for specific learning scenarios, based on the experimental results reported in this thesis. This objective is addressed in Sections 4.4, 5.4 and 6.4 of Chapters 4, 5 and 6 (for each of the three aforementioned types of Auto-PU systems).

1.4 Contributions

This section lists the contributions of this work, as follows. Firstly, this work has proposed a PU learning evaluation framework. That is, through a literature review the primary evaluation metrics of PU learning (in terms of predictive accuracy) were identified and mapped to their appropriate PU learning goals. An evaluation methodology for utilising different types of datasets (both real-world and synthetic datasets) was identified, and datasets created specifically for PU learning evaluation were made publicly available for PU learning researchers[1]. This results of this contribution have been published in [20].

Second, this work has proposed an Auto-ML framework specific to PU learning. This framework will make it easier for other PU learning researchers to develop their own Auto-ML systems, with defined search spaces and an objective function to serve as a starting point. Parts of the proposed framework were published in [21][22].

Third, this work has proposed three new Auto-ML systems specific to PU learning, each of which a contribution in itself. The first system, GA-Auto-PU (Chapter 4), performed a global search in the defined space of PU learning algorithms using a Genetic Algorithm as the optimiser, which as mentioned earlier led to very long runtimes. The design of GA-Auto-PU and parts of its computational results reported in this thesis were published in [21][22]. The second system, BO-Auto-PU (Chapter 5), performed a much more computationally efficient (faster) search using Bayesian optimisation, enabling the use of the Auto-PU framework for researchers without access to high performance computing systems. Finally, EBO-Auto-PU (Chapter 6) was built based on a new hybrid approach combining aspects of evolutionary computation and Bayesian optimisation.

Out of these three systems, EBO-Auto-PU is the most novel contribution since it is based on a new hybrid evolutionary and Bayesian optimisation method. That is, EBO-Auto-PU can be deemed a novel contribution to both the area of PU learning and the area of Auto-ML. GA-Auto-PU and BO-Auto-PU are using standard GA and BO methods. Hence, although arguably they are not new contributions to the area of Auto-ML (since they use standard optimisers), they can still be deemed

[1] https://github.com/jds39/Unlabelled-Datasets/

novel contributions to the area of PU learning, since they were the first Auto-ML systems proposed specifically for PU learning.

All the three systems were shown in the experiments to achieve statistically significantly better predictive accuracy than some baseline PU learning methods. Overall, regarding predictive performance, EBO-Auto-PU had somewhat higher performance than GA-Auto-PU and BO-Auto-PU. In addition, EBO-Auto-PU was much faster than GA-Auto-PU and somewhat slower than BO-Auto-PU.

Finally, this work provides an analysis of the PU learning algorithm components most frequently selected by these systems, in order to provide guidance to researchers designing PU learning algorithms. Two major outcomes of this were a demonstration of the preference for simple linear classifiers used in the first step of the two-step procedure, and a preference against utilising the spy method.

1.5 Thesis Structure

Chapter 2 details the relevant background information needed to understand this thesis. This covers the fundamental concepts of classification (a type of supervised learning) and classifier evaluation, evolutionary algorithms, Bayesian optimisation, Auto-ML and positive-unlabelled learning.

Chapter 3 proposes a novel framework for Auto-ML applied to PU learning, called Auto-PU. This chapter details the search spaces used by the optimisation methods, and how the PU learning algorithms produced by the proposed Auto-PU systems are evaluated. The evaluation is conducted on two types of datasets, engineered PU datasets and synthetic datasets. The engineered datasets are created from standard binary datasets, the procedure for which is described in Section 3.3.

Chapter 4 details GA-Auto-PU [21][22], the first Auto-ML system for PU learning, utilising a Genetic Algorithm (GA) as the optimiser. This chapter details the GA procedure that the system follows and compares the system with a well-established binary classification Auto-ML system and two PU learning baseline methods, before discussing the PU learning algorithm components most frequently selected by the system.

Chapter 5 details BO-Auto-PU, a Bayesian optimisation-based Auto-ML for PU learning. This chapter details the BO procedure and compares the system against GA-Auto-PU and the two PU

learning baselines, before discussing the PU learning algorithm components most frequently selected by the system.

Chapter 6 details EBO-Auto-PU, utilising a new hybrid evolutionary/Bayesian optimisation method. This chapter details the hybrid optimiser as applied to the system, before comparing against GA-Auto-PU, BO-Auto-PU, and the two PU learning baselines, before discussing the PU learning algorithm components most frequently selected by the system.

Chapter 7 concludes this work, summarising the main research contributions and suggesting research directions for future work.

Appendix A reports the detailed results of precision and recall measures for each dataset for all the evaluated systems, since those results were presented only in summarised form across the main chapters with computational results (Chapters 4, 5 and 6).

Appendix B briefly describes and reports the results for another type of Auto-ML system for PU learning, which optimises the hyperparameters of a multi-layer perceptron (neural network) algorithm, among other hyperparameters of PU learning algorithms. Hence, this system can be considered a type of neural architecture search system. This system's brief description and its results are reported in this Appendix, rather than in the main body of the thesis, mainly because its predictive accuracy results were quite poor in general, clearly inferior to the other three Auto-ML systems proposed in this thesis (GA-Auto-ML, BO-Auto-ML and EBO-Auto-ML).

1.6 Publications Derived from this Research

This section provides the bibliographical details of the author's three papers that were peer-reviewed and accepted for publication throughout the course of this work.

Saunders, J.D. and Freitas, A.A., 2022. GA-auto-PU: a Genetic Algorithm-based Automated Machine Learning system for Positive-Unlabeled learning. In *Proceedings of the 2022 Genetic and Evolutionary Computation Conference Companion* (pp. 288-291). ACM Press, 2022. ISBN: 978-1-4503-9268-6/22/07. DOI: https://doi.org/10.1145/3520304.3528932.

This work introduced GA-Auto-PU, the first Auto-ML system for PU learning. As detailed in Chapter 4, GA-Auto-PU utilised a simple genetic algorithm as the optimiser and, in this work, it outperformed a state-of-the-art PU learning algorithm.

Saunders, J.D and Freitas A. A., 2022. Evaluating a new Genetic Algorithm for Automated Machine Learning in Positive-Unlabelled learning. In *Proceedings of the 15th International Conference on Artificial Evolution* (*EA 2022*). *Lecture Notes in Computer Science, Vol. 14091*, 42-57. Springer.

This work presented an extension of the previous work by proposing a second version of GA-Auto-PU, with an extended search space of candidate PU learning algorithms, and conducting a more in-depth analysis of the system, comparing it with two (rather than just one) baseline PU learning methods.

Saunders, J.D. and Freitas, A.A., 2022. Evaluating the Predictive Performance of Positive-Unlabelled Classifiers: a brief critical review and practical recommendations for improvement. *ACM SIGKDD Explorations Newsletter*, *24*(2), pp. 5-11.

In this work, a literature review was conducted assessing the current PU learning literature and the evaluation methods utilised by work proposing new PU learning algorithms. In this work, guidelines for evaluation of PU learning methods were established.

Chapter 2
Background

This chapter details the relevant background information necessary for this thesis. Section 2.1 outlines the fundamental concepts of supervised learning, detailing performance metrics and the classifiers used throughout this work. Section 2.2 and 2.3 detail evolutionary algorithms and Bayesian optimisation respectively, two optimisation methods used in this work. Section 2.4 outlines Automated Machine learning (Auto-ML). Section 2.5 explains Positive-Unlabelled (PU) learning.

2.1 Supervised Learning

2.1.1 Basic Concepts

Supervised learning is a fundamental area of machine learning that involves training a model to make predictions based on a set of labelled examples (*instances*) [1][2]. These instances, known as the training set, consist of pairs of input and output data, where the input data represents the characteristics of an instance (also referred to as *features* or *attributes*), and the output (labelled) data represents the desired predictions for those instances (also known as the *class variable* for classification tasks, or *output/target variable* for regression tasks). The goal of supervised learning is to learn a generalisable model that can make accurate predictions or decisions for new instances (not observed in the training set), based on the patterns learned from the training set.

This differs from unsupervised learning, a machine learning area that involves finding relationships among variables without distinguishing between input and output variables, i.e., from

data which is not labelled. A typical unsupervised learning task is clustering, where the goal is to divide the instances into clusters (groups) based on their similarities (i.e., maximising the similarities of instances within each cluster and minimising the similarities of instances between different clusters). The produced clusters can then be interpreted as "classes" for the subsequent application of a classification algorithm.

There are two main types of supervised learning tasks: classification and regression. In classification, the goal is to predict a categorical class label for each instance, such as whether an email is spam or not. In this example, the features of the instance could be characteristics of the text included in the body of the email, like a set of binary variables indicating whether or not a given word occurs in the text of the email. The class variable of this instance would be a binary variable, indicating "spam" (often represented as 1) or "not spam" (often represented as 0). In regression, the goal is to predict a continuous value for each instance, such as the price of a house. In this example, the features of the instance are the characteristics of the property, such as its location, square footage, and architectural style. The output variable of this instance would be a real-valued number, representing the price of the property.

Formally, the supervised learning task is defined as:

Given a training set T, consisting of pairs of a feature vector and a scalar label [$(\bar{x}_1, y_1)$, $(\bar{x}_2, y_2)$, …, $(\bar{x}_n, y_n)$], where y is related to $\bar{x}$ by way of a function f and n is the number of training instances, discover a function h that approximates f [23].

Training a supervised learning model generally involves splitting the data into *training* and *test* sets. The training set is a set of labelled instances used to train a predictive model. The learned model captures patterns and relationships between the features and the class labels. The test dataset is a separate set of labelled instances used to evaluate the performance of the trained model. The test set allows for an estimation of how well the learned model generalises to new, unseen data (i.e., not included in the training set) [1].

It is important to note that the training and test sets must be independent and non-overlapping. Independent meaning that the data for the sets are selected in such a way that the characteristics of the data in the training set do not influence the selection of the test set. Non-overlapping meaning

that no data point appears in both the training and the test set, the test set should contain completely unseen data. That is, the test set should not be used during the model training process. Maintaining separation gives a more realistic indication of predictive performance on unseen data.

For evaluating a predictive model, it is common to use a *k*-fold cross-validation procedure, where the data is divided into *k* folds of approximately equal size, and the model is trained on *k*-1 of the folds before being tested on the remaining fold. This process is repeated *k* times, with a different fold being used as the test set in each iteration. The predictive performance of the model is then averaged over all *k* iterations.

Cross-validation is advantageous over using of a single test set, providing a more robust estimate of model performance, by evaluating it on multiple subsets of the data which are used as test sets (separated from the data subsets used as training sets). This helps to avoid reporting an overly optimistic measure of predictive performance that may result from overfitting the training set, if the model was learned from the full data (as a training set) and evaluated on the same full data. It also provides an indication of the generalisability of the model, as the performance is evaluated over multiple test sets.

Overfitting is a common problem in machine learning that occurs when a model is too complex and becomes too specialised to the training data, making it unable to generalise to new, unseen data. This generally occurs as a result of over tuning a model to the input dataset [1].

2.1.2 Predictive Performance Evaluation

A key part of supervised learning is the evaluation of the learned predictive model. This is typically done by comparing the model's predictions on a test set to the true labels of those instances and measuring the extent to which those predictions match the true labels. Popular evaluation metrics include accuracy, precision, recall, and F-measure for classification tasks [24] and mean squared error (MSE) and mean absolute error (MAE) for regression tasks [25].

Before detailing the specifics of these metrics, the following definitions are required:

- True Positives (TP): The number of instances that truly belong to the positive class and are correctly predicted as belonging to the positive class.

- True Negatives (TN): The number of instances that truly belong to the negative class and are correctly predicted as belonging to the negative class.
- False Positives (FP): The number of instances that truly belong to the negative class but are falsely predicted as belonging to the positive class.
- False Negatives (FN): The number of instances that truly belong to the positive class but are falsely predicted as belonging to the negative class.
- y_i: The true (actual) value of the class/target variable in the ith instance of the dataset.
- $\hat{y}_i$: The predicted value of the class/target variable for the ith instance.
- $\bar{y}$: The arithmetic mean of the true values of the target variable over all instances of the dataset.
- n: The number of instances in a given dataset.

Note that TP, TN, FP, and FN are defined only for classification and $\bar{y}$ is defined only for regression tasks, whilst y_i and $\hat{y}_i$ are defined for both classification and regression tasks. It is also worth emphasising that, although these statistics can be calculated for the training and test sets, in order to measure generalisation performance what matters are the values of these statistics in the test set.

Accuracy

Accuracy is defined as the ratio of the number of correct predictions to the total number of predictions [24][26] (Equation 2.1).

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \qquad (2.1)$$

Accuracy is a popular metric due to its ease of interpretation as it provides a direct measure of the proportion of correct predictions made by the classifier, out of all predictions made [24]. Furthermore, it is a simple and intuitive measure that can be easily understood by a wide range of audiences and does not require the use of probability estimates or threshold settings, unlike performance metrics such as the Receiver Operating Characteristic curve metric [27].

However, accuracy also has a number of limitations when used as the sole measure of performance. One of the most significant, particularly for the datasets used in the experiments reported later in this thesis, is that it can be misleading when applied to datasets with imbalanced

class distributions. That is, data where there is not an equal proportion of instances belonging to each class. In such scenarios, a classifier can achieve a high accuracy by simply predicting all instances as belonging to the majority class, despite not providing any useful information about the minority class. This can result in a misleadingly high representation of predictive performance as the classifier may not actually be capable of identifying instances of the minority class [24][26]. This is particularly problematic for supervised learning tasks that involve anomaly detection, as the instances that users are interested in identifying as anomalies will constitute a very small minority class, i.e., only a very small proportion of the full set of instances.

Another important limitation of accuracy is that it does not account for false negatives (FN) and false positives (FP) separately [26]. As a result, it is incapable of capturing the trade-offs between these different types of errors. Obtaining values for these statistics separately is important for tuning a supervised learning model to a specific learning task where the cost of identifying a FN is substantially greater or smaller than the cost of identifying a FP. For example, in the case of medical diagnosis, a false negative may result in a missed diagnosis, which is potentially far more costly than a false positive, which may result in the patient undergoing some unnecessary treatment. In these cases, a high accuracy may not reflect the real-world predictive performance of the classifier.

In summary, accuracy is a widely used and easily interpretable measure of classification performance. However, accuracy has limitations when used on datasets with imbalanced classes and does not consider false positives and false negatives separately, limitations which are addressed by the following metrics.

Note that the metrics of Precision, Recall and F-measure, described next, are defined with respect to a given class of interest, out of all classes. Typically, the class of interest is the minority class, usually referred to as the "positive" class, whilst the other class(es) is(are) referred to as the "negative" class. Sometimes, however, these measures are calculated for each class separately and then its results are averaged over all the classes. Two common approaches for performing such average will be discussed later, after the description of these three measures considering only the minority (positive) class as the class of interest.

Precision

Precision is defined as the proportion of true positive predictions out of all positive predictions, calculated as shown in Equation 2.2 [24]. Precision is a useful metric for evaluating the predictive performance of a classification model in situations where it is more important to avoid false positives than to identify all actual positive instances. Unlike accuracy, precision considers only the positive predictions made by the classifier, making it more informative when working with imbalanced-class datasets where the minority class is of interest.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad (2.2)$$

One advantage of precision as a metric is that is provides a way to measure how exact a classifier is in its positive predictions. That is, it provides information regarding precisely how many of the instances predicted as positive are actually positive. It is a particularly useful measure when the class of interest is rare and false positives are highly undesired, such as gene function prediction. Conducting experiments to verify gene function are very time consuming and expensive. Therefore, any classifier looking to identify gene function should minimise false positives to provide a list of genes that are promising.

However, precision, when used in isolation, can be misleading, as classifiers that just identify a few positive instances correctly can have high precision even though they are not actually identifying the majority of the positive instances. Additionally, precision does not consider the false negatives, and therefore it does not give the full picture of the model's performance [26].

In summary, precision measures the proportion of true positive predictions out of all positive predictions. It is an especially useful metric in scenarios where false positive are highly undesirable, and it is informative in situations where there is a high cost of false positives. Due to the limitations identified, precision is not generally used a solitary evaluation metric. Generally, precision is reported in conjunction with recall.

Recall

Recall, also known as sensitivity or the true positive rate, is defined as the proportion of true positive predictions out of all actual positive instances, calculated as shown in Equation 2.3 [24]. Recall is a

useful metric for evaluating the performance of a classification model in situations where it is more important to identify all actual positive instances rather than avoiding false positives. Recall considers both the true positive predictions made by the classifier and the actual number of positive instances in the dataset, making it more informative when working with imbalanced-class datasets where the minority class is of interest.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad (2.3)$$

One advantage of recall as a metric is that it provides a way to evaluate how well a classifier is able to identify all of the positive instances within a dataset. That is, how many of the actual positive instances are identified by the classifier. This is particularly useful when the class of interest is rare and false negatives are highly undesired, such as in the medical diagnosis example given previously.

However, when used in isolation, recall can be misleading, as a model that simply predicts the positive class for all instances would achieve the maximum recall score (100%), despite not providing any useful information about the negative class and having a high number of false positives (low precision) [26].

In summary, recall measures the proportion of true positive predictions out of all actual positive instances. It is especially useful in scenarios where false negatives are highly undesired. However, it has limitations and should be used in conjunction with other metrics such as precision.

F-measure

F-measure, also known as F1-score, is defined as the harmonic mean of precision and recall, calculated as shown in Equation 2.4 [24]. F-measure is a useful metric for evaluating the performance of a classification model in situations where both precision and recall are important, as it provides a balance between them. F-measure is high when both precision and recall are high, and low when either precision or recall are low. As such, it does not have the same drawbacks described for using either precision or recall as a solitary metric.

$$\text{F-measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \qquad (2.4)$$

One advantage of F-measure is that it provides a way to balance the trade-off between precision and recall. In many cases, precision and recall can be in conflict with each other, where an increase in one metric leads to a decrease in the other. In these cases, using F-measure can give a better understanding of the model's performance than using precision and recall separately.

Additionally, F-measure is not impacted by class imbalance in the way described for accuracy, as a model that overwhelmingly predicts the negative majority class with a few minority class predictions can still be shown to be performing poorly by utilising the F-measure.

However, F-measure also has its limitations as precision and recall are combined into a single metric. As such, detailed performance of the classifier is not forthcoming and no description of the specific FP and FN errors is given. Therefore, it is often useful to report F-measure in conjunction with precision and recall in order to better understand the predictive performance of a model.

In summary, F-measure combines precision and recall into a single metric, providing a balance between them. It is particularly useful in scenarios where precision and recall are both important and when the dataset's class distribution is highly imbalanced. However, it has limitations and should often be reported in conjunction with other metrics, such as precision and recall.

Micro/macro averaging

The aforementioned metrics of precision, recall, and F-measure can be calculated with different degrees of granularity by using micro or macro averaging. Micro-average is a method of calculating precision, recall, and F-measure by considering the performance of the model over all the samples, regardless of the class. Micro-average is calculated by summing the TP, FP, and FN values for all the classes together, and then calculating precision, recall, and F-measure using these summed values. Note that the labels 'True' and 'False' in this case refer to the class under consideration. That is, to consider TP as an example, a TP instance is a True Positive in the context of a specific class. Thus, when calculating the metrics for the next class, TP will then refer to that new class of interest. As a more practical example, if the two classes are "A" and "B", first the class "A" is considered as the positive class, and then TP is the number of examples annotated with class "A" in the dataset which were correctly predicted by the classifier as class "A", and analogously for the calculation of FP and FN. Next, the class "B" is considered as the positive class, and then TP is the number of

examples annotated with class “B” in the dataset which were correctly predicted by the classifier as class “B”, and analogously for the calculation of FP and FN. Macro-average is calculated by calculating precision, recall, and F-measure for each class, and then taking the average of these values [28].

Micro-average is useful when the samples are imbalanced across classes and the classification of each instance is considered equally important. In this case, since micro-average considers the performance of the model over all instances regardless of the class, micro-average will in practice assign greater importance to the classification of the majority class (with most instances). However, it is important to note that in certain situations, such as when the performance on a minority class is of particular importance, other measures like macro-average may be more suitable. Macro-average is useful when it is necessary to consider the performance of the model separately for each class. This is the case when the classes are of equal importance and the performance of the model on each class should be considered equally. Macro-average calculates precision, recall, and F-measure for each class and then takes the average of these values, providing a representation of the model's performance on each class.

Mean squared error

Mean Squared Error (MSE) is a metric used for evaluating the performance of regression models. It measures the average squared difference between the predicted values and the true values of the target variable, calculated as shown in Equation 2.5 [29]. MSE gives an indication of how far the predictions are from the true values. A low MSE indicates that the model is making accurate predictions and a high MSE indicates that the model’s predictions are far from the true values. For the metrics discussed previously, the goal is to *maximise* the metrics in order to increase model performance. For metrics which describe the error of a system, like MSE, the goal is to *minimise* the metric.

$$\text{MSE} = \frac{\sum(y_i - \hat{y}_i)^2}{n} \qquad (2.5)$$

One advantage of MSE as a metric is that it is differentiable, that is, the rate of change can be calculated at any point, which makes it a useful metric for optimising model parameters, enabling

the use of optimisation methods such as gradient descent. Additionally, it penalises large errors more than small errors, so it is particularly useful in cases where it is important to minimise large predicted errors [29].

However, MSE has some limitations. Primarily, it is sensitive to outliers. That is, a single large error can significantly increase the overall value of the MSE. In addition, it does not provide information about the direction of the errors, i.e., whether they are overestimations or underestimations of the true value of the target variable.

In summary, MSE is a widely used metric for evaluating the performance of regression models; it measures the average squared difference between the predicted values and the true values. It is particularly useful in cases where it is important to minimise large prediction errors, or for tuning model performance in the training process. However, it is limited by sensitivity to outliers, meaning that a single large error can decrease this interpretation of model performance; and it does not provide any information about the direction of the errors.

Mean absolute error

Mean Absolute Error (MAE) is a metric used for evaluating the performance of regression models. It measures the average absolute difference between the predicted values and the true values, calculated as shown in Equation 2.6 [30]. It gives an indication of the average magnitude of prediction errors. A low MAE indicates that the model is making accurate predictions and a high MAE indicates that the model's predictions are far from the true values.

$$\text{MAE} = \frac{\sum |y_i - \hat{y}_i|}{n} \qquad (2.6)$$

One advantage of MAE as a metric is that it is robust to outliers, meaning that it is not affected by a small number of large errors, unlike MSE [30]. This can be particularly useful for data that contains outliers or extreme values that may not be representative of the majority of the data. Additionally, it also provides a more intuitive interpretation of the error since it reflects the magnitude of the errors.

However, MAE is not without limitations. Like MSE, MAE does not give any information about the direction of the errors. That is, there is no information regarding whether an error is the result of an overestimation or underestimation of the true value of the target variable.

In summary, MAE is a widely used metric for evaluating the performance of regression models. It measures the average absolute difference between the predicted values and the true values. It is robust to outliers, providing an intuitive interpretation of the errors, but does not provide any specific details about the direction of the errors.

2.1.3 Base Classification Algorithms

Throughout this work, we utilise 18 base classification algorithms in the proposed Auto-ML systems (to be described in detail in Chapters 4, 5, 6). These are standard binary classification algorithms, all implemented with Sci-Kit Learn [31], with the exception of the deep forest algorithm which was implemented with the deep-forest library[2]. These classification algorithms were selected simply because they are popular and easily accessible through Sci-Kit Learn. Deep forest was selected for use as it is used by one of the baseline methods. Furthermore, their implementations each allow for the prediction of both a discrete class variable (1 or 0 in our case) and the (continuous) probability of an instance belonging to the positive class. Throughout this work, these classification algorithms are implemented with the default value of their hyperparameters as given in the Sci-Kit Learn documentation[3]. The remainder of this subsection gives a brief overview of each of these 18 classification algorithms.

Gaussian naïve Bayes

Gaussian Naïve Bayes is an algorithm based on Bayes' Theorem that makes the strong assumption that features are independent from each other given the class variable [32]. It is particularly useful for classification tasks that involve continuous features which are normally distributed. This algorithm is called "naïve" as its aforementioned assumption is often not the case in real-world data.

Given a set of features $X = \{x_1, x_2, \dots, x_n\}$ and a target variable y, the Gaussian naïve Bayes algorithm estimates the probability of y given X as shown in Equation 2.7.

$$\mathrm{P}(y|X) = \frac{\mathrm{P}(y) \times \mathrm{P}(X|y)}{\mathrm{P}(X)} \qquad (2.7)$$

[2] https://pypi.org/project/deep-forest/
[3] https://scikit-learn.org/stable/user_guide.html

Where $P(y)$ is the prior probability of y, $P(X|y)$ is the likelihood of the features given the class, and $P(X)$ is the probability of the set of features X. $P(X|y)$ is estimated as the product of the probability density functions of each feature, assuming a Gaussian distribution of each feature. The class that maximises $P(y|X)$ is chosen as the predicted class.

The Gaussian naïve Bayes algorithm is simple and computationally efficient, a distinct advantage over more complex classification algorithms when working with large datasets. However, it is important to emphasise that the class-conditional independence assumption between features is often violated, which can potentially lead to learn an ineffective classification model. Though, despite the assumption often not holding, the naïve Bayes algorithm still performs well in practice [33]

Bernoulli naïve Bayes

The Bernoulli naïve Bayes algorithm, like the Gaussian naïve Bayes algorithm, is based on Bayes' Theorem, also assuming that the features are independent from each other given the class variable [32]. Unlike Gaussian naïve Bayes, Bernoulli naïve Bayes works best on datasets with binary features. The formula for determining a class is the same as for Gaussian naïve Bayes, shown in Equation 2.7. The difference lies in the estimation of $P(X|y)$.

$P(X|y)$ is estimated as the product of the Bernoulli probabilities of each feature. The Bernoulli probability of a feature conditioned on the class, $P(x_i|y)$ is the probability that feature x_i takes the value 1 given the class y. The class that maximises $P(y|X)$ is chosen as the predicted class.

The advantages and disadvantages are the same as those for Gaussian naïve Bayes.

Logistic regression

The logistic regression algorithm learns a type of generalised linear model that predicts the probability of an instance belonging to a specific class by calculating the function [33] shown in Equation 2.8.

$$P(y = 1|X) = \frac{1}{1 + e^{-z}} \qquad (2.8)$$

Where $P(y = 1|X)$ is the probability of instance X belonging to the positive class, and z is the linear combination of the input features and the model's parameters (the features coefficients, or weights).

Logistic regression is another fast and simple classification algorithm that is efficient on large datasets. However, Logistic Regression assumes linearity and, as such, is often unsuitable for complex datasets [32].

Linear discriminant analysis

Linear Discriminant Analysis (LDA) is a classification algorithm that generates a linear decision boundary by fitting class conditional densities to the data and using Bayes' rule [33]. That is, the classes are linearly separated in the feature space by calculating the probability densities that describe the probability of an instance belonging to a particular class. Bayes' rule (given in Equation 2.7) is then used to calculate the probability of new instances belonging to a given class.

LDA suffers the same advantages and disadvantages as the previous classification algorithms, it is a simple and efficient algorithm, but assumes linearity. Furthermore, LDA is sensitive to outliers, making it a potentially poor choice for complex datasets [33].

K-nearest neighbours

K-Nearest Neighbours (kNN) is a simple classification algorithm that is based on measuring the distances between instances in the feature space. In essence, *k* training instances are found which are closest to the current test instance, and that test instance is assigned the majority class (or the mean value for regression tasks) of those *k* nearest training instances [32]. The distance between instances can be calculated using a variety of metrics, but a common choice is the well-known Euclidean distance. However, whilst the kNN algorithm is conceptually simple, the resulting model can be highly non-linear, and thus effective for complex datasets.

The kNN algorithm is very simple but can be computationally expensive for large datasets with a large number of instances and features. However, it is generally robust to outliers in the data and is non-parametric, meaning that no assumptions are made about the distribution of the data [32].

Support vector machine

The Support Vector Machine (SVM) algorithm learns a decision boundary separating the classes in the feature space [34]. The decision boundary is chosen such that it maximizes the margin, the distance between the decision boundary and the closest training instances from each class (known as

the “support vectors”). SVMs can be used for both linear and non-linear classification, an advantage over some of the previously defined classifiers. For linearly separable data, the decision boundary is simply defined as the linear hyperplane separating the data. For non-linear data, the data is transformed into a higher dimension space, where a linear boundary can potentially be found.

One major advantage of SVMs is their ability to handle high dimensional spaces and their versatility, allowing for a wide range of kernel functions for the decision function. However, this does entail some complexity as a suitable kernel needs to be found for a given task [32].

Decision tree

A decision tree algorithm is a versatile type of classification algorithm which learns decision trees from the data [35]. The basic idea behind decision trees is to divide the feature space into smaller regions, *leaves*, that correspond to a particular value of the class variable. The process of dividing the feature space is done by successfully splitting the data on one feature at a time, based on a condition that maximises the separation of the different classes. The result is a tree-like structure, where each internal node represents a test on a feature, each branch represents the outcome of that test, and each leaf node represents a predicted class [32].

This process is based on a greedy search strategy that recursively selects the feature with the highest information gain [32]. The information gain is a measure of how much a feature helps to reduce the uncertainty of the class variable. Decision trees are usually pruned to prevent overfitting by removing branches that do not contribute much to the accuracy of the tree.

Given a set of features $X = \{x_1, x_2, \dots, x_n\}$ and a class variable y, a decision tree classifier estimates the probability of y given X by traversing the tree from the root to a leaf node. The path followed depends on the values of the features and the conditions specified at each internal node.

The primary advantage of decision trees is that they are often simple to understand and interpret, since they are graphical models. One can understand the decision making process simply by following the path taken through the tree. Furthermore, they are computationally efficient, making them an effective choice for large datasets in regard to speed. However, sometimes the decision trees learned from the data are too large to be interpreted by users (even after some tree pruning). Also,

they are sensitive to small variations in the data. A small variation can lead to a completely different decision tree, which is one of the reasons why they are sensitive to overfitting [35].

Random forest

The Random Forest algorithm produces an ensemble of decision trees, meaning that they utilise a large number of decision trees to perform their classification. The basic idea of a Random Forest algorithm is to combine (typically via voting) the predictions of multiple decision trees, each generated using a technique called bootstrap aggregating, or *bagging* [36]. Bagging consists of randomly sampling instances from the data, with replacement (a single instance can be sampled multiple times) and building a decision tree from those sampled instances [37].

As previously discussed, a single decision tree is prone to overfitting. By averaging the predictions of many trees, each built on a different sample of the data, random forests can reduce overfitting and improve the generalisation of the model.

Given a set of features $X = \{x_1, x_2, \dots, x_n\}$ and a target variable y, a random forest algorithm learns a large number of decision trees, each built with a randomly sampled subset of the data, and aggregates the predictions of the trees by assigning the class y as the majority vote, in the case of classification, or the average predicted value, in the case of regression.

Random forests are effective classifiers as they are less prone to overfitting than a single decision tree and can efficiently handle a large number of features and instances [37].

Extra tree

The extra tree algorithm, also referred to as the extremely randomised tree algorithm, learns a model similar to a decision tree in that a tree-like structure is built and traversed for calculating the class of an instance. The difference is that the extra tree algorithm does not use a greedy approach to calculate the split at a given node. Instead, a random value is used, hence “extremely randomised” [38].

This results in a tree that is far less predictable than a standard decision tree, and as a result generally less effective when used in isolation. As such, the extra tree classifier is generally used as the ensemble extra trees classifier, described next. However, the extra tree classifier has still be included in this work as an option for our systems for completeness.

Extra trees

The extra trees algorithm is a variation on the standard random forest, utilising an ensemble of extra trees, as opposed to standard decision trees [38]. The increased randomness can result in a learned model that is more robust to overfitting than a standard decision tree, but the model is generally less easily interpretable as the randomness of the splitting threshold can diminish the ease of inference regarding feature significance.

Bagging

Bagging, or bootstrap aggregating, is an ensemble learning technique used to improve the performance of machine learning models by training the base classification algorithm many times, each time on a different (randomly sampled) subset of the instances and aggregating their predictions [39]. The Sci-Kit Learn implementation used in this work uses a decision tree algorithm as the base algorithm. As such, when the base classification algorithm is a decision tree algorithm, the bagging technique is similar to the random forest algorithm. The primary difference is that the random forest algorithm uses a randomly sampled subset of the features for learning the decision trees, whilst all features are used with the bagging technique for learning the decision trees. As such, the trees learned by the bagging technique are less diverse (i.e., more similar) than the trees learned by the random forest algorithm. This is because bagging is trained using data subsets differing only by the training instances used, whilst random forest is trained using data subsets different by both the features and the training instances used. It is important to emphasise, though, that bagging is a generic ensemble method that can be used with any base classification algorithm, not just decision tree algorithms.

Bagging shares the advantage of the random forest algorithm regarding preventing overfitting.

AdaBoost

AdaBoost, or *adaptive boosting*, is an ensemble learning technique that iteratively trains weak models and combines them to create a final, stronger model [32]. A weak model is defined as a model that performs only slightly better than random guessing. The algorithm chosen as the base classification algorithm for producing weak models in the Sci-Kit Learn implementation is the decision stump algorithm (which learns a decision tree with a single internal node).

In each iteration, AdaBoost adjusted the weight of the misclassified instances, so that the next model pays more attention to the instances that were misclassified in the previous iteration. By iteratively adjusting the weights, AdaBoost helps the model focus on areas of the search space which are more difficult to interpret. After many iterations, the final model is a weighted combination of the models, where the weights are proportional to the accuracy of each model.

This implementation of AdaBoost shares the advantage of the random forest algorithm in terms of reducing overfitting but is slightly more prone to overfitting as it is sensitive to noise and outliers [33].

Gradient boosting

Gradient boosting is an ensemble learning technique that builds a model by iteratively building decision trees, with each tree used to correct the mistakes of the previous tree – like AdaBoost. This technique is called gradient boosting as it optimises a loss function using gradient descent [33] – unlike AdaBoost. The loss function calculates the rate of error of the classification models. In the Sci-Kit Learn implementation used in this work, the loss function is the log loss, also referred to as the cross-entropy loss, which compares the predicted probability with the true class label.

The algorithm starts by fitting a simple base model (decision tree) to the data and then iteratively adds new decision trees. In each iteration, the algorithm uses the gradient of the loss function with respect to the predictions of the current ensemble of trees to fit the next tree. By iteratively fitting new trees, the algorithm is able to improve the accuracy of the model by reducing the residual errors.

Gradient boosting can handle large datasets with high-dimensional feature spaces but requires a high computational cost and is sensitive to overfitting with too many iterations [33].

Histogram-based gradient boosting classification tree

The histogram-based gradient boosting classification tree (HGBoost) algorithm is an extension of the gradient boosting algorithm that improves the accuracy and scalability by using histograms to approximate the distributions of the feature values. Instead of using the traditional decision tree structure, the algorithm uses histograms to represent the feature values. This approach allows the algorithm to handle large datasets and high-dimensional feature spaces more efficiently by discretizing the feature values into bins and storing the bin frequencies, used by the histograms to

approximate the distribution of the features [40]. In each iteration, HGBoost fits a new histogram-based decision tree to the residuals of the previous iteration, using the histograms to select the best split points for each feature and using gradient descent to find the optimal histograms.

The primary advantage of HGBoost is that it is better able to handle large datasets and high-dimensional feature spaces than the gradient boosting algorithm. However, it has several parameters to tune and as such an ineffective classification model can be built with poor parameter selection.

Deep forest

The deep forest algorithm was proposed as an attempted improvement upon some drawbacks of deep neural networks (particularly their very large computational time). Rather than using a layered network of neuron-like nodes, deep forest uses a layered network of forests of decision trees. As such, it is described as "an ensemble of ensembles" [41]. Each forest outputs an estimated class probability vector, with each component of the vector representing the class probability estimated by the random forest for the corresponding class label. The class probability vectors of each forest in the layer are concatenated and used, together with the original features in the dataset, as predictive features for the random forests to be trained in the next layer [41].

The deep forest algorithm is much more recent and lesser known that several of the other algorithms included in the Auto-ML systems proposed in this thesis, and it does not have a Sci-Kit Learn implementation. However, it is the base classification algorithm for one of the baseline Positive-Unlabelled learning methods in this work (see Section 2.5), and as such it has been included. The deep forest algorithm has been implemented with the deep-forest python package[4].

Stochastic gradient descent

Gradient descent is an optimisation algorithm used to minimize a function by iteratively moving in the direction of the steepest descent as defined by the negative of the gradient. It involves adjusting the parameters step-by-step, based on the learning rate and the gradient of the loss function at the current position. This process is repeated until the algorithm converges to a minimum (ideally the global minimum but it can become trapped in local minima) of the function. The stochastic gradient

[4] https://pypi.org/project/deep-forest/

descent (SGD) algorithm is an extension of the basic gradient descent algorithm that updates the model's parameters using random subsets, or mini-batches, of the data. The Sci-Kit Learn implementation utilises SVM as the base classifier that is optimised using the SGD algorithm. Firstly, the SVM is initialised with random parameter values, which, in each iteration, are updated to minimise the loss function using gradient descent [32].

The SGD algorithm is relatively computationally efficient as it uses mini-batches of the data to update the parameters, meaning that it can handle large datasets [32]. However, there are many parameters for the algorithm, each of which can have a substantial impact on the output.

Multilayer perceptron

A multilayer perceptron (MLP) is a type of artificial neural network that consists of multiple layers of interconnected neurons, known as perceptrons [42]. The perceptron, originally proposed by Rosenblatt in 1958, is a linear classification device modelled on a neuron. Essentially, the perceptron assigns a weight to each feature value of an input instance and determines if the sum of the weighted feature values is greater than a given threshold. See Figure 2.1.

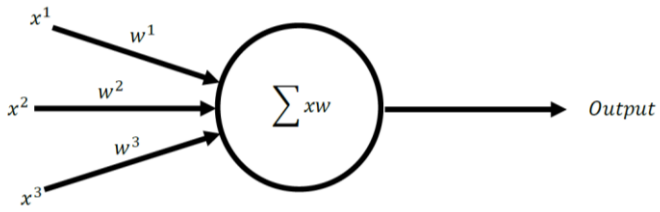


Figure 2.1. Rosenblatt's Perceptron. $\boldsymbol{x^n}$ is input $\boldsymbol{n}$, $\boldsymbol{w^n}$ is the weight applied to input $\boldsymbol{n}$, $\sum \boldsymbol{xw}$ is the sum of the weighted feature values, and $\boldsymbol{output}$ is the class assigned to the input instance.

The limitations of a single perceptron were highlighted by Minsky and Papert in their seminal 1988 book "Perceptron" [43]. However, many of these limitations are overcome when using a network of perceptrons, producing an MLP. The layers in the MLP are typically fully connected, meaning that each neuron in one layer is connected to every neuron in the next layer. The MLP consists of three different types of layers: an input layer, hidden layers, and an output layer. The input layer receives the input and will have a number of neurons equal to the number of features of the data. The hidden layers sit between the input and output layers and extract complex features from the input data. The output layer uses the processed data from the hidden layers to calculate an output class value.

MLPs are effective classifiers for complex datasets, able to learn both linear and non-linear relationships between inputs and outputs. However, they are computationally expensive (particularly when they have many hidden layers, which is the case in deep neural networks), and they can be prone to overfitting. They also have a number of hyperparameters that need to be tuned to achieve a high predictive accuracy. For a review of MLPs and their applications see [44].

Gaussian process

The Gaussian Process Classifier (GPC) uses a probabilistic model that defines a distribution over functions, which can be used to make predictions about the target variable given new input features. In GPC, it is assumed that the target variable is a random variable distributed according to a Gaussian distribution, with the mean and the variance of the distribution being determined by the input features. Predictions are made about the target variable given new input by finding the mean and variance of the distribution at the new input values [32].

GPC is generally accurate and robust, even for datasets with a limited number of instances. However, the classifier is computationally expensive as its run-time increases cubically with the number of instances [32].

2.2 Evolutionary Algorithms (EAs)

EAs are powerful methods for solving complex optimization and search problems across various domains [45]. Inspired by the principles of natural evolution, EAs simulate the processes of selection, variation, and adaptation to efficiently explore solution spaces. With their ability to handle high-dimensional, non-linear, and multi-modal problem landscapes, EAs have gained significant attention from researchers [46][47][48][49][50][51]. In recent years, EAs have been extensively applied in diverse fields such as engineering [49], computer science [47], biology [52], economics [53], and social sciences [54]. In addition, EAs are frequently applied to machine learning tasks [55][56][57][58]. The inherent flexibility and adaptability of EAs allow them to tackle problems that are challenging for traditional optimization techniques, making them particularly useful in scenarios where analytical models or problem-specific algorithms are not readily available or feasible [59].

The core idea behind EAs is the concept of population-based search. Unlike traditional optimization methods that focus on generating and evaluating a single solution at a time, EAs maintain a population of candidate solutions and iteratively improve them over iterations (generations, in EA terminology). By employing mechanisms inspired by natural selection, such as selection, crossover, and mutation, EAs can explore a vast search space in parallel, effectively searching for promising regions that tend towards optimal or near-optimal solutions, although this is dependent on computational budget and hyperparameter settings. To that end, [59] defined six primary components of EAs, namely:

- Individual representation
- Fitness function
- Population initialisation
- Variation operators
- Parent selection mechanism
- Survivor selection mechanism

The next subsections outline these components as well as termination criteria, primarily in reference to Genetic Algorithms (GAs) as GAs are the type of EAs that are the focus of this work.

One of the key advantages of EAs is their ability to handle complex and dynamic problem environments. EAs possess inherent mechanisms for robustness and adaptability, as they can continuously explore the solution space, responding to changes in the problem landscape. This makes EAs particularly suitable for problems that involve uncertainty, noisy or incomplete information, and time-varying conditions [59].

This background section aims to provide an overview of EAs, their fundamental concepts and principles. Sections 2.2.1 – 2.2.7 detail the primary components of EAs described above. Section 2.2.8 briefly discusses genetic programming, and 2.2.9 outlines practical considerations and challenges.

2.2.1 Individual Representation

In EAs, the individual representation defines how candidate solutions are encoded and therefore manipulated. Each element within the encoding is commonly referred to as a "gene". Different types

of individual representations have been developed to accommodate diverse problem domains and solution spaces. This section will explore several commonly used individual representations, including binary encoding, real-valued encoding, and categorical encoding, highlighting their characteristics and applications.

Binary encoding is one of the most widely employed individual representations in EAs. It utilizes binary genes, where a value of 1 represents the presence or truth of a certain attribute or feature, while a value of 0 denotes its absence or falsehood. This encoding scheme is particularly suitable for problems that involve binary decision variables or binary-encoded features. For example, in a GA for feature selection, each gene can represent the inclusion or exclusion of a specific feature from a set of features.

Real-valued encoding employs genes that are real numbers within a predefined range. This representation is suitable for optimization problems involving continuous variables, such as parameter tuning or function optimization. By allowing the genes to take on real values, the algorithm can search for optimal solutions in a continuous solution space. Real-valued encoding offers a more fine-grained representation, enabling the algorithm to explore and exploit the solution space with greater precision. Additionally, real-valued encoding facilitates the application of mathematical operators, such as arithmetic crossover and mutation, which can be used to generate offspring with intermediate values of features, as opposed to just binary feature values.

In certain problem domains, categorical encoding is utilized to represent individuals. With categorical encoding, each gene takes on a value from a predefined set of categorical values. This representation is commonly used when the problem involves discrete or categorical variables that are not naturally represented by binary or real-valued encodings. For instance, in a classification problem, each gene may correspond to a specific category or class label. Categorical encoding allows EAs to effectively explore and select from a discrete set of options, making it suitable for problems with a finite number of possibilities.

It is important to note that an individual need not be represented by a sole type of representation. That is, an individual can be composed of a combination of binary, real-valued, and categorical representations. This is the case for the GA that forms the basis of the GA-Auto-PU system proposed in Chapter 4. This hybrid representation approach enables the algorithm to effectively represent

various types of variables or features within the candidate solutions, accommodating the complexity and heterogeneity of the problem domain.

In summary, individual representation is a crucial aspect of EAs as it defines how candidate solutions are encoded. Binary encoding is commonly used for problems with binary decision variables or binary-encoded features. Real-valued encoding is suitable for optimization problems involving continuous variables, facilitating fine-grained exploration of the solution space. Categorical encoding is employed when the problem involves discrete or categorical variables. The hybrid approach used in GA-Auto-PU (Chapter 4) combines multiple encoding schemes to handle the complexity and diversity of the problem domain, ensuring the algorithm's effectiveness and versatility, as will be discussed in greater detail later in this work.

2.2.2 Fitness Function

In EAs, the fitness function plays a pivotal role in evaluating the quality of candidate solutions within the population. It measures how well an individual performs in solving the target problem. The fitness function serves as a guide for the evolutionary process, allowing an EA to distinguish between better and poorer solutions and driving the search towards more promising regions of the solution space.

The design of a fitness function is problem-dependent and requires careful consideration to ensure its effectiveness and relevance. The fitness function should capture the key performance criteria or objectives of the problem at hand. For instance, if the EA aims to optimize the hyperparameters of a predictive model, the fitness function might assess the model's performance using a specific metric, such as accuracy, F-measure, or mean squared error. By evaluating the candidate solutions based on their ability to achieve the desired outcome, the fitness function guides the algorithm towards finding solutions that exhibit desirable characteristics.

It is essential to note that the fitness function is often the most computationally intensive component of an EA. Evaluating the fitness of each individual in the population can be very time-consuming, especially for complex problems or when the evaluation requires resource-intensive computations, such as running simulations or training machine learning models. To mitigate the computational burden, it is crucial to evaluate fitness only when necessary. By minimising the number of fitness evaluations, the algorithm can allocate computational resources more efficiently.

Various techniques, such as surrogate modelling and adaptive fitness evaluation, can be employed to reduce the computational cost associated with fitness evaluations. These approaches aim to strike a balance between the accuracy of fitness assessment and the computational efficiency of the algorithm. See [60][51] for in-depth discussions of these methods.

In summary, the fitness function plays a critical role in EAs by evaluating the quality of candidate solutions based on their ability to solve the target problem or achieve the desired objectives. It guides the EA's search by distinguishing between better and poorer solutions. Due to its computational intensity, it is important to evaluate fitness only when necessary, employing techniques to reduce the number of fitness evaluations.

2.2.3 Population Initialisation

The population in an EA represents a set of individuals, where each individual represents a potential solution to the problem at hand. The most basic and commonly used population constraint is size. That is, the number of individuals is specified by the user, and the system sticks to this number each generation. However, implementations do exist with variable population size and additional selection criteria to compensate (for examples see [53][61]). Population initialization is the process of creating the initial set of individuals that will form the starting point for the evolutionary search. This section explores different approaches to population initialization, namely: random, deterministic, and heuristic initializations.

Random initialization is one of the most commonly used approaches in population initialization. In this method, individuals are generated randomly within the search space. Random initialization leads to high population diversity, as the individuals are distributed randomly across the solution space. This diversity is beneficial as it allows the algorithm to explore a wide range of potential solutions from the start. Random initialization is generally a good choice for most applications, particularly when the problem involves high-dimensional search spaces. However, it is important to note that random initialization does not guarantee the generation of high-quality solutions, as the initial population might contain individuals with poor fitness values.

Deterministic initialization, on the other hand, generates a predefined set of individuals according to a specific rule or pattern. This approach is typically less desirable in high-dimensional search

spaces, as it may result in a less diverse population, unless diversity is explicitly enforced, which can involve substantially more implementation time. However, deterministic initialization can be useful in certain scenarios where prior knowledge or specific constraints guide the generation of individuals.

Heuristic initialization utilizes a rule or principle to generate individuals. This approach aims to direct the search space towards potentially good solutions based on domain-specific knowledge or problem characteristics. Heuristic initialization can prevent computational "waste" by starting the search closer to promising regions of the solution space. However, one challenge of heuristic initialization is the risk of getting trapped in local optima, as the search may be biased towards a particular region and fail to explore other potentially better solutions. The selection of an appropriate heuristic for initialization is crucial, as it should strike a balance between computational efficiency and exploration capabilities.

Experiments conducted by Surry and Radcliffe [62] provided insights into the impact of different initialization methods. They compared the effects of heuristic and random initializations on the fitness improvement of populations. The results showed that while the average fitness of a population increased more with heuristic initialization, random initialization resulted in better fitness improvement, especially for the best individuals. These findings were supported by subsequent studies in various domains, where "chaotic" approaches were found to generally enhance the performance of EAs [63].

In summary, population initialization is a critical step in EAs as it determines the starting point for the search. Random initialization is commonly used due to its ability to generate diverse populations. Deterministic initialization may be suitable in specific scenarios with prior knowledge or constraints. Heuristic initialization can guide the search space towards potentially good solutions, but careful consideration is required to prevent getting trapped in local optima. The choice of initialization method should be based on the problem characteristics and desired trade-offs between exploration and exploitation. The findings from previous studies highlight the advantages of random initialization in terms of fitness improvement and the potential benefits of "chaotic" approaches.

2.2.4 Variation Operators

Variation operators are fundamental components of EAs that generate new candidate solutions from existing ones. The two main types of variation operators are crossover and mutation.

Crossover

Crossover is analogous to natural reproduction, where children inherit genetic material from their parents. In the context of EAs, parents are selected from the population based on their fitness, and children are generated by combining the genetic material of these parents. Each child inherits specific genes from each parent with a certain probability. Several mechanisms determine how genes are inherited, with single-point, multipoint, uniform, and arithmetic crossover being common approaches [64][65][66]. All types described below are in reference to two-parent crossover, where two parents are selected from a population and two children are generated from these two parents. Other multi-parent crossover mechanisms are possible, but two-parent crossover is by far the most commonly used.

In single-point crossover, the child inherits genes up to a specific point from one parent and genes after that point from the other parent. The crossover point is typically randomly determined. This process is illustrated in Figure 2.2. Single-point crossover is straightforward to implement but may result in a lack of diversity compared to other crossover mechanisms. One limitation of single-point crossover is its positional bias, meaning that adjacent genes are more likely to be swapped together.

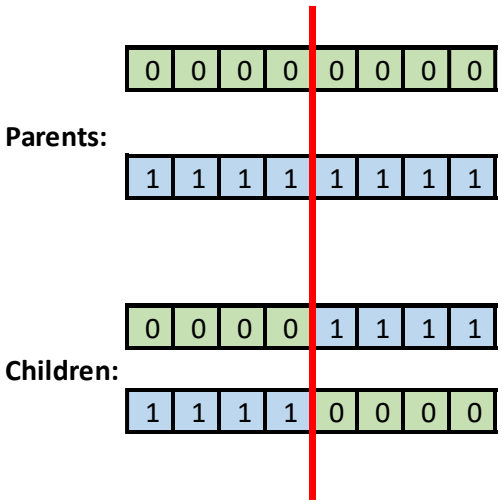


Figure 2.2. Example of single-point crossover. The red line indicates the crossover point.

Multipoint crossover is a variation of crossover commonly used in EAs to generate offspring from parent individuals. Unlike single-point crossover that involves a single crossover point, multipoint crossover involves multiple crossover points along the chromosome. The crossover points are randomly selected or predetermined, and segments between these points are exchanged between the parents to create the offspring. This process leads to a more diverse recombination of genetic material, as multiple segments from each parent contribute to the offspring's genetic makeup. Multipoint crossover can facilitate the exploration of different regions of the solution space and promote the combination of favourable traits from both parents. By allowing for more intricate combinations of genetic information, multipoint crossover enhances the algorithm's ability to search for promising solutions and adapt to complex problem landscapes. However, multipoint crossover is not without positional bias. An example of multipoint crossover is shown in Figure 2.3.

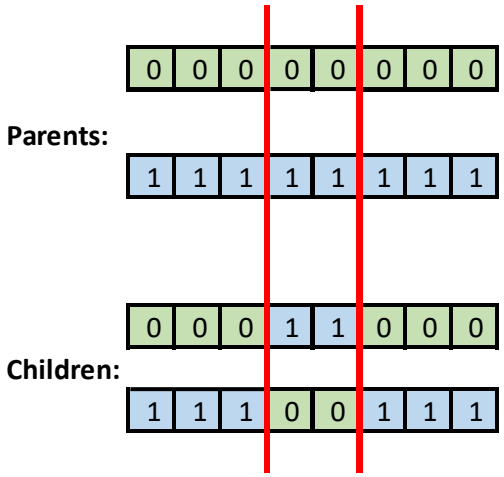


Figure 2.3. Example of multipoint crossover. The red lines indicate the crossover points.

Uniform crossover overcomes the positional bias by treating each gene individually. Instead of specifying specific crossover points, each gene has a probability of being inherited from either parent. Figure 2.4 provides an example of uniform crossover. This mechanism allows for greater variance in the offspring compared to single-point crossover or multipoint crossover and is not victim to positional bias. However, it may not be suitable for all encoding schemes, especially when certain genes need to be grouped together based on the decoding procedure of a candidate solution.

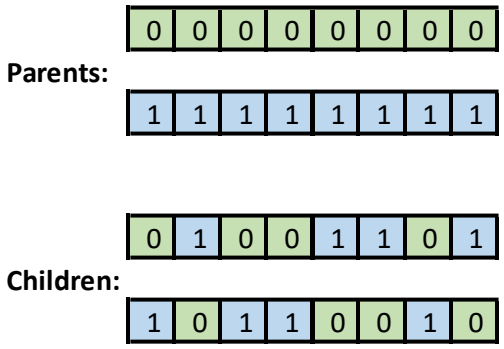


Figure 2.4. Example of uniform crossover.

For individuals utilising real-valued encoding, arithmetic crossover is an option. In arithmetic crossover, the genetic material of two parents is combined by performing a weighted average of their corresponding gene values. Each gene in the offspring is computed as a linear combination of the corresponding genes from the parents, where the weights are determined by a crossover parameter. The crossover parameter controls the degree of influence each parent has on the gene values of the offspring. This parameter should be set depending on the relative importance of the parents, which is scenario dependent. For example, if one wishes for each parent to have the same influence on the genes of the offspring, the crossover parameter should assign equal weighting to each parent. However, if one wishes for parent 1 to have a greater influence on the genes of the offspring, the crossover parameter should assign as greater weighting to parent 1.

Mutation

Mutation is a critical operator in EAs that introduces additional variation into the population by modifying individual genes. The purpose of mutation is to explore new regions of the solution space and prevent the algorithm from converging prematurely to suboptimal solutions.

For binary genes, the mutation process involves swapping the value of the gene for its opposite. If a gene has a value of 1, it is mutated to 0, and vice versa. This simple operation introduces variability into the population and allows for the exploration of different binary configurations.

In the case of real-valued genes, mutation typically involves adding or subtracting a predetermined or randomly generated value to or from the gene's current value. This value, often referred to as the mutation step, is typically small to ensure that the mutation introduces only minor

changes. By perturbing the gene's value, mutation allows the algorithm to explore the neighbourhood of the current solution and potentially discover better solutions.

Mutation of categorical values can be more complex, particularly when there is a specified order among the candidate values. In such cases, the mutation process may involve setting the gene to the previous or next candidate value from its current assignment. This preserves the ordering and allows for controlled exploration within the categorical space. Alternatively, if no specific order is defined, a random candidate value can be selected for mutation.

The probability of mutation determines how often the mutation operator is applied. Typically, the mutation probability is set to a low value, such as ranging from 0 to 0.1. This ensures that only a small proportion of genes undergo mutation in each generation. The low mutation probability strikes a balance between introducing variability into the population and maintaining the genetic information that contributes to good solutions.

Figure 2.5 provides an illustrative example of mutation, showcasing how the mutation operator alters the gene values in a population. This example mutation shows how to make slight variations to the gene values.

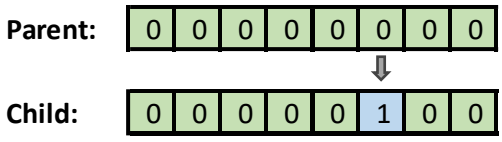


Figure 2.5. Example of mutation.

By incorporating mutation into the evolutionary process, an EA can prevent stagnation, escape local optima, and explore diverse areas of the solution space. Mutation complements the crossover operator by introducing additional genetic diversity and facilitating the exploration of the search landscape. Together, these variation operators contribute to the algorithm's ability to adapt and improve over generations.

2.2.5 Parent Selection Mechanism

In EAs, selecting appropriate parents for crossover and/or mutation is crucial to ensure the overall improvement of the population's fitness over generations. Two commonly used parent selection mechanisms are tournament selection and roulette wheel selection, which are summarised next.

Tournament selection

Tournament selection is a widely employed parent selection mechanism in EAs. This method involves randomly sampling a subset of individuals, known as the tournament subset, from the population. The size of the tournament subset is typically much smaller than the total population size. The fitness values of the individuals within the subset are then compared, and the individual with the highest fitness is chosen as a parent for crossover and/or mutation. This process is repeated until the desired number of parents is obtained.

Tournament selection strikes a balance between promoting variation and favouring individuals with higher fitness. By randomly sampling individuals for each tournament, all individuals in the population have a chance to participate in the selection process, ensuring diversity. At the same time, the selection of the fittest individual within the subset increases the probability of selecting individuals with superior fitness. This favours the propagation of favourable traits and improves the overall fitness of the population over time.

The size of the tournament subset is an important parameter that needs to be carefully determined. It represents a trade-off between variation and short-term population fitness. A larger tournament size includes more individuals in each competition, increasing the likelihood of certain individuals being selected multiple times. This can lead to higher population fitness for a short term, due to the concentration of individuals with superior fitness. However, a larger tournament size also poses a risk of premature convergence, where the population may converge to a suboptimal solution due to a lack of diversity. On the other hand, a smaller tournament size reduces the competition among individuals, resulting in a lower chance for those with high fitness to be selected. This can temporarily decrease population fitness. However, a smaller tournament size can also benefit long-term diversity, as it allows a wider range of individuals to have opportunities for crossover.

A commonly used tournament size is 2 individuals. However, the optimal tournament size depends on the specific problem and population characteristics. It is recommended to consider the size of the entire population when determining the tournament size, as a larger population may require a larger tournament size to ensure sufficient competition.

Roulette wheel selection

Roulette wheel selection assigns the probability of an individual being selected directly proportional to its fitness value. This selection mechanism, resembling a roulette wheel, ensures that all individuals have a chance of being selected, but those with higher fitness have a higher probability of being selected. Roulette wheel selection promotes population diversity while still favouring individuals with better fitness.

In roulette wheel selection, each individual is assigned a segment on the roulette wheel proportional to its fitness value. The size of the segment corresponds to the individual's probability of being selected. The hypothetical roulette wheel is then spun, and the selection process proceeds by landing on one of the segments, determining the selected individual. In practice, the individuals are generally assigned a value between 0 to 1, with the value of all individuals summing to equal 1. The size of the value assigned to the individual is directly proportional to their fitness, with a higher range between the value of the previous individual and the value of the current individual if the current individual has a high fitness. A random number is then generated, and if it falls within the range for a given individual, that individual is selected. For example, individual 1 may be assigned the value of 0.1, meaning that if the randomly generated number is between 0 and 0.1, individual 1 will be selected. Individual 2 may be assigned the value of 0.15, meaning that if the randomly generated value is between 0.1 and 0.15, individual 2 is selected. It can thus be inferred that individual 1 has a higher fitness than individual 2, as the range for individual 1 is 0 to 0.1, whilst the range for individual 2 is half the size at 0.1 to 0.15 (thus a range of 0.05). Figure 2.6 provides an example illustrating the concept of roulette wheel selection.

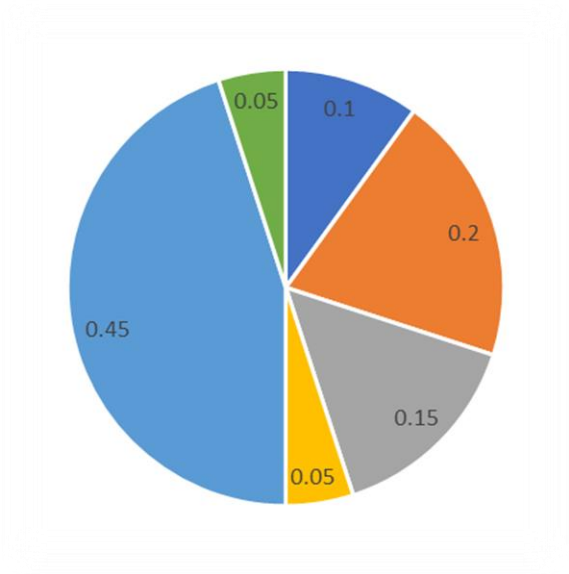


Figure 2.6. Example of roulette wheel selection. Fitness values have been scaled to sum to 1 and are represented by the slices of the pie chart.

Both tournament selection and roulette wheel selection offer advantages in terms of allowing all individuals to be considered for selection while still favouring individuals with higher fitness. The choice between these mechanisms depends on the specific problem, population characteristics, and the desired balance between variation and population fitness improvement.

In summary, parent selection mechanisms play a crucial role in EAs by determining which individuals are chosen for crossover. Tournament selection and roulette wheel selection are widely used methods. Tournament selection ensures a balance between variation and short-term population fitness, while roulette wheel selection allows for diversity while favouring individuals with higher fitness. The selection mechanism should be carefully chosen based on the problem requirements and the desired trade-off between exploration and exploitation.

2.2.6 Survivor Selection Mechanism

The survivor selection mechanism plays a crucial role in determining which individuals from the current generation will be included in the next generation of the population. While the majority of the next generation typically consists of offspring generated through variation operators, survivor selection mechanisms allow for the survival of some individuals from the current generation. This mechanism ensures that the genetic material of high-performing individuals is preserved and carried forward in the evolutionary process.

One commonly used survivor selection mechanism is elitism, which is based on the fitness of the individuals. In elitism, the individual or individuals with the highest fitness in the current generation are directly carried over to the next generation without any changes. This strategy guarantees that the highest fitness observed in the population will not decrease over generations. By preserving the best individuals, elitism ensures that valuable genetic material is maintained, potentially leading to the preservation and propagation of desirable traits. Elitism serves as a form of exploitation, as it focuses on preserving the best solutions found so far and prevents their loss due to selection pressure.

Alternatively, survivor selection mechanisms can be based on the age of the individuals. In such strategies, younger (i.e., more recently created) individuals are given a higher chance of survival into the next generation. This approach recognizes that younger individuals may not have had enough opportunities to contribute their genetic material to the population and allows them to persist for a longer period, increasing their potential impact on the evolutionary process. However, these strategies often require a variable population size or a low crossover rate to accommodate the inclusion of younger individuals without overwhelming the population.

The choice of survivor selection mechanism depends on the problem requirements and the specific goals of the EA. Elitism is a popular choice when the aim is to maintain the best solutions over generations and promote convergence towards optimal or near-optimal solutions. Age-based strategies can be advantageous when there is a need to ensure sufficient exploration and diversity in the population, particularly in dynamic or changing environments. The appropriate survivor selection mechanism should be carefully selected to strike a balance between preserving high-performing individuals and promoting the exploration of new solutions.

In summary, the survivor selection mechanism determines which individuals from the current generation are allowed to survive and be included in the next generation. Elitism, based on fitness, preserves the best individuals, preventing a decrease in the highest fitness observed. Age-based strategies prioritize younger individuals, providing them with a higher chance of survival. The choice of survivor selection mechanism depends on the desired goals of the algorithm and should be tailored to strike a balance between exploitation and exploration.

2.2.7 Termination Criteria

While not explicitly listed as a key component of an EA by Eiben and Smith [59], the specification of termination criteria is essential for defining when the algorithm should stop. Jain et al. [67] categorize termination criteria into three types: direct termination criteria, derived termination criteria, and cluster-based termination criteria.

Direct termination criteria are predefined. Common examples include a maximum generation count (most commonly used), a maximum run time threshold, and (less commonly) a specific fitness value. The primary issue with the first two criteria is that their specification is likely arbitrary with regard to algorithm performance. Whilst a maximum run time threshold may be necessary in terms of computational resources, there is no guarantee that the EA will reach a desired solution within that timeframe. This also applies for a maximum generation count. Specifying a specific fitness value to be reached has a separate but possibly greater issue: there is no guarantee that the system will reach this particular value. Thus, the user could end up with a system that would run indefinitely if left undisturbed.

Derived termination criteria are calculated based on data regarding the current generation. Examples of derived termination criteria are: (a) the best-vs-average (referred to as the "running mean" by [67]), where the difference between the best fitness and the average fitness is calculated and execution is terminated if the difference is less than a specified threshold; (b) the standard deviation, where the execution is terminated if the standard deviation of the population fitness is less than a specified threshold; and (c) the best-vs-worst criteria, where the execution is terminated if the difference between the best fitness and the worst is less than a specified threshold. Other derived termination criteria exist (see [68] for a more extensive list), but the vast majority suffer from the same drawback: an arbitrary threshold value. Whilst criteria such as the best-vs-average may appear a less ad-hoc stopping criterion than a direct termination criterion, a value for which we consider the best fitness and the population fitness mean similar enough to justify terminating execution must be defined, a value which is also likely arbitrary. However, a derived termination criteria is potentially preferable as a more adaptive stopping criterion. Termination will be based on a reasonably justifiable quantity, rather than computational resource.

Cluster-based termination criteria employ clustering techniques to analyse the current fitness diversity. An example of such a method is ClusTerm [67] where clusters of high-fitness individuals (termed “elitist clusters”) are identified, and execution is terminated when the aggregated size of the elitist clusters reach a predetermined threshold. As this relies on a predetermined threshold, the same drawbacks described regarding derived termination criteria apply. In fact, as cluster-based techniques determine stopping criteria based on data regarding the current generation and very few examples of this type of stopping criteria exist, they can be considered a subset of derived termination criteria.

In summary, termination criteria are crucial for determining when an EA should stop. Direct termination criteria, derived termination criteria, and cluster-based termination criteria offer different approaches to defining stopping conditions. However, most of these criteria rely on arbitrary threshold values, which can make termination decisions subjective and lacking a solid justification. While derived termination criteria and cluster-based termination criteria offer more adaptive stopping criteria, the choice of threshold values remains a challenge. Determining appropriate termination criteria should consider the problem at hand and strike a balance between computational resources, diversity, and the desired quality of the obtained solutions.

2.2.8 Genetic Programming (GP)

Discussion on EAs thus far has focussed on GAs with linearly encoded individuals. However, GP is an important topic to introduce before discussing Auto-ML. The primary difference between GP and previously discussed EAs is that GP methods evolve programs. That is, a GP method will evolve computer programs as a solution to a given task [69]. Despite the conceptual difference, GP follows a similar general procedure as GAs, including population initialization, fitness assessment, reproduction through variation operators, and survivor selection. However, the common approaches to individual representation in GP differ significantly from linear encoding employed in GAs. In GP, individuals are typically represented using a tree structure, making it tree-based GP (TGP) [70]. The tree structure represents the program's structure, with the tree nodes corresponding to functions or operators (e.g., +, -, ÷, ×), and the leaf nodes representing variables or constants [71]. The tree nodes serve as the building blocks for creating more complex programs, and the leaf nodes provide the necessary inputs or values for the program's execution. This tree-based representation allows for the

expression of complex program structures and facilitates the exploration of a wide range of program architectures. An extensive list of TGP systems across a variety of domains can be found in Table 2 of [72].

Another approach is grammar-based GP (GGP). GGP systems replace the set of terminals and functions by a grammar [73]. There are two types of GGP individual encoding, namely solution-encoding and production-rule-sequence-encoding. In solution-encoding, an individual is represented by a tree. In production-rule-sequence-encoding, the individual is mapped from its original encoding to the desired output. Often, this is mapping from a linear encoding according to the grammar. Recent examples of GGP systems are [74][75][76].

In addition to TGP and GGP, several other GP approaches exist, each with its own individual encoding and manipulation techniques. Stack-based GP, cartesian GP, and linear GP are among the notable variations. These different approaches provide flexibility in encoding programs and manipulating their structures, enabling a diverse range of applications. For a more extensive introduction to different GP frameworks, see [72].

In summary, GP is a powerful paradigm that allows for the evolution of programs as solutions to complex problems. Unlike linearly encoded GAs, GP employs a tree-based or grammar-based representation of individuals. This representation enables the evolution of program structures and facilitates the exploration of complex solution spaces. TGP and GGP are popular branches of GP, with TGP utilizing tree structures and GGP employing grammars. The variety of GP approaches provides a versatile framework for solving a wide range of problems that require the evolution of program-like structures.

2.2.9 Practical Considerations and Challenges

While EAs offer powerful problem-solving capabilities, there are several practical considerations and challenges that researchers and practitioners need to address when applying EAs in practice. These considerations can significantly impact the success and efficiency of the algorithm, and understanding these challenges is crucial for achieving optimal results.

Computational complexity

Computational complexity is a crucial practical consideration when implementing and applying EAs. EAs are computationally expensive, since they need to evaluate the fitness of a large population of individuals for many generations, and fitness evaluation is usually a very expensive operation – particularly in machine learning, where fitness often involves running a supervised learning algorithm. As the population size, number of generations and problem complexity (affecting the fitness evaluation time) increase, the computational cost of these operations an EA can grow exponentially, posing challenges in terms of time and resource requirements.

Efficient implementation plays a significant role in managing computational complexity. Researchers often strive to optimize the algorithm's performance by leveraging efficient data structures and algorithms. For example, when evaluating solutions against large datasets, employing algorithmic optimizations, such as caching or approximation techniques, can significantly reduce the computational burden and improve the efficiency of fitness evaluations [60].

Parallelization is another strategy to address computational complexity. EAs can benefit from parallel and distributed computing techniques to accelerate the execution time by performing multiple fitness evaluations or variation operations simultaneously. Parallelization can be achieved through multi-core processors, cluster computing, or even utilizing cloud computing resources. Distributing the computational load across multiple processing units allows for more efficient exploration of the search space and can help tackle larger and more complex optimization problems within a reasonable time frame [60].

Furthermore, efficient utilization of distributed computing resources can enable researchers to explore multiple regions of the search space concurrently, enhancing the diversity of the population and potentially improving the quality of solutions. Load balancing techniques and task scheduling strategies become crucial in distributed EAs to ensure optimal resource utilization and minimize communication overhead.

However, it is important to note that parallelization introduces additional challenges, such as synchronization, load balancing, and scalability. Researchers need to carefully design parallel algorithms and address potential bottlenecks to fully exploit the computational power of parallel and distributed computing systems.

Additionally, advancements in hardware technologies, such as graphics processing units (GPUs) and field-programmable gate arrays (FPGAs), provide opportunities to accelerate the execution of EAs further. These specialized hardware platforms can be leveraged to perform parallel computations and accelerate fitness evaluations and variation operators. However, harnessing the potential of these hardware platforms requires specific expertise and tailored algorithm implementations.

Parameter tuning

Parameter tuning is a critical aspect of effectively applying EAs. EAs involve several parameters that directly influence the algorithm's behaviour and performance, such as population size, mutation probability, crossover probability, selection mechanisms, and termination criteria. Selecting appropriate parameter values is a challenging task because different parameter settings can lead to vastly different optimization outcomes [77].

One of the key goals in parameter tuning is to strike a balance between exploration and exploitation. Exploration aims to search a wide range of the solution space to discover diverse and potentially superior solutions, while exploitation focuses on refining promising solutions to converge towards the optimal or near-optimal regions. Finding the optimal balance is problem-specific and often requires a deep understanding of the problem domain.

Empirical analysis is a common approach to parameter tuning. Researchers perform multiple runs of the EA with different parameter settings and analyse the impact on the algorithm's performance metrics, such as convergence speed, solution quality, and diversity. By systematically varying one or more parameters while keeping others fixed, researchers can gain insights into how changes in parameter values affect the optimization process. This analysis helps identify parameter configurations that yield desirable outcomes and informs the selection of appropriate values.

Sensitivity studies are another valuable technique for parameter tuning. Sensitivity analysis involves systematically varying one parameter while keeping others constant and observing the corresponding changes in the algorithm's performance. This analysis helps determine which parameters have a significant impact on the optimization process and which have a minor effect. Sensitivity studies guide researchers in prioritizing parameters that require more attention during the tuning process [78].

It is also possible to use adaptive techniques that utilize feedback mechanisms or statistical analyses to dynamically adjust parameter values based on the observed behaviour of the algorithm during runtime. Adaptive parameter control methods enable EAs to adapt to the problem's characteristics and the changing landscape of the solution space, enhancing the algorithm's robustness and convergence performance [77]. Self-adaptive frameworks allow the algorithm to autonomously adjust its internal parameters based on the feedback from the optimization process. Self-adaptive EAs use evolutionary principles to evolve the parameter values themselves, by encoding candidate parameter values into an individual together with the encoding of a candidate solution and letting both the parameter values and the candidate solution components be optimised by the evolutionary process [79].

It is worth noting that parameter tuning is an iterative process that often requires several iterations of experimentation, analysis, and refinement. Thus, this will increase the difficulties around computational expense discussed previously. Researchers should consider the problem's characteristics, domain knowledge, and insights gained from previous studies to guide the parameter tuning process effectively.

Premature convergence

Convergence is a fundamental goal in EAs, as they aim to reach optimal or near-optimal solutions. However, a common challenge in EAs is premature convergence, where the algorithm gets trapped in suboptimal regions of the search space and fails to explore potentially better solutions. Premature convergence hinders the algorithm's ability to fully exploit the search space and can lead to suboptimal or unsatisfactory results. The issue of premature convergence is closely tied with the issue of hyperparameter tuning, in the sense that successful hyperparameter tuning can mitigate premature convergence, as discussed next.

To address the issue of premature convergence, a delicate balance between exploitation and exploration is essential. Exploitation involves intensifying the search around promising solutions, refining them to achieve higher quality solutions. On the other hand, exploration focuses on expanding the search to discover new regions of the search space that may contain better solutions.

Striking the right balance between exploitation and exploration allows EAs to escape local optima and explore the search space more effectively.

One strategy to mitigate premature convergence is to employ diverse selection mechanisms. Traditional selection methods, such as roulette wheel selection, tend to favour individuals with higher fitness values. While this bias towards selecting fitter individuals helps exploit good solutions, it can also lead to a loss of diversity within the population. Employing selection mechanisms that explicitly promote diversity, such as rank-based selection or tournament selection with a small tournament size, can help maintain a diverse population and prevent premature convergence. These mechanisms ensure that individuals with lower fitness values have a chance to contribute to the next generation, preserving genetic diversity and exploration potential [79].

Introducing genetic diversity through mutation is another crucial approach to combat premature convergence. Mutation operators introduce random changes to individuals' genetic material, promoting exploration by generating novel solutions. By using a low mutation probability, the algorithm explores the search space beyond the immediate neighbourhood of the current solutions. Mutation enables the algorithm to escape local optima and encourages the discovery of new regions with potentially better solutions. However, it is important to strike a balance with the mutation probability, as an excessively high mutation probability may hinder convergence and impede the exploitation of promising solutions [80].

Interpretability and transparency

Interpretability and transparency are crucial considerations when applying EAs in certain domains where understanding the evolved solutions is important. While EAs are powerful tools for discovering complex solutions, these solutions can often be difficult to interpret or explain, posing challenges in domains where interpretability is a requirement. Achieving transparency and interpretability in the evolved solutions is a complex task, particularly when using highly flexible representation schemes.

One of the challenges in ensuring interpretability and transparency is the trade-off between complexity and performance. EAs can generate solutions that are highly complex and exhibit intricate interactions among variables or components. While these complex solutions may achieve

high performance on the optimization task, they can be challenging to interpret and understand, especially for domain experts who need to make informed decisions based on the solutions. In this case, a balance must be struck between complex solutions that achieve high performance and solutions that are comprehensible and interpretable to domain experts.

To address this challenge, researchers can adopt specific strategies during the optimization process. One approach is to utilize representations that naturally lend themselves to interpretability. For example, in the area of automated machine learning, an EA can be designed such that it favours machine learning pipelines utilising interpretable classifiers, such as decision trees, over black box classifiers such as neural networks.

Furthermore, post-processing techniques can be employed to enhance the interpretability of evolved solutions. These techniques involve extracting useful insights, visualizing the solutions, or summarizing their behaviour in a way that is more understandable to domain experts. For example, sensitivity analysis can help identify the most influential variables or components in the evolved solutions. Visualization techniques, such as heat maps, decision trees, or rule-based representations, can provide a visual understanding of the returned solution's behaviour and decision-making process.

Addressing these practical considerations and challenges requires a combination of domain expertise, algorithmic knowledge, and careful experimentation. Researchers and practitioners need to understand the nuances of the problem at hand, select appropriate algorithmic techniques, and continuously refine and adapt the algorithm to achieve the best possible results in practical applications. By addressing these challenges, EAs can be effectively applied to a wide range of real-world problems, offering valuable insights and solutions that would otherwise be difficult to obtain using traditional optimization methods.

2.3 Bayesian Optimisation (BO)

BO is a powerful meta-learning technique that involves learning a surrogate function to optimise an objective function [81][82]. That is, we optimise an objective function by calculating an estimation, where in general the estimation can be computed much faster than the evaluation of the objective function. The core idea behind BO is to iteratively select the next set of input parameters to evaluate

based on the previous evaluations. These previous evaluations are used to train a surrogate model that is then used to predict the score that would be assigned by the objective function, and thus identifies areas of the search space to be evaluated. By calculating (relatively fast) an estimation, rather than assessing all input parameters according to the computationally expensive objective function, BO efficiently identifies promising areas of the search space. This characteristic makes BO particularly well-suited for problems with limited resources or in situations where the cost of experimentation is high.

Specifically, a set of instances are evaluated according to an objective function. The objective function will depend on the specific application domain but should reflect the purpose that the instances have been created for. Recall the concept of a fitness function in the context of an EA, the objective function is analogous. For example, in the context of machine learning, the input could be a decision tree algorithm's configurations. That is, the hyperparameters that the candidate decision tree algorithm configuration should utilise. The objective score in this scenario could be the accuracy achieved by the decision tree learned by the algorithm on a specific dataset. A model, generally a Gaussian process regressor [83][84] or random forest [85], referred to as the surrogate model, is then trained using the characteristics of the input as features, and a performance metric obtained from the evaluation by the objective function as the target. To continue the decision tree example, the input to the surrogate model may have the maximum depth as one feature, the minimum samples required to split an internal node as another, etc., and the accuracy achieved by the learned decision tree as the target. This model is then used to estimate the performance of newly generated algorithm configurations, assigning a "surrogate score" to each. This process of performance estimation is generally much faster than calculating the performance of a configuration using the objective function when applied to an optimisation task that involves an expensive objective function.

These new configurations are generated according to the defined search strategy, a process that guides exploration of the search space (consisting of all candidate algorithm configurations) by determining how new candidate solutions are generated. A very naive search strategy would generate individuals randomly, playing no role in "guiding" exploration, whereas a more intelligent approach would utilise heuristics, leveraging information from the surrogate model to determine promising areas within the search space.

Once the generated configurations have been evaluated by the fast surrogate model, the most promising configurations (i.e., the ones with the highest estimated score) are evaluated with the slow objective function (the real optimisation target), and their configuration details and scores are used to update the surrogate model. There are several approaches to selecting the most promising individuals, defined by the acquisition function, discussed later in this text.

This section will discuss the main components of the BO algorithm in more detail, beginning with surrogate models in Section 2.3.1. Section 2.3.2 will focus on acquisition functions, whilst Section 2.3.3 will outline the specific algorithm steps, including a pseudocode of a basic implementation. Finally, Section 2.3.4 will describe practical considerations and challenges.

2.3.1 Surrogate Models

Surrogate models serve as approximations to the true objective function and are used to predict the performance of new candidate solutions in the search space. By utilizing the surrogate model, BO can efficiently explore the search space and identify promising areas for further evaluation, without needing to assess all configurations according to the objective function [81][82]. The choice of surrogate model in BO depends on various factors, including the problem domain, available data, and computational resources. Two commonly used surrogate models are Gaussian process regressors [83][84] and random forests [85].

Gaussian process regressors are flexible and powerful models that can capture complex relationships between input parameters and the objective function's performance [32]. They are particularly well-suited for problems with relatively small datasets or when the underlying relationship is expected to be smooth and continuous. A Gaussian process defines a prior distribution over functions and updates this distribution based on observed data, providing a posterior distribution that represents the uncertainty in predictions. The posterior distribution can be used to estimate the objective function's performance for unobserved configurations.

Random forests, on the other hand, are an ensemble learning method that combines multiple decision trees to make predictions. Each tree is trained on a subset of the available data, and the final prediction is obtained by averaging the predictions of individual trees. Random forests are known for their ability to handle high-dimensional data and capture complex interactions between input

parameters [37]. They can be used as surrogate models in BO by training them on the evaluated configurations and their corresponding objective function values.

To train the surrogate model, the input configurations are represented by their characteristics or features, which could include hyperparameters, design choices, or any other relevant attributes. The objective function's performance on these configurations serves as the target variable. For example, in the context of optimizing machine learning algorithms, the features could be the algorithm's hyperparameters, and the target variable could be the algorithm's accuracy on a specific dataset. The features will need to be encoded in such a way as to be interpreted by the surrogate model. For example, unordered categorical variables may be one-hot encoded.

Once the surrogate model is trained, it can be used to predict the performance of new, unobserved configurations. These predictions provide an estimate of the objective function's value without the need for expensive function evaluations. BO utilizes these estimates to guide the search process towards regions of the search space that are likely to yield better performance.

It is important to note that the surrogate model approximates the true objective function and carries some inherent uncertainty. The uncertainty captures the surrogate model's estimation error and plays a crucial role in the selection of candidate solutions for evaluation. Acquisition functions, as discussed in Section 2.3.2, leverage this uncertainty to balance exploration and exploitation and guide the search towards promising areas of the search space [84]. Unlike random forests, Gaussian process regressors return an uncertainty value with their predictions, making them a natural choice when working with acquisition functions such as probability of improvement and expected improvement. A random forest regressor would need to be modified in order to return these values. However, in general, a random forest regressor can be trained much faster than a Gaussian processor regressor.

2.3.2 Acquisition Functions

Acquisition functions play a crucial role in BO by guiding the selection of the most promising configurations to evaluate [84]. These functions determine the utility or potential of different candidate solutions within the search space. The choice of an appropriate acquisition function is

essential to balance exploration and exploitation, as it influences the trade-off between exploring new areas of the search space and exploiting regions that are likely to yield high performance.

The acquisition function considers the surrogate model's predictions and its associated uncertainty or confidence. The uncertainty captures the model's estimation error, which is crucial for efficient exploration. There are several popular acquisition functions used in BO, each with its own characteristics and advantages. Three such acquisition functions are described next.

Predicted value

This is the simplest acquisition function as it takes only the value predicted by the surrogate model without alteration. That is, the promising areas of the search space are identified only as those which contain instances for which the surrogate model calculates high estimated values. The following acquisition functions attempt a trade-off between exploration and exploitation, whereas this simple approach looks only to exploit existing knowledge regarding the search space.

Probability of improvement

The Probability of Improvement (PI) acquisition function is a popular choice in BO. It aims to select the configuration that has the highest probability of improving upon the best-known performance observed thus far during the optimization process. The PI function focuses on exploiting regions within the search space that have a high probability of yielding better results [86].

The PI acquisition function calculates the probability that a new configuration will improve upon the current best-known performance. It does this by comparing the surrogate model's predictions with the current best performance. If the predicted value at a particular configuration exceeds the current best performance plus a certain threshold, that configuration is considered promising. PI is calculated as shown in Equation 2.9.

$$\mathrm{PI}(x) = \Phi(\mathrm{f}(x) > x_{best} + \varepsilon) \qquad (2.9)$$

Where $\mathrm{PI}(x)$ is the probability of improvement at a particular configuration x in the search space, $\mathrm{f}(x)$ is the surrogate model's prediction for configuration x, x_{best} is the best-known performance observed so far during the optimisation process, Φ is the cumulative distribution function of the standard normal distribution, and ε is the threshold parameter that determines the level of

improvement required for a configuration to be considered valuable. ε can be defined based on a user-specified parameter or adaptively adjusted during the optimization process. A higher threshold encourages more exploration, as it allows for the consideration of configurations that may have a lower probability of improvement but still offer a significant potential gain. On the other hand, a lower threshold favours exploitation, focusing on configurations that have a higher probability of surpassing the current best performance.

By selecting configurations based on their probability of improvement, the PI acquisition function tends to direct the optimization process towards promising areas of the search space. It exploits regions that are likely to yield better results while gradually refining the estimate of the objective function. However, this behaviour leads to a system that is overly exploitative, rather than exploratory [86].

Expected improvement

The expected improvement (EI) acquisition function is another popular choice in the BO literature that quantifies the expected improvement over the current best performance. First proposed by Močkus et al. [87], it provides a balance between exploration and exploitation by considering both the probability of improvement and the potential magnitude of improvement.

To calculate the expected improvement, the EI function utilizes the predictions of the surrogate model and its associated uncertainty. The surrogate model estimates the performance of different configurations within the search space based on the observed evaluations. The uncertainty captures the model's estimation error or lack of confidence in its predictions. EI is calculated as shown in Equation 2.10.

$$\mathrm{EI}(x) = \mathrm{E}[\max(\mathrm{f}(x) - x_{best}, 0)] \qquad (2.10)$$

Where $\mathrm{EI}(x)$ is the probability of improvement at a particular configuration x in the search space, $\mathrm{f}(x)$ is the surrogate models prediction for configuration x, and x_{best} is the best-known performance observed so far during the optimisation process. E[...] is the expected value operator. This calculates the average value of improvement considering the uncertainty associated with the surrogate model's predictions. It considers the probability distribution of $\mathrm{f}(x)$ and computes the weighted average of $\max(\mathrm{f}(x) - x_{best}, 0)$ over the distribution [86].

The EI function evaluates the potential improvement by comparing the surrogate model's predictions with the current best performance observed so far. It considers areas where the surrogate model predicts a higher improvement than the current best performance and weighs this improvement by the probability of its occurrence. In other words, EI encourages exploration by giving more importance to configurations with higher likelihood of improving upon the current best performance.

One of the key advantages of the EI function is its ability to explore regions of the search space where the surrogate model is uncertain. By considering areas with high uncertainty, EI promotes exploration and helps to avoid premature convergence to suboptimal solutions. This exploration aspect is crucial, especially in the early stages of optimization when the surrogate model has limited information about the search space.

The magnitude of improvement also plays a role in the calculation of the expected improvement. The EI function considers not only the probability of improvement but also the potential gain in performance. It prioritizes configurations that not only have a higher probability of improvement but also offer a larger expected gain in performance. This consideration ensures that the algorithm focuses on configurations that have the potential for significant improvements, rather than just minor incremental gains. Furthermore, it is non-parametric, unlike the PI acquisition function.

2.3.3 Optimisation Algorithm

The BO algorithm follows a systematic process to iteratively search for the optimal solution within the search space. This section outlines the key steps involved in the algorithm, including the evaluation of candidate configurations, updating the surrogate model, and selecting the most promising configurations for further evaluation. This description is outlined in ref [84].

The algorithm begins by initialising the surrogate model using an initial set of evaluated configurations. These configurations are typically selected based on random sampling or domain knowledge to cover a representative portion of the search space. The surrogate model is trained using the characteristics of these configurations as features and their corresponding objective function values as targets. Based on the surrogate model and a defined search strategy, new candidate configurations are generated for evaluation. The search strategy determines how the algorithm explores the search space to generate promising configurations. It can be as simple as random

sampling or incorporate more intelligent techniques that leverage information from the surrogate model to guide the exploration.

The surrogate model is then used to estimate the performance of the newly generated configurations. Each configuration is assigned a "surrogate score" based on its predicted performance. The acquisition function plays a crucial role in selecting the most promising configurations for further evaluation. Depending on the acquisition function utilised, it can balance exploration and exploitation by considering the surrogate model's predictions and associated uncertainties. Different acquisition functions, such as Probability of Improvement or Expected Improvement, can be used to guide the selection process.

The most promising configuration(s), as determined by the acquisition function, are selected for evaluation using the objective function. These configurations undergo the costly evaluation process to obtain their actual objective function values. The evaluations provide additional information to update the surrogate model in the following iteration and refine the estimation of the objective function's performance.

The algorithm repeats the evaluation, surrogate model update, and selection steps for a predefined number of iterations or until a convergence criterion is met. The convergence criterion can be based on the improvement in the objective function value or the stability of the surrogate model's predictions. If the convergence criterion is met, the algorithm terminates, and the configuration with the highest observed objective function value is considered the optimal solution. Otherwise, the algorithm continues to iteratively refine the estimation and search for better solutions.

Procedure 2.1 Outline of the Bayesian optimization procedure

---

1. *Candidate_solutions* = randomly generate *#Candidate_solutions* configurations;
2. *Scores* = run objective function for all configurations in *Candidate_solutions*;
3. Fit *Surr_model* with *Candidate_solutions* as features, *Scores* as target;
4. $i = 0$;
5. **While** $i <$ *It_count*:
   a. *New_Candidate_solutions* = randomly generate *#Candidate_solutions* configurations;
   b. $\hat{Y}$ = calculate a surrogate score for each new config with *Surr_model*;
   c. *Best_config* = config with highest score according to $\hat{Y}$;
   d. *Score* = run objective function for *Best_config*;
   e. Add *Best_config* to *Candidate_solutions*, add *Score* to *Scores*;
   f. Retrain *Surr_model* on *Candidate_solutions* and *Scores*;
   g. i += 1;

Output: Best configuration according to objective score;

An example of a basic implementation of BO is given in Procedure 2.1, which works as follows. First, *#Candidate_solutions* PU learning configurations are randomly generated (step 1) and evaluated, with their scores as calculated by the objective function saved as *Scores* (step 2). A *Surr_model*, is then trained, using *Candidate_solutions* as features, and *Scores* as the target variable (step 3). This involves processing the *Candidate_solutions* in such a way as to be suitable input for *Surr_model.* How they are processed will depend on the component types. The iteration index i is set to 0 (step 4). A new set of *#Candidate_solutions* configurations, *New_Candidate_solutions*, are randomly generated (step 5.a) and a surrogate score for each is calculated by *Surr_model* and saved as $\hat{Y}$ (step 5.b). The best configuration, *Best_config*, with the highest score according to $\hat{Y}$ is evaluated using the objective function (steps 5.c,d), and added to *Candidate_solutions*, with the objective Score (F-measure) added to *Scores* (step 5.e). *Surr_model* is then retrained with *Best_config* added to *Config* (step 5.f). i is incremented by 1 (step 5.g). This process (steps 5.a-g) is repeated *It_count* times. Finally, the best configuration, according to the objective score, is returned.

This is an example of a basic implementation, using the predicted value as the acquisition function and random candidate solution initialisation. This procedure can be easily adapted to include more complex acquisition functions, candidate solution initialisation, and specific components of the given optimisation task.

2.3.4 Practical Considerations and Challenges

While BO offers a powerful framework for optimizing complex objective functions, there are, as with any optimisation approach, several practical considerations and challenges that need to be considered.

Computational expense

Training surrogate models, such as Gaussian process regressors or random forests, involves fitting the models to the evaluated configurations and their corresponding objective function values. The complexity of these models can vary depending on the problem domain and the amount of available

data. Training more complex models may require more computational resources and time. Therefore, it is essential to consider the trade-off between model complexity and computational expense. In some cases, simpler models may be preferred to reduce the computational burden, especially when the computational resources are limited.

Evaluating candidate configurations using the surrogate model is generally computationally cheaper than evaluating the objective function directly. The surrogate model provides estimates of the objective function's performance without the need for time-consuming evaluations. However, the efficiency of this estimation process depends on the surrogate model's complexity and the number of candidate configurations to be evaluated. It is important to strike a balance between the number of evaluations performed by the surrogate model and the computational expense required for each evaluation. Techniques such as parallelization or efficient sampling strategies can be employed to mitigate the computational cost of evaluating candidate configurations.

The choice of acquisition function also affects the computational expense. Some acquisition functions, such as the Probability of Improvement or Expected Improvement, require additional computations to determine the most promising configurations for evaluation. These computations may involve optimizing acquisition functions or estimating probability distributions. The complexity of these computations can impact the overall computational expense of the optimization process. However, given that the objective function is expensive enough to justify the use of a surrogate model, these extra computations may prove negligible.

Note that the computational expense should be justified by the complexity of the objective function and the optimization problem. If the objective function is relatively simple and inexpensive to evaluate, the benefits of using a surrogate model and BO may be outweighed by the computational overhead. In such cases, alternative optimization methods that directly evaluate the objective function may be more suitable.

Surrogate model selection

The choice of the surrogate model depends on several factors, including the characteristics of the problem domain, the available data, and the computational resources at hand. Different surrogate

models have different strengths and limitations, and selecting the most appropriate one is essential for achieving accurate predictions and efficient optimization.

Gaussian process regressors are commonly used surrogate models in Bayesian optimization [83][84]. They are flexible and powerful models that can capture complex relationships between input parameters and the objective function's performance. Gaussian process regressors define a prior distribution over functions and update this distribution based on observed data, resulting in a posterior distribution that represents the uncertainty in predictions. The posterior distribution can be used to estimate the objective function's performance for unobserved configurations [32]. Gaussian process regressors are particularly suitable when the underlying relationship between input parameters and the objective function is expected to be smooth and continuous. They are also advantageous when dealing with relatively small datasets, as they provide a probabilistic framework for incorporating uncertainty in predictions.

Random forests are another popular choice for surrogate models in Bayesian optimization. They are ensemble learning models that combine multiple decision trees to make predictions. Each tree is trained on a subset of the available data, and the final prediction is obtained by averaging the predictions of individual trees [36]. Random Forests are known for their ability to handle high-dimensional data and capture complex interactions between input parameters. They are especially effective when dealing with noisy or heterogeneous data [37]. Random forests can be used as surrogate models in BO by training them on the evaluated configurations and their corresponding objective function values. However, utilising a random forest classifier with an acquisition function such as EI or PI (see Section 2.3.2) requires a modification of the standard implementation to return uncertainty metrics. For example, the approach utilised by Thornton et al. [90] calculates the predictive variance for all the trees in the forest for each prediction and uses this as the uncertainty metric. The higher the variance, the higher the uncertainty.

In addition to Gaussian process regressors and random forests, other surrogate models can also be considered based on the specific problem characteristics. Neural networks, for example, have shown success in various domains and can capture complex non-linear relationships. Support vector models can handle high-dimensional data and incorporate kernel functions to capture non-linearities.

When selecting a surrogate model, it is important to assess the strengths and limitations of each model and consider how well it aligns with the problem domain and available data. Factors to consider include the complexity of the model, the interpretability of the predictions, the ability to handle noisy or heterogeneous data, and the computational resources required for training and prediction. It may be necessary to experiment with different surrogate models and evaluate their performance using appropriate metrics.

Hyperparameter tuning

Surrogate models and acquisition functions often involve hyperparameters that control their behaviour and performance. These hyperparameters determine important aspects such as the flexibility of the surrogate model, the balance between exploration and exploitation, and the level of uncertainty considered in the optimization process. Properly tuning these hyperparameters is essential to ensure optimal performance and achieve meaningful results. The choice of hyperparameter values of a classifier can significantly impact the output of that classifier [88], thus it follows that it can drastically alter performance of the surrogate models and acquisition functions. Suboptimal hyperparameter settings may lead to poor predictive accuracy, inadequate exploration or exploitation, or inefficient use of computational resources. Therefore, it is important to carefully tune these hyperparameters to achieve the best possible performance.

It is worth noting that hyperparameter tuning is an iterative and time-consuming process [88]. It may require multiple rounds of experimentation and evaluation to find the optimal hyperparameter settings. The computational expense involved in hyperparameter tuning should be considered alongside other practical considerations such as the computational cost of the optimization process and the available resources.

Convergence

Bayesian optimization aims to find the optimal solution within the search space. However, several challenges can hinder convergence, such as premature convergence to suboptimal solutions. Proper exploration of the search space is essential to overcome these challenges and improve the convergence of the optimization process. Exploration and exploitation are two key components in BO that address the trade-off between searching for new, unexplored regions and exploiting the

currently known promising areas of the search space. The choice of acquisition function significantly influences this balance [84].

It is important to note that achieving convergence in BO can be a complex task, and the optimal strategy for exploration and exploitation may vary depending on the problem domain and the specific optimization goals. Therefore, it is crucial to carefully select or design the acquisition functions and employ appropriate techniques that suit the characteristics of the problem at hand.

2.4 Automated Machine Learning (Auto-ML)

Auto-ML is a rapidly growing field of machine learning (ML) that looks to limit the human involvement in ML applications [8], reducing the demand for domain experts and allowing those without extensive ML knowledge to operate complex ML pipelines [9]. Algorithm performance is largely dependent on input data [89]. Auto-ML can assuage this issue by searching for the best model specific to the target ML task (e.g., the best model for a given classification dataset).

Whilst [9] defines an Auto-ML system as developing a full ML pipeline, from data preparation to model evaluation, some researchers approach Auto-ML as a combined algorithm selection and hyperparameter optimisation (CASH) task, defined by [90] as automatically and simultaneously choosing a learning algorithm and corresponding hyperparameter settings to optimise empirical predictive performance on a given input dataset.

Early approaches to Auto-ML involved grid search, in which a grid of configurations is developed, and each configuration is evaluated [91]. This has the benefit of completeness (in the context of the grid's elements), as it evaluates all desired configurations for a pre-defined set of algorithms and hyper-parameter settings. Furthermore, implementation is trivial [92] in comparison to other optimisation frameworks. The primary drawback of grid search is the computational expense. The number of evaluations grows exponentially with the number of hyperparameters [93]. By evaluating all desired configurations, many low-quality configurations will be evaluated that would be overlooked by a more intelligent optimisation method. As the evaluation function is often the largest bottleneck in an Auto-ML system, an intelligent (heuristic) selection of promising configurations to be evaluated is a fundamental advantage that many more sophisticated systems

have over grid search. This drawback renders grid search infeasible for high-dimensional grids (with many hyperparameters having their candidate values as elements of the grid), see e.g., [94].

Random search alleviates the expense of grid search in high-dimensional search spaces to a degree by evaluating fewer configurations [91]. Random search draws hyperparameter configurations randomly from the configuration search space, independently from previously drawn configurations [92]. This has the benefit of a more exploratory search (since the latter evaluated only pre-defined configurations), but no guarantee of an optimal solution within the search space [95].

Neither grid search nor random search exploits information gained from previous evaluations and thus are not, in their traditional form, considered intelligent optimisation approaches.

Due to the computational expenses of grid and random searches, there is a strong need for approaches which utilise information from previously tested configurations. This section will focus on two of the most popular, namely EAs and BO. Other strategies exist, such as gradient descent-based methods, which was used by [96] for neural architecture search, and reinforcement learning, which was used by [97]. However, these optimisation techniques are outside the scope of this work and will not be discussed further. For a relatively recent review of such topics see [9].

2.4.1 Evolutionary Algorithms (EAs) for Auto-ML

As previously discussed, EAs are powerful optimisation tools and are well established within the literature. As such, they are a suitable choice for Auto-ML and much work has been done applying EAs to particular Auto-ML tasks. For a general review of EAs, the reader is referred to Section 2.2. In this current subsection the discussion is focused on EAs specifically for Auto-ML, where an individual represents a candidate algorithm or classification pipeline configuration, i.e., a combination of one or more classification algorithms or methods (e.g., data pre-processing methods) and their hyperparameters. In this context, EAs provide a flexible and robust framework for exploring the space of algorithms and hyperparameter settings.

EAs offer several advantages for Auto-ML. They can handle high-dimensional search spaces and explore a wide range of algorithms and hyperparameter combinations. EAs perform a global, rather than a local, search [98][99], making them suitable for highly non-linear environments, such as those often present for Auto-ML tasks. Moreover, the population-based nature of EAs allows for parallel

evaluation of candidate configurations, making them suitable for distributed computing environments. However, even when parallelising the evaluation of candidate solutions, computational expense is a large drawback of EAs when applied to Auto-ML tasks. Unlike BO (see Section 2.3), EAs generally evaluate all candidate solutions in the population. The fitness function associated with evaluating candidate solutions is generally expensive, especially for classifiers that are already expensive, such as neural networks. This computational expense, however, has not deterred research into Auto-ML utilising EAs, as discussed next.

Olson et al. [100] proposed the Tree-based Pipeline Optimisation Tool (TPOT), an Auto-ML system using Genetic Programming (GP). The GP uses tree-based encoding such that the individuals in the population are ML pipelines. Rather than mathematical operators as the functions (as shown in Section 2.2.8), the functions are pipeline operators and hyper-parameters, e.g., specifying the number of trees in a random forest or the number of features selected during feature selection. Each individual is evaluated by the classification accuracy of the pipeline produced. Experiments on 150 benchmark datasets showed statistically significant improvement over random forest on 21 of the datasets, no statistically significant difference on 125 datasets, and statistically significantly worse results on 4 datasets by a metric of "balanced accuracy", where accuracy is adjusted for class imbalance [101]. Whilst random forest is an excellent classification algorithm, as it is not an Auto-ML system its usefulness as a comparison is limited. A major drawback of the original version of TPOT is that it can produce individuals that represent invalid pipelines, with a large computational cost in terms of evaluation and generation [74]. This issue has been addressed by other EA-based Auto-ML systems, such as the Resilient Classification Pipeline Evolution system (RECIPE).

Like TPOT, RECIPE, proposed by [74], is an GP-based Auto-ML system that evolves ML pipelines. However, RECIPE uses a grammar to ensure that all generated individuals are valid, so that it does not waste resources on invalid individuals. Furthermore, RECIPE evaluates a larger search space than TPOT and Auto-Sklearn (see Section 2.4.2) which, whilst making for a more complex search space, allows for a greater variety of solutions [74]. Experiments [74] showed RECIPE outperforming TPOT and Auto-Sklearn with regards to F-measure with statistical significance 2 out of 20 times. 13 out of 20 experiments showed no statistically significant difference. Of the further 5 experiments, TPOT outperformed RECIPE and Auto-Sklearn 4 times, and Auto-

Sklearn performed best 1 time. The authors suggest that the increased search space of RECIPE, whilst having the advantage of diverse solutions, may have hindered its performance in these experiments.

A more recent version of TPOT named layered TPOT (LTPOT) looked to assuage the issue of wasted computational resource by evaluating individuals on a subset of the data before allowing them to be fully evaluated [102]. Whilst this does not eliminate the issue of invalid individuals as RECIPE does, it limits the computational resources used by them. Experimental results showed that LTPOT generally found a pipeline as good as that found by TPOT sooner than TPOT found it [102].

Another very recent version of TPOT has been proposed utilising Bayesian optimisation. This will be discussed in Section 2.4.2.

Zöller & Huber [91] provide a comparison between several Auto-ML frameworks that produce whole ML pipelines. The results showed that TPOT outperformed the other methods on the majority of the 137 classification tasks tested. Also, it is estimated that TPOT overfit the data less than all other techniques.

Whilst TPOT is a broad Auto-ML system in that it generates whole pipelines for generic learning tasks, there are several specialist Auto-ML approaches. For example, [103] used an EA to develop an Auto-ML system that focuses on classifier ensembles. Ensemble classifiers, such as random forest, utilise multiple models of a single type of classifier (e.g., decision tree) to make a classification. Ensembles often outperform their base classifiers [104] and thus a specialised Auto-ML system specific to ensembles may outperform a more generic Auto-ML system. The proposed method, named PBIL-Auto-Ens, uses an Estimation of Distribution Algorithm (EDA), a type of EA that differs from the genetic algorithm (GA) approach outlined in Section 2.2. EDAs generate a population of individuals by sampling from a probability distribution, and after sorting by fitness, a proportion of the individuals are selected, and the probability distribution is re-estimated. New solutions are generated according to this distribution, without using any crossover or mutation, and the process is repeated until a stopping criterion (like a fixed number of generations) is satisfied. The proposed system was compared against Auto-WEKA (see Section 2.4.2) over 15 datasets and outperformed it on 12 of the 15 according to the error rate.

Another recent example of a specialised Auto-ML system is Auto-MEKAGGP, proposed by [74], which focusses on multilabel classification (MLC). An MLC task is one in which an instance

can be associated with more than one class label. Much like RECIPE, Auto-MEKAGGP uses Grammar-based Genetic Programming (GGP). Unlike RECIPE, Auto-MEKAGGP is a CASH system, rather than one that builds whole ML pipelines. As Auto-ML systems built for standard binary classification (single label classification) associate an instance with only one class, they are not (without extension) applicable to MLC. As such, systems specific to MLC are needed. Auto-MEKAGGP searches the space of algorithms and configurations available in the MEKA tool (an open-source extension to the WEKA library (see Section 2.4.2) that provides access to multi-label classifiers). Compared to another EA-based Auto-ML system specific to MLC and two baseline approaches, Auto-WEKAGGP showed best performance on average [74].

Whilst generic systems such as TPOT and RECIPE are good tools for a wide range of learning scenarios, often better results can be achieved by finding a system specific to the target problem, which could involve e.g., a given type of algorithm like ensembles or multilabel classification algorithms. As such, systems such as PBIL-Auto-Ens and Auto-MEKAGGP are invaluable.

A recent Auto-ML system is AutoML-Zero [105] developed by researchers at Google. The primary idea behind AutoML-Zero is to evolve machine learning algorithms from scratch, without prior knowledge or human expertise. Instead of relying on predesigned algorithms or architectures, AutoML-Zero utilises an EA to evolve sequences of mathematical operations which are then used as classifiers. This is a relatively novel approach, primarily due to the computational expense involved in the search, prohibiting the development of such a system by many researchers without access to state-of-the-art high-performance computing systems, and achieved seemingly excellent results, developing, without human intervention, advanced ML techniques such as stochastic gradient descent, the ReLU activation function, and gradient normalisation. It was even applied to traditionally challenging machine learning scenarios, such as datasets with few training examples, and datasets with multiple classes. In the case of few training examples, a technique that has been established in the machine learning literature referred to as noisy ReLU [106][107] was discovered by the AutoML-Zero system. In the case of multiple classes, AutoML-Zero developed a technique that used the transformed mean of the weight matrix as the learning rate. It was unclear as to why, but it appeared to aid in the case of multiple classes.

However, a major drawback of the paper proposing AutoML-Zero is that the system was not compared with any other Auto-ML system. That said, it remains an exciting contribution to the Auto-ML field, considering its high degree of novelty, and a promising tool for future work. Recently, AutoML-Zero has been extended in the work of Guha et al. [108] for multi-objective optimisation. This new version of AutoML-Zero considers computational efficiency as well as predictive performance. However, this new version is only compared against the old version of AutoML-Zero, and thus suffers the same limitations.

In summary, EAs have emerged as powerful optimization tools, making them well-suited for Auto-ML tasks. These algorithms offer a flexible and robust framework for exploring the space of algorithms or pipelines and their hyperparameter settings. The process essentially involves initializing a population of candidate configurations representing algorithm/pipeline-hyperparameter combinations, evaluating their performance, selecting the fittest configurations, introducing variation through genetic operators, replacing individuals in the population, and terminating the process based on predefined criteria. EAs have several advantages for Auto-ML, including their ability to handle high-dimensional search spaces, explore a wide range of solutions, and perform global searches in highly non-linear environments. However, computational expense is a drawback, as EAs often evaluate all candidate solutions at each generation. Despite this, researchers have proposed both EA-based generic Auto-ML systems such as TPOT, RECIPE AutoML-Zero, and EA-based Auto-ML systems tailored to specific learning scenarios such as PBIL-Auto-Ens and Auto-MEKAGGP, which complement more generic Auto-ML frameworks, In general, these EA-based Auto-ML systems have demonstrated promising results in experiments and comparisons with other non-EA-based techniques, highlighting the importance of finding solutions specific to the target problem for improved performance.

2.4.2 Bayesian Optimisation (BO) for Auto-ML

BO is a type of sequential model-based optimisation (SMBO) method. The configurations of previously tested solutions are assessed to inform where in the search space to consider the generation of new candidate solutions [91]. Once a new configuration is assessed, this is also used alongside the

previous configurations to inform the future generation of candidate configurations. As such, BO systems will exploit promising regions and often converge to a local minima in the search space [91].

In the context of Auto-ML, an initial set of candidate configurations, representing classification pipelines, is sampled based on prior knowledge or randomly generated. Each candidate configuration is evaluated using a performance metric, typically through cross-validation, to estimate its quality. The performance metric provides a measure of how well the configuration performs on the given task, such as accuracy or F-measure. A probabilistic surrogate model, often a Gaussian process or random forest, is then fitted to the evaluated data, capturing the relationship between the configurations and their corresponding performance. The surrogate model serves as a proxy for the true performance landscape and allows for efficient exploration and exploitation of the design space. Using the surrogate model, an acquisition function is defined to determine the next configuration to evaluate. The acquisition function balances the exploration of unexplored regions and the exploitation of promising regions in the design space. Popular acquisition functions include Expected Improvement (EI) and Probability of Improvement (PI) (see Section 2.3.2). Once a new configuration is selected, it is evaluated, and its performance is added to the existing data. The surrogate model is retrained, incorporating the new information. This iterative process of evaluating, updating the model, and selecting new configurations continues until a termination criterion is met, such as reaching a maximum number of evaluations or convergence of the surrogate model.

The primary advantage of BO is its ability to efficiently handle expensive-to-evaluate functions, which is particularly beneficial for Auto-ML tasks with computationally expensive algorithms or pipelines. BO focuses the evaluation efforts on promising configurations, exploiting the knowledge gained from previous evaluations to estimate the performance of a classification algorithm or pipeline, rather than calculating the actual value using the often expensive objective function. Zöller and Huber [91] explain that the trade-off between exploration and exploitation is determined by the use of an acquisition function such as Expected Improvement (EI). However, if an acquisition function that focusses only on exploitation is used, rather than exploration, this will lead to the optimisation procedure becoming susceptible to being trapped in local optima [86].

BO is commonly used throughout the Auto-ML literature and is the basis for several well-known Auto-ML systems, such as Auto-WEKA and Auto-Sklearn. Auto-WEKA is a tool that approaches

Auto-ML using BO to search the classification and regression search-space of WEKA algorithms and hyperparameter configurations. WEKA is an open source machine learning library with tools for a variety of machine learning tasks [109]. Auto-WEKA was initially proposed by [90] as an Auto-ML tool for classification, and later updated by [110] to handle regression tasks. Auto-WEKA was compared to 39 baseline classifiers with default parameter settings in WEKA and a random grid search (an approach where a grid of candidate solutions is specified, and a random search performed on that grid) on 21 datasets [90]. Auto-WEKA outperformed the baseline classifiers according to error rate on 14/21 datasets and performed equally on the remaining 7. In comparison to random grid search, Auto-WEKA outperformed on 20/21 datasets, and performed worse on 1. Whilst [90] did not compare to any other Auto-ML framework, it is worth noting that Auto-WEKA was generally outperformed by the PBIL-Auto-Ens method described in Section 2.4.1 [103]. It is also worth noting the performance of Auto-Sklearn, a sister package of Auto-WEKA specific to Scikit-Learn, in experiments by [91].

Auto-Sklearn is largely similar to Auto-WEKA, also utilising BO, but specific to the Scikit-Learn python library [111]. Whilst results comparing Auto-Sklearn to other Auto-ML methods are not directly analogous to Auto-WEKA as Auto-Sklearn has a slightly reduced search space, the results are still indicative. Auto-Sklearn was generally outperformed by TPOT, but generally outperformed all other techniques [91]. However, it is estimated that Auto-Sklearn tended to overfit more than all other techniques except random search. This indicates that TPOT is a more desirable framework than Auto-Sklearn generally due to the increased performance and lack of overfitting.

Two new versions of TPOT, TPOT-BO-S and TPOT-BO-ALT, were proposed by Kenny et al. [112]. These new versions loosely couple the standard TPOT, utilising GP, with BO. The first version TPOT-BO-S, switches to a BO procedure at a specified point in the procedure. TPOT-BO-ALT alternates between the standard TPOT and BO procedures throughout the run. Overall, these new variations did not add substantial improvement over the original TPOT system.

In summary, BO is a sequential model-based optimization technique used in Auto-ML tasks. It leverages Bayesian inference to efficiently search the space of machine learning algorithms or pipelines and their hyperparameters. BO involves sampling an initial set of candidate configurations, evaluating their performance, fitting a surrogate model to capture the performance landscape, and

using an acquisition function to select new configurations for evaluation. The iterative process continues until a termination criterion is met. BO excels in handling expensive evaluations and focuses on promising configurations, exploiting knowledge gained from previous evaluations. However, it can become trapped in local optima due to a lack of population diversity. BO has been successfully employed in Auto-ML frameworks like Auto-WEKA and Auto-Sklearn, which utilize BO to search algorithm and hyperparameter spaces. These frameworks have shown competitive performance compared to baseline classifiers and other Auto-ML methods. Auto-WEKA and Auto-Sklearn, however, have demonstrated some limitations, such as being outperformed by specific methods like PBIL-Auto-Ens and TPOT in certain scenarios. Overall, BO is a valuable approach in Auto-ML, but careful consideration of its limitations and appropriate framework selection is essential for achieving optimal results.

2.4.3 Practical Considerations and Challenges

While Auto-ML holds promise for automating the machine learning pipeline, there are practical considerations and challenges that need to be addressed to ensure its effective implementation and deployment, as follows.

Computational expense

Searching a large space of algorithms and hyperparameters requires substantial computational resources and can be time-consuming. The complexity of the dataset and the use of classifiers that are themselves expensive, such as deep learning classifiers, further exacerbate this challenge. To assuage this issue, efficient resource management techniques can be utilised. Optimizing the utilization of available computational resources, such as CPUs and GPUs, can significantly speed up the Auto-ML process. Techniques like parallel computing, where multiple computations are performed simultaneously, help distribute the workload and reduce time required for optimization.

Another approach to mitigate computational expense is the use of approximation techniques, such as those used in BO. Instead of exhaustively evaluating all possible configurations, surrogate models can be used to estimate the performance of candidate configurations. These surrogate models

provide a fast and computationally inexpensive alternative to direct evaluation, allowing for more efficient exploration of the algorithm-hyperparameter space.

Furthermore, embedding computational expense as a criterion to be minimised by the Auto-ML system can benefit the process by favouring pipelines that need fewer computational resources. However, it is important to strike a balance between computational expense and the quality of the Auto-ML results. While it is desirable to minimize the computational time and resources required, it is crucial to ensure that the optimization process explores a wide area of the algorithm-hyperparameter space and finds high-performing configurations.

Evaluation

Another consideration is the selection of appropriate evaluation metrics and validation strategies. The performance of different algorithm-hyperparameter configurations needs to be evaluated to identify the best models for a given task. However, the choice of evaluation metrics can vary depending on the specific problem domain and the goals of the application. Using inappropriate metrics may lead to suboptimal results and misinterpretation of the model's performance.

The selection of evaluation metrics should align with the objectives of the problem. For example, in classification tasks, metrics like accuracy, precision, recall, F-measure, or area under the receiver operating characteristic curve (AUC-ROC) are commonly used. These metrics measure different aspects of the model's performance, such as overall correctness, trade-offs between precision and recall, or the ability to distinguish between classes. It is important to carefully consider which metrics are most relevant and meaningful for the specific problem at hand. For example, for perfectly balanced datasets, accuracy may be the most appropriate metric. It is important to consider the limitations and potential biases of the chosen evaluation metrics and validation strategies. For example, in imbalanced datasets, accuracy alone may not provide a comprehensive understanding of the model's performance. Metrics like precision, recall, or F-measure are often more appropriate for evaluating the model's performance on minority classes.

Generalisation

Ensuring the generalization and transferability of Auto-ML models is a critical consideration in the development and deployment of machine learning systems. While Auto-ML aims to automate the

model selection and hyperparameter optimization process, it is important to ensure that the selected models can perform well on unseen data.

One of the main challenges in generalization is the potential for overfitting. Overfitting occurs when a model becomes overly specialized to the training data and fails to generalize well to new, unseen data (test data). This is a concern in Auto-ML because the optimization procedure may inadvertently select models that perform well on the training data but fail to generalize to new instances. This problem also occurs in standard machine learning, but Auto-ML has been shown to be especially vulnerable to overfitting [113].

To mitigate the risk of overfitting and enhance generalization, robust validation techniques are crucial. Cross-validation is a commonly used approach that helps assess the model's performance on unseen data. By dividing the data into multiple subsets or folds, and iteratively training and evaluating the model on different combinations of these subsets, cross-validation provides a more reliable estimate of the model's generalization performance. Additionally, the selection of appropriate evaluation metrics can contribute to better generalization. Evaluation metrics that focus on the overall performance and robustness of the model, such as area under the precision-recall curve, can provide a more comprehensive assessment of the model's ability to generalize.

2.5 Positive-Unlabelled (PU) Learning

PU learning is a field of machine learning that focusses on learning models from datasets that consist of only positive-class and unlabelled instances [3]. PU learning shares the goal of binary classification – to accurately predict the class of an unseen example by learning to distinguish between two classes. However, since a standard binary classifier requires a training set with two class labels, a standard binary classifier built using a PU dataset would have to treat all unlabelled instances as a separate class, and so such classifiers will predict the probability of an instance being labelled (Pr (s=1)) as opposed to the probability of an instance belonging to the positive class (Pr (y=1)) [4] – where s is a variable taking 1 or 0 to indicate whether or not an instance is labelled, and y is the true label of an instance, taking values 1 or 0 to denote the positive or negative class, respectively. PU learning models, on the other hand, are trained to predict Pr(y=1) using PU data and have been

shown theoretically to improve upon standard binary classification models when applied to PU datasets [114].

PU learning is an important area of machine learning as it naturally arises in many different domains, such as bioinformatics [115][116][117], text mining [118][119][120], and cyber security [121][122]. For example, reference [115] utilises PU learning for the prediction of genes associated with diseases. This is a PU learning task where disease-associated genes are positive instances, as confirmed by biomedical experiments. However, the vast majority of the genes not associated with diseases have not undergone such experiments, since these experiments are expensive. As such, the genes not associated with diseases are better thought of as unlabelled instances as there is no experimental evidence indicating either association or disassociation. An example from the domain of text mining is found in [118], which proposed a text classification system utilising PU learning for web page classification. This is another learning task where PU learning is appropriate. Scraping web pages is an easy and quick task, so assembling a large dataset is a simple process. However, the majority of the instances (webpages) will be unlabelled as manually labelling each instance is an expensive task. As illustrated by these examples, PU learning is appropriate when the dataset consists of a small sample of reliable positives and a larger remaining sample of unknown-label instances.

PU learning is related to semi-supervised learning [123] in the sense that it specialises the semi-supervised scenario [3]. In both semi-supervised and PU learning, typically the large majority of training instances is unlabelled; but a semi-supervised learning's training set includes small proportions of both positive and negative instances, whilst a PU learning's training set does not include any negative instance.

Over the past two decades, many PU learning algorithms have been developed for a wide array of applications [3][124]. However, little has been written on the subject of evaluation metrics for PU learning, which is a challenging task. To address this shortcoming, Saunders & Freitas [20] reviewed evaluation approaches for PU learning and provided practical recommendations for improvement. The evaluation approaches will be discussed in Section 2.5.3.

The next three subsections detail common assumptions made to enable PU learning, the three most popular PU learning frameworks, and practical considerations.

2.5.1 PU Learning Assumptions

To enable PU learning, assumptions are commonly made about the data. The most common assumptions are negativity, separability and smoothness, selected completely at random, and selected at random [3].

Negativity

The most basic assumption to enable learning from PU data is the assumption of negativity. That is, it is assumed that all unlabelled data is simply negative [3]. This is the most naïve of the four assumptions discussed in this section, as the unlabelled set can contain a substantial amount of unlabelled positive instances, depending on the dataset. However, despite the naivety of this assumption, it can be effective if utilising a classification algorithm that produces a model that is robust to noise within the data.

Despite the assumption of negativity not holding in practice, it is still widely used [3]. The popularity of this assumption is due to the fact that it allows for the use of standard machine learning methods, without any modification to the classification algorithm or processing pipeline. In other words, making the negativity assumption means transforming the original PU learning problem into a standard classification problem, denying the true unlabelled nature of the original data. Conversely, treating the original data as a PU learning problem and using a PU learning algorithm means rejecting this naïve negativity assumption, which is the approach followed in this thesis.

Separability and smoothness

The assumption of separability states that the positive and negative instances are separable in the feature space. That is, it is assumed that a classifier hypothetically exists that can perfectly separate the positive and negative instances [3]. The assumption of smoothness states that instances which are close to each other in the feature space are likely to belong to the same class [3].

These assumptions are foundational to the two-step approach (see Section 2.5.2), arguably the most popular approach to PU learning.

Selected completely at random (SCAR)

The SCAR assumption, formalised by [4], states that the positive instances are labelled irrespective of their features, and thus the labelled set is an independent and identically distributed sample from the positive distribution. That is, for the given data, $\Pr(s{=}1) = \Pr(s{=}1|x)$, where $\Pr(s{=}1)$ represents the probability of an instance being labelled and x is an instance's feature vector. Or, put simply, the sample of positive instances in the labelled positive set is representative of the entire set of positive instances, both labelled and unlabelled. Making the SCAR assumption allows us to assume that the instances in the labelled positive set are representative of the instances within the positive unlabelled set, and thus, if a classifier can accurately predict the labelled positive instances, it should, in theory, be able to predict the unlabelled positive instances also. For this reason, the SCAR assumption is foundational to some PU learning approaches.

Elkan & Noto [4] show that an implication of the SCAR assumption is Equation 2.11.

$$f(x) = \frac{g(x)}{c} \qquad (2.11)$$

In other words, $g(x)$ differs from $f(x)$ by a constant factor, where $g(x)$ is a probabilistic classifier trained to distinguish the labelled set and the unlabelled set and thus predicts $\Pr(s{=}1|x)$ (referred to as a non-traditional classifier by [4]), and $f(x)$ is a probabilistic classifier trained to distinguish a positive and negative set and thus predicts $\Pr(y{=}1|x)$ (referred to as a traditional classifier).

One major implication of this formula is that if we are simply looking to rank instances by their predicted probability of belonging to the positive class, e.g., in target prioritisation, we can simply use $g(x)$, as the instances predicted as having the highest probability of belonging to the positive class will be the same for both $g(x)$ and $f(x)$. This approach alone is satisfactory for simple PU problems, such as a scenario where we have a reasonably balanced set of positive and unlabelled instances. However, PU problems are rarely this simple. The majority of PU datasets will consist of a small number of positive instances and a large number of unlabelled instances. As such, the standard approach to model training will yield poor results with the positive class being overwhelmed by the unlabelled class, resulting in poor recall of labelled instances. It is for this reason that the instances predicted as having the highest probability of belonging to the positive class by $g(x)$ cannot be trusted without further model evaluation, $g(x)$ may simply be an inaccurate classifier. As such,

PU learning methods that produce more reliable classifiers are needed. PU learning model evaluation is discussed in Section 2.5.3.

Selected at random (SAR)

The SAR assumption states that an instance is labelled depending on its features. That is, the labelling mechanism depends on the features of an instance [125]. It is thought that many PU learning datasets suffer from a labelling bias [3], thus the motivation for this assumption. Formalised, this assumption states that $e(x) = \Pr(s{=}1|x, y{=}1)$. Where e is the propensity score (the probability of a selected instance being labelled). In order to enable use of the SAR assumption, one must know the labelling mechanism. If the labelling mechanism is known, a standard classifier can be trained with the output values processed to incorporate it. However, if it is unknown, the SAR assumption cannot be used.

2.5.2 Approaches to PU Learning

In this section, the three major approaches to PU learning are discussed. Namely, the two-step framework, biased learning, and incorporation of the class prior.

Two-step framework

The most common PU learning framework is the "two-step" approach. The first step of this approach consists of identifying a set of reliable negative instances among the unlabelled set. That is, a set of instances that are substantially different from the labelled positive instances and are likely not unlabelled positive instances. The second step consists of building a classifier to distinguish the labelled positive instances from the reliable negative set. These two steps use only the training set. The resulting classifier is then used to classify the remaining unlabelled instances in the testing set [126]. Providing that the reliable negative set is an accurate representation of the negative class, this model will predict $\Pr(y{=}1)$ rather than $\Pr(s{=}1)$. This approach assumes separability and smoothness of the data. That is, it is assumed that there is a natural separation between the positive and negative classes (separability), and it is assumed that instances that are similar to each other have a similar probability of belonging to the positive class (smoothness) (see Section 2.5.1) [3].

Arguably the most well-known two-step technique in the PU literature is the "S-EM" method [120]. This technique selects a subset of positive instances (known as "spies") to be added to the unlabelled set. All instances in the unlabelled set (spies included) are then assigned the negative class label, and a naïve Bayes classifier is built to distinguish between the positive and the unlabelled set. The resulting classifier is then used to classify the unlabelled set (spies included) and the Pr(y=1) of each of the spy instances is used to determine a threshold under which an unlabelled instance's Pr(y=1) must fall to be considered a "reliable negative" instance. Several variations on this technique have been proposed, including that by [127] which utilises the multilayer perceptron classifier and an altered threshold calculation. Other well-known two-step techniques include the "Roc-SVM" method specific to PU text classification proposed in [5] which utilises the Rocchio classifier and an iterative SVM approach, and the "Positive Example Based Learning" (PEBL) method for web page classification, as proposed in [128].

Whilst the literature generally refers to two individual steps (Step 1, Step 2) when discussing two-step methods, this thesis uses a slightly different terminology. Instead, we reference the steps as phases, and recognise that "Step 1" actually often consists of two distinct phases. As such, when discussing two-step techniques, this work will reference Phase 1A, used to extract an initial reliable negative set, Phase 1B, an optional step that several methods take to use the initial reliable negative to further extract reliable negative instances from the unlabelled set, and Phase 2, "Step 2" in the usual description, which builds a classifier using the positive and reliable negative set and classifies the remaining unlabelled instances.

This notation is advantageous as it recognises that "Step 1" often consists of two distinct phases, and the use of "phase" rather than "step" allows us to reference the individual steps of the algorithms in each phase without confusion.

A generic implementation of Phase 1A of a two-step PU learning algorithm is given in Procedure 2.2 for the reader's reference.

In Procedure 2.2, U is the set of unlabelled instances, P is the set of labelled positive instances, *Classifier* is the classifier used in the implementation, RN is the set of reliable negative instances, y(x) is the class predicted for instance x by the classifier, and *threshold* is a predefined threshold for classing an instance as reliably negative.

Procedure 2.2 Basic Phase 1A implementation of a two-step PU learning algorithm

1. $RN \leftarrow \{\}$;
2. Split U into multiple subsets;
3. **For** each subset::
   a. $N =$ subset;
   b. Build *Classifier(P, N)*;
   c. Classify(U);
   d. **For** every instance x in U:
      i. If y(x) < *threshold* **then** $RN \leftarrow RN \cup x$
      ii. $U \leftarrow U - x$;

Output: *U, RN*;

Several Phase 1A methods follow the generic procedure shown in Procedure 2.2 with minor alterations. One variation on this procedure is the method proposed by [129]. The difference is that Phase 1A iterates a set number of times (5) and then only the top 1% of the instances predicted by the classifier to most likely be negative are added to RN. Also, the classifier used is the deep forest classifier (see Section 2.1). The [129] method will henceforth be referenced as "DF-PU" and is shown in Procedure 2.5.

To complement Procedure 2.2, Procedure 2.3 gives a basic implementation of Phase 1B.

Procedure 2.3 Basic Phase 1B implementation of a two-step PU learning algorithm

1. $RN \leftarrow$ predefined reliable negative set, determined in Phase 1A;
2. **While** loop condition:
   a. Build *Classifier(P, RN)*;
   b. Classify(U);
   c. **For** every instance x in U:
      i. If y(x) < *threshold* **then** $RN \leftarrow RN \cup x$
      ii. $U \leftarrow U - x$;

Output: *RN*;

In Procedure 2.3, *RN* is the set of reliable negative instances output by Phase 1A, *P* is the set of labelled positive instances, *Classifier* is the classifier used in the implementation, *U* is the set of unlabelled instances returned by Phase 1A, with the reliable negative instances removed, and y(x) is the class predicted for instance x by the classifier. Phase 1B is more akin to a standard semi-supervised machine learning algorithm. Phase 1B is an optional step, not always employed by two-step PU learning algorithms. Furthermore, the *while* loop in both Phase 1A and Phase 1B is optional, as a single iteration is a valid approach. The while loop generally involves splitting the unlabelled

set into multiple sets in order to handle the class imbalance often present in PU learning datasets. This is illustrated in the pseudocode by splitting the unlabelled set (U) into multiple subsets. However, should no class imbalance be present, a single iteration is valid. In fact, the systems proposed in this thesis do not utilise a *while* loop condition in Phase 1B at all. The reasons for this are two-fold. Firstly, class imbalance is often not present in the Phase 1B (when discriminating between the positive and the reliable negative sets), since the parameter that determines whether to classify an instance as reliably negative is often conservative. Secondly, inclusion of this parameter increases the size of the search space and offered no improvements in performance in preliminary experiments. As such, it can be argued that a *while* loop condition in Phase 1B is not necessary.

No procedure is given for Phase 2 as this phase simply consists of building a classifier to distinguish the positive and the reliable negative set and using that classifier to predict the class of the unlabelled instances.

A variation on the standard implementation that is used as a baseline approach in this work is the S-EM method, proposed by Liu et al. [120]. As previously mentioned, S-EM (also referred to as the "Spy" method) is one of the most well-known PU learning methods in the literature and, despite having been proposed two decades ago, remains a popular choice of baseline given its impressive performance [130-136]. S-EM primarily differs from the standard implementation due to the use of hidden positive instances to determine which instances to classify as reliably negative. That is, a set of the labelled positive instances are hidden in the unlabelled set and, when classified, their predicted probability of belonging to the positive class is used to determine the predicted probability under which genuine unlabelled instances should fall to be considered reliably negative. A pseudocode of this implementation is given in Procedure 2.4.

In Procedure 2.4, U is the set of unlabelled instances, P is the set of labelled positive instances, S is the spy set, *sample size%* is the percentage of labelled positive instances to be hidden in the unlabelled set, *Classifier* is the classifier used in the implementation, *RN* is the set of reliable negative instances, Pr(y=1) is the probability of belonging to the positive class, *noise level* is the level of noise to account for in the positive set, and $|S|$ is the number of instances in the spy set. Naïve Bayes is used as the classifier in both Phase 1A and Phase 2 of the S-EM method.

Procedure 2.4 S-EM ("Spy" method)

1. $N \leftarrow U$;
2. $S \leftarrow$ sample(P, *sample size%*), $P \leftarrow P - S$;
3. $N + S \leftarrow N \cup S$;
4. Run EM(*Classifier*, P, $N + S$);
5. Sort instances by their Pr(y=1);
6. $\theta \leftarrow$ Pr(y=1) of the instance in position (*noise level* $\times$ $|S|$) in sorted S;
7. $RN \leftarrow \{\}$; # initialised empty set
8. **For** every instance x in N: # i.e., instances in $N + S$ that are not spies
   a. **If** Pr(y=1|x) $< \theta$ **then** $RN \leftarrow RN \cup x$;
   b. $U \leftarrow U - x$;
9. Build *Classifier*(P, RN);

Output: Classifier

First, the data sets are initialised (steps 1-3). The Expectation Maximisation (EM) algorithm is run using *classifier*, computing Pr(y=1) for each instance in N+S, and rebuilding *classifier* with the updated Pr(y=1) values as an additional feature (step 4). The process iterates until the values of Pr(y=1) no longer change. A full explanation of how the EM procedure is applied can be found in [120]. After sorting the instances in S in decreasing order of their Pr(y=1) values (step 5), the probability threshold θ is set (step 6) and used to determine which instances in N+S that are not spies are added to the RN set and removed from U (step 8). The *classifier* is then built on P and RN (step 9). Steps 1-8 are Phase 1A, Phase 1B is skipped in this method, and step 9 is Phase 2. This method differs from the generic procedure of Phase 1A (Procedure 2.2) with the addition of the "spy" component as well as the inclusion of the Expectation Maximisation (EM) algorithm in step 4. So, the convergence criterion of the EM algorithm is analogous to the for loop of Procedure 2.2.

Several variations on this method have been proposed, but the original implementation remains a popular choice for comparison of newly proposed PU learning algorithms. Preliminary experiments showed S-EM outperforming a more recently proposed modified version, so the original implementation was selected as a baseline method for the experiments reported later in this thesis.

Procedure 2.5 outlines the DF-PU procedure. First, RN is initialised as an empty set, and U is split into 5 sets, each with 20% of the data, randomly sampled. For each *Set*, a deep forest classifier is trained to distinguish the positive instances and the instances in *Set*, treated as the negative instances. All instances in U are then classified, and the 1% of instances with the lowest Pr(y=1) are added to RN. Finally, a deep forest classifier is trained on P and RN.

Procedure 2.5 DF-PU

1. $RN \leftarrow \{\}$; # initialised empty set
2. *Sets* = *U* randomly split into 5 sets, each with 20% of the data;
3. **For** *Set* in *Sets:*
   a. Train deep forest classifier on *P* and *Set*;
   b. Classify U and sort instances by their $\Pr(y=1)$;
   c. $RN \leftarrow RN \cup$ 1% of instances with lowest $\Pr(y=1)$;
4. Build deep forest classifier on P and *RN;*

Output: Classifier

Implementations of both versions of the baselines can be found on GitHub[5].

Biased learning

Whilst biased learning is not utilised in this work, it is a prominent PU learning framework and as such will be briefly described in this section.

Biased learning is a PU learning framework that treats the unlabelled set as negative class instances and applies a higher penalty to the misclassification of positive instances. The unlabelled set is, as such, treated as a negative set with noise [3]. Deciding exactly how much more to penalise the misclassification of positive instances is non-trivial. Some papers, e.g., [137][138] tune their models according to an evaluation metric proposed by [139], shown in Equation 2.12.

$$\frac{p \times r}{\Pr(y = 1)} \qquad (2.12)$$

Where p is the precision, r is the recall, and $\Pr(y = 1)$ is the probability of an instance belonging to the positive class. This metric, however, is a weighted formula that considers both precision and recall equally. This approach is potentially inefficient as, depending on the goal of the PU learning classifier (PU learning goals are discussed in Section 2.5.3), it may be that either precision or recall should be considered more than the other. This metric has not been widely adopted in the literature, and there is no commonly used evaluation or tuning criteria specific to PU learning.

Many biased learning methods for PU learning are based on the support vector machine (SVM) classifier (see Section 2.1), mostly stemming from the biased SVM classifier proposed by Liu et al. [126]. This is, essentially, a standard SVM that applies a higher penalty to misclassification of the

[5] https://github.com/jds39/GA-Auto-PU

positive class, thus increasing the number of identified true positive instances, whilst also increasing the number of false positives. However, it is worth noting that these may not be truly false positives as their labels are unknown. Several extensions to the biased SVM approach have been proposed [15,119,138,140].

Incorporation of the class prior

As with biased learning, class-prior based learning is not utilised in this work but will be briefly described in this section. Class-prior based PU learning techniques use the known or estimated class-prior at various stages in the classification pipeline. Those that utilise it in the preprocessing stage seek to change the dataset before training the classifier [3]. One approach is to weight the instances in the dataset. I.e., assign a weight to the instances of each class that reflects the class prior [141]. Another approach is to consider unlabelled instances as both positive and negative when training the model. This can be done by duplicating the unlabelled instances and assigning a weight equivalent to the class-prior for the specific class [4].

Some studies look to alter standard binary classification methods to utilise the true class prior. The positive naïve Bayes algorithm, proposed by [142] and extended by [143], is a naïve Bayes classifier specific to PU learning. Rather than calculating the class probability as described in Section 2.1, another formula that calculates a higher prior probability for the positive class is used. This has a benefit over the preprocessing techniques discussed above as the prior probability does not need to be known, it can be estimated.

Finally, some studies use post-processing techniques that alter the class probabilities assigned to instances after classification. As discussed earlier, the SCAR assumption implies that the predictions of a model trained on a PU dataset differ from the predictions of a model trained on positive and negative data by a constant factor. If this constant factor is calculated or estimated, a standard binary classifier could be trained on the PU data and the assigned probabilities can be altered [3]. However, as previously discussed, a classifier trained on PU data may be inaccurate.

2.5.3 Practical Considerations and Challenges

This section details practical considerations and challenges of PU learning. Several of these have been highlighted in this chapter already but will now be discussed in detail.

Model evaluation

The absence of negative instances presents an issue to the evaluation of PU learning models as predictive accuracy metrics usually rely on knowledge of the true class labels of each instance. However, in PU learning we have only knowledge of the true class label of a sample of positive instances. The remaining positive instances, and all negative instances, are unlabelled. Due to these unlabelled instances, popular metrics such as true positive and false negative rates, precision, recall, and the F-measure [144], cannot be correctly calculated.

Under the SCAR assumption, given that the sample of positive instances in the labelled positive set is representative of the entire set of positive instances, both labelled and unlabelled, we can estimate several performance metrics for models tested on genuine PU data. That is, PU data that has not been engineered from a standard positive-negative (PN) dataset (with positive and negative labels). However, as these metrics represent performance estimates, they are not entirely robust. Arguably, a more robust approach is to evaluate a PU learning method on an engineered PU dataset before applying that method to a genuine PU learning task.

As identified in [20], the approach most frequently taken in the literature is to evaluate proposed methods on engineered PU data created from a standard PN dataset by hiding a certain percentage of positive instances in the negative set, thus creating an unlabelled set (i.e., all negatives and the hidden positives will be indistinguishably treated as ‘unlabelled’). This is done for the training set, whilst leaving the test set untouched. That is, the test set will contain positive and negative instances as in the original dataset. Hence, the model is trained on PU data but evaluated on fully labelled data. Therefore, we can accurately calculate standard PN metrics. This is arguably a more robust approach as the performance is not estimated based on the SCAR assumption (i.e., that assumption is not required) for the test set, as the test set is left untouched (i.e., positive instances are only hidden in the negative set in the training set, not the test set); we can rely on values of performance metrics that are accurately calculated based on the known class labels of the instances in the test set. However,

this approach assumes that a method that demonstrates good performance on an engineered PU dataset will also perform well on a genuine PU dataset. A table demonstrating the approach taken (either using genuine PU data or engineered) and the evaluation metrics used was given in [20] and repeated here as Table 2.1 for the readers reference.

Regarding metrics used to evaluate PU learning models, F-measure, precision, and recall should be reported, with an emphasis on the metric that most closely matches the goal of the learning task [20]. As shown by Table 2.1, F-measure was the most reported metric, reported in 37 of the 51 reviewed papers. There are two primary goals of PU learning – prioritisation and anomaly detection. Depending on the goal, either precision or recall may be more important than the other. As explained in [138], if the goal of the learning task is prioritisation, precision is the most important metric. If the goal is anomaly detection, recall is the most important metric, as follows.

In the task of prioritisation, one wishes to identify highly ranked targets. That is, instances that have the highest predicted probability of belonging to the positive class. As we are interested in prioritising instances, it is important that our model identify few unlabelled positives. Prioritisation is required when we need to identify a few top-ranked (most likely positive) instances for performing expensive or time-consuming future experiments on those few high-priority instances, and so minimising the number of false positives (maximising precision) is particularly important, to avoid doing experiments that produce negative results. Maximising recall is not so crucial because it would be too expensive or too time-consuming to perform future experiments to validate a large number of instances predicted as positives. An example is gene prioritization, where each gene is an instance and the positive class represents a biological function (or associated disease) of the gene, since biological experiments to verify gene functions tend to be expensive and time-consuming.

In the task of anomaly detection, one wishes to accurately identify positive class instances, which are usually a very small minority (“anomalies”). In anomaly detection, usually the cost of a false negative is usually much higher than the cost of a false positive. Therefore, maximising recall (minimising false negatives) is usually more important than maximising precision (minimising false positives). For example, when classifying a bank’s transactions into fraud (anomaly) vs non-fraud (normal), the cost of misclassifying a fraud transaction as a non-fraud transaction is usually much higher than vice-versa.

Table 2.2, also given in [20] but repeated here for the reader's reference, shows, for the 12 papers from Table 2.1 using real-world datasets for PU learning (i.e., genuine PU datasets, rather than engineered PU datasets), whether their goal is anomaly detection or prioritisation and whether they reported precision or recall. Unfortunately, out of the 3 papers addressing anomaly detection in Table 2.2, only one is reporting recall, and out of the 9 papers addressing prioritisation, only 4 are reporting precision. Without these results, the suitability of the proposed method for the given target application cannot be determined. This shows that the importance of reporting precision and recall separately (particularly in prioritisation or anomaly detection tasks) is still not well appreciated in the PU learning area.

Whilst precision and recall may be important metrics for a given learning task, considering either precision or recall in isolation is flawed, since it is well-known that it is relatively easy to maximise one of these measures at the expenses of obtaining a poor value for the other. Hence, it is important to report the F-measure, precision, and recall. This will allow researchers looking to utilise a PU learning method to make an informed decision on which algorithm is most appropriate for their use case, favouring those with a high F-measure and precision for prioritisation tasks, and those with a high F-measure and recall for anomaly detection. To further analyse the performance of a PU learning algorithm, it is important, when feasible, to assess the performance of its learned model on multiple distributions of unlabelled instances. That is, testing on different versions of the same dataset with differing percentages of the positive instances hidden in the unlabelled set in the training set, when doing experiments with engineered PU datasets. Due to the nature of PU learning, it is often hard to know the distribution of positive instances, and what proportion of them remain unlabelled. However, there are scenarios in which the distribution is known, or can be estimated [145][146]. In such scenarios, by providing results of experiments conducted on multiple distributions, we can provide a more comprehensive analysis of PU methods and inform on their appropriate use case.

To summarise, the evaluation of PU learning models poses challenges due to the absence of negative instances. Traditional metrics that rely on true class labels cannot be accurately calculated. The SCAR assumption allows for estimating performance metrics on genuine PU data, but they are not entirely robust. Alternatively, models can be evaluated on engineered PU datasets, where a percentage of positive instances is hidden in the negative set. This approach relies on accurately

calculated standard metrics but assumes the method will perform well on genuine PU datasets. Assessing the model's performance on multiple distributions of unlabelled instances provides a more comprehensive analysis. The choice of evaluation metrics depends on the goal of the learning task: prioritization or anomaly detection. Precision is crucial for prioritization, while recall is more important for anomaly detection. The importance of these metrics is determined by the cost associated with false positives and false negatives in each task, respectively.

Table 2.1. Evaluation approaches used by papers proposing PU learning algorithms.

Reference	Engineered or Genuine PU Data	F-measure	Accuracy	Precision	Recall	AUROC
[147]	E	✓	·	·	·	·
[148]	E	✓	·	·	·	·
[143]	E	✓	✓	·	·	·
[149]	E	·	✓	·	·	·
[150]	E	✓	·	·	·	·
[138]	E	·	·	·	·	·
[151]	E	·	✓	·	·	·
[146]	E	·	✓	·	·	·
[4]	E	✓	·	·	·	✓
[152]	E	✓	·	·	·	·
[153]	E	✓	·	·	·	·
[154]	E	·	✓	·	·	·
[155]	E	✓	✓	·	·	·
[156]	E	·	✓	·	·	·
[157]	E	✓	·	·	·	·
[158]	E	·	✓	✓	✓	·
[140]	E	✓	✓	·	·	·
[119]	E	✓	·	·	·	·
[132]	E	✓	·	·	·	·
[159]	G	·	·	·	·	✓
[142]	E	✓	✓	·	·	·
[139]	E	✓	·	·	·	·
[160]	G	·	✓	·	·	·
[5]	E	✓	✓	·	·	·
[161]	E	✓	·	·	·	·
[162]	E	✓	·	·	·	·
[163]	E	✓	·	·	·	·
[11]	E&G	✓	✓	·	·	✓
[164]	E	✓	✓	·	·	·
[126]	E	✓	·	·	·	·
[120]	E	✓	·	·	·	·
[118]	E	·	✓	·	·	✓
[122]	G	✓	✓	·	✓	·
[165]	E&G	·	·	·	·	✓
[166]	E	✓	·	·	·	·
[115]	G	✓	·	✓	✓	·
[167]	E	✓	·	·	·	·
[168]	E	✓	✓	·	·	·
[116]	G	✓	·	✓	✓	✓
[169]	E	✓	✓	·	·	·
[117]	G	✓	·	·	·	·
[170]	E&G	✓	·	·	✓	·
[171]	E	✓	·	·	·	·
[129]	G	·	·	✓	✓	·
[172]	E	✓	·	·	·	·
[173]	E	✓	·	·	·	·
[121]	G	·	✓	·	·	·
[174]	E	✓	·	·	·	·
[6]	G	✓	·	✓	✓	✓
[175]	E	✓	·	·	·	·
[176]	E	·	·	·	·	✓
Totals	E:42 G:12	37	19	5	7	8

Table 2.2. PU learning goals in reviewed papers using genuine PU data.

Reference	Anomaly detection	Prioritisation	Precision	Recall
[35]	·	✓	·	·
[38]	✓	·	·	·
[43]	·	✓	·	·
[10]	✓	·	·	✓
[46]	·	✓	·	·
[3]	·	✓	✓	✓
[4]	·	✓	✓	✓
[5]	·	✓	·	·
[50]	·	✓	·	✓
[52]	·	✓	✓	✓
[9]	✓	·	·	·
[56]	·	✓	✓	✓
Total	3	9	4	6

Imbalanced data

As previously stated, class imbalance is often present in PU learning datasets, given the expense or difficulty associated with identifying positive instances. Therefore, PU learning datasets commonly consist of a large number of unlabelled instances and a very small number of labelled positive instances. Handling the imbalanced data presents a challenge [177][178], but each of the three previously discussed PU learning frameworks employ strategies to do so. If utilising the two-step framework, the unlabelled set can be split into multiple subsets that create a more even distribution with the positive set. This is often referred to as undersampling in machine learning literature. In the biased learning approach, class imbalance is handled by modifying the error or loss function in the learning algorithm. This modification, often called cost-sensitive learning, adjusts the penalties associated with misclassification of instances from different classes. Specifically, it increases the penalty for misclassifying instances from the minority class (positive instances in this case). By doing so, the algorithm is pushed a higher rate of classification of the positive class, hence countering the effects of class imbalance. When incorporating the class prior, the approach directly incorporates the prior probability of the classes into the learning algorithm. This prior probability can be estimated from the dataset itself or provided based on domain knowledge. When class priors are incorporated correctly, the learning algorithm should be able to naturally handle the class imbalance. The decision threshold is adjusted based on these priors, thus compensating for the imbalance in the class distribution.

In summary, class imbalance is a significant challenge in PU learning, but it can be effectively handled using various strategies, depending on the specific learning approach adopted. These methods aim to adjust the learning process in a way that ensures the algorithm does not overlook the minority class, leading to more accurate and robust models.

Assumption violation

As previously discussed, PU learning is underpinned by various assumptions, such as negativity, separability, smoothness, Selected Completely at Random (SCAR), and Selected at Random (SAR). These assumptions guide the learning process and form the basis of the algorithm's "understanding" of the data. However, when the actual data does not conform to these assumptions, the performance of the PU learning method may be significantly impacted.

In practice, it is unlikely that the data used will adhere completely to the assumptions made. However, PU learning methods still show good predictive performance when applied. Therefore, so long as the data adheres to the assumptions made somewhat, it can be argued that the impact of areas of the data that do not adhere is relatively small.

Chapter 3

The Proposed Auto-ML Framework for Positive-Unlabelled Learning

This chapter details the proposed Automated Machine Learning (Auto-ML) framework used for Positive-Unlabelled (PU) learning throughout this work.

Recall that there are three main approaches to PU learning (outlined in Chapter 2.5), namely the two-step framework, biased learning, and methods that incorporate the class prior. The Auto-ML systems proposed in this thesis focus only on the development of two-step PU learning methods, given that this is the most popular approach.

This chapter is organised as follows. Section 3.1 presents a summary of the two-step approach for PU learning – for details, see Section 2.5. Section 3.2 specifies the search spaces and the objective function used by all Auto-ML systems proposed in this thesis. This section also gives pseudocodes with specific implementation details of Phases 1A, 1B, and 2. Section 3.2 is the core of the proposed Auto-ML framework for positive-unlabelled learning, and the search spaces and objective function specified in this section will be used by all three types of Auto-ML systems proposed in Chapters 4, 5 and 6. Section 3.3 describes the classification datasets used in this thesis' experiments, which includes both synthetic datasets and real-world biomedical datasets which were engineered for PU learning. Section 3.4 details the experimental methodology used throughout this work.

3.1 A Summary of the Two-Step Approach for PU Learning

Recall that a two-step PU learning algorithm consists of three main components, namely: Phase 1A, Phase 1B, and Phase 2 [21][22]. Phase 1A and IB curate a set of reliable negative instances from the unlabelled set, and Phase 2 builds a classifier to distinguish the labelled positive and reliable negative instances. In Phase 1A, a probabilistic classifier is trained using the labelled positive instances as the positive set and a subset of the unlabelled instances as the negative set. Typically, the unlabelled set is divided into multiple subsets, and each is used in turn as the negative set in this Phase 1A, so that this phase typically involves multiple iterations of classifier training. The number of such subsets, which is also the number of iterations training a classifier, is a user-specified parameter. In each iteration, the learned model classifies the instances in the current unlabelled subset and those instances that have a probability of belonging to the positive class of less than a given threshold are added to the set of reliable negative instances and removed from the unlabelled set. This process is then repeated for each unlabelled subset. Note that this subset count parameter can simply be 1, meaning that the entire unlabelled set would be used to learn the classifier, and only one iteration would be performed in Phase 1A. The result of Phase 1A is the initial reliable negative set, sometimes referred to as the reliable negative seed set in methods that opt to use Phase 1B.

Phase 1B is an optional phase that some methods choose to undertake to further expand the reliable negative set [5][167][179]. This phase is more akin to traditional semi-supervised classification, expanding the reliable negative set using the initial reliable negative instances as a seed set. A classifier is built to distinguish the positive and reliable negative instances, and the resulting model is used to classify the unlabelled set. Those instances classified by the model as having a predicted probability of belonging to the positive class of less than a given threshold are then added to the reliable negative set.

The final Phase 2 simply involves learning a classifier to distinguish the positive and the reliable negative set.

Each of the phases described here involve their own set of hyperparameters. These hyperparameters and the values they can take in the Auto-ML systems are defined in Section 3.2.

3.2 Search Spaces and Objective Function

In Auto-ML in general, a search space is defined as the set of all possible candidate solutions that can be found by the search algorithm, consisting of a pre-defined set of algorithms with their hyperparameters and their respective values. For our Auto-ML systems for PU learning, the search space is defined by the two-step PU learning framework. That is, a candidate solution is a two-step PU learning method, consisting of Phases 1A, 1B, and 2, as defined in Section 3.1 and discussed in detail in Section 2.5.2. Each phase has a distinct set of hyperparameters and values that these hyperparameters can take. It is these hyperparameters and values that define the search space of the proposed Auto-ML systems.

Throughout our experiments we have utilised two variations of the search space. The first, referred to as the base search space, is detailed in Section 3.2.1. The second, referred to as the extended search space, is detailed in Section 3.2.2.

3.2.1 Base Search Space

The base search space, proposed in previous work [21], allows the system to build simple two-step PU learning methods that do not utilise any heuristics for determining the values of the hyperparameters. Specifically, the search space is defined by the following 7 hyperparameters and their corresponding candidate values:

- Iteration_count_1A: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }
- Threshold_1A: { 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5 }
- Classifier_1A: { Candidate_classifiers }
- Threshold_1B: { 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5 }
- Classifier_1B: { Candidate_classifiers }
- Flag_1B: { True, False }
- Classifier_2: { Candidate_classifiers }

Where Candidate_classifiers represents 18 different candidate classification algorithms, namely: Gaussian naïve Bayes, Bernoulli naïve Bayes, random forest, decision tree, multilayer perceptron, support vector machine, stochastic gradient descent classifier, logistic regression, k-nearest

neighbour, deep forest, AdaBoost, gradient boosting classifier, linear discriminant analysis, extra tree classifier, extra trees classifier (an ensemble of extra trees), bagging classifier, Gaussian process classifier, and histogram-based gradient boosting classification tree. For an overview of these classification algorithms, see Section 2.1.

Phase 1A			Phase 1B			Phase 2
Iteration_count_1A	Threshold_1A	Classifier_1A	Threshold_1B	Classifier_1B	Flag_1B	Classifier_2

Figure 3.1. Representation of a candidate solution, with a linear encoding.

Together, these hyperparameters constitute Phase 1A, Phase 1B, and Phase 2 of the two-step PU learning framework described in Section 2.5.2 and outlined in Section 3.1. Figure 3.1 shows how these hyperparameters form these phases.

Phase 1A consists of the hyperparameters *Iteration_count_1A*, *Threshold_1A*, and *Classifiers_1A*. The iteration count determines the number of subsets to split the unlabelled set into when learning a classifier to distinguish between the positive and the unlabelled set, and also the number of iterations that a classification algorithm is run in Phase 1A. E.g., if the iteration count is 5, the unlabelled set will be split into 5 subsets, each with 20% of the unlabelled data, and the classification algorithm will be run 5 times, each using a different subset of unlabelled instances in the training set. This helps to handle the class imbalance present in many PU learning datasets. The *Threshold_1A* hyperparameter determines the predicted probability of belonging to the positive class that an instance must fall under to be considered a reliable negative instance. The *Classifier_1A* is simply the classifier used to predict the reliable negative instances.

Phase 1B consists of the hyperparameters *Threshold_1B*, *Classifier_1B*, and *Flag_1B*. *Threshold_1B* and *Classifier_1B* are analogous to those used in Phase 1A. The *Flag_1B* hyperparameter indicates whether to skip Phase 1B or not. Phase 1B is not always utilised in PU learning techniques, and therefore the Auto-ML system can generate individuals that are able to skip this phase. Given the similarities between Phase 1A and Phase 1B, a natural question arises as to why we exclude an iteration count parameter from Phase 1B. There are two main reasons for this exception. Firstly, the iteration count parameter was introduced in order to handle the class imbalance inherent to PU learning datasets. However, this is not generally an issue once an initial reliable negative set has been created as this set is simply a small subset of the unlabelled set. Furthermore, class imbalance is indirectly handled by the *Threshold_1A* parameter, which will evolve to be a

smaller value (and thus fewer instances will be added to the reliable negative set) if the reliable negative set becomes large enough to detriment predictive accuracy. Secondly, this hyperparameter increases the size of the search space and, in preliminary experiments, did not improve predictive performance.

Phase 2 simply consists of the hyperparameter *Classifier_2*. This classifier will be trained to distinguish the positive set and the reliable negative set extracted from the unlabelled set in phases 1A and potentially 1B. The size of the original search space is thus calculated as follows:

$10 \times 10 \times 18 \times 10 \times 18 \times 2 \times 18 = 11{,}664{,}000$ possible candidate solutions.

However, this calculation is an upper bound calculation of the number of candidate solutions, given the dependencies between the Phase 1B hyperparameters. That is, given that whether or not the hyperparameters *Threshold_1B, and Classifier_1B* have an impact on the candidate solution is determined by the *Flag_1B* hyperparameter.

Figure 3.2 shows an example candidate solution.

Phase 1A			Phase 1B			Phase 2
3	0.4	Random forest	0.25	Linear Discriminant Analysis	TRUE	Bernoulli Naïve Bayes

Figure 3.2. Example candidate solution for the base search space.

So, in Phase 1A of the example candidate solution shown in Figure 3.2, the unlabelled set would be split into 3 subsets (defined by the *Iteration_count_1A* hyperparameter). Each of these subsets in turn, along with the labelled positive set, would be used to train a random forest classifier (*Classifier_1A*) which would then predict the probability of the unlabelled instances in the current subset belonging to the positive class. Those instances with a predicted probability of less than 0.4 (the *Threshold_1A* parameter) would be added to the reliable negative set and removed from the unlabelled set. Then, as the *Flag_1B* parameter is set to True, a linear discriminant analysis classifier (*Classifier_1B*) would be built using the labelled positive instances as the positive set and the reliable negative instances identified in Phase 1A as the negative set. It would then be used to classify the remaining unlabelled instances, and those with a predicted probability of belonging to the positive class of less than 0.25 (the *Threshold_1B* hyperparameter) would be added to the reliable negative set. Finally, a Bernoulli naïve Bayes classifier (*Classifier_2*) would be trained on the labelled positive and the reliable negative sets.

3.2.2 Extended Search Space (Based on the Spy Technique)

The second search space, proposed in previous work [22] and referred to as the extended search space, introduces three new hyperparameters based on the spy technique for PU learning. Essentially, spy-based approaches are used to heuristically determine the *Threshold_1A* parameter. A percentage of labelled positive instances (determined by *Spy_rate*) are hidden in the unlabelled set. A classifier (*Classifier_1A*) is built, using the labelled positive instances as the positive set and the unlabelled instances with the spy instances as the negative set. The spy instances are then classified, and *Threshold_1A* is determined such that a percentage of spy instances (determined by *Spy_tolerance*) have a predicted probability of belonging to the positive class of less than *Threshold_1A* (e.g., if *Spy_tolerance* is set to 0.05, 5% of the spy instances can have a predicted probability of belonging to the positive class of less than *Threshold_1A*). Note that the *Threshold_1A* parameter defined by the candidate solution is thus redundant and its value is not used when building the PU learning model for candidate solutions with a value of True for *Spy_flag*. However, *Threshold_1A* is still needed as a component of a candidate solution during the search performed by the Auto-ML system, since some candidate solutions generated along the search will not use the spy technique (depending on the value of the *Flag_1B* hyperparameter).

Hence, the three new hyperparameters introduced into the extended search space are as follows:

- Spy_flag: { True, False }
- Spy_rate: { 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35 }
- Spy_tolerance: { 0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 }

Spy_flag is a Boolean value used to indicate whether or not to use a spy-based method in Phase 1A. *Spy_rate* determines the percentage of positive instances to use as spies. *Spy_tolerance* determines what percentage of spies can remain in the unlabelled set when the threshold is calculated. The inclusion of these three new hyperparameters increases the size of the size space to:

$10 \times 10 \times 18 \times 10 \times 18 \times 2 \times 18 \times 2 \times 7 \times 11 = 1{,}796{,}256{,}000$ candidate solutions.

The extended search space is thus 154 times larger than the original search space. Note, however, that this is also an upper bound, given that the spy hyperparameters have the same dependency considerations as the *Phase 1B* hyperparameters.

The motivation for expanding the search space was to attempt to increase predictive performance of the system by utilising spy-based methods, as initially proposed by [120]. This approach has been used frequently in the PU learning literature with success [180][127][163][181][182].

Phase 1A						Phase 1B			Phase 2
3	0.4	Random forest	TRUE	0.15	0.02	0.25	Linear Discriminant Analysis	TRUE	Bernoulli Naïve Bayes

Figure 3.3. Example candidate solution for extended search space.

Figure 3.3 shows an example candidate solution for the extended search space. As *Spy_flag* is set to True, 15% of the labelled positive instances (determined by *Spy_rate*) are hidden in the unlabelled set in Phase 1A. The RN threshold is determined as the value at which only 2% of spy instances have a predicted probability of belonging to the positive class of less than the determined value.

Spies are utilised in Phase 1A, but not in Phase 1B. This decision was made as preliminary experiments showed no increase in predictive performance when the system allowed spies in Phase 1B. Also, the search space would be greatly expanded if spies were used in Phase 1B, as the three new hyperparameters introduced in this expanded search space would all need to be repeated for Phase 1B. Thus, if spies were used in Phase 1B, the size of the search space would be:

$$10 \times 10 \times 18 \times 10 \times 18 \times 2 \times 18 \times 2 \times 7 \times 11 \times 2 \times 7 \times 11 = 276{,}623{,}424{,}000$$

Therefore, given that no increase in performance was shown in our preliminary experiments using spies in Phase 1B and the system had a much larger search space to explore (154 times larger than the extended search space, 23,716 times larger than the base search space), we opted simply for inclusion of the spy-based heuristic method in Phase 1A only.

3.2.3 Objective Function

The objective function assesses the quality of a given configuration of PU learning hyperparameter settings for a specific PU learning task, i.e., a specific input dataset. This is done by applying the PU method configuration defined by the candidate solution to the training set. To describe the process of obtaining the objective score of a candidate solution, we use the following notation:

RN: The set of reliable negative instances.
P: The set of labelled positive instances.
U: The set of unlabelled instances.

P(y=1): The probability of an instance belonging to the positive class, as calculated by the classifier.

Procedure 3.1 Objective function (*Candidate solution*, *Training set*)

1. Split *Training set* into 5 *Learning* and *Validation* sets;
2. **For** each *Learning set* and corresponding *Validation set*:
   a. *P* = all labelled positive instances in *Learning set*;
   b. *U* = all unlabelled instances in *Learning set*;
   c. **If** *Spy_flag* **then** *RN*, *U* = Phase 1A-Spies(*P*, *U*) // **call Procedure 3.2**
      else *RN*, *U* = Phase 1A(*P*, *U*); // **call Procedure 3.3**
   d. **If** *Flag_1B* **then** *RN*, *U* = Phase 1B(*P, RN*, *U*); // **call Procedure 3.4**
   e. Train *Classifier_2* (using the *Learning set*) to distinguish *P* and *RN*;
   f. Classify *Validation set*;

Output *Candidate solution's* objective score = average F-measure over the 5 *Validation sets*;

The objective score of each candidate solution is computed as specified in Procedure 3.1. The *Training set* is split into 5 folds for internal cross-validation, creating 5 pairs of *Learning* and *Validation* sets (step 1). For each pair of *Learning* and *Validation* sets, all labelled positive instances are added to *P* (step 2.a) and all unlabelled instances are added to *U* (step 2.b). The *RN* set is determined with either the Phase 1A-Spies(*P*, *U*) or Phase 1A(*P*, *U*) algorithm, depending on the *Spy_flag* parameter, which returns a refined *U* set (step 2.c, executing Procedure 3.2 or 3.3). If the flag for running Phase 1B is set to true, *RN* and *U* sets are further refined with the Phase 1B(*P, RN*, *U*) algorithm (step 2.d, executing Procedure 3.4). *Classifier_2* is then trained to distinguish *P* and *RN* (step 2.e), and then used to classify the *Validation set* (step 2.f). The objective score of the Individual is assigned as the F-measure over the 5 *Validation set* classifications (output). The F-measure is calculated by evaluating the individuals on a validation set considering all unlabelled instances as negative. That is, whilst in the test set there are positive and negative instances (i.e., no unlabelled set), in the validation set there is a positive and an unlabelled set, with the unlabelled set considered as the negative set. An alternative approach would be to have the validation set reflect the true class labels of the instances (i.e., a positive and a negative set, rather than a positive and an unlabelled set) so that the F-measure can be accurately calculated. That is, in the procedure for creating a PU dataset from a PN dataset, rather than simply changing the training set to a PU dataset, and leaving the test set as a PN dataset, we could have gone a step further and, for the learning and validation sets, converted the learning set into a PU dataset, but leave the validation set as a PN dataset, rather than having both the learning and validation sets as PU datasets, as they are created from the training set,

which is also a PU dataset. However, by evaluating the candidate solution on positive and unlabelled data, it is a truer reflection of how the individual would perform when given genuine PU data. This procedure, and the subsequent procedures 3.2-3.4, have also been presented in [22].

Procedure 3.2 Phase 1A-Spies(*P, U*)

1. *RN* = { };
2. *Sets* = split *U* into *Iteration_count_1A* subsets;
3. **For** every *Set* in *Sets:*
   - **a.** *Spies* = *Spy_rate%* instances, randomly selected from *P*; *P = P – Spies;*
   - **b.** *Set_with_spies* = *Set* ∪ *Spies*
   - **c.** Train *Classifier_1A* on *P* and *Set_with_spies;*
   - **d.** Classify all instances in *Set_with_spies;*
   - **e.** Set *threshold* to a value such that S*py_tolerance*% spies have Pr*(y=1)* less than *threshold*;
   - **f.** **For** each unlabelled *Instance* in *Set_with_spies*:
     - **i.** **If** Pr(*y=1*) < *threshold* **then** *RN = RN* ∪ *Instance*, *U = U – Instance*;

Output *RN, U*;

Procedure 3.2 describes Phase 1A of the two-phase PU learning method, executed when *Spy_flag* is True. The *RN* set is initialised empty (step 1). The set *U* of unlabelled instances is split into *Iteration_count_1A* subsets (step 2). For each *Set* in the list of subsets, *Spies* is initialised with *Spy_rate*% of instances of *P*, randomly selected and removed from *P* (step 3.a), and *Set* and *Spies* are combined to form *Set_with_spies* (step 3.b). Next, *Classifier_1A* is trained on *P* and *Set_with_spies* (step 3.c) and all instances in *Set_with_spies* are classified and the *threshold* is set so that *Spy_tolerance*% of spies have Pr*(y=1)* less than *threshold* (step 3.d-e). For each unlabelled *Instance* in *Set_with_spies* (excluding the spies), if Pr*(y=1)* is less than *threshold,* they are added to *RN* and removed from *U* (step 3.f). The resulting *RN* and *U* sets are then returned.

Procedure 3.3 Phase 1A(*P, U*)

1. *RN* = { };
2. Sets = split U into *Iteration_count_1A* subsets;
3. **For** every *Set* in *Sets:*
   - **a.** Train *Classifier_1A* on *P* and *Set*;
   - **b.** Classify all unlabelled instances in *Set;*
   - **c.** **For** each unlabelled *Instance* in *Set*:
     - **i.** **If** P(*y=1*) < *Threshold_1A* **then** *RN = RN* ∪ *Instance*, *U = U – Instance*;

Output *RN, U*;

Phase 1A of the two-phase PU learning method, executed when *Spy_flag* is False, is described in Procedure 3.3. The *RN* set is initialised as an empty set (step 1). The set *U* of unlabelled instances is

split into *Iteration_count_1A* subsets (step 2). For each *Set* in the list of subsets, *Classifier_1A* is trained to distinguish *P* and *Set* (step 3.a) and used to classify all unlabelled instances in *Set* (instances previously treated as the negative set during training) (step 3.b). For each unlabelled *Instance*, if the instance's calculated *P(y=1)* is less than *Threshold_1A* then *Instance* is added to *RN* and removed from *U* (step 3.c.i). The resulting *RN* and *U* sets are then returned.

Procedure 3.4 Phase 1B(*P, RN*, *U*)

---

1. Train *Classifier_1B* on *P, RN*;
2. Classify *U*;
3. **For** each *Instance* in *U*:
  a. **If** P(y=1) < *Threshold_1B* **then** *RN* = *RN* ∪ *Instance*, *U* = *U* – *Instance*;

Output *RN*, *U*;

Phase 1B of the two-phase learning method is described in Procedure 3.4. *Classifier_1B* is trained to distinguish the positive and reliable negative instances in *P, RN* (step 1) and the resulting classifier is then used to classify *U* (step 2). For each *Instance* in *U*, if the *Instance's* calculated *P(y=1)* is less than *Threshold_1B*, *Instance* is added to *RN* and removed from *U* (step 3). The resulting *RN* and *U* sets are returned (step 4).

As Phase 2 simply consists of building Classifier_2 from *P* and *RN*, no pseudocode is needed.

This objective-function evaluation process is used for all Auto-PU systems described in this work, which includes systems based on evolutionary algorithms, Bayesian optimisation, and a hybrid evolutionary Bayesian optimisation system. For those experiments that utilise the base search space, rather than the extended search space, Procedure 3.2 is not used as there are no spy parameters, and therefore the step 2.c of Procedure 3.1 is simplified, as follows: "RN, U = Phase 1A(P, U); // **call Procedure 3.3**".

3.3 Classification Datasets

To assess the proposed Auto-PU systems, experiments were conducted on two types of datasets, namely real-world biomedical datasets and synthetic datasets. Both types of datasets are originally binary classification datasets and therefore need to be adapted for PU learning, as discussed in Section 2.5. To do so, we have hidden δ% of the positive instances in the negative set (where δ is a

user-specified parameter), thus creating an unlabelled set. This process of engineering a PU dataset from a binary dataset is common throughout the PU learning literature [120][155][183][20]. δ takes the values 20%, 40%, and 60% throughout this work, meaning that each dataset is engineered into three datasets, thus creating 120 datasets total (20 original real-world datasets, 20 original synthetic datasets, with each original dataset being used to produce three datasets, using the aforementioned three different values of δ).

3.3.1 Real-World Biomedical Datasets

The experiments reported in this thesis used 20 publicly available biomedical datasets, including 13 classical benchmark classification datasets from the well-known UCI dataset repository [184], and 7 datasets introduced in [185-191]. These datasets all involve real-world learning scenarios in the task of disease or health-risk prediction. The main characteristics of these datasets are shown in Table 3.1. All these 20 datasets are originally binary-class datasets (with positive and negative instances), and they were engineered for PU learning as described earlier.

Table 3.1. Main characteristics of the biomedical datasets used in the experiments.

Dataset	No. instances	No. features	%Pos
Alzheimer's [185]	354	9	10.73
Autism [184]	288	15	48.26
Breast cancer Coimbra [184]	116	9	55.17
Breast cancer Wisconsin [184]	569	30	37.26
Breast cancer mutations [186]	1416	53	32.42
Cervical cancer [184]	668	30	2.54
Cirrhosis [187]	277	17	25.72
Dermatology [184]	359	34	13.41
Pima Indians Diabetes [184]	769	8	34.90
Early Stage Diabetes [188]	521	17	61.54
Heart Disease [184]	304	13	54.46
Heart Failure [189]	300	12	32.11
Hepatitis C [184]	590	13	9.51
Kidney Disease [184]	159	24	27.22
Liver Disease [184]	580	11	71.50
Maternal Risk [184]	1014	6	26.82
Parkinsons [184]	196	22	75.38
Parkinsons Biomarkers [190]	131	29	23.08
Spine [184]	311	6	48.39
Stroke [191]	3427	15	5.25

Biomedical datasets are good candidates for PU learning given the inherent uncertainty involved in labelling biomedical data. For example, consider a learning task that involves assessing a person's

risk of developing a given disease (as is the scenario for many of these datasets). A classifier may learn to distinguish between a positive set, consisting of data from patients who have been diagnosed with a specific disease, and a negative set, consisting of data from patients who have not been diagnosed with a specific disease. From this data, we wish to identify whether an unseen patient has a specific disease. However, consider the wording of this scenario. We are looking to identify whether a patient *has* a disease, by learning from data consisting of patients who have or have not been *diagnosed* with a disease. In other words, we are looking to identify true positives by learning only from labelled positives. The negative set, in this scenario, can be more precisely considered an unlabelled set, given that "not diagnosed" does not mean that a patient does not have a disease. It might simply be that this patient has not undergone any tests to determine whether the disease is present. Or, this patient may have undergone some tests, but the tests may not be wholly accurate, or the disease may be undetectable with the given test. For examples of studies detailing the reliability of specific diagnostic tests see [192-197]. Furthermore, biomedical tests are expensive, and thus the presence of unlabelled data may simply be a practicality to minimise the cost of data curation. Thus, we have used biomedical datasets in our experiments as they are appropriate for PU learning and have been referred to as "one of the most significant usage areas in PU learning" [198]. For examples of PU learning applications to biomedical datasets, see [3,115,199,200,201].

3.3.2 Synthetic Datasets

The second type of dataset used in the experiments reported in this thesis were synthetic datasets, which have been computationally generated using sklearn's make_classification method [31]. 20 datasets were created, using the following parameter settings of the make_classification method:

- Number of samples: 500 to 2,000
- Number of features: 50 to 200
- Number of informative features: 2 to (number of features / 2)
- Number of redundant features: 0 to (number of features – number of informative features)
- Number of clusters per class: 1 to 10
- Percentage of instances belonging to positive class: 1 – 50%

All other parameters were kept at their default. The exact characteristics of each dataset can be found in Table 3.2. Samples is the number of instances in the dataset, features is the number of features (or attributes) of an instance, informative is the number of informative (relevant) features, redundant is the number of redundant features, clusters is the number of clusters per class, and Pos is the percentage of instances in the dataset that are positive. Note that the %Pos columns in both Tables 3.1 and 3.2 show the percentage of positive instances before δ% are hidden in the unlabelled set.

Testing on synthetic datasets is also used in the PU learning literature [154,156] and allows us to evaluate our systems on datasets with a variety of characteristics.

Table 3.2. Main characteristics of the synthetic datasets used in the experiments.

Dataset	Samples	Features	Informative	Redundant	Clusters	%Pos
1	1209	167	12	74	5	15.76
2	1366	147	7	57	7	29.31
3	944	192	83	65	2	12.78
4	1799	97	29	10	3	21.57
5	1156	148	56	59	5	19.27
6	1489	113	54	1	2	25.30
7	1365	95	8	76	1	12.98
8	761	169	16	67	4	32.02
9	1258	100	17	2	3	11.56
10	1428	79	12	36	3	26.83
11	1903	58	27	25	5	38.02
12	1969	56	10	38	9	23.93
13	1502	73	7	15	4	24.45
14	1342	78	35	31	3	1.64
15	898	98	41	42	5	43.48
16	1132	75	25	8	1	35.64
17	976	82	39	24	8	6.45
18	640	116	15	55	8	23.35
19	1271	56	8	7	9	32.04
20	516	72	19	47	8	22.45

3.4 Experimental Methodology

3.4.1 Cross-Validation

Throughout this work, the experiments use a nested cross-validation procedure, with an external cross-validation used to measure predictive performance (generalisation ability) and an internal cross-validation used to evaluate candidate solutions during a run of an Auto-PU system.

For the external cross-validation, the experiments use the well-known stratified 5-fold cross-validation procedure. This involves randomly splitting the data into 5 folds and using those folds as training and test sets. Training sets are created by combining 4 of the 5 folds, and the test set is the remaining fold. This process is repeated for all folds, so each is used as the test set exactly once. The cross-validation is stratified in the sense that in each of the 5 folds the distribution of class labels is approximately the same as the distribution in the full dataset. We chose 5 folds, rather than the more popular 10 folds, as the number of positive instances in some of our classification datasets are small. Thus, in some datasets 10 folds would split the positive set into folds that are so small as to be practically unsuitable. As we utilise the stratified cross-validation procedure, the folds each have roughly the same number of positive instances.

Inside the 5-fold external cross-validation, the Auto-PU system runs a 5-fold internal cross-validation procedure. This involves splitting the training set into 5 pairs of learning and validation sets. The model is then built using the learning set and tested on the validation set. This process is repeated for each pair of learning and validation sets, with the performance of the model averaged over the 5 sets. Performing this internal cross-validation procedure helps to prevent overfitting, by having the model performance generalised over multiple subsets of the training data.

For each version of the Auto-PU system, for each training set, we run the system to evolve the best candidate solution that it is able to find within the search space. During the search performed by the Auto-PU system, each candidate solution is assessed on the 5 pairs of learning and validation sets, and the performance of the candidate solution is determined by the average F-measure value achieved over the 5 validation sets. When the Auto-PU system's search ends, the best candidate solution returned by the system is thus the one with the highest average F-measure calculated over the 5 validation sets inside the training set. Then, a PU learning classifier is built from the training set with the configuration defined by that best candidate solution. The classifier is then used to predict the class of all instances in the test set. This process is repeated for the 5 pairs of training and test sets in the 5-fold cross-validation.

We report precision, recall, and F-measure as the evaluation metrics for comparison. F-measure is the most relevant measure in the experiments reported in this thesis, since it is the measure being optimised by the Auto-PU systems, but the separate values of precision and recall are also important

metrics for determining the suitability of a PU learning method for a given learning task [20]. For details of precision, recall, and F-measure, see Section 2.1. When comparing a version of the Auto-PU system against another version or method, all systems/methods tested use the same 5-fold cross-validation procedure, with the same folds, to ensure a fair comparison.

3.4.2 Statistical Significance Analysis

Regarding statistical analysis of the computational results, for each performance measure (F-measure, recall, and precision), we compare the performance of the system tested against the performance of the other methods using the non-parametric Wilcoxon Signed-Rank test [202]. Since this involved testing multiple null hypotheses, we use the well-known Holm correction [203] for multiple hypothesis testing. This procedure involves comparing the best method against each of the other methods, ranking the p-values from the smallest to largest (i.e., from most to least significant), and adjusting the significance level α according to the p-values' ranking. We set α = 0.05 as usual before adjusting it according to the position of the p-value in the ranked list. p_1 (the smallest p-value) is deemed significant if less than $\frac{\alpha}{n}$, where n is the number of hypotheses tested, which is the number of methods tested minus 1. For example, n = 2 for 3 methods, since the best method is compared against each of the other two methods, and so 2 hypothesis are tested. If this condition is not satisfied, the procedure stops and all p values are deemed non-significant. If p_1 is deemed significant, p_2 is deemed significant if less than $\frac{\alpha}{n-1}$, etc.

3.4.3 Correlation Coefficient Analysis

In this work, the correlation between a hyperparameter's values and a dataset's characteristic is analysed using the Pearson's linear correlation coefficient. This correlation coefficient, denoted as r in Equation 3.1, measures the strength and direction of the linear relationship between two variables. It can have a value between -1 and 1, where -1 indicates a perfectly negative linear correlation, 1 indicates a perfectly positive linear correlation, and 0 signifies no linear correlation. The closer the coefficient is to either -1 or 1, the stronger the correlation between the variables. The formula for the correlation coefficient is given in Equation 3.1, as defined in [204].

$$r = \frac{\sum z_x z_y}{n - 1} \qquad (3.1)$$

Where z_x and z_y are the z-scores of the two variables being analysed for their correlation, and n is the number of observed instances. For an explanation of z-scores, see [204].

For the purposes of this work, we use the categorisation of correlation coefficient values as defined in [205]. The converse of these values apply with the same categorisations for the negative correlations. The categorisations are as follows:

- 0.00 – 0.09: Negligible correlation
- 0.10 – 0.39: Weak correlation
- 0.40 – 0.69: Moderate correlation
- 0.70 – 0.89: Strong correlation
- 0.9 – 1.00: Very strong correlation

Chapter 4

A Genetic Algorithm-based Auto-ML System for Positive Unlabelled Learning (GA-Auto-PU)

GA-Auto-PU is a Genetic Algorithm (GA)-based Automated Machine Learning (Auto-ML) system for Positive-Unlabelled (PU) learning. Before proposing this system in previous work [21], no Auto-ML system for PU learning existed in the literature. The value of Auto-ML systems was discussed in Section 2.4, but to briefly summarise, the performance of any machine learning algorithm is largely dependent on the input data. Thus, constructing an algorithm customised to the input data, from a set of algorithmic components, is a valuable approach for any machine learning research area; and it is particularly important in PU learning, given the lack of Auto-ML systems in this area.

In this chapter we evaluate the performance of the GA-Auto-PU system against TPOT (see Section 2.4), an Auto-ML system for standard binary classification, and against two baseline PU learning methods (see Section 2.5). For evaluation we test on two types of datasets, real-world biomedical datasets, and synthetic datasets, each with three different values of δ (20%, 40%, 60%), indicating the percentage of positive instances hidden in the negative class to create an unlabelled dataset. For details of both types of datasets, see Section 3.3.

This chapter first gives a detailed description of the GA-Auto-PU system (Section 4.1), including details of its main procedure and its hyperparameters. Then, the experimental setup is detailed in Section 4.2, with a description of the experimental datasets, the nested cross-validation procedure,

the statistical significance testing, and the structure of the results sections. Next, the results are presented (Section 4.3), firstly the GA-Auto-PU with the base search space, and then for the GA-Auto-PU version with the extended search space. For details of the difference between the base and search spaces, see Section 3.2. For each of these two GA-Auto-PU versions, the reported results compare GA-Auto-PU against TPOT and two baseline PU learning methods. Next, the algorithmic components most frequently selected by GA-Auto-PU in the experiments are discussed and analysed (Section 4.4), before summarising this chapter (Section 4.5).

4.1 Description of GA-Auto-PU

As mentioned in the previously, GA-Auto-PU is a Genetic Algorithm (GA)-based Automated Machine Learning (Auto-ML) system specific to PU learning. This section describes the pseudocodes detailing the procedure followed by the GA-Auto-PU system. Recall that a GA iteratively evolves a population of individuals, where each individual represents a candidate solution, and the quality of an individual is evaluated by a fitness (objective) function. In GA-Auto-PU, in essence, an individual represents a PU learning algorithm configuration. This is encoded as a list of genes, where each gene represents the value chosen for a hyperparameter of a PU learning algorithm. Details of the individual representation (encoding), as well the fitness (objective) function used by GA-Auto-PU can be found in Chapter 3. Details of evolutionary computing concepts such as selection, crossover, mutation, and elitism can be found in Section 2.2.

4.1.1 The GA Procedure

Procedure 4.1 outlines the procedure that the GA follows to evolve a PU learning algorithm configuration. Initially, a *Population* of *Pop_size* individuals (candidate solutions) is randomly generated (step 1). This random generation involves, for each gene, randomly selecting a value from the list of candidate values of that specific gene. The probability of a specific gene value being ramdomly selected is proportional to the number of candidate solutions. For example, for the hyperparameter (gene) *Iteration_count_1A* there are 10 candidate values, being 1 to 10. The chance of the number 1 being randomly selected as the value for *Iteration_count_1A* is 10%.

Phase 1A			Phase 1B			Phase 2
2	0.45	Random forest	0.25	Gaussian NB	FALSE	Logistic regression

Figure 4.1. Example of a randomly generated individual in GA-Auto-PU.

An example of a randomly generated individual could have the values shown in Figure 4.1. As described in Section 3.2, the 7 gene values in Figure 4.1 represent the values of the following PU learning algorithm hyperparameters, respectively: (a) *Phase_1A_Iteration_Count*, (b) *Phase_1A_RN_Threshold*, (c) *Phase_1A_Classifier*, (d) *Phase_1B_RN_Threshold*, (e) *Phase_1B_Classifier*, (f) *Phase_1B_Flag*, (g) *Phase_2_Classifier*.

Procedure 4.1 Outline of the GA Procedure

1. *Population* = Generate population();
2. **Repeat** #*generations* times:
   - **a. For** each *Individual* in *Population*:
     - **i. If** *Individual* configuration has not already been assessed, **then** assess fitness(*Individual, Training set*); // see Procedures 3.1-3.4, Chapter 3.
     - **ii. Else** *Individual*'s *Fitness* values are assigned as the output of the previous assessment;
   - **b.** *Fittest_individual* = Get fittest individual(*Population*);
   - **c.** *New_pop* = Select individuals from *Population* using tournament selection;
   - **d.** *New_pop*'s individuals undergo crossover with probability *cross_prob*;
   - **e.** *New_pop*'s individuals undergo mutation with probability *mutat_prob*;
   - **f.** *Population* = *New_Pop* ∪ *Fittest_individual*;

Return Best Individual in Population

At each generation, for each *Individual*, its configuration (genome) is checked against a list of previously assessed configurations, and if it has not already been assessed, the *Fitness* of *Individual* is calculated (step 2.a.i), as described in detail in Section 3.2.3, including Procedures 3.1-3.4. To recap briefly, this fitness calculation is conducted by running 5-fold cross-validation over the training set (without using the test set). That is, the two-step PU learning procedure is executed with the hyperparameter values encoded in the individual 5 times, each time with 4/5 of the training set used as a "learning set" (to learn a PU model) and with 1/5 of the training set used as a "validation set" (to measure the predictive performance of the learned model); and then the individual's fitness value is set as the average F-measure achieved over these 5 validation sets.

If the configuration has already been assessed, the fitness values of the previous assessment are assigned to *Individual* (step 2.a.ii). This saves unnecessary execution time, meaning that whilst, hypothetically, the system could be assessing 100 unique PU learning algorithm configurations at

each generation, this is a worst-case scenario in regard to computational efficiency, and in practice the system is likely assessing fewer configurations, simply because some configurations have already been assessed.

Once all individuals have been evaluated, the fittest *Individual* is saved for the following generation (step 2.b). This is elitism, which was covered in Section 2.2; but to explain briefly, the best individual of each generation is passed without any modification (i.e., without undergoing crossover or mutation) to the next generation. This is to ensure that the potentially highest quality individual is not lost as the generations progress, and to help maintain a high quality population.

Population then undergoes tournament selection (step 2.c), uniform crossover (step 2.d), and mutation (step 2.e). These three steps are evolutionary operations that were described in detail in Section 2.2 but will be briefly described here for the reader's reference. In tournament selection, a fixed number of individuals, determined by the tournament size, are randomly sampled from the population, and the individual with the highest fitness among the sampled ones is selected for potentially undergoing uniform crossover. In GA-Auto-PU, a tournament size of 2 is used.

Uniform crossover is a process whereby new (child) individuals are created by swapping genes from selected (parent) individuals. For each pair of individuals selected by tournament selection (where selected individuals are randomly assigned to pairs), the chance of that pair of individuals undergoing uniform crossover is determined by the crossover probability (90% in GA-Auto-PU). If the two individuals do not undergo uniform crossover, they skip this evolutionary stage. GA-Auto-PU uses 2 selected parents to produce 2 children. The 2 children begin as clones (copies) of the parents, with child 1 as a clone of parent 1 and child 2 as a clone of parent 2. Then, for each gene, a random number is generated which, if lower than a predefined value referred to as the gene crossover probability (0.5 in GA-Auto-PU), the values of that gene for the two children are swapped. An example of uniform crossover in GA-Auto-PU is given in Figure 4.2.

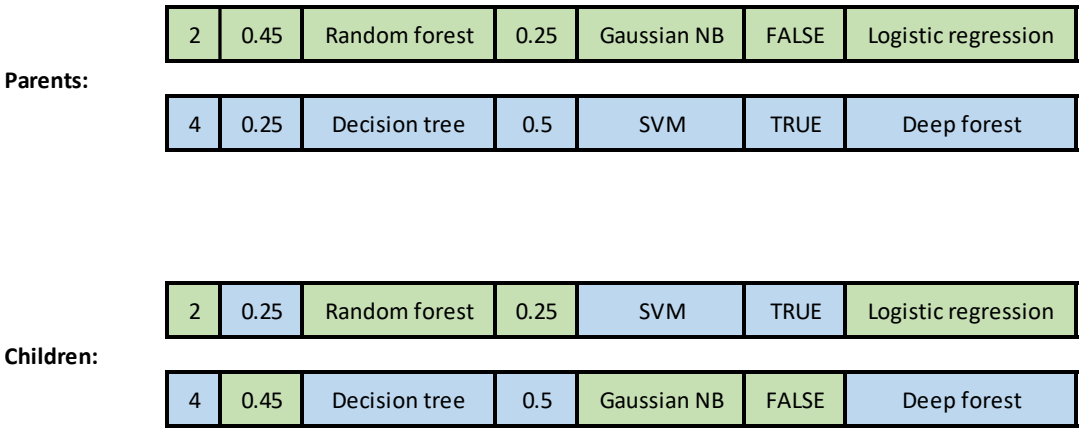


Figure 4.2. Example of uniform crossover in GA-Auto-PU.

After potentially undergoing crossover, individuals can potentially undergo mutation. This involves altering a gene in order to introduce further genetic diversity to the population. Mutation works as follows for each gene. For the *Iteration_count_1A* hyperparameter, as this gene takes an integer value, its value is mutated by adding or subtracting 1 from the current value, within the bounds specified by the candidate values (1 – 10). Whether the value is added or subtracted is a random choice, with both actions having a 50% probability of occurring. For the two *threshold* hyperparameters (*Phase_1A_RN_Threshold* and *Phase_1B_Threshold*), as these can take values from 0.05 to 0.5 in increments of 0.05, a value of 0.05 is added or subtracted to the current value, within the specified bounds. For the *Flag_1B* hyperparameter, the mutation is a simple bit flip, changing to *false* if the value is *true*, and vice versa. For the three *Classifier* hyperparameters (*Classifier_1A*, *Classifier_1B*, *Classifier_2*), as these are categorical, a new value is randomly selected from the candidate classifier names. For each gene, the probability of undergoing mutation is determined by the mutation probability, set to 10% in GA-Auto-PU. An example of the effect of the mutation operator is shown in Figure 4.3, where the gene encoding the *Flag_1B* hyperparameter has undergone mutation.

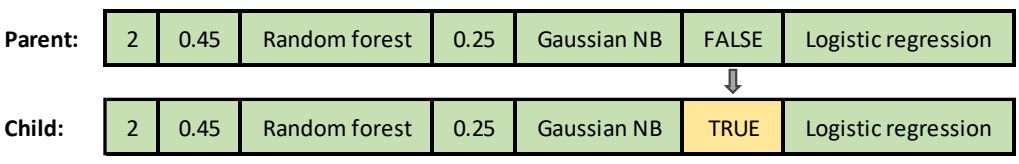


Figure 4.3. Example of mutation in GA-Auto-PU.

After undergoing the operations of selection, crossover and mutation, the evolved individuals are added to *Population* (step 2.f in Procedure 4.1).

Finally, the fittest *Individual* is re-added to *Population* (elitism) (also in step 2.f). This process of fitness calculation, selection, crossover, mutation, and elitism is repeated *#generations* times. The fitness of an individual is assigned as the F-measure achieved over the 5 folds of the cross-validation procedure applied to the training set (see Section 3.2.3).

4.1.2 The GA's Hyperparameters

Table 4.1 shows the default hyperparameter settings of the GA underlying GA-Auto-PU. The *#generations* parameter determines the number of generations to evolve the population. *Pop_size* determines the number of individuals in the population. *Cross_prob* is the probability that two individuals will undergo uniform crossover. *Gene_cross_prob* is the probability that each specific gene will be swapped when two individuals undergo uniform crossover. *Mutat_prob* is the probability that each gene of an individual will undergo mutation. *Tournament_size* is the number of individuals randomly sampled for tournament selection.

Table 4.1. Hyperparameters of the GA-Auto-PU system, with their values used in this thesis' experiments.

Hyperparameter	Value
#generations	50
Pop_size	101
Cross_prob	0.9
Gene_cross_prob	0.5
Mutation_prob	0.1
Tournament_size	2

4.2 Experimental Setup

The experimental procedure is explained in detail in Chapter 3. However, to briefly recap, two types of datasets are used in these experiments (biomedical and synthetic), each with 3 versions (varying the % of positive instances hidden in the unlabelled set), thus creating 120 datasets total.

A nested cross-validation procedure is used, with a simple 5-fold cross-validation procedure as the external layer. The internal layer splits the training set into 5 learning and validation sets, which is used to evaluate the candidate solutions.

To compare the performance of the methods, we use the Wilcoxon signed rank test [202], with Holm correction for testing multiple hypotheses [203]..

4.2.1 Structure of the Results' Sections

In the next section, we present experimental results evaluating two versions of the GA-Auto-PU system, implemented with the two search spaces described in Chapter 3. Firstly, GA-Auto-PU is compared against TPOT. Secondly, GA-Auto-PU is compared against the two PU learning baselines. Experiments were conducted on both the real-world biomedical datasets and the synthetic datasets, for three values of δ (the percentage of positives hidden in the unlabelled set): 20%, 40%, and 60%. Each section will report the F-measure results in full and will provide a summary of the precision and recall results. The full precision and recall results (for each dataset) can be found in the Appendix. For the sake of brevity, the GA-Auto-PU system utilising the base search space will be referred to as GA-1, whilst the system utilising the extended search space (which includes the Spy technique of PU learning) will be referred to as GA-2.

4.3 Results for GA-Auto-PU

4.3.1 Results comparing GA-Auto-PU with TPOT

In this section, results for GA-Auto-PU are given and compared to TPOT, beginning with a comparison of GA-1 (using the base search space) and TPOT on the biomedical datasets, as shown in Table 4.2. Recall that TPOT was designed for standard classification, rather than PU learning, so this comparison is unnatural, but it is still justifiable, given that PU learning datasets are often treated as standard binary datasets as discussed in Section 2.5, and that no Auto-ML system for PU learning existed before GA-Auto-PU. By comparing with a state-of-the-art Auto-ML system for standard binary classification, if GA-Auto-PU substantially outperforms TPOT, it can be argued that this shows the benefits of using an Auto-ML system for PU learning, rather than a standard binary classification Auto-ML system, and thus show the limits of the assumption of negativity (see Section 2.5).

Although TPOT was not designed for PU learning, it can still be applied to a PU learning dataset by simply treating all unlabelled instances as negative instances, and then learning a model to discriminate between positive and 'negative' (in reality unlabelled) instances. In this case TPOT will make no attempt to identify 'reliable negatives', it will implicitly consider all unlabelled instances as 'reliable negatives'. As a result, intuitively, TPOT is expected to achieve a smaller predictive accuracy than a proper Auto-ML system for PU learning, which first learns to identify the reliable negatives among the unlabelled instances and then uses only the reliable negatives for learning the final classification model. Therefore, the comparison of GA-Auto-PU with TPOT is, of course, unfair for TPOT, as it is not designed for PU learning. However, this comparison serves a purpose by showing the improvement on predictive performance that can be achieved by using an Auto-ML specific to PU learning for PU learning datasets, rather than using the naïve approach of treating such datasets as standard binary-classification (positive-negative) datasets and simply applying a standard Auto-ML system to such datasets. In addition, note that the truly positive instances hidden in the unlabelled instance set will be effectively acting as 'noisy data' for TPOT (since TPOT will treat all unlabelled instances as negative instances), and TPOT has been shown to outperform other Auto-ML systems on datasets with noise [206], further suiting it for comparison with the GA-Auto-PU system.

For a fair comparison, TPOT is evaluated using the same nested cross-validation procedure used to evaluate GA-Auto-PU. In addition, GA-Auto-PU uses the default hyper-parameter settings – reported in Table 4.1, whilst TPOT uses the default settings (reported in [100]) with the exception of number of candidate solutions and number of generations, which are set to 101 and 50 respectively to match those of the GA systems. Also, TPOT tunes for accuracy as default, but we have changed this to F-measure for a fair comparison with GA-Auto-PU.

Table 4.2. F-measure results of GA-1 and TPOT on real-world biomedical datasets.

	δ = 20%		δ = 40%		δ = 60%	
Dataset	**GA-1**	**TPOT**	**GA-1**	**TPOT**	**GA-1**	**TPOT**
Alzheimer's	0.529	0.531	0.551	0.400	0.456	0.313
Autism	0.960	0.964	0.927	0.956	0.910	0.896
Breast cancer Coi.	0.705	0.559	0.687	0.586	0.510	0.466
Breast cancer Wis.	0.954	0.946	0.932	0.915	0.906	0.673
Breast cancer mut.	0.893	0.891	0.868	0.890	0.854	0.833
Cervical cancer	0.828	0.733	0.903	0.000	0.714	0.000
Cirrhosis	0.573	0.494	0.464	0.466	0.443	0.356
Dermatology	0.860	0.776	0.780	0.761	0.828	0.698
PI Diabetes	0.677	0.661	0.649	0.548	0.606	0.575
ES Diabetes	0.958	0.950	0.895	0.850	0.930	0.821
Heart Disease	0.843	0.818	0.801	0.806	0.785	0.784
Heart Failure	0.770	0.660	0.652	0.615	0.674	0.563
Hepatitis C	0.953	0.865	0.771	0.804	0.588	0.458
Kidney Disease	0.976	0.988	0.988	0.687	0.754	0.667
Liver Disease	0.834	0.726	0.803	0.446	0.804	0.628
Maternal Risk	0.476	0.838	0.812	0.766	0.735	0.649
Parkinsons	0.860	0.906	0.836	0.664	0.818	0.628
Parkinsons Biom.	0.476	0.237	0.265	0.192	0.233	0.111
Spine	0.652	0.963	0.907	0.877	0.818	0.728
Stroke	0.474	0.218	0.255	0.203	0.255	0.164

Table 4.3 summarises the statistical significance of the results from Table 4.2 (for biomedical datasets), as well as the results for precision and recall. In Table 4.3, for each combination of a performance measure (F-measure, precision, recall) and a δ value (δ= 20%, 40%, 60%), the table reports the average (Avg.) rank of GA-1 vs TPOT (GA-1 is the left rank, TPOT is the right one) and the corresponding p-value. The better (lower) avg. rank in each cell is shown in boldface, and significant p-values (smaller than α) are also shown in boldface. For example, in the cell for F-measure, δ = 20%, the average ranks for GA-1 is 1.3 and TPOT is 1.7. Hence, GA-1 was the winner, but the p-value (0.09) was greater than the significant level α (0.05), so this result was not statistically significant. The following discussion of results will focus mainly on the F-measure, the most important measure in Table 4.3, whilst precision and recall results are reported for completeness.

Table 4.3. Results of Wilcoxon signed-rank tests when comparing GA-1 against TPOT regarding F-measure, Precision and Recall, for the 3 δ values on the biomedical datasets.

δ (%)	**F-measure**		**Precision**		**Recall**	
	Avg. ranks	**p-value**	**Avg. ranks**	**p-value**	**Avg. ranks**	**p-value**
20%	**1.3** vs 1.7	0.09	1.6 vs **1.4**	0.913	**1.1** vs 1.9	**0.0002**
40%	**1.25** vs 1.75	**0.001**	**1.4** vs 1.6	0.202	**1.2** vs 1.8	**0.0007**
60%	**1.0** vs 2.0	**0.00002**	**1.02** vs 1.98	**0.0001**	**1.22** vs 1.78	**0.003**

The results in Table 4.3 show GA-1 outperforming TPOT for F-measure and recall, with statistical significance in 5 out of the 6 cases. For precision, TPOT performs best for δ=20% but does not

achieve statistical significance. GA-1 performs best for δ = 40% and 60%, achieving statistical significance for 60%. Regarding recall, GA-1 outperforms TPOT in all cases, achieving statistically significantly better performance for all values of δ.

In summary, on the biomedical datasets, GA-1 consistently outperforms TPOT for F-measure and recall, whilst TPOT performs slightly better for precision.

Moving next to a comparison of GA-2 (with the extended search space) and TPOT on the biomedical datasets, Table 4.4 presents the results of both systems. Note that the TPOT results in these tables are the same as those given in Table 4.2 showing results for GA-1 (with the base search space), but these results are included in Tables 4.4 for the reader's reference.

Table 4.4. F-measure results of GA-2 and TPOT on real-world biomedical datasets.

	δ = 20%		**δ = 40%**		**δ = 60%**	
Dataset	**GA-2**	**TPOT**	**GA-2**	**TPOT**	**GA-2**	**TPOT**
Alzheimer's	0.548	0.531	0.576	0.400	0.529	0.313
Autism	0.982	0.964	0.940	0.956	0.927	0.896
Breast cancer Coi.	0.711	0.559	0.671	0.586	0.553	0.466
Breast cancer Wis.	0.956	0.946	0.936	0.915	0.866	0.673
Breast cancer mut.	0.896	0.891	0.739	0.890	0.872	0.833
Cervical cancer	0.867	0.733	0.839	0.000	0.350	0.000
Cirrhosis	0.446	0.494	0.397	0.466	0.204	0.356
Dermatology	0.901	0.776	0.896	0.761	0.692	0.698
PI Diabetes	0.642	0.661	0.646	0.548	0.634	0.575
ES Diabetes	0.978	0.950	0.887	0.850	0.894	0.821
Heart Disease	0.836	0.818	0.780	0.806	0.786	0.784
Heart Failure	0.751	0.660	0.670	0.615	0.671	0.563
Hepatitis C	0.944	0.865	0.863	0.804	0.610	0.458
Kidney Disease	0.925	0.988	0.951	0.687	0.806	0.667
Liver Disease	0.831	0.726	0.817	0.446	0.748	0.628
Maternal Risk	0.862	0.838	0.813	0.766	0.738	0.649
Parkinsons	0.935	0.906	0.843	0.664	0.792	0.628
Parkinsons Biom.	0.282	0.237	0.259	0.192	0.280	0.111
Spine	0.923	0.963	0.917	0.877	0.761	0.728
Stroke	0.241	0.218	0.239	0.203	0.243	0.164

Table 4.5 details the statistical significance of the F-measure results shown in Table 4.4 and summarises the results for precision and recall. These results are an improvement on the results for GA-1 (shown in Table 4.3), with GA-2 performing best for all metrics across all values of δ and achieving statistical significance in 7 out of the 9 cases.

Table 4.5. Results of Wilcoxon signed-rank tests when comparing GA-2 against TPOT regarding F-measure, Precision and Recall, for the 3 δ values on the biomedical datasets.

δ (%)	F-measure		Precision		Recall	
	Avg. ranks	**p-value**	**Avg. ranks**	**p-value**	**Avg. ranks**	**p-value**
20%	**1.2** vs 1.8	**0.021**	**1.45** vs 1.55	0.396	**1.25** vs 1.75	**0.015**
40%	**1.2** vs 1.8	**0.004**	**1.4** vs 1.6	0.154	**1.2** vs 1.8	**0.0005**
60%	**1.1** vs 1.9	**0.0004**	**1.0** vs 2.0	**0.000002**	**1.25** vs 1.75	**0.033**

Looking now to a comparison of the systems on the synthetic datasets, Table 4.6 compares GA-1 and TPOT.

Table 4.6. F-measure results of GA-1 and TPOT on synthetic datasets.

	δ = 20%		**δ = 40%**		**δ = 60%**	
Dataset	**GA-1**	**TPOT**	**GA-1**	**TPOT**	**GA-1**	**TPOT**
1	0.661	0.524	0.718	0.419	0.603	0.369
2	0.136	0.065	0.044	0.033	0.065	0.017
3	0.788	0.760	0.693	0.585	0.637	0.576
4	0.831	0.781	0.818	0.571	0.674	0.449
5	0.618	0.563	0.616	0.446	0.609	0.365
6	0.759	0.804	0.769	0.681	0.684	0.619
7	0.520	0.536	0.515	0.315	0.478	0.309
8	0.525	0.436	0.477	0.509	0.381	0.308
9	0.111	0.037	0.080	0.000	0.146	0.000
10	0.903	0.843	0.872	0.643	0.742	0.637
11	0.604	0.525	0.567	0.450	0.531	0.436
12	0.674	0.700	0.666	0.642	0.609	0.626
13	0.644	0.596	0.623	0.565	0.516	0.508
14	0.975	0.936	0.962	0.903	0.925	0.806
15	0.601	0.411	0.593	0.329	0.519	0.324
16	0.477	0.505	0.388	0.356	0.301	0.145
17	0.347	0.267	0.496	0.231	0.412	0.270
18	0.559	0.242	0.389	0.186	0.326	0.163
19	0.472	0.412	0.468	0.279	0.381	0.230
20	0.705	0.613	0.692	0.551	0.625	0.502

The results in Table 4.7 show the superiority of GA-1 over TPOT for F-measure and recall, achieving statistically significantly better performance than TPOT across all values of δ. For precision, TPOT performs best and achieves statistical significance when δ=20%, whilst GA-1 achieves best performance (although not with statistical significance) when δ = 40% and 60%.

Table 4.7. Results of Wilcoxon signed-rank tests when comparing GA-1 against TPOT regarding F-measure, Precision and Recall, for the 3 δ values on the synthetic datasets.

δ (%)	**F-measure**		**Precision**		**Recall**	
	Avg. ranks	**p-value**	**Avg. ranks**	**p-value**	**Avg. ranks**	**p-value**
20%	**1.2** vs 1.8	**0.0002**	1.7 vs **1.3**	**0.007**	**1.1** vs 1.9	**0.0002**
40%	**1.05** vs 1.95	**0.00001**	**1.45** vs 1.55	0.841	**1.1** vs 1.9	**0.0002**
60%	**1.05** vs 1.95	**0.00001**	**1.45** vs 1.55	0.368	**1.2** vs 1.8	**0.0003**

Table 4.8. F-measure results of GA-2 and TPOT on synthetic datasets.

	δ = 20%		δ = 40%		δ = 60%	
Dataset	**GA-2**	**TPOT**	**GA-2**	**TPOT**	**GA-2**	**TPOT**
1	0.640	0.524	0.709	0.419	0.545	0.369
2	0.176	0.065	0.105	0.033	0.111	0.017
3	0.759	0.760	0.702	0.585	0.612	0.576
4	0.824	0.781	0.809	0.571	0.692	0.449
5	0.612	0.563	0.571	0.446	0.559	0.365
6	0.762	0.804	0.751	0.681	0.672	0.619
7	0.528	0.536	0.496	0.315	0.448	0.309
8	0.571	0.436	0.484	0.509	0.390	0.308
9	0.098	0.037	0.000	0.000	0.143	0.000
10	0.896	0.843	0.850	0.643	0.716	0.637
11	0.574	0.525	0.579	0.450	0.525	0.436
12	0.681	0.700	0.692	0.642	0.599	0.626
13	0.648	0.596	0.612	0.565	0.576	0.508
14	0.977	0.936	0.966	0.903	0.934	0.806
15	0.595	0.411	0.575	0.329	0.565	0.324
16	0.431	0.505	0.402	0.356	0.299	0.145
17	0.384	0.267	0.470	0.231	0.382	0.270
18	0.576	0.242	0.408	0.186	0.373	0.163
19	0.462	0.412	0.483	0.279	0.385	0.230
20	0.701	0.613	0.664	0.551	0.594	0.502

Table 4.8 details the results of GA-2 and TPOT on the synthetic datasets; whilst Table 4.9 details the statistical significance of the F-measure results shown in Table 4.8 and summarises the results for precision and recall. These results largely reflect the results for GA-1 when compared with TPOT (shown in Table 4.7). However, GA-2 performs slightly worse in regard to the average ranks for precision and fails to achieve statistical significance in any case regarding precision. However, statistically significantly better performance against TPOT is achieved by GA-2 for F-measure and recall for all values of δ.

Table 4.9. Results of Wilcoxon signed-rank tests when comparing GA-2 against TPOT regarding F-measure, Precision and Recall, for the 3 δ values on the synthetic datasets.

δ (%)	**F-measure**		**Precision**		**Recall**	
	Avg. ranks	**p-value**	**Avg. ranks**	**p-value**	**Avg. ranks**	**p-value**
20%	**1.25** vs 1.75	**0.001**	1.7 vs **1.3**	**0.021**	**1.15** vs 1.85	**0.0004**
40%	**1.05** vs 1.95	**0.000004**	**1.3** vs 1.7	0.475	**1.1** vs 1.9	**0.00005**
60%	**1.05** vs 1.95	**0.000004**	**1.5** vs **1.5**	0.756	**1.15** vs 1.85	**0.0003**

Figure 4.4 shows graphically how the average F-measure of each of the Auto-ML systems changes over the different values of δ, for the biomedical datasets. In order to reduce the number of figures across this chapter, this Figure shows the results for all Auto-ML systems (GA-1, GA-2 and TPOT) and all baseline PU learning methods (DF-PU and S-EM) investigated in this chapter, but in this

current part of the text the analysis is focussed on the results for TPOT, GA-1 and GA-2 only – the results for DF-PU and S-EM will be discussed later.

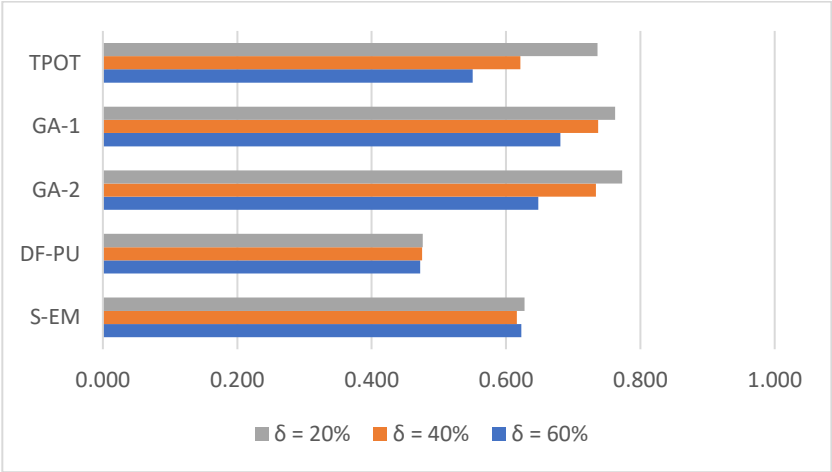


Figure 4.4. Average F-measure results comparison for TPOT, GA-1, GA-2, DF-PU and S-EM, across the three values of δ for the biomedical datasets.

Focusing on the results of GA-1 and GA-2 vs. TPOT, it is evident that as the value of δ increases, the performance of TPOT rapidly declines. The performance of GA-1 and GA-2 also decline, though at a smaller rate than that of TPOT. In fact, the average F-measure of TPOT for all values of δ does not exceed the average F-measure of GA-1 or GA-2 at δ=40%, with the average F-measure of TPOT at δ=40% below even the average F-measure of GA-2 at δ=60%. This chart shows that, whilst the average F-measure for TPOT at δ=20% is somewhat comparable to GA-1 and GA-2 for δ=20%, as the percentage of positive instances available in the labelled positive set decreases, the performance decline is very substantial, making TPOT no longer comparable for more challenging PU learning tasks.

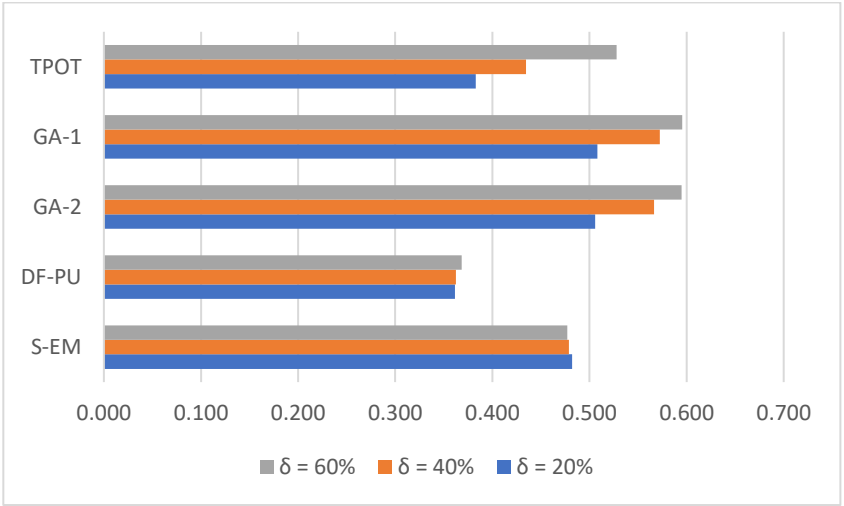


Figure 4.5. Average F-measure results comparison for TPOT, GA-1, GA-2, DF-PU and S-EM, across the three values of δ for the synthetic datasets.

Figure 4.5 shows graphically how the average F-measure of each of the systems changes over the different values of δ, for the synthetic datasets. The results in this figure largely reflect the results shown in Figure 4.4. As with the biomedical datasets, this chart shows that, whilst the average F-measure for TPOT at δ=20% is somewhat comparable to GA-1 and GA-2 for δ=20%, as the percentage of positive instances available in the labelled positive set decreases, the performance decline is significant, making TPOT no longer comparable for more challenging PU learning tasks.

In order to further analyse the results, Table 4.10 shows the values of Pearson's linear correlation coefficient between the F-measure values achieved by GA-1, GA-2, TPOT, DF-PU, and S-EM and percentages of positive examples in the original dataset, for each δ value, for the biomedical datasets. Again, in order to reduce the number of tables across this chapter, the results for all the aforementioned systems or methods are reported in Table 4.10, but in this current part of the text the analysis is focused on the results for GA-1, GA-2 and TPOT only – the results for the baselines will be discussed later. The purpose of this analysis is to understand how reliant the performance of each method is on the percentage of positive examples in the original dataset.

Table 4.10. Linear (Pearson's) correlation coefficient value between the F-measure and the percentage of positive examples in the original dataset (before hiding some positive examples in the unlabelled set) for each combination of a method and a δ value, for the biomedical datasets, for all methods.

Method	δ = 20%	δ = 40%	δ = 60%
GA-1	0.333	0.385	0.504
GA-2	0.340	0.357	0.580
TPOT	0.406	0.432	0.606
DF-PU	0.988	0.988	0.988
S-EM	0.646	0.558	0.652

Table 4.10 shows the same trends of positive correlations as Figure 4.4. For δ=20% and 40%, GA-1 and GA-2 exhibit a weak correlation between percentage of positive instances and F-measure, whilst for δ=60% the correlation is moderate, with correlation categorisation defined as outlined in Section 3.4.3. Whereas, for TPOT, the correlation is moderate for all values of delta. In other words, the performance of TPOT is more closely tied to the percentage of positive instances than the performance of GA-Auto-PU.

Table 4.11. Linear (Pearson's) correlation coefficient value between the F-measure and the percentage of positive examples in the original dataset (before hiding some positive examples in the unlabelled set) for each combination of a method and a δ value, for the synthetic datasets, forall methods.

Method	δ = 20%	δ = 40%	δ = 60%
GA-1	0.712	0.682	0.687
GA-2	0.700	0.702	0.696
TPOT	0.624	0.667	0.674
DF-PU	0.990	0.990	0.990
S-EM	0.794	0.793	0.776

Table 4.11 shows the values of Pearson's linear correlation coefficient between the F-measure values achieved by GA-1, GA-2, TPOT, DF-PU, and S-EM and the percentages of positive examples in the original dataset, for each δ value for the synthetic datasets. Table 4.11 shows much stronger trends than those shown in Table 4.10 for the biomedical datasets. For now, we focus on the results of GA-1, GA-2 and TPOT, with the other results discussed in their appropriate sections. GA-1, GA-2 and TPOT show moderate correlations for δ=40% and 60%, but GA-1 and GA-2 show a strong correlation for δ=20%. This is in contrast to the results of Table 4.10, which showed TPOT exhibiting a stronger trend than GA-1 and GA-2. This helps to highlight the importance of examining methods on multiple types of datasets to gain a fuller understanding of the performance of that system. Due to the differing results of Tables 4.10 and 4.11, it is hard to draw conclusions on this analysis. The purpose was to investigate which methods are more reliant on a high number of positive instances, but limited distinctions between GA-Auto-PU and TPOT are shown. These distinctions are more pronounced for the baseline methods, as discussed later.

4.3.2 Results comparing GA-Auto-PU with two baseline PU learning methods

This section details the results achieved by GA-Auto-PU and two baseline PU learning methods (DF-PU and S-EM, see Section 2.5) when applied to 20 real-world biomedical datasets and 20 synthetic datasets. Note that the results in the GA-1 and GA-2 columns in the tables reported in this section are the same as those reported in the previous section, but they are repeated in this section for the reader's convenience.

Table 4.13 summarises the statistical significance of the F-measure results from Table 4.12, as well as the results for precision and recall.

Table 4.12. F-measure results of GA-Auto-PU with base search space and baseline PU learning methods on real-world biomedical datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**GA-1**	**DF-PU**	**S-EM**	**GA-1**	**DF-PU**	**S-EM**	**GA-1**	**DF-PU**	**S-EM**
Alzheimer's	0.529	0.195	0.321	0.551	0.194	0.370	0.456	0.171	0.373
Autism	0.960	0.648	0.820	0.927	0.648	0.841	0.910	0.645	0.835
Breast cancer Coi.	0.705	0.697	0.711	0.687	0.711	0.704	0.510	0.697	0.699
Breast cancer Wis.	0.954	0.543	0.898	0.932	0.543	0.903	0.906	0.539	0.904
Breast cancer mut.	0.893	0.489	0.892	0.868	0.489	0.893	0.854	0.485	0.892
Cervical cancer	0.828	0.061	0.054	0.903	0.042	0.053	0.714	0.044	0.046
Cirrhosis	0.573	0.405	0.436	0.464	0.401	0.442	0.443	0.405	0.459
Dermatology	0.860	0.228	0.718	0.780	0.229	0.718	0.828	0.219	0.719
PI Diabetes	0.677	0.516	0.534	0.649	0.516	0.525	0.606	0.515	0.544
ES Diabetes	0.958	0.762	0.792	0.895	0.756	0.859	0.930	0.759	0.793
Heart Disease	0.843	0.705	0.811	0.801	0.705	0.828	0.785	0.702	0.829
Heart Failure	0.770	0.487	0.529	0.652	0.486	0.508	0.674	0.481	0.557
Hepatitis C	0.953	0.176	0.695	0.771	0.171	0.708	0.588	0.160	0.609
Kidney Disease	0.976	0.428	1.000	0.988	0.428	1.000	0.754	0.428	0.951
Liver Disease	0.834	0.834	0.816	0.803	0.832	0.587	0.804	0.834	0.788
Maternal Risk	0.476	0.403	0.454	0.812	0.395	0.433	0.735	0.390	0.438
Parkinsons	0.860	0.856	0.815	0.836	0.860	0.748	0.818	0.860	0.762
Parkinsons Biom.	0.476	0.354	0.333	0.265	0.354	0.261	0.233	0.367	0.331
Spine	0.652	0.652	0.820	0.907	0.652	0.839	0.818	0.652	0.830
Stroke	0.474	0.086	0.102	0.255	0.094	0.102	0.255	0.094	0.102

The results in Table 4.13 show GA-1 largely outperforming the baseline PU learning methods in regard to F-measure and precision, but the baseline methods largely outperforming GA-1 in regard to recall. Regarding F-measure, GA-1 outperforms the baseline method in regard to average rank for every value of δ. All of these results are statistically significant, with the exception of against S-EM when δ = 60%. Regarding precision, the results for GA-1 are even better, achieving statistical significance in all cases. Regarding recall, the results are the opposite, with GA-1 outperformed by the baseline methods with statistical significance in almost all cases. However, the performance regarding recall by the baseline methods was largely due to their overprediction of the positive class, coming at a substantial cost to precision. It is for this reason that, even though the recall of the baseline methods is such that they outperform GA-1 in most cases with statistical significance, they are themselves outperformed with statistical significance by GA-1 regarding F-measure as their precision is lacking so much as to largely reduce their F-measure. DF-PU performed best in regard to recall, but worst for both precision and F-measure. This was likely due to the default hyperparameter settings of DF-PU, which selected a subset of 20% of the unlabelled instances as the negative set and only the bottom 1% of instances with the lowest probability of belonging to the positive class as the reliable negative set in Phase 1A of the PU learning algorithm. These parameters may work well for large and highly imbalanced datasets, but they are not suitable as a general use

case as they may identify very few reliable negative instances, resulting in a classifier that greatly over-predicts the positive class.

These results highlight the importance of the iteration count and the reliable negative threshold hyperparameters of GA-Auto-PU and the need for a system that can tune these parameters for a specific dataset.

Table 4.13. Results of Wilcoxon signed-rank tests with Holm correction for multiple hypothesis when comparing GA-1 against S-EM and DF-PU regarding F-measure, Precision and Recall, for the 3 δ values for the biomedical datasets.

δ (%)	Methods compared	F-measure			Precision			Recall		
		Avg. ranks	p-value	α	Avg. ranks	p-value	α	Avg. ranks	p-value	α
20%	GA-1 vs DF-PU	**1.05** vs 1.95	**0.0002**	0.025	**1.05** vs 1.95	**0.0002**	0.025	1.75 vs **1.25**	**0.013**	0.025
	GA-1 vs S-EM	**1.15** vs 1.85	**0.001**	0.05	**1.18** vs 1.82	**0.003**	0.05	1.58 vs **1.42**	0.523	0.05
40%	GA-1 vs DF-PU	**1.2** vs 1.8	**0.0001**	0.025	**1.0** vs 2.0	**0.000002**	0.025	1.98 vs **1.02**	**0.0001**	0.025
	GA-1 vs S-EM	**1.2** vs 1.8	**0.0003**	0.05	**1.08** vs 1.92	**0.0003**	0.05	1.8 vs **1.2**	**0.008**	0.05
60%	GA-1 vs DF-PU	**1.2** vs 1.8	**0.0007**	0.025	**1.0** vs 2.0	**0.000002**	0.025	2.0 vs **1.0**	**0.000002**	0.025
	GA-1 vs S-EM	**1.4** vs 1.6	0.216	0.05	**1.02** vs 1.98	**0.0001**	0.05	1.85 vs **1.15**	**0.0004**	0.05

Next, the performance of GA-2 is compared with the baseline methods on the biomedical datasets.

Table 4.14. F-measure results of GA-Auto-PU with extended search space and two baseline PU learning methods on real-world biomedical datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**GA-2**	**DF-PU**	**S-EM**	**GA-2**	**DF-PU**	**S-EM**	**GA-2**	**DF-PU**	**S-EM**
Alzheimer's	0.548	0.195	0.321	0.576	0.194	0.370	0.529	0.171	0.373
Autism	0.982	0.648	0.820	0.940	0.648	0.841	0.927	0.645	0.835
Breast cancer Coi.	0.711	0.697	0.711	0.672	0.711	0.704	0.553	0.697	0.699
Breast cancer Wis.	0.956	0.543	0.898	0.936	0.543	0.903	0.866	0.539	0.904
Breast cancer mut.	0.895	0.489	0.892	0.739	0.489	0.893	0.872	0.485	0.892
Cervical cancer	0.867	0.061	0.054	0.839	0.042	0.053	0.350	0.044	0.046
Cirrhosis	0.446	0.405	0.436	0.397	0.401	0.442	0.204	0.405	0.459
Dermatology	0.901	0.228	0.718	0.896	0.229	0.718	0.692	0.219	0.719
PI Diabetes	0.642	0.516	0.534	0.645	0.516	0.525	0.634	0.515	0.544
ES Diabetes	0.978	0.762	0.792	0.887	0.756	0.859	0.894	0.759	0.793
Heart Disease	0.836	0.705	0.811	0.780	0.705	0.828	0.786	0.702	0.829
Heart Failure	0.751	0.487	0.529	0.670	0.486	0.508	0.671	0.481	0.557
Hepatitis C	0.944	0.176	0.695	0.863	0.171	0.708	0.610	0.160	0.609
Kidney Disease	0.925	0.428	1.000	0.951	0.428	1.000	0.806	0.428	0.951
Liver Disease	0.831	0.834	0.816	0.817	0.832	0.587	0.748	0.834	0.788
Maternal Risk	0.862	0.403	0.454	0.813	0.395	0.433	0.737	0.390	0.438
Parkinsons	0.935	0.856	0.815	0.842	0.860	0.748	0.792	0.860	0.762
Parkinsons Biom.	0.282	0.354	0.333	0.259	0.354	0.261	0.280	0.367	0.331
Spine	0.923	0.652	0.820	0.917	0.652	0.839	0.761	0.652	0.830
Stroke	0.241	0.086	0.102	0.241	0.094	0.102	0.247	0.094	0.102

Table 4.15. Results of Wilcoxon signed-rank tests with Holm correction for multiple hypothesis when comparing GA-2 against S-EM and DF-PU regarding F-measure, Precision and Recall, for the 3 δ values for the biomedical datasets.

δ (%)	Methods compared	F-measure			Precision			Recall		
		Avg. ranks	p-value	α	Avg. ranks	p-value	α	Avg. ranks	p-value	α
20%	GA-2 vs DF-PU	**1.1** vs 1.9	**0.00002**	0.025	**1.0** vs 2.0	**0.000002**	0.025	2.0 vs **1.0**	**0.000002**	0.025
	GA-2 vs S-EM	**1.12** vs 1.88	**0.0008**	0.05	**1.12** vs 1.88	**0.0003**	0.05	1.72 vs **1.28**	0.053	0.05
40%	GA-2 vs DF-PU	**1.25** vs 1.75	**0.0003**	0.025	**1.05** vs 1.95	**0.00001**	0.025	2.0 vs **1.0**	**0.000002**	0.025
	GA-2 vs S-EM	**1.3** vs 1.7	**0.007**	0.05	**1.12** vs 1.88	**0.001**	0.05	1.75 vs **1.25**	**0.014**	0.05
60%	GA-2 vs DF-PU	**1.25** vs 1.75	**0.002**	0.025	**1.0** vs 2.0	**0.000002**	0.025	2.0 vs **1.0**	**0.000002**	0.025
	GA-2 vs S-EM	**1.5** vs **1.5**	0.546	0.05	**1.02** vs 1.98	**0.0001**	0.05	1.9 vs **1.1**	**0.00001**	0.05

Table 4.15 details the statistical significance of the F-measure results shown in Table 4.14 and summarises the results for precision and recall. Overall, GA-2 outperforms the baseline PU learning methods for F-measure and precision across all values of δ in all 12 cases, and the results are statistically significant in nearly 11 cases. As was the case for GA-1, the baseline methods outperform GA-2 for recall for all values of δ and with statistical significance in 5 out of the 6 cases. The reasons for this are the same as those given when discussing the comparison between the baseline methods and GA-1, namely the large overprediction of the positive class by the baseline methods.

Looking now to the synthetic datasets, Table 4.16 presents the results for GA-1 and the baseline methods.

Table 4.17 summarises the statistical significance of the F-measure results from Table 4.16, as well as the results for precision and recall for the synthetic datasets. The results follow a largely similar trend to those reported in Table 4.13, with GA-1 outperforming the baseline methods with statistical significance in all but one case for F-measure and precision, failing to achieve statistical significance when δ = 60%, whilst being outperformed with statistical significance in 2 cases for recall, both by DF-PU when δ = 20% and 60% respectively. The performance of the baseline methods in regard to recall is due to the same reasons discussed previously, i.e., the massive overprediction of the positive class instances.

Table 4.16. F-measure results of GA-Auto-PU with base search space and baseline PU learning methods on synthetic datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**GA-1**	**DF-PU**	**S-EM**	**GA-1**	**DF-PU**	**S-EM**	**GA-1**	**DF-PU**	**S-EM**
1	0.661	0.484	0.616	0.718	0.484	0.602	0.603	0.483	0.613
2	0.136	0.125	0.194	0.044	0.120	0.130	0.065	0.112	0.120
3	0.788	0.552	0.589	0.693	0.552	0.587	0.637	0.552	0.600
4	0.831	0.454	0.644	0.818	0.416	0.633	0.674	0.417	0.630
5	0.618	0.357	0.402	0.616	0.356	0.436	0.609	0.357	0.465
6	0.759	0.403	0.477	0.769	0.402	0.525	0.684	0.402	0.582
7	0.520	0.285	0.433	0.515	0.283	0.462	0.478	0.282	0.451
8	0.525	0.326	0.468	0.477	0.326	0.457	0.381	0.326	0.439
9	0.111	0.035	0.099	0.080	0.000	0.044	0.146	0.000	0.120
10	0.903	0.233	0.612	0.872	0.234	0.627	0.742	0.233	0.663
11	0.604	0.491	0.505	0.567	0.491	0.520	0.531	0.490	0.517
12	0.674	0.397	0.550	0.666	0.397	0.567	0.609	0.394	0.586
13	0.644	0.500	0.551	0.623	0.460	0.556	0.516	0.456	0.549
14	0.975	0.529	0.817	0.962	0.529	0.840	0.925	0.529	0.873
15	0.601	0.387	0.423	0.593	0.387	0.425	0.519	0.385	0.422
16	0.477	0.239	0.414	0.388	0.239	0.401	0.301	0.240	0.299
17	0.347	0.214	0.262	0.496	0.214	0.281	0.412	0.214	0.267
18	0.559	0.372	0.444	0.389	0.373	0.433	0.326	0.373	0.422
19	0.472	0.378	0.426	0.468	0.378	0.429	0.381	0.376	0.413
20	0.705	0.610	0.615	0.692	0.610	0.620	0.625	0.610	0.613

Table 4.17. Results of Wilcoxon signed-rank tests with Holm correction for multiple hypothesis when comparing GA-1 against S-EM and DF-PU regarding F-measure, Precision and Recall, for the 3 δ values for the synthetic datasets.

δ (%)	**Methods compared**	**F-measure**			**Precision**			**Recall**		
		Avg. ranks	**p-value**	**α**	**Avg. ranks**	**p-value**	**α**	**Avg. ranks**	**p-value**	**α**
20%	GA-1 vs DF-PU	**1.1** vs 1.9	**0.00002**	0.025	**1.0** vs 2.0	**0.000002**	0.025	1.82 vs **1.18**	**0.006**	0.025
	GA-1 vs S-EM	**1.12** vs 1.88	**0.0008**	0.05	**1.1** vs 1.9	**0.000002**	0.05	1.55 vs **1.45**	0.701	0.05
40%	GA-1 vs DF-PU	**1.25** vs 1.75	**0.0003**	0.025	**1.05** vs 1.95	**0.000004**	0.025	1.75 vs **1.25**	0.026	0.025
	GA-1 vs S-EM	**1.3** vs 1.7	**0.007**	0.05	**1.1** vs 1.9	**0.0001**	0.05	1.55 vs **1.45**	0.430	0.05
60%	GA-1 vs DF-PU	**1.25** vs 1.75	**0.002**	0.025	**1.15** vs 1.85	**0.00004**	0.025	1.78 vs **1.22**	**0.020**	0.025
	GA-1 vs S-EM	**1.5** vs **1.5**	0.546	0.05	**1.35** vs 1.65	0.245	0.05	1.58 vs **1.42**	0.573	0.05

Table 4.18 details the results of GA-2 and the baseline methods on the synthetic datasets.

Table 4.19 details the statistical significance of the F-measure results shown in Table 4.18 and summarises the results for precision and recall. GA-2 largely outperforms the baseline methods regarding F-measure and precision for all values of δ, with the results being statistically significant in 10 out of the 12 cases. For recall, as expected based on the results previously reported in this chapter, the baseline methods outperform GA-2 for all values of δ with statistical significance in 2 out of the 6 cases.

Table 4.18. F-measure results of GA-Auto-PU with extended search space and two baseline PU learning methods on synthetic datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**GA-2**	**DF-PU**	**S-EM**	**GA-2**	**DF-PU**	**S-EM**	**GA-2**	**DF-PU**	**S-EM**
1	0.640	0.484	0.616	0.709	0.484	0.602	0.545	0.483	0.613
2	0.176	0.125	0.194	0.105	0.120	0.130	0.111	0.112	0.120
3	0.759	0.552	0.589	0.702	0.552	0.587	0.612	0.552	0.600
4	0.824	0.454	0.644	0.809	0.416	0.633	0.692	0.417	0.630
5	0.612	0.357	0.402	0.571	0.356	0.436	0.559	0.357	0.465
6	0.762	0.403	0.477	0.751	0.402	0.525	0.672	0.402	0.582
7	0.528	0.285	0.433	0.496	0.283	0.462	0.448	0.282	0.451
8	0.571	0.326	0.468	0.484	0.326	0.457	0.390	0.326	0.439
9	0.098	0.035	0.099	0.000	0.000	0.044	0.143	0.000	0.120
10	0.896	0.233	0.612	0.850	0.234	0.627	0.716	0.233	0.663
11	0.574	0.491	0.505	0.579	0.491	0.520	0.525	0.490	0.517
12	0.681	0.397	0.550	0.692	0.397	0.567	0.599	0.394	0.586
13	0.648	0.500	0.551	0.612	0.460	0.556	0.576	0.456	0.549
14	0.977	0.529	0.817	0.966	0.529	0.840	0.934	0.529	0.873
15	0.595	0.387	0.423	0.575	0.387	0.425	0.565	0.385	0.422
16	0.431	0.239	0.414	0.402	0.239	0.401	0.299	0.240	0.299
17	0.384	0.214	0.262	0.470	0.214	0.281	0.382	0.214	0.267
18	0.576	0.372	0.444	0.408	0.373	0.433	0.373	0.373	0.422
19	0.462	0.378	0.426	0.483	0.378	0.429	0.385	0.376	0.413
20	0.701	0.610	0.615	0.664	0.610	0.620	0.594	0.610	0.613

Table 4.19. Results of Wilcoxon signed-rank tests with Holm correction for multiple hypothesis when comparing GA-2 against S-EM and DF-PU regarding F-measure, Precision and Recall, for the 3 δ values for the synthetic datasets.

δ (%)	**Methods compared**	**F-measure**			**Precision**			**Recall**		
		Avg. ranks	**p-value**	**α**	**Avg. ranks**	**p-value**	**α**	**Avg. ranks**	**p-value**	**α**
20	GA-2 vs DF-PU	**1.05** vs 1.95	**0.0002**	0.05	**1.0** vs 2.0	**0.000002**	0.025	1.88 vs **1.12**	**0.007**	0.025
	GA-2 vs S-EM	**1.15** vs 1.85	**0.0001**	0.025	**1.15** vs 1.85	**0.0004**	0.05	1.65 vs **1.35**	0.277	0.05
40	GA-2 vs DF-PU	**1.2** vs 1.8	**0.0001**	0.025	**1.1** vs 1.9	**0.00001**	0.025	1.75 vs **1.25**	0.053	0.025
	GA-2 vs S-EM	**1.2** vs 1.8	**0.0003**	0.05	**1.15** vs 1.85	**0.0001**	0.05	1.55 vs **1.45**	0.784	0.05
60	GA-2 vs DF-PU	**1.2** vs 1.8	**0.0007**	0.025	**1.02** vs 1.98	**0.0001**	0.025	1.75 vs **1.25**	**0.017**	0.025
	GA-2 vs S-EM	**1.4** vs 1.6	0.216	0.05	**1.4** vs 1.6	0.154	0.05	**1.35** vs 1.65	0.522	0.05

Referring back to Figures 4.4 and 4.5, the trend in the average performance for both DF-PU and S-EM is interesting. Whilst the Auto-PU systems have a decline in performance as the δ value increases, the performance of both DF-PU and S-EM remains relatively stable. This is, again, likely due to the massive overprediction of the positive class instances. That is, if we predict almost all instances to be positive for all datasets, the performance is going to be about the same regardless of the value of δ, since the percentage of positive instances in the test set has not been altered. This poor

performance is likely due to the use of the default hyperparameter settings for the algorithms. That is, the hyperparameter settings used in the papers proposing the original methods were used, rather than tuning the parameters to fit the learning task for which we are using the methods. Recent work [207] showed that tuning the hyperparameter settings improved the performance of these methods.

Referring back to Tables 4.10 and 4.11, analysing the correlation coefficient between percentage of instances that belong to the positive set and F-measure, S-EM exhibits moderate to strong correlations, whilst DF-PU exhibits a near perfect correlation. This latter result can also be explained by the fact that DF-PU massively overpredicts the positive class, since in this case an increase in the percentage of positive-class instances tends to lead to an increase in precision without substantially reducing recall, therefore leading to an increased F-measure. These results indicate that the performance of the baseline methods is more reliant on a high proportion of positive instances in the dataset than GA-Auto-PU. Thus, indicating that GA-Auto-PU may perform more favourably in learning scenarios where there are fewer positive instances to learn from.

4.4 The Most Frequently Selected Hyperparameter Values of the Optimised PU Learning Algorithm

This section reports the PU learning hyperparameter values most frequently selected by GA-Auto-PU utilising the base search space (GA-1) and the extended search space (GA-2). It reports the selection frequency, baseline frequency, and their difference. The selection frequency of a hyperparameter value is calculated as the ratio of the number of GA-Auto-PU runs where value was used in the optimised PU learning algorithm returned by GA-Auto-PU over the total number of GA-Auto-PU runs, which is 300 for each type of search space (base or extended spaces) and each type of dataset (biomedical or synthetic), considering 20 datasets times 3 values of δ, times 5 runs of a GA-Auto-PU version per dataset, due to the use of 5-fold cross-validation. For each type of search space (i.e., base and extended search spaces), two separate tables are reported, firstly for the biomedical datasets and secondly for the synthetic datasets. In these tables, the baseline frequency is the expected selection frequency of a hyperparameter value if all values of that hyperparameter were randomly selected for use in a PU learning algorithm. I.e., it is calculated by simply dividing 1 (one)

by the number of candidate values for that hyperparameter. The difference between these two frequencies is simply the selection frequency minus the baseline frequency.

Little has been written on the topic of suitable algorithm configuration for PU learning, and no guidelines exist in the literature. By analysing the most frequently selected hyperparameter values in this thesis' experiments, we can begin to understand which PU learning algorithm configurations perform well and under what circumstances. This information could prove useful for future research into improving the performance of PU learning algorithms.

Throughout this section, the term "classifier" is used to refer to a classification algorithm (rather than a classification model learned by an algorithm), unless explicitly mentioned otherwise.

4.4.1 The Hyperparameter Values Most Frequently Selected by GA-1 (with Base Search Space)

Tables 4.20 and 4.21 report the most frequently selected values of the hyperparameters of the optimised PU learning algorithms returned by all runs of GA-1 on the biomedical datasets and on the synthetic datasets, respectively. Considering first the *Phase_1A_Classifier* and *Phase_2_Classifier* hyperparameters, it is noteworthy that Linear Discriminant Analysis (LDA) and Gaussian Naïve Bayes (NB) were most frequently selected for both types of datasets, albeit as classifiers for different phases of the optimised PU learning algorithm depending on the type of dataset. More precisely, Gaussian NB and LDA were most frequently selected as Phase 1A and Phase 2 classifiers, respectively, for the biomedical datasets (Table 4.20); whilst the preference for these two classifiers was reversed in these two phases for the synthetic datasets (Table 4.21). Recall from Section 2.5 two assumptions of the two-step PU learning framework, separability and smoothness. Separability refers to a natural separation between the two classes, and smoothness states that instances which are close in the feature space are more likely to belong to the same class. Both LDA and Gaussian NB are linear classifiers, making them suitable classifiers for data that adheres to both of these assumptions.

The most frequently selected *Phase 1B classifier* for the biomedical datasets was Deep forest, whilst for the synthetic datasets it was k-nearest neighbour (kNN). These classifiers were selected with relatively low frequency (in relation to difference between the baseline frequency and selection frequency) and thus the conclusions about their preferences in that phase are not strong. Considering

the *Phase 1B Flag* hyperparameter, its two candidate values (*True* and *False*) have almost exactly the same selection frequency for both types of datasets, implying that whether to utilise Phase 1B strongly depends on the specifics of the individual dataset.

Table 4.20. Selection frequency of hyperparameter values by GA-1 for the biomedical datasets.

Hyperparameter	Most selected value	Selection Freq. (%)	Baseline Freq. (%)	Diff. (%)
Phase 1A Iteration Count	1	19.67	10.00	9.67
Phase 1A RN Threshold	0.3	15.00	10.00	5.00
Phase 1A Classifier	Gaussian NB	14.00	5.56	8.44
Phase 1B Flag	False	52.33	50.00	2.33
Phase 1 B RN Threshold	0.15	19.00	10.00	9.00
Phase 1B Classifier	Deep forest	11.00	5.56	5.44
Phase 2 Classifier	LDA	21.00	5.56	15.44

Table 4.21. Selection frequency of hyperparameter values by GA-1 for the synthetic datasets.

Hyperparameter	Most selected value	Selection Freq. (%)	Baseline Freq. (%)	Diff. (%)
Phase 1A Iteration Count	4	19.00	10.00	9.00
Phase 1A RN Threshold	0.35	21.33	10.00	11.33
Phase 1A Classifier	LDA	14.00	5.56	8.44
Phase 1B Flag	True	52.00	50.00	2.00
Phase 1 B RN Threshold	0.2	15.17	10.00	5.17
Phase 1B Classifier	kNN	10.33	5.56	4.77
Phase 2 Classifier	Gaussian NB	22.33	5.56	16.77

The hyperparameter *Phase_1A_Iteration_Count* is utilised to handle the class imbalance that is often present in PU learning datasets. It handles the class imbalance by splitting the unlabelled set into a given number of sets in Phase 1A of the two-step procedure. Therefore, it intuitively follows that when the percentage of positive instances is low, the iteration count should be high, and vice versa. To check if this is really the case, a more detailed analysis of this hyperparameter's optimisation was conducted, as described next.

For each dataset, for each value of δ, the average value of the *Phase_1A_Iteration_Count* hyperparameter over the cross-validation procedure was calculated. Then, the Pearson's correlation coefficient between those average iteration counts and the percentage of instances which are positive in the original datasets (i.e., the full datasets before hiding the positive instances in the negative set or splitting into training and test sets) was calculated, over the 20 datasets for each type of dataset (biomedical and synthetic datasets), for each of the three. For the biomedical datasets, the correlation coefficient values were -0.646, -0.655, and -0.689 for δ = 20%, 40%, and 60% respectively. For the

synthetic datasets, the coefficient values were -0.653, -0.685 and -0.696 for δ = 20%, 40%, and 60% respectively. All of these represent a moderate negative correlation (categorised as described in Section 3.4.3), indicating that when the percentage of positive instances is low, the average iteration count value selected by GA-1 tends to be high, and vice versa. This supports the idea that the iteration count parameter is handling class imbalance. If these results hold for the analyses later in this work, this can help to provide general guidance on what the iteration count parameter should be set to for a dataset with a given class distribution, when using the two-step framework.

The other hyperparameters listed in Table 4.20 offer no clear patterns or conclusions to be drawn, and as such will not be discussed further.

4.4.2 The Hyperparameter Values Most Frequently Selected by GA-2 (with Extended Search Space)

Tables 4.22 and 4.23 report the most frequently selected values of the hyperparameters of the optimised PU learning algorithms returned by all runs of GA-2 on the biomedical datasets and the synthetic datasets, respectively. Interestingly, these results for GA-2 differ from the results for GA-1 in the previous section. That is, whilst GA-1 showed a clear preference for simple classifiers (predominantly LDA and Gaussian NB) in Phase 1A and Phase 2, the story is somewhat more mixed for GA-2. *Phase 1A Classifier* was most frequently Logistic regression for the synthetic datasets, which is a relatively simple classifier; and Random forest for the biomedical datasets. As random forest is an ensemble classifier, it cannot be said that it is simple. For *Phase 1B Classifier*, the most frequently selected classifier for the biomedical datasets was SVM, and for the synthetic datasets was random forest. Finally, for *Phase 2 classifier*, the most frequently selected classifier was Deep forest for the biomedical datasets and Multilayer perceptron for the synthetic datasets. Deep forest and Multilayer perceptron cannot be considered simple classifiers, perhaps implying that the reliable negative sets created when utilising the spy method are somewhat more complex than those created when utilising the base search space. However, the use of a complex classifier does not necessarily imply complex data, given that a complex classifier may work effectively on a simple dataset, and there were no criteria set for favouring computational simplicity when evaluating candidate solutions during the GA-2's search.

Table 4.22. Selection frequency of hyperparameter values by GA-2 for the biomedical datasets.

Hyperparameter	Most selected value	Selection Freq. (%)	Baseline Freq. (%)	Diff. (%)
Phase 1A Iteration Count	1	26.67	10.00	16.67
Phase 1A RN Threshold	0.3	20.33	10.00	10.33
Phase 1A Classifier	Random forest	12.67	5.56	7.11
Phase 1B Flag	False	66.00	50.00	16.00
Phase 1 B RN Threshold	0.25	15.33	10.00	5.33
Phase 1B Classifier	SVM	10.67	5.56	5.11
Spy rate	0.1	18.67	14.29	4.38
Spy tolerance	0.06	18.47	9.09	9.38
Spy flag	FALSE	73.33	50.00	23.33
Phase 2 Classifier	Deep forest	10.00	5.56	4.44

Table 4.23. Selection frequency of hyperparameter values by GA-2 for the synthetic datasets.

Hyperparameter	Most selected value	Selection Freq. (%)	Baseline Freq. (%)	Diff. (%)
Phase 1A Iteration Count	4	24.33	10.00	14.33
Phase 1A RN Threshold	0.35	23.33	10.00	13.33
Phase 1A Classifier	Logistic reg.	10.67	5.56	5.11
Phase 1B Flag	FALSE	67.00	50.00	17.00
Phase 1 B RN Threshold	0.15	15.33	10.00	5.33
Phase 1B Classifier	Random forest	11.33	5.56	5.77
Spy rate	0.3	18.00	14.29	3.71
Spy tolerance	0.05	13.94	9.09	4.85
Spy flag	FALSE	79.33	50.00	29.33
Phase 2 Classifier	MLP	15.67	5.56	10.11

The *Phase_1B_Flag* parameter has been set to *False* more frequently than it was for GA-1, 73.33% for biomedical datasets and 67% for synthetic datasets. This may be due to a more accurate reliable negative set being assembled in Phase 1A, through the use of the spy method. However, considering the spy method, the *Spy flag* was set to *False* far more frequently than it was set to *True*, at 73.33% for the biomedical and 79.33% for the synthetic datasets. This means that only 26.67% and 20.67% of optimised candidate solutions did utilise the spy method. It is too early to draw conclusions about the efficacy of including the spy-based methods in the search space, given that so far this thesis has reported only the results of GA-Auto-PU. However, if this trend continues for other Auto-PU systems (whose results will be reported in later chapters), it could indicate that the inclusion of spy-based methods in the search space is not efficient, given that it increases the search space from 11,664,000 possible candidate solutions to 1,796,256,000 possible candidate solutions (see Section 3.2), thus

creating a much more complex search landscape and a higher computational cost of finding an optimal solution.

As in the previous section, a more detailed analysis of the *Phase_1A_Iteration_Count* hyperparameter's optimisation is reported next. As with the previous section, there is a predominantly moderate negative correlation between the iteration count and the percentage of instances in the dataset that belong to the positive class, indicating that the hyperparameter is utilised to handle class imbalance. GA-2 actually displays a strong correlation when $\delta = 60\%$ for the biomedical datasets, further supporting this hypothesis. For the biomedical datasets, the correlation coefficient values are -0.631, -0.687, and -0.723 for $\delta = 20\%$, 40%, and 60% respectively. For the synthetic datasets, the values are -0.689, -0.698, and -0.695 for $\delta = 20\%$, 40%, and 60% respectively.

The other parameters listed in Table 4.22 offer no clear patterns or conclusions to be drawn, and as such will not be discussed further.

4.5 Summary

This Chapter has introduced GA-Auto-PU, a Genetic Algorithm-based Automated-Machine Learning system for Positive-Unlabelled learning. We evaluated the predictive performance of two versions of GA-Auto-PU, utilising two search spaces (distinguished by whether or not they allow for the generation of spy-based PU learning methods). Each version of GA-Auto-PU was compared firstly against an Auto-ML baseline (TPOT), and secondly against two baseline PU learning methods. Whilst TPOT was built for standard classification rather than for PU learning, at the time of proposing GA-Auto-PU, no other Auto-ML system for PU learning existed. As such, a direct comparison could not be made. However, comparing GA-Auto-PU with TPOT has shown the predictive performance that can be gained from utilising a PU learning system, rather than an Auto-ML system for standard binary classification.

In general, the two versions of GA-Auto-PU (named GA-1 and GA-2) outperformed TPOT and the two baseline PU learning methods with regard to F-measure with statistical significance. The analysis of the algorithmic components most frequently selected by GA-Auto-PU showed a preference for non-spy-based methods for both biomedical and synthetic datasets. This could indicate that the spy-based method is an unnecessary or not cost-effective expansion of the search space,

considering that the inclusion of spy-based methods in the search space greatly increases the size of that space. However, this conclusion may be premature, given the limited number of results reported so far in this thesis. Therefore, this will be explored further in later chapters when more results are available.

Overall, the performance of GA-1 and GA-2 across the datasets, and the varied selected values for the algorithmic components discussed in Section 4.4, confirm the need for an Auto-ML system such as GA-Auto-PU, which can configure and tune the hyper-parameters of a PU learning algorithm in order to maximise predictive performance for a specific input dataset.

Chapter 5

A Bayesian Optimisation-based Auto-ML System for Positive-Unlabelled Learning (BO-Auto-PU)

In this chapter we introduce BO-Auto-PU, a Bayesian Optimisation (BO)-based Automated Machine Learning (Auto-ML) system for Positive-Unlabelled (PU) learning. GA-Auto-PU, detailed in Chapter 4, was the first Auto-ML system specific to PU learning, and showed statistically significant improvements in predictive performance over two baseline PU learning methods and an Auto-ML system for standard binary classification. However, the GA-Auto-PU system is computationally expensive, with GA-1 averaging 226.3 minutes to run 5-fold cross-validation per dataset and GA-2 averaging 223.2 minutes to run 5-fold cross-validation per dataset.

BO is generally a much more computationally efficient procedure than a standard Genetic Algorithm, given that it assesses most of the generated individuals (candidate solutions) with a fast executed surrogate model as opposed to the slowly executed objective function (see Section 2.3 for details of the BO procedure). As such, BO-Auto-PU has been developed in an attempt to reduce the computational expense of GA-Auto-PU without sacrificing predictive performance. Therefore, there are two primary research questions to explore in this chapter: firstly, whether BO-Auto-PU improves on the performance of GA-Auto-PU in regard to computational efficiency, and secondly whether BO-Auto-PU improves on the performance of GA-Auto-PU in regard to predictive performance.

These research questions will be addressed through the experiments and the discussion of their results in this chapter, and these questions will be directly answered in Section 5.5 (Summary).

Whilst the question of improvement in predictive accuracy is generally considered to be the primary concern of researchers when proposing a new machine learning method, the primary contribution of BO-Auto-PU is the increase in computational efficiency. That is, a decrease in runtime. One of the primary goals of Auto-ML is to allow the field of machine learning to be more accessible for a wider array of researchers and practitioners (see Section 2.4). A system with a very high computational runtime is not a generally accessible system to those with limited computational resources. Thus, strides must be made to develop systems which can be utilised by all researchers and practitioners, not just those with access to high powered resources. So, to restate, the primary contribution of BO-Auto-PU is the improvement in computational efficiency. As repeated throughout this work, PU learning is a growing field of machine learning. Whilst GA-Auto-PU performed well with regard to predictive performance, it was computationally inefficient. In order for the field of PU learning to continue growing, the systems that enable it must be accessible to all. Thus, the motivation for the development of BO-Auto-PU.

The structure of this chapter is as follows: Section 5.1 gives a detailed description of the BO-Auto-PU system, presenting the pseudocode and hyperparameters of the system. Section 5.2 outlines the experimental setup, detailing the datasets used, the cross-validation procedure, and the statistical significance tests. Section 5.3 presents the results comparing BO-Auto-PU against GA-Auto-PU, before presenting the results comparing BO-Auto-PU with two baseline PU learning methods. Note that BO-Auto-PU utilises the same two search spaces as GA-Auto-PU (both the base search space and the extended search space with the Spy method); and the reporting and analyses of the computational results in Section 5.3 will also be performed separately for each search space. Section 5.4 discusses the hyperparameter values of the optimised PU learning methods most frequently selected by BO-Auto-PU, before Section 5.5 (Summary) concludes this chapter.

5.1 Description of BO-Auto-PU

As outlined in the Introduction of this chapter, BO-Auto-PU is a Bayesian Optimisation (BO)-based Automated Machine Learning (Auto-ML) system specific to PU learning. This section describes the

pseudocodes detailing the procedure followed by the BO-Auto-PU system. Details of the individual (candidate solution) encoding, as well the fitness (objective) function can be found in Chapter 3. Recall that the reason why individual encoding and fitness function are discussed separately in Chapter 3 (rather than in the current chapter) is because these Auto-ML system components are the same for both GA-Auto-PU and BO-Auto-PU. A generic version of BO is discussed in detail in Section 2.3.

5.1.1 The BO Procedure for PU learning

Procedure 5.1 outlines the procedure that the BO follows to evolve a PU learning algorithm. Initially, *#Configs* PU learning algorithm configurations are randomly generated (step 1) and evaluated, with their F-measures saved as *Scores* (step 2). The random generation procedure is the same as that described for GA-Auto-PU in Chapter 4, but to reiterate briefly, the random generation of a candidate solution involves, for each gene, randomly selecting a value from the list of candidate values of that specific gene. E.g., for the hyperparameter *Iteration_Count_1A*, the options for the value of that hyperparameter are 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10; and each of these values can be selected with equal probability (10% in this case).

Procedure 5.1 Outline of the Bayesian optimization procedure for Positive-Unlabelled Learning

1. *Configs* = randomly generate *#Configs* PU learning configurations;
2. *Scores* = run objective function for all configurations in *Configs*; // see Procedures 3.1-3.4
3. Fit *Surr_model* with *Configs* as features, *Scores* as target;
4. $i = 0$;
5. **While** $i <$ *It_count*:
    a. *New_configs* = randomly generate *#Configs* configurations;
    b. $\hat{Y}$ = calculate a surrogate score for each new config with *Surr_model*;
    c. *Best_config* = config with highest surrogate score $\hat{Y}$;
    d. *Score* = run objective function for *Best_config*; // see Procedures 3.1-3.4
    e. Add *Best_config* to *Configs*, add *Score* to *Scores*;
    f. Retrain *Surr_model* on *Configs* and *Scores*;
    g. i += 1;

Output: Best configuration according to objective score;

A random forest regressor, *Surr_model* (surrogate model), is then trained, using *Configs* as predictive features, and *Scores* as the target variable (step 3). Common choices for *Surr_model* are random forest and Gaussian process regressor (see Sections 2.1 and 2.3). Preliminary experiments conducted

to compare utilisation of Gaussian process vs random forest showed that random forest produced the best results for this use case, in addition to random forests being much faster than Gaussian process in general, and as such random forest has been used to learn the surrogate models in BO-Auto-PU.

Configs are processed as follows for training *Surr_model*: for the base search space (without the Spy method), the numeric components of each configuration (*Threshold_1A, Iteration_Count_1A, Threshold_1B*) are treated as numeric features, the Boolean component (*Flag_1B*) is treated as a binary feature, and the nominal components (*Classifier_1A*, *Classifier_1B*, *Classifier_2*) are one-hot encoded, with a binary value for each potential value of the component, indicating whether or not that value is used. The resulting instance for the regression algorithm (which will be used to learn *Surr_model*) consists of 58 features. For the extended search space (with the Spy method), all the previously mentioned components are treated as they are in the base search space. However, we also have the additional Spy method's components, with *Spy_rate* and *Spy_tolerance* treated as numeric features, and the Boolean component "*Spy_flag*" treated as a binary feature – indicating whether or not the Spy method is used. This results in an instance consisting of 61 features for the regression algorithm. A truncated example of the dataset used as input for learning *Surr_model* is shown in Figure 5.1.

F1	F2	F3	F4	F5	F6 … 20	F21	F22	F23	F24	F25 … 39	F40	F41	F42	F43	F44 … 58	Score
6	0.3	1	0	0	…	0.15	1	0	0	…	0	0	0	0	…	0.545
2	0.45	0	0	1	…	0.25	0	1	0	…	1	0	1	0	…	0.632
2	0.5	0	0	0	…	0.5	1	0	0	…	0	0	0	0	…	0.123
2	0.2	0	0	0	…	0.35	0	0	0	…	0	1	0	0	…	0.958
2	0.15	0	1	0	…	0.1	0	0	1	…	1	0	0	0	…	0.342

Figure 5.1. Example input dataset for learning *Surr_model* in BO-Auto-PU with the base search space.

Note that for Figure 5.1, the feature headings have been replaced with the values F1, F2, etc. to save space. However, they are (in order) as follows:

F1 is *Iteration_count_1A*.

F2 is *RN_threshold_1A*.

F3...F20 represent the one-hot encoded values for *Classifier_1A,* namely: Gaussian naïve Bayes (F3), Bernoulli naïve Bayes (F4), random forest (F5), etc, with the other 15 candidate classifiers truncated as F6 … 20 to save space. For example, in the second row of the dataset shown in Figure

5.1, the value F5 = 1 means that random forest was selected as the classifier for phase 1A in that candidate configuration.

F21 is *RN_threshold_1B*.

F22…F39 represent the one-hot encoded values for *Classifier_1B,* namely: Gaussian naïve Bayes (F22), Bernoulli naïve Bayes (F23), random forest (F24), etc.

F40 is *Flag_1B* with 1 representing a value of *True* and 0 representing a value of *False*.

F41…F58 represent the one-hot encoded values for *Classifier_2,* namely: Gaussian naïve Bayes (F41), Bernoulli naïve Bayes (F42), random forest (F43), etc..

The iteration index i is set to 0 (step 4), and the *while* loop in step 5 is started, proceeding *It_count* times. In each iteration of this loop, a new set of *#Configs* configurations, *New_configs*, are randomly generated (step 5.a) and a surrogate score for each is calculated by *Surr_model* and saved as $\hat{Y}$ (step 5.b). Note that this random population generation is the same procedure as the initial random population generation in step 1 of Procedure 5.1 (which is also the same procedure used by GA-Auto-PU to generate an initial population, as described in Chapter 4). That is, at each iteration, a new population is created just as it was at the start of the execution of BO-Auto-PU.

The best configuration, *Best_config*, with the highest surrogate score $\hat{Y}$ is evaluated using the objective function (steps 5.c,d), and added to *Configs*, with the objective Score (F-measure) added to *Scores* (step 5.e). This is the key step in the Bayesian optimization procedure. The reason this optimization procedure is so much more computationally efficient than a standard genetic algorithm (GA) is that only a single candidate solution is evaluated by the computationally expensive objective at each iteration; unlike a GA, which evaluates the whole population using the objective function. Selection of the candidate solution to evaluate is performed by the acquisition function. There are many varieties of acquisition function, with options such as Expected Improvement (EI) and Probability of Improvement (see Section 2.3.2), as well as the simple approach of using the score predicted by *Surr_model*. Preliminary experiments comparing this simple approach and EI showed that the simple approach of selecting the candidate solution with the best score predicted by *Surr_model* gave the best results. That is, the predictive performance of this simple approach was higher than when EI was used. Hence, this approach has been taken for BO-Auto-PU.

Surr_model is then retrained with *Best_config* added to *Configs* (step 5.f). That is, *Surr_model* is updated to include the new candidate solution that was selected from the population, thus updating *Surr_model*'s knowledge of the search space. This increases the size of *Configs* by 1 at each iteration, providing more information to *Surr_model*. The iteration index i is incremented by 1 (step 5.g). This process (steps 5.a-g) is repeated *It_count* times. Finally, the best configuration, according to the objective score (F-measure), is returned. Note that it is the objective score, not the surrogate score, that determines the best candidate solution to be returned. The surrogate score is an estimation that can be computed relatively fast, whereas the objective score, which is takes much longer to be computed, is the predictive performance measure that the system really has to optimise. Therefore, the best candidate solution is selected from the population of candidate solutions for which the objective score has been calculated, to return an accurate best candidate solution. Best, that is, relative to the population.

The objective function cited in steps 2 and 5.d is defined in Section 3.2.3.

5.1.2 The BO's Hyperparameters

Table 5.1 shows the hyperparameter settings of the BO underlying BO-Auto-PU. The *It_count* parameter determines the number of iterations to perform the optimisation. *#Configs* determines the number of individuals in the population. *Surr_model* is the surrogate model used to calculate the surrogate score. The acquisition function is the method for selecting which candidate solution to assess with the objective function. For this, we simply use the predicted value from *Surr_model*. In preliminary experiments, we varied the acquisition function to use the Expected Improvement (EI) algorithm (see Section 2.3) but found that just using *Surr_model* predicted value gave better results.

Table 5.1. Hyperparameters of the BO-Auto-PU system, with their default values.

Hyperparameter	Value
It_count	50
#Configs	101
Surr_model	Random Forest Regressor
Acquisition function	Surr_model predicted value

The *It_count* and *#Configs* parameters were set to match the *#generations* and *Pop_size* parameters for GA-Auto-PU (see Section 4.1.2), in order to perform a controlled comparison between BO-Auto-PU and GA-Auto-PU in the experiments reported later in this chapter.

5.1.3 Computational Efficiency

In both the GA-based and the BO-based Auto-ML systems for PU learning, the running time is by far dominated by the time to evaluate the candidate solutions along the iterations of the search, i.e., the time to learn a PU model on part of the training set and evaluate the learned model's F-measure on the remaining part of the training set, for each candidate PU learning method. GA and the BO-based methods perform the same number of iterations (50) in our experiments. However, in each generation (iteration) of GA-Auto-PU the GA must learn and evaluate n PU models, where n is the number of individuals (candidate solutions) in the population, whilst each iteration of BO-Auto-PU needs to learn and evaluate a single PU model. Learning a PU model can be very computationally expensive, depending not only on the size of the dataset but also on the time complexity of the 3 classification algorithms chosen for Phases 1A, 1B and 2 of the 2-step method, and the number of iterations the classifier is applied in Phase 1A.

GA-Auto-PU and BO-Auto-PU also must perform other steps for generating candidate solutions to be evaluated, but these take in general much less time than the time to evaluate candidate solutions using the objective function (F-measure) as described above. More precisely, at each iteration the GA must perform tournament selection, crossover and mutation, but these are all simple operations, which are much faster than computing the fitness function (learning a PU model for each individual).

Unlike the GA, at each iteration BO learns a surrogate model, but again, the time for this is much shorter than the time taken to learn a PU model in each iteration of BO. This is because the surrogate model is learned by a relatively fast random forest algorithm using a small dataset of PU algorithm configurations, whilst learning a PU model involves running multiple classifiers (one of them for several iterations in Phase 1A), each classifier can be much slower than a random forest. In addition, each classifier is learned using the training data of the current dataset, which is typically much larger in number of instances than the small dataset of PU method configurations. Regarding the number of features, the training set for learning a PU model has in general more features than the training set for learning the surrogate model in the case of the synthetic datasets; whilst the converse is true in the case of the biomedical datasets – but even for this latter group of datasets, the overall time taken to learn a surrogate model is still much faster than the time to learn a PU model, making BO-Auto-PU much faster than GA-Auto-PU, as will be reported later in this thesis.

5.2 Experimental Setup

The experimental procedure is explained in detail in Chapter 3. However, to briefly recap, two types of datasets are used in these experiments (biomedical and synthetic), each with 3 versions (varying the % of positive instances hidden in the unlabelled set), thus creating 120 datasets total.

A nested cross-validation procedure is used, with a simple 5-fold cross-validation procedure as the external layer. The internal layer splits the training set into 5 learning and validation sets, which is used to evaluate the candidate solutions.

To compare the performance of the methods, we use the Wilcoxon signed rank test [202], with Holm correction for testing multiple hypotheses [203].

5.2.1 Structure of the Results' Sections

In the next sections, we present experimental results comparing the BO-Auto-PU system with both search spaces (without and with the Spy method). Firstly, BO-Auto-PU is compared against GA-Auto-PU. Secondly, BO-Auto-PU is compared against the two PU learning baselines. Experiments were conducted on both the real-world biomedical datasets and the synthetic datasets, for three values of δ (the percentage of positives hidden in the unlabelled set): 20%, 40%, and 60%. Each section will report the F-measure results in full and will provide a summary of the precision and recall results. The full precision and recall results can be found in the Appendix.

For the sake of brevity, the BO-Auto-PU and the GA-Auto-PU systems utilising the base search space (without the Spy method) will be referred to as BO-1 and GA-1 respectively, whilst the systems using the extended search space will be referred to as BO-2 and GA-2 respectively.

5.3 Results for BO-Auto-PU

5.3.1 Results comparing BO-Auto-PU with GA-Auto-PU

In this section, results for BO-Auto-PU are given and compared to GA-Auto-PU, beginning with a comparison of BO-1 and GA-1 (using the base search space) on the biomedical datasets, as shown in Table 5.2.

Table 5.2. F-measure results of BO-1 and GA-1 on real-world biomedical datasets.

	δ = 20%		δ = 40%		δ = 60%	
Dataset	**GA-1**	**BO-1**	**GA-1**	**BO-1**	**GA-1**	**BO-1**
Alzheimer's	0.529	0.615	0.551	0.600	0.456	0.436
Autism	0.960	0.967	0.927	0.956	0.910	0.863
Breast cancer Coi.	0.705	0.694	0.687	0.701	0.510	0.586
Breast cancer Wis.	0.954	0.949	0.932	0.969	0.906	0.895
Breast cancer mut.	0.893	0.893	0.868	0.873	0.854	0.841
Cervical cancer	0.828	0.839	0.903	0.903	0.714	0.645
Cirrhosis	0.573	0.545	0.464	0.529	0.443	0.489
Dermatology	0.860	0.872	0.780	0.905	0.828	0.725
PI Diabetes	0.677	0.647	0.649	0.645	0.606	0.594
ES Diabetes	0.958	0.983	0.895	0.877	0.930	0.902
Heart Disease	0.843	0.844	0.801	0.830	0.785	0.777
Heart Failure	0.770	0.753	0.652	0.605	0.674	0.704
Hepatitis C	0.953	0.907	0.771	0.838	0.588	0.708
Kidney Disease	0.976	0.988	0.988	0.964	0.754	0.806
Liver Disease	0.834	0.820	0.803	0.817	0.804	0.795
Maternal Risk	0.476	0.837	0.812	0.780	0.735	0.689
Parkinsons	0.860	0.935	0.836	0.875	0.818	0.732
Parkinsons Biom.	0.476	0.167	0.265	0.192	0.233	0.182
Spine	0.652	0.954	0.907	0.926	0.818	0.742
Stroke	0.474	0.244	0.255	0.153	0.255	0.208

Table 5.3 summarises the statistical significance of the F-measure results from Table 5.2 (for biomedical datasets), as well as the results for precision and recall. In Table 5.3, for each combination of a performance measure (F-measure, precision, recall) and a δ value (δ= 20%, 40%, 60%), the table reports the average (Avg.) rank of BO-1 vs GA-1 (BO-1 is the left rank, GA-1 is the right one) and the corresponding p-value. The better (lower) avg. rank in each cell is shown in boldface. For example, in the cell for F-measure, δ = 20%, the average rank for BO-1 is 1.48 and for GA-1 is 1.52. Hence, BO-1 was the winner, but the p-value (0.952) was greater than the significance level α (0.05), so this result was not statistically significant. The following discussion of results will focus mainly on the F-measure, the most important measure in Table 5.3, whilst precision and recall results are reported for completeness.

The results in Table 5.3 show BO-1 performing best in general for δ=20% and 40%, whilst GA-1 performs best for δ=60%. Perhaps, given that PU learning when δ=60% is a harder task than δ=20% and 40% due to the limited labelled data, a more diverse population is beneficial to find more complex solutions. GA-1 naturally introduces diversity through its population of candidate solutions and its evolutionary operations, and thus may be more adept for a more complex PU learning task associated with a larger δ value. However, no results in Table 5.3 were statistically significant.

Table 5.3. Results of Wilcoxon signed-rank tests when comparing BO-1 against GA-1 regarding F-measure, Precision and Recall, for the 3 δ values on the biomedical datasets.

δ (%)	F-measure		Precision		Recall	
	Avg. ranks	p-value	Avg. ranks	p-value	Avg. ranks	p-value
20%	**1.48** vs 1.52	0.952	**1.45** vs 1.55	0.983	1.6 vs **1.4**	0.114
40%	**1.38** vs 1.62	0.334	**1.45** vs 1.55	0.601	**1.42** vs 1.58	0.398
60%	1.75 vs **1.25**	0.177	1.52 vs **1.48**	0.513	1.75 vs **1.25**	0.064

The results comparing BO-2 and GA-2 (using the extended search space) on the biomedical datasets are given in Table 5.4.

Table 5.4. F-measure results of BO-2 and GA-2 on real-world biomedical datasets.

	δ = 20%		δ = 40%		δ = 60%	
Dataset	**BO-2**	**GA-2**	**BO-2**	**GA-2**	**BO-2**	**GA-2**
Alzheimer's	0.580	0.548	0.603	0.576	0.492	0.529
Autism	0.963	0.982	0.937	0.940	0.914	0.927
Breast cancer Coi.	0.667	0.711	0.618	0.671	0.000	0.553
Breast cancer Wis.	0.959	0.956	0.942	0.936	0.889	0.866
Breast cancer mut.	0.890	0.896	0.853	0.739	0.845	0.872
Cervical cancer	0.867	0.867	0.867	0.839	0.839	0.350
Cirrhosis	0.497	0.446	0.515	0.397	0.472	0.204
Dermatology	0.876	0.901	0.841	0.896	0.795	0.692
PI Diabetes	0.653	0.642	0.648	0.646	0.615	0.634
ES Diabetes	0.954	0.978	0.891	0.887	0.912	0.894
Heart Disease	0.844	0.836	0.817	0.780	0.805	0.786
Heart Failure	0.757	0.751	0.652	0.670	0.600	0.671
Hepatitis C	0.964	0.944	0.761	0.863	0.612	0.610
Kidney Disease	0.976	0.925	0.976	0.951	0.789	0.806
Liver Disease	0.822	0.831	0.815	0.817	0.722	0.748
Maternal Risk	0.847	0.862	0.786	0.813	0.729	0.738
Parkinsons	0.936	0.935	0.837	0.843	0.800	0.792
Parkinsons Biom.	0.286	0.282	0.000	0.259	0.000	0.280
Spine	0.941	0.923	0.936	0.917	0.700	0.761
Stroke	0.256	0.241	0.255	0.239	0.233	0.243

Table 5.5 details the statistical significance of the F-measure results shown in Table 5.4 and summarises the results for precision and recall. These results show BO-2 and GA-2 performing very similarly, with not more than a difference of 0.2 in any of the rankings and no statistically significant differences in performance observed.

Table 5.5. Results of Wilcoxon signed-rank tests when comparing BO-2 against GA-2 regarding F-measure, Precision and Recall, for the 3 δ values on the biomedical datasets.

δ (%)	F-measure		Precision		Recall	
	Avg. ranks	p-value	Avg. ranks	p-value	Avg. ranks	p-value
20%	1.5 vs 1.5	0.514	**1.48** vs 1.52	0.394	**1.35** vs 1.65	0.760
40%	**1.45** vs 1.55	0.729	**1.48** vs 1.52	0.387	1.62 vs **1.38**	0.519
60%	1.6 vs **1.4**	0.388	1.58 vs **1.42**	0.387	1.55 vs **1.45**	0.784

Looking now to a comparison of the systems on the synthetic datasets, Table 5.6 shows the results comparing BO-1 and GA-1.

Table 5.6. F-measure results of BO-1 and GA-1 on synthetic datasets.

	δ = 20%		**δ = 40%**		**δ = 60%**	
Dataset	**GA-1**	**BO-1**	**GA-1**	**BO-1**	**GA-1**	**BO-1**
1	0.661	0.663	0.718	0.619	0.603	0.554
2	0.136	0.128	0.044	0.046	0.065	0.061
3	0.788	0.761	0.693	0.671	0.637	0.555
4	0.831	0.811	0.818	0.785	0.674	0.627
5	0.618	0.651	0.616	0.496	0.609	0.514
6	0.759	0.763	0.769	0.676	0.684	0.600
7	0.520	0.598	0.515	0.478	0.478	0.465
8	0.525	0.512	0.477	0.473	0.381	0.391
9	0.111	0.043	0.080	0.051	0.146	0.087
10	0.903	0.918	0.872	0.868	0.742	0.756
11	0.604	0.568	0.567	0.575	0.531	0.531
12	0.674	0.671	0.666	0.683	0.609	0.574
13	0.644	0.662	0.623	0.603	0.516	0.511
14	0.975	0.978	0.962	0.969	0.925	0.900
15	0.601	0.635	0.593	0.588	0.519	0.493
16	0.477	0.432	0.388	0.333	0.301	0.282
17	0.347	0.389	0.496	0.302	0.412	0.218
18	0.559	0.502	0.389	0.436	0.326	0.245
19	0.472	0.423	0.468	0.426	0.381	0.406
20	0.705	0.696	0.692	0.670	0.625	0.622

Table 5.7 summarises the statistical significance of the results from Table 5.6 (for synthetic datasets), as well as the results for precision and recall. Table 5.7 is structured in the same manner as Table 5.3. The results from Table 5.7 rather conclusively show GA-1 outperforming BO-1, overall. Whilst BO-1 performs best for precision across all values of δ, GA-1 performs best for both F-measure and recall across all values of δ, with statistical significance in 5 of the 6 cases (the significant p-values are shown in boldface in the table).

Table 5.7. Results of Wilcoxon signed-rank tests when comparing BO-1 against GA-1 regarding F-measure, Precision and Recall, for the 3 δ values on the synthetic datasets.

δ (%)	**F-measure**		**Precision**		**Recall**	
	Avg. ranks	**p-value**	**Avg. ranks**	**p-value**	**Avg. ranks**	**p-value**
20%	1.55 vs **1.45**	0.475	**1.45** vs 1.55	0.216	1.65 vs **1.35**	0.009
40%	1.75 vs **1.25**	0.006	**1.2** vs 1.8	0.007	1.75 vs **1.25**	0.007
60%	1.8 vs **1.2**	0.001	**1.3** vs 1.7	0.058	1.85 vs **1.15**	0.002

Next, Table 5.8 details the performance of BO-2 and GA-2 on the synthetic datasets and Table 5.9 summarises the statistical significance test results. These results differ from the results of Table 5.5, detailing the results of BO-2 and GA-2 applied to the biomedical datasets, with GA-2 largely

outperforming BO-2 for F-measure and recall on the synthetic datasets. The results for precision are more favourable for BO-2, performing best in 2/3 cases and achieving statistical significance in 1.

Table 5.8. F-measure results of BO-2 and GA-2 on synthetic datasets.

	δ = 20%		δ = 40%		δ = 60%	
Dataset	**BO-2**	**GA-2**	**BO-2**	**GA-2**	**BO-2**	**GA-2**
1	0.649	0.640	0.586	0.709	0.578	0.545
2	0.203	0.176	0.022	0.105	0.094	0.111
3	0.770	0.759	0.676	0.702	0.537	0.612
4	0.831	0.824	0.795	0.809	0.622	0.692
5	0.603	0.612	0.614	0.571	0.499	0.559
6	0.746	0.762	0.672	0.751	0.652	0.672
7	0.495	0.528	0.456	0.496	0.416	0.448
8	0.531	0.571	0.480	0.484	0.341	0.390
9	0.019	0.098	0.039	0.000	0.105	0.143
10	0.900	0.896	0.854	0.850	0.756	0.716
11	0.573	0.574	0.554	0.579	0.474	0.525
12	0.713	0.681	0.647	0.692	0.586	0.599
13	0.667	0.648	0.578	0.612	0.533	0.576
14	0.974	0.977	0.948	0.966	0.908	0.934
15	0.589	0.595	0.545	0.575	0.415	0.565
16	0.507	0.431	0.298	0.402	0.320	0.299
17	0.289	0.384	0.304	0.470	0.070	0.382
18	0.529	0.576	0.459	0.408	0.297	0.373
19	0.444	0.462	0.405	0.483	0.361	0.385
20	0.710	0.701	0.632	0.664	0.554	0.594

Table 5.9. Results of Wilcoxon signed-rank tests when comparing BO-2 against GA-2 regarding F-measure, Precision and Recall, for the 3 δ values on the synthetic datasets.

δ (%)	**F-measure**		**Precision**		**Recall**	
	Avg. ranks	**p-value**	**Avg. ranks**	**p-value**	**Avg. ranks**	**p-value**
20%	1.65 vs **1.35**	0.08	**1.2** vs 1.8	**0.001**	1.95 vs **1.05**	**0.00003**
40%	1.85 vs **1.15**	**0.002**	1.5 vs 1.5	0.812	1.75 vs **1.25**	**0.002**
60%	1.85 vs **1.15**	**0.001**	**1.25** vs 1.75	0.177	1.85 vs **1.15**	**0.0001**

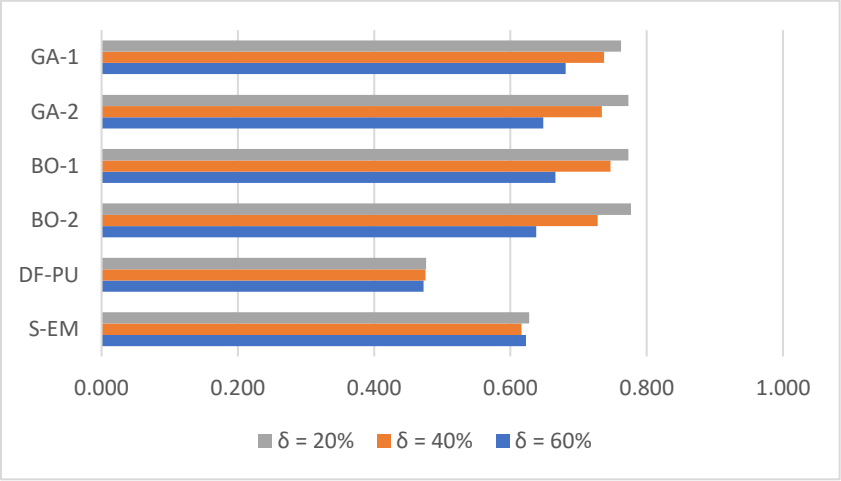


Figure 5.2. Average F-measure results comparison for BO-1, BO-2, GA-1, GA-2, DF-PU and S-EM, across the three values of δ for the biomedical datasets.

Figure 5.2 shows graphically how the average F-measure of each of the Auto-PU systems and the baseline methods changes over the different values of δ for the biomedical datasets. In order to reduce the number of figures across this chapter, this figure shows the results for all Auto-PU systems (BO-1, BO-2, GA-1, GA-2) and all baseline PU learning methods (DF-PU and S-EM) investigated in this chapter, but in this current part of the text the analysis is focussed on the results for BO-1, BO-2, GA-1, and GA-2 only – the results for DF-PU and S-EM were discussed in Chapter 4. Note that the charted data for GA-1, GA-2, DF-PU, and S-EM were also shown in Chapter, 4 but have been included here for the reader's reference.

As with the results for GA-1 and GA-2, discussed in Section 4.3, the performance of BO-1 and BO-2 does also decline monotonically with the increase in the value of δ. The decline for δ=60% for BO-1 and BO-2 is sharper than that of GA-1, which was reflected in the results shown in Tables 5.2 and 5.4.

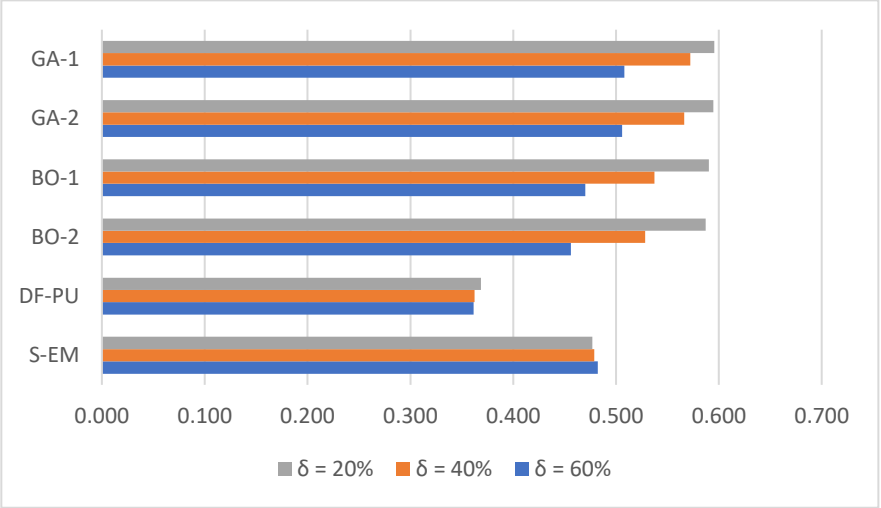


Figure 5.3. Average F-measure results comparison for BO-1, BO-2, GA-1, GA-2, DF-PU and S-EM, across the three values of δ for the synthetic datasets.

Figure 5.3 shows graphically how the average F-measure of each of the systems changes over the different values of δ, for the synthetic datasets. It is evident that GA-1 performs better across the values of δ, given the overall lower average value of F-measure for BO-1 and BO-2 and the sharper decline in performance as δ increases.

Table 5.10. Linear (Pearson's) correlation coefficient value between the F-measure and the percentage of positive examples in the original dataset (before hiding some positive examples in the unlabelled set) for each combination of a method and a δ value, for the biomedical datasets, for all methods.

Method	δ = 20%	δ = 40%	δ = 60%
BO-1	0.398	0.360	0.498
BO-2	0.339	0.348	0.225
GA-1	0.333	0.385	0.504
GA-2	0.340	0.357	0.580
DF-PU	0.988	0.988	0.988
S-EM	0.646	0.558	0.652

In order to further analyse the results, Table 5.10 shows the values of Pearson's linear correlation coefficient between the F-measure values achieved by BO-1, BO-2, GA-1, GA-2, DF-PU, and S-EM and percentages of positive examples in the original dataset, for each δ value, for the biomedical datasets. Again, in order to reduce the number of tables across this chapter, the results for all the aforementioned systems or methods are reported in Table 5.10, but in this current part of the text the analysis is focused on the results for BO-1, BO-2, GA-1 and GA-2 – the results for the baseline methods were discussed in Chapter 4.

Table 5.10 shows the same trends of positive correlations as Figure 5.2. For δ=20% and 40%, BO-1 and BO-2 exhibit a weak correlation between percentage of positive instances and F-measure, whilst for δ=60% the correlation is moderate, with correlation categorisation defined as outlined in Section 3.4.3. These correlation values are reflective of the results for GA-1 and GA-2, which follow a similar trend.

Table 5.11. Linear (Pearson's) correlation coefficient value between the F-measure and the percentage of positive examples in the original dataset (before hiding some positive examples in the unlabelled set) for each combination of a method and a δ value, for the synthetic datasets, for all methods.

Method	δ = 20%	δ = 40%	δ = 60%
BO-1	0.682	0.710	0.700
BO-2	0.706	0.695	0.659
GA-1	0.712	0.682	0.687
GA-2	0.700	0.702	0.696
DF-PU	0.990	0.990	0.990
S-EM	0.794	0.793	0.776

Table 5.11 shows the values of Pearson's linear correlation coefficient between the F-measure values achieved by BO-1, BO-2, GA-1, GA-2, DF-PU, and S-EM and the percentages of positive examples in the original dataset, for each δ value for the synthetic datasets. Comparing the BO-1 and BO-2 results in Tables 5.10 and 5.11, the correlation between percentage of positive instances and F-measure for both is substantially higher for the synthetic datasets than it was for the biomedical

datasets. (A similar trend was also observed for GA-1 and GA-2 in Chapter 4.) For δ=20%, the correlation for BO-1 was moderate in Table 5.7. However, for δ=40% and 60%, BO-1 displayed a high correlation. This was the opposite of GA-1 and BO-2, which displayed high correlations for δ=20% and moderate correlations for δ=40% and 60%. GA-2 displayed high correlations for δ=20% and 40%. However, the real differences in the correlation values were very small.

Considering the overall results in this section, it is hard to draw a clear conclusion regarding the best system. Regarding predictive performance, the results are somewhat in favour of GA-1. GA-1 was outperformed by BO-1 by most of the results shown in Table 5.3, but without statistical significance. Whereas GA-1 performed best by the results shown in Table 5.8 with statistical significance. Comparing BO-2 and GA-2, the results in Table 5.5 show very similar performance between the two systems, whilst the results in Table 5.9 show GA-2 largely outperforming BO-2, with statistical significance in many cases. However, the BO-Auto-PU systems offer a key advantage over GA-Auto-PU in regard to computational runtime. GA-1 took 226.3 minutes on average to run a 5-fold cross-validation procedure per dataset, averaged over the two types of dataset, whilst BO-1 took only 8.4 minutes. GA-2 took 223.2 minutes on average whilst BO-2 took 9.8 minutes. The learning curves of the systems are reported in Section 6.5. Thus, BO-Auto-PU performed approximately 23-27 times faster than GA-Auto-PU. It is important to note that one of the primary aims of Auto-ML is to make machine learning more accessible to users or data analysis without expertise on machine learning. GA-Auto-PU makes machine learning more accessible by removing the need for trial-and-error approaches to PU learning parameter tuning. However, it falls short in that the computational expense needed may make the system impractical for some users or data analysts. This is the primary contribution of BO-Auto-PU; BO-Auto-PU still achieves high predictive performance, but at a much lower computational expense than GA-Auto-PU, thus making it more accessible.

5.3.2 Results comparing BO-Auto-PU with two baseline PU learning methods

This section details the results achieved by BO-Auto-PU and two baseline PU learning methods (DF-PU and S-EM, see Section 2.5) when applied to 20 real-world biomedical datasets and 20 synthetic

datasets. Note that the results in the BO-1 and BO-2 columns in the tables of results reported in this section are the same as those reported in the previous section, but they are repeated in this section for the reader's convenience.

Table 5.12. F-measure results of BO-1 and baseline PU learning methods on real-world biomedical datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**BO-1**	**DF-PU**	**S-EM**	**BO-1**	**DF-PU**	**S-EM**	**BO-1**	**DF-PU**	**S-EM**
Alzheimer's	0.615	0.195	0.321	0.600	0.194	0.370	0.436	0.171	0.373
Autism	0.967	0.648	0.820	0.956	0.648	0.841	0.863	0.645	0.835
Breast cancer Coi.	0.694	0.697	0.711	0.701	0.711	0.704	0.586	0.697	0.699
Breast cancer Wis.	0.949	0.543	0.898	0.969	0.543	0.903	0.895	0.539	0.904
Breast cancer mut.	0.893	0.489	0.892	0.873	0.489	0.893	0.841	0.485	0.892
Cervical cancer	0.839	0.061	0.054	0.903	0.042	0.053	0.645	0.044	0.046
Cirrhosis	0.545	0.405	0.436	0.529	0.401	0.442	0.489	0.405	0.459
Dermatology	0.872	0.228	0.718	0.905	0.229	0.718	0.725	0.219	0.719
PI Diabetes	0.647	0.516	0.534	0.645	0.516	0.525	0.594	0.515	0.544
ES Diabetes	0.983	0.762	0.792	0.877	0.756	0.859	0.902	0.759	0.793
Heart Disease	0.844	0.705	0.811	0.830	0.705	0.828	0.777	0.702	0.829
Heart Failure	0.753	0.487	0.529	0.605	0.486	0.508	0.704	0.481	0.557
Hepatitis C	0.907	0.176	0.695	0.838	0.171	0.708	0.708	0.160	0.609
Kidney Disease	0.988	0.428	1.000	0.964	0.428	1.000	0.806	0.428	0.951
Liver Disease	0.820	0.834	0.816	0.817	0.832	0.587	0.795	0.834	0.788
Maternal Risk	0.837	0.403	0.454	0.780	0.395	0.433	0.689	0.390	0.438
Parkinsons	0.935	0.856	0.815	0.875	0.860	0.748	0.732	0.860	0.762
Parkinsons Biom.	0.167	0.354	0.333	0.192	0.354	0.261	0.182	0.367	0.331
Spine	0.954	0.652	0.820	0.926	0.652	0.839	0.742	0.652	0.830
Stroke	0.244	0.086	0.102	0.153	0.094	0.102	0.208	0.094	0.102

Table 5.13. Results of Wilcoxon signed-rank tests with Holm correction for multiple hypothesis when comparing BO-1 against S-EM and DF-PU regarding F-measure, Precision and Recall, for the 3 δ values for the biomedical datasets.

δ (%)	**Methods compared**	**F-measure**			**Precision**			**Recall**		
		Avg. ranks	**p-value**	**α**	**Avg. ranks**	**p-value**	**α**	**Avg. ranks**	**p-value**	**α**
20%	BO-1 vs DF-PU	**1.15** vs 1.85	**0.0001**	0.025	**1.05** vs 1.95	**0.00001**	0.025	2.0 vs **1.0**	**0.000002**	0.025
	BO-1 vs S-EM	**1.15** vs 1.85	**0.0009**	0.05	**1.12** vs 1.88	**0.0005**	0.05	1.7 vs **1.3**	**0.0266**	0.05
40%	BO-1 vs DF-PU	**1.15** vs 1.85	**0.0002**	0.025	**1.0** vs 2.0	**0.000002**	0.025	1.98 vs **1.02**	**0.0001**	0.025
	BO-1 vs S-EM	**1.2** vs 1.8	**0.0006**	0.05	**1.08** vs 1.92	**0.0003**	0.05	1.75 vs **1.25**	**0.021**	0.05
60%	BO-1 vs DF-PU	**1.2** vs 1.8	**0.002**	0.025	**1.05** vs 1.95	**0.000004**	0.025	2.0 vs **1.0**	**0.000002**	0.025
	BO-1 vs S-EM	**1.4** vs 1.6	0.498	0.05	**1.12** vs 1.88	**0.0002**	0.05	1.98 vs **1.02**	**0.0001**	0.05

Table 5.13 summarises the statistical significance of the F-measure results from Table 5.12, as well as the results for precision and recall. In Table 5.13, for each combination of a performance measure (F-measure, precision, recall) and a δ value (δ= 20%, 40%, 60%), the table reports the average (Avg.) rank of BO-1 vs a baseline method (BO-1 is the left rank, the baseline is the right one), as well as the corresponding p-value and adjusted α value (significance level). The better (lower) avg. rank in each

cell is shown in boldface; and significant p-values (smaller than the adjusted α) are also shown in boldface. The following discussion of results will focus mainly on the F-measure, the most important measure in Table 5.13, whilst precision and recall results are reported for completeness.

Table 5.13 shows BO-1 largely outperforming DF-PU and S-EM across all values of δ for F-measure and precision with statistical significance in 11 out of 12 cases. As was the case for GA-1, the baseline methods substantially outperform BO-1 for recall, with statistical significance in all 6 cases. However, the reasons for this are the same as those outlined in Section 4.3, namely that the baseline methods massively overpredict the positive class, thus resulting in high recall but low precision.

Next, Table 5.14 reports the results of BO-2 against the baseline methods on the biomedical datasets and Table 5.15 summarises the statistical significance test results. As has been the case with the previously discussed methods, BO-2 largely outperforms the baselines for F-measure and precision across all values of δ, with statistical significance in 11 out of 12 cases. As has been previously discussed, the baseline methods overpredict the positive class, achieving significantly higher recall than BO-2 in all 6 cases, but at a substantial loss to precision.

Table 5.14. F-measure results of BO-2 and two baseline PU learning methods on real-world biomedical datasets.

	δ = 20%			**δ = 40%**			**δ = 60%**		
Dataset	**BO-2**	**DF-PU**	**S-EM**	**BO-2**	**DF-PU**	**S-EM**	**BO-2**	**DF-PU**	**S-EM**
Alzheimer's	0.580	0.195	0.321	0.603	0.194	0.370	0.492	0.171	0.373
Autism	0.963	0.648	0.820	0.937	0.648	0.841	0.914	0.645	0.835
Breast cancer Coi.	0.667	0.697	0.711	0.618	0.711	0.704	0.000	0.697	0.699
Breast cancer Wis.	0.959	0.543	0.898	0.942	0.543	0.903	0.889	0.539	0.904
Breast cancer mut.	0.890	0.489	0.892	0.853	0.489	0.893	0.845	0.485	0.892
Cervical cancer	0.867	0.061	0.054	0.867	0.042	0.053	0.839	0.044	0.046
Cirrhosis	0.497	0.405	0.436	0.515	0.401	0.442	0.472	0.405	0.459
Dermatology	0.876	0.228	0.718	0.841	0.229	0.718	0.795	0.219	0.719
PI Diabetes	0.653	0.516	0.534	0.648	0.516	0.525	0.615	0.515	0.544
ES Diabetes	0.954	0.762	0.792	0.891	0.756	0.859	0.912	0.759	0.793
Heart Disease	0.844	0.705	0.811	0.817	0.705	0.828	0.805	0.702	0.829
Heart Failure	0.757	0.487	0.529	0.652	0.486	0.508	0.600	0.481	0.557
Hepatitis C	0.964	0.176	0.695	0.761	0.171	0.708	0.612	0.160	0.609
Kidney Disease	0.976	0.428	1.000	0.976	0.428	1.000	0.789	0.428	0.951
Liver Disease	0.822	0.834	0.816	0.815	0.832	0.587	0.722	0.834	0.788
Maternal Risk	0.847	0.403	0.454	0.786	0.395	0.433	0.729	0.390	0.438
Parkinsons	0.936	0.856	0.815	0.837	0.860	0.748	0.800	0.860	0.762
Parkinsons Biom.	0.286	0.354	0.333	0.000	0.354	0.261	0.000	0.367	0.331
Spine	0.941	0.652	0.820	0.936	0.652	0.839	0.700	0.652	0.830
Stroke	0.256	0.086	0.102	0.255	0.094	0.102	0.233	0.094	0.102

Table 5.15. Results of Wilcoxon signed-rank tests with Holm correction for multiple hypothesis when comparing BO-2 against S-EM and DF-PU regarding F-measure, Precision and Recall, for the 3 δ values for the biomedical datasets.

δ (%)	Methods compared	F-measure			Precision			Recall		
		Avg. ranks	p-value	α	Avg. ranks	p-value	α	Avg. ranks	p-value	α
20%	BO-2 vs DF-PU	**1.1** vs 1.9	**0.0002**	0.025	**1.0** vs 2.0	**0.000002**	0.025	2.0 vs **1.0**	**0.000002**	0.025
	BO-2 vs S-EM	**1.15** vs 1.85	**0.001**	0.05	**1.08** vs 1.92	**0.0001**	0.05	1.75 vs **1.25**	**0.024**	0.05
40%	BO-2 vs DF-PU	**1.2** vs 1.8	**0.001**	0.025	**1.05** vs 1.95	**0.00004**	0.025	2.0 vs **1.0**	**0.000002**	0.025
	BO-2 vs S-EM	**1.25** vs 1.75	**0.006**	0.05	**1.12** vs 1.88	**0.001**	0.05	1.8 vs **1.2**	**0.006**	0.05
60%	BO-2 vs DF-PU	**1.2** vs 1.8	**0.019**	0.025	**1.1** vs 1.9	**0.0004**	0.025	1.95 vs **1.05**	**0.000004**	0.025
	BO-2 vs S-EM	**1.4** vs 1.6	0.571	0.05	**1.12** vs 1.88	**0.0006**	0.05	1.85 vs **1.15**	**0.0001**	0.5

Moving on to the synthetic datasets, Table 5.16 details the results comparing BO-1 vs the baseline methods and Table 5.17 summarises the statistical significance tests. Table 5.16 shows BO-1 largely outperforming DF-PU and S-EM across all values of δ for F-measure and precision, with statistical significance in 11 out of 12 cases. As was the case for GA-1, the baseline methods substantially outperform BO-1 for recall, with statistical significance in all 6 cases. However, the reasons for this are the same as those outlined in Section 4.3, namely that the baseline methods massively overpredict the positive class, thus resulting in high recall but low precision.

Table 5.16. F-measure results of BO-1 and baseline PU learning methods on synthetic datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**BO-1**	**DF-PU**	**S-EM**	**BO-1**	**DF-PU**	**S-EM**	**BO-1**	**DF-PU**	**S-EM**
1	0.663	0.484	0.616	0.619	0.484	0.602	0.554	0.483	0.613
2	0.128	0.125	0.194	0.046	0.120	0.130	0.061	0.112	0.120
3	0.761	0.552	0.589	0.671	0.552	0.587	0.555	0.552	0.600
4	0.811	0.454	0.644	0.785	0.416	0.633	0.627	0.417	0.630
5	0.651	0.357	0.402	0.496	0.356	0.436	0.514	0.357	0.465
6	0.763	0.403	0.477	0.676	0.402	0.525	0.600	0.402	0.582
7	0.598	0.285	0.433	0.478	0.283	0.462	0.465	0.282	0.451
8	0.512	0.326	0.468	0.473	0.326	0.457	0.391	0.326	0.439
9	0.043	0.035	0.099	0.051	0.000	0.044	0.087	0.000	0.120
10	0.918	0.233	0.612	0.868	0.234	0.627	0.756	0.233	0.663
11	0.568	0.491	0.505	0.575	0.491	0.520	0.531	0.490	0.517
12	0.671	0.397	0.550	0.683	0.397	0.567	0.574	0.394	0.586
13	0.662	0.500	0.551	0.603	0.460	0.556	0.511	0.456	0.549
14	0.978	0.529	0.817	0.969	0.529	0.840	0.900	0.529	0.873
15	0.635	0.387	0.423	0.588	0.387	0.425	0.493	0.385	0.422
16	0.432	0.239	0.414	0.333	0.239	0.401	0.282	0.240	0.299
17	0.389	0.214	0.262	0.302	0.214	0.281	0.218	0.214	0.267
18	0.502	0.372	0.444	0.436	0.373	0.433	0.245	0.373	0.422
19	0.423	0.378	0.426	0.426	0.378	0.429	0.406	0.376	0.413
20	0.696	0.610	0.615	0.670	0.610	0.620	0.622	0.610	0.613

Table 5.17. Results of Wilcoxon signed-rank tests with Holm correction for multiple hypothesis when comparing BO-1 against S-EM and DF-PU regarding F-measure, Precision and Recall, for the 3 δ values for the synthetic datasets.

δ (%)	Methods compared	F-measure			Precision			Recall		
		Avg. ranks	p-value	α	Avg. ranks	p-value	α	Avg. ranks	p-value	α
20%	BO-1 vs DF-PU	**1.0** vs 2.0	**0.000002**	0.025	**1.0** vs 2.0	**0.000002**	0.025	1.95 vs **1.05**	**0.00001**	0.025
	BO-1 vs S-EM	**1.15** vs 1.85	**0.0002**	0.05	**1.1** vs 1.9	**0.00003**	0.05	1.75 vs **1.25**	**0.002**	0.05
40%	BO-1 vs DF-PU	**1.05** vs 1.95	**0.00002**	0.025	**1.0** vs 2.0	**0.000002**	0.025	1.95 vs **1.05**	**0.000004**	0.05
	BO-1 vs S-EM	**1.15** vs 1.85	**0.002**	0.05	**1.1** vs 1.9	**0.00001**	0.05	2.0 vs **1.0**	**0.000002**	0.025
60%	BO-1 vs DF-PU	**1.1** vs 1.9	**0.0007**	0.025	**1.0** vs 2.0	**0.000002**	0.025	1.95 vs **1.05**	**0.000004**	0.05
	BO-1 vs S-EM	1.6 vs **1.4**	0.410	0.05	**1.15** vs 1.85	**0.0001**	0.05	2.0 vs **1.0**	**0.000002**	0.025

Table 5.19 details the statistical significance of the F-measure results shown in Table 5.18 and summarises the results for precision and recall. BO-2 again largely outperforms the baseline methods regarding F-measure and precision for all values of δ, with the exception of S-EM for F-measure when δ=60%. The difference in F-measure and precision values is statistically significant in all cases when BO-2 outperforms the baseline methods (i.e., in 11 out of 12 cases). For recall, as expected based on previous results, the baselines outperform BO-2 for all δ values with statistical significance.

Table 5.18. F-measure results of BO-2 and two baseline PU learning methods on synthetic datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**BO-2**	**DF-PU**	**S-EM**	**BO-2**	**DF-PU**	**S-EM**	**BO-2**	**DF-PU**	**S-EM**
1	0.649	0.484	0.616	0.586	0.484	0.602	0.578	0.483	0.613
2	0.203	0.125	0.194	0.022	0.120	0.130	0.094	0.112	0.120
3	0.770	0.552	0.589	0.676	0.552	0.587	0.537	0.552	0.600
4	0.831	0.454	0.644	0.795	0.416	0.633	0.622	0.417	0.630
5	0.603	0.357	0.402	0.614	0.356	0.436	0.499	0.357	0.465
6	0.746	0.403	0.477	0.672	0.402	0.525	0.652	0.402	0.582
7	0.495	0.285	0.433	0.456	0.283	0.462	0.416	0.282	0.451
8	0.531	0.326	0.468	0.480	0.326	0.457	0.341	0.326	0.439
9	0.019	0.035	0.099	0.039	0.000	0.044	0.105	0.000	0.120
10	0.900	0.233	0.612	0.854	0.234	0.627	0.756	0.233	0.663
11	0.573	0.491	0.505	0.554	0.491	0.520	0.474	0.490	0.517
12	0.713	0.397	0.550	0.647	0.397	0.567	0.586	0.394	0.586
13	0.667	0.500	0.551	0.578	0.460	0.556	0.533	0.456	0.549
14	0.974	0.529	0.817	0.948	0.529	0.840	0.908	0.529	0.873
15	0.589	0.387	0.423	0.545	0.387	0.425	0.415	0.385	0.422
16	0.507	0.239	0.414	0.298	0.239	0.401	0.320	0.240	0.299
17	0.289	0.214	0.262	0.304	0.214	0.281	0.070	0.214	0.267
18	0.529	0.372	0.444	0.459	0.373	0.433	0.297	0.373	0.422
19	0.444	0.378	0.426	0.405	0.378	0.429	0.361	0.376	0.413
20	0.710	0.610	0.615	0.632	0.610	0.620	0.554	0.610	0.613

Table 5.19. Results of Wilcoxon signed-rank tests with Holm correction for multiple hypothesis when comparing BO-2 against S-EM and DF-PU regarding F-measure, Precision and Recall, for the 3 δ values for the synthetic datasets.

δ (%)	Methods compared	F-measure			Precision			Recall		
		Avg. ranks	p-value	α	Avg. ranks	p-value	α	Avg. ranks	p-value	α
20%	BO-2 vs DF-PU	**1.05** vs 1.95	**0.00001**	0.025	**1.0** vs 2.0	**0.000002**	0.025	1.95 vs **1.05**	**0.00001**	0.05
	BO-2 vs S-EM	**1.1** vs 1.9	**0.0005**	0.05	**1.05** vs 1.95	**0.000004**	0.05	2.0 vs **1.0**	**0.000002**	0.025
40%	BO-2 vs DF-PU	**1.05** vs 1.95	**0.00004**	0.025	**1.05** vs 1.95	**0.000004**	0.025	1.9 vs **1.1**	**0.00001**	0.05
	BO-2 vs S-EM	**1.3** vs 1.7	**0.017**	0.05	**1.08** vs 1.92	**0.00002**	0.05	2.0 vs **1.0**	**0.000002**	0.025
60%	BO-2 vs DF-PU	**1.35** vs 1.65	**0.019**	0.025	**1.0** vs 2.0	**0.000002**	0.025	1.95 vs **1.05**	**0.000004**	0.025
	BO-2 vs S-EM	1.75 vs **1.25**	0.083	0.05	**1.15** vs 1.85	**0.0003**	0.05	1.92 vs **1.08**	**0.00001**	0.05

5.4 The PU Learning Algorithm's Hyperparameter Values Most Frequently Selected by BO-Auto-PU

This section reports the optimised PU learning algorithm's hyperparameter values which were most frequently selected by BO-Auto-PU utilising the base search space (BO-1) and the extended search space (BO-2). It reports the selection frequency, baseline frequency, and their difference. The selection frequency of a PU learning algorithm's hyperparameter value is calculated as the ratio of the number of times that value was used in the optimised PU learning algorithm returned by BO-Auto-PU over the total number of BO-Auto-PU runs, which is 300 per type of dataset (biomedical or synthetic), considering 20 datasets times 3 values of δ times 5 runs of a BO-Auto-PU version per dataset, due to the use of 5-fold cross-validation. The baseline frequency is the expected selection frequency of a hyperparameter value if all values of that hyperparameter were randomly selected for use in a PU learning algorithm. I.e., it is calculated by simply dividing 1 (one) by the number of candidate values for that hyperparameter. The difference between these two frequencies is simply the selection frequency minus the baseline frequency.

As mentioned in Section 4.4, little has been written on the topic of suitable algorithm configuration for PU learning, and no guidelines exist in the literature. Hence, the information about

the most frequently selected hyperparameter values in our experiments can give some insights about how to improve the performance of PU learning algorithms.

Throughout this section, the term "classifier" is used to refer to a classification algorithm (rather than a classification model learned by an algorithm), unless explicitly mentioned otherwise.

5.4.1 The Hyperparameter Values Most Frequently Selected by BO-1

Tables 5.20 and 5.21 report the most frequently selected values of the hyperparameters of the optimised PU learning algorithms returned by all runs of BO-1 on the biomedical datasets and synthetic datasets, respectively. The most frequently selected *Phase 1A Classifiers* were naïve Bayes classifiers in both Table 5.20 and Table 5.21, i.e., Bernoulli for the biomedical datasets and Gaussian for the synthetic datasets. This reinforces the hypothesis from the previous chapter that the Auto-PU system favours simple classifiers in the early phase.

The values most frequently selected for *Phase 1B Classifiers* (histogram-based boosting classifier (HGBoost) and support vector machine (SVM)) give little in the way of obvious conclusions to be drawn, as with the results in Chapter 4. Perhaps this is an indication that, whilst there is a trend emerging for *Phase 1A Classifiers*, the choice of *Phase 1B Classifier* is somewhat less determined.

Table 5.20. Selection frequency of hyperparameter values by BO-1 for the biomedical datasets.

Hyperparameter	Most selected value	Selection Freq. (%)	Baseline Freq. (%)	Diff. (%)
Phase 1A Iteration count	2	19.00	10.00	9.00
Phase 1A RN Threshold	0.05	14.33	10.00	4.33
Phase 1A Classifier	Bernoulli NB	8.67	5.56	3.11
Phase 1B Flag	TRUE	52.67	50.00	2.67
Phase 1 B RN Threshold	0.2	14.00	10.00	4.00
Phase 1B Classifier	HGBoost	8.00	5.56	2.44
Phase 2 Classifier	LDA	32.67	5.56	27.11

The most frequently selected values for *Phase 2 Classifier* are linear discriminant analysis (LDA) for the biomedical datasets and deep forest for the synthetic datasets, which are two classifiers which also came up frequently when analysing the results for GA-Auto-PU. LDA is a linear classifier, and

thus fits with the previously discussed assumptions of separability and smoothness. Deep forest is a powerful classifier (see Section 2.1) so it is little surprise that it performs well.

Table 5.21. Selection frequency of hyperparameter values by BO-1 for the synthetic datasets.

Hyperparameter	Most selected value	Selection Freq. (%)	Baseline Freq. (%)	Diff. (%)
Phase 1A Iteration count	4	14.67	10.00	4.67
Phase 1A RN Threshold	0.45	13.00	10.00	3.00
Phase 1A Classifier	Gaussian NB	16.00	5.56	10.44
Phase 1B Flag	FALSE	52.00	50.00	2.00
Phase 1 B RN Threshold	0.45	14.67	10.00	4.67
Phase 1B Classifier	SVM	10.33	5.56	4.77
Phase 2 Classifier	Deep Forest	23.67	5.56	18.11

The *Phase 1B Flag*, as with the results of for GA-Auto-PU in Chapter 4, is almost exactly a 50/50 split for both biomedical and synthetic datasets.

Interestingly, the *Phase 1A RN Threshold* hyperparameter for the biomedical datasets has the lowest value, 0.05, but the second highest value for synthetic datasets, 0.45. However, these values were selected by such a small margin from the baseline frequency that no conclusions can be drawn from this.

As in Chapter 4 analysing the results of GA-Auto-PU, a more detailed analysis of the *Phase 1A Iteration Count* hyperparameter's optimisation is reported next. As was the case for GA-Auto-PU, there is a moderate correlation between iteration count and the percentage of instances in the dataset that belong to the positive class, indicating that the hyperparameter is utilised to handle class imbalance. For the biomedical datasets, the correlation coefficient values are -0.677, -0.700, and -0.700 for δ = 20%, 40%, and 60% respectively. For the synthetic datasets, the values are -0.656, -0.692, and -0.683 for δ = 20%, 40%, and 60% respectively. The correlations for the biomedical datasets when δ = 40% and 60% are classified as strong, but only just at the 0.7 threshold from moderate to strong.

5.4.2 The Hyperparameter Values Most Frequently Selected by BO-2

Tables 5.22 and 5.23 report the most frequently selected values of the hyperparameters of the optimised PU learning algorithms returned by all runs of BO-2 on the biomedical datasets and

synthetic datasets, respectively. The most frequently selected values for *Phase 1A Classifier* were logistic regression for the biomedical datasets and Gaussian naïve Bayes (NB) for the synthetic datasets, which are two classifiers that have also come up frequently throughout this work in the previous hyperparameter analysis for *Phase 1A Classifier.* This further reinforces the hypothesis regarding the assumptions of separability and smoothness.

Table 5.22. Selection frequency of hyperparameter values by BO-2 for the biomedical datasets.

Hyperparameter	**Most selected value**	**Selection Freq. (%)**	**Baseline Freq. (%)**	**Diff. (%)**
Phase 1A Iteration count	2	21.00	10.00	11.00
Phase 1A RN Threshold	0.25	13.00	10.00	3.00
Phase 1A Classifier	Logistic reg.	8.67	5.56	3.11
Phase 1B Flag	TRUE	50.67	50.00	0.67
Phase 1 B RN Threshold	0.2	14.67	10.00	4.67
Phase 1B Classifier	Bagging clas.	7.67	5.56	2.11
Spy rate	0.3	18.00	14.29	3.71
Spy tolerance	0.08	12.18	9.09	3.09
Spy flag	FALSE	74.00	50.00	24.00
Phase 2 Classifier	LDA	51.67	5.56	46.11

Table 5.23. Selection frequency of hyperparameter values by BO-2 for the synthetic datasets.

Hyperparameter	**Most selected value**	**Selection Freq. (%)**	**Baseline Freq. (%)**	**Diff. (%)**
Phase 1A Iteration count	2	16.67	10.00	6.67
Phase 1A RN Threshold	0.3	17.59	10.00	7.59
Phase 1A Classifier	Gaussian NB	17.00	5.56	11.44
Phase 1B Flag	FALSE	55.33	50.00	5.33
Phase 1 B RN Threshold	0.45	17.33	10.00	7.33
Phase 1B Classifier	Deep Forest	11.00	5.56	5.44
Spy rate	0.2	29.00	14.29	14.71
Spy tolerance	0.03	14.05	9.09	4.96
Spy flag	FALSE	88.00	50.00	38.00
Phase 2 Classifier	Deep Forest	20.67	5.56	15.11

For *Phase 1B Classifier*, the bagging classifier was most frequently selected for the biomedical datasets. This is a classifier that has not yet come up in the hyperparameter analysis. By contrast, deep forest was most frequently selected for the synthetic datasets. This is a classifier which, like logistic regression and Gaussian NB, is appearing frequently in the analysis of most frequently selected classifiers.

The *Phase 2 Classifier* presents some interesting results – deep forest was selected most frequently for the synthetic datasets, which is unsurprising given the previous results, and linear

discriminant analysis (LDA) is selected most frequently for the biomedical datasets. The selection of LDA is not, itself, surprising, given the frequency at which has been selected in the previous analysis. However, it was selected for over half of all optimised candidate solutions, which is very substantial, considering that there are 18 candidate classifiers. For the acquisition function of BO-Auto-PU, we used simply the value predicted by the random forest regressor, as opposed to a more exploratory approach, such as Expected Improvement (see Section 2.3.2). This means that rather than focusing on exploring the search space, BO-Auto-PU focuses on exploiting known well-performing areas of the search space. This approach has served relatively well, given the predictive performance results, but could be the reason as to why the LDA classifier occurs so frequently for *Phase 2 Classifier* for the biomedical datasets. That is, it could be that candidate solutions utilising LDA for *Phase 2 Classifier* performed well in the search space, and as such BO-Auto-PU selected those candidate solutions more frequently for objective function assessment.

Spy-based methods were, again, not preferred in most cases, with a value of "False" selected for the Spy flag hyperparameter in 74% of cases for the biomedical datasets, and 88% of cases for the synthetic datasets. These results, along with those from Section 4.4, are beginning to show an emerging lack of favour towards the spy methods.

Looking now towards a more in depth analysis of the *Phase 1A Iteration Count* hyperparameter, BO-2 displayed moderate to strong correlations between the average value of *Phase 1A Iteration Count* and the percentage of positive instances in the original, unaltered datasets, with observed correlation coefficient values of -0.641, -0.706, and -0.736 for the biomedical datasets when $\delta = 20\%$, 40%, and 60%, respectively; and -0.772, -0.667, and -0.674 for the synthetic datasets when $\delta = 20\%$, 40%, and 60%, respectively.

5.5 Summary

Overall, considering a comparison between the two BO-based systems and the two baseline PU learning methods, both BO-1 and BO-2 outperformed the baseline methods in general, with statistical significance regarding F-measure and precision in several cases. S-EM did, however, show a large increase in performance for the $\delta = 60\%$ datasets, outperforming both BO-1 and BO-2 in terms of F-

measure for δ = 60% on the synthetic datasets. Whilst the F-measures of BO-1, BO-2, and DF-PU declined by 0.121, 0.131, and 0.007 respectively from δ = 20% to 60%, the performance of S-EM made a marginal increase of 0.005.

The results when comparing the BO-based systems against the GA-based systems, in terms of predictive performance, were mixed. Regarding predictive performance, for the biomedical datasets there was very little difference between the two systems, with no statistical significance observed. However, for the synthetic datasets, GA-1 arguably outperformed BO-1.

However, the BO-based methods do have a very distinct advantage over the GA-based methods with regard to computational runtime. The BO-based methods were developed in an attempt to improve upon the long runtime required by the GA-based methods, ideally simultaneously increasing (or at least not decreasing) predictive performance. Whilst predictive performance was not increased, there was no practical loss in the real-world biomedical datasets, arguably the most important of the two types of datasets.

The considerations regarding computational efficiency noted in Section 5.1.3 are supported by the empirical runtimes observed in the experiments, namely: GA-1 took 226.3 minutes, GA-2 took 223.2 minutes, while BO-1 took 8.4 minutes and BO-2 took 9.8 minutes on average to run a 5-fold cross-validation per dataset (so, BO was about 23-27 times faster than GA). All experiments were run on a 48-core GPU with 256GB of memory.

At the beginning of this chapter, two research questions were posed. Firstly, does BO-Auto-PU improve on the performance of GA-Auto-PU in regard to computational efficiency? The response to this question is overwhelmingly *yes*, BO-Auto-PU runs much faster than GA-Auto-PU. The second research question was: does BO-Auto-PU improve on the performance of GA-Auto-PU in regard to predictive accuracy? The response to this question is less positive. BO-Auto-PU failed to improve upon GA-Auto-PU in regard to predictive accuracy, performing similarly for the biomedical datasets and somewhat poorly in regard to GA-Auto-PU for the synthetic datasets. Thus, the answer to the second question is no, BO-Auto-PU failed to improve upon the predictive accuracy of GA-Auto-PU and actually exhibited a small decline in performance overall.

To conclude, whilst the BO-based systems have greatly improved upon the GA-based systems with regard to computational runtime, the former do so at a loss to predictive performance when

utilizing the extended search space. Thus, finding a finer balance between the BO-based and the GA-based systems, looking to combine the speed of the BO with the population diversity of the GA, may improve upon both types of systems. It is this that has motivated the development of the EBO-Auto-PU system, proposed next in Chapter 6.

Chapter 6

A Hybrid Evolutionary/Bayesian Optimisation-based Auto-ML System for Positive-Unlabelled Learning (EBO-Auto-PU)

This chapter introduces EBO-Auto-PU, a hybrid Evolutionary/Bayesian Optimisation (EBO)-based Automated Machine Learning (Auto-ML) system for Positive-Unlabelled (PU) learning. GA-Auto-PU, detailed in Chapter 4, was the first Auto-ML system specific to PU learning, and showed statistically significant improvements in predictive performance over two baseline PU learning methods and an Auto-ML system for standard binary classification. However, the GA-Auto-PU system is computationally expensive, with GA-1 and GA-2 averaging 226.3 and 223.2 minutes, respectively, to run a 5-fold cross-validation per dataset. In an effort to reduce the computational expense of GA-Auto-PU, BO-Auto-PU was introduced in Chapter 5. BO-Auto-PU proved much more computationally efficient than GA-Auto-PU, with BO-1 and BO-2 averaging 8.4 and 9.8 minutes, respectively, to run a 5-fold cross-validation per dataset.

The improvement in computational efficiency, however, was gained at a small cost to predictive performance. Of the four systems tested (GA vs BO, both with two search spaces), GA-1 (with the base search space) was arguably the best performing system in terms of predictive accuracy, whilst BO-2 (with the extended search space) was undoubtably the worst performing system. It seems that the BO-based system was unable to cope with the expanded search space. It can be hypothesised that

the disparity in predictive performance was due to the differing levels of population diversity between the two systems. That is, GA-Auto-PU has large population diversity as it assesses many PU learning configurations in each iteration according to the objective (fitness) function and creates diversity through crossover and mutation applied to configurations selected based on their fitness (predictive accuracy, more precisely F-measure). Whereas BO-Auto-PU assesses only a single configuration at each iteration, selected according to the predictive accuracy value predicted by the random forest regressor. At each iteration, a population is randomly generated to be evaluated by the faster surrogate model, rather than evolved to be evaluated by the slower objective (fitness) function as in the case of GA-Auto-PU. So, whilst diversity may occur in the BO search, there is no guarantee that it will benefit the system as the diverse configurations may not be selected for assessment by the objective function.

EBO-Auto-PU was developed in an attempt to bridge this gap between the GA- and the BO-based systems, introducing diversity into BO by evolving a population rather than random population generation, whilst using a surrogate model to reduce computational expense and prioritise candidate solutions for assessment according to the objective function. To assess whether EBO-Auto-PU has been successful in this aim, two research questions are presented. Firstly, does EBO-Auto-PU present a good trade-off, regarding computational efficiency, between GA-Auto-PU and BO-Auto-PU? And, secondly, does EBO-Auto-PU achieve good predictive performance compared with GA-Auto-PU?

To clarify the first question, a ‘good’ trade-off can be considered as EBO-Auto-PU performing with a run time faster that of GA-Auto-PU. Given that EBO-Auto-PU evaluates more candidate solutions according to the objective function at each iteration than BO-Auto-PU (as will be explained later), it is extremely unlikely that EBO-Auto-PU will perform better than or on par with BO-Auto-PU in regard to computational runtime. Thus, setting an aim of improving upon the runtime of BO-Auto-PU would be an act of folly. To clarify the second question, EBO-Auto-PU should achieve good predictive performance compared with GA-Auto-PU, rather than BO-Auto-PU, as GA-Auto-PU outperformed BO-Auto-PU in regard to predictive performance. To this end, ‘good’ predictive performance can be considered predictive performance that exceeds that of GA-Auto-PU. These research questions will be address in the conclusions of this chapter.

The remainder of this chapter is structured as follows. Section 6.1 details EBO-Auto-PU. Section 6.2 describes the experimental setup. Section 6.3 compares EBO-Auto-PU against GA-Auto-PU and BO-Auto-PU, and compares EBO-Auto-PU against the two baseline PU learning methods used in Chapters 4 and 5. Section 6.4 analyses the most frequently selected hyperparameters of the EBO-Auto-PU system, and Section 6.5 summarises this chapter.

6.1 Description of EBO-Auto-PU

As outlined previously, EBO-Auto-PU is a hybrid Evolutionary/Bayesian Optimisation (EBO)-based Auto-ML system specific to PU learning. This section describes the pseudocodes detailing the procedure followed by the EBO-Auto-PU system. Details of the individual encoding of a candidate solution, as well the fitness (objective) function can be found in Chapter 3. Note that this section refers to standard GA procedures such as crossover, mutation, and tournament selection. These procedures are described in detail in Section 2.2.

6.1.1 The EBO Procedure for PU Learning

Procedure 6.1 outlines the procedure that the EBO component of EBO-Auto-PU follows to evolve a PU learning algorithm. *#Configs* PU learning configurations are randomly generated (step 1) and evaluated, with their F-measures saved as *Scores* (step 2). Note that this random population generation is the same procedure as described in Chapters 4 and 5 for GA-Auto-PU and BO-Auto-PU respectively. However, to briefly restate, for each candidate solution, for each component (hyperparameter) of that candidate solution, there are a fixed set of values that may be set. Each of these values has an equal probability of being assigned to that component. For example, the hyperparameter *Phase 1A iteration count* may take the values {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. As there are ten potential values, each of these have a 10% probability of being assigned. Regarding candidate solution evaluation, this is evaluation according to the objective function described in Section 3.2.3. Briefly described, this evaluation is 5-fold cross-validation on the training set, splitting the training set into 5 learning and validation sets, for which the average F-measure achieved by the PU learning

algorithm is calculated and saved as the candidate solutions score. The test set is untouched in this procedure.

Procedure 6.1 Outline of the Evolutionary Bayesian Optimization procedure for Positive-Unlabelled Learning

---

1. *Configs* = randomly generate *#Configs* PU learning configurations;
2. *Scores* = run objective function for all configurations in *Configs*; // see Procedures 3.1-3.4
3. Learn *Surr_model* with *Configs* as features, *Scores* as target;
4. **For** *#It_count* times:
   a. *Temp_Configs* = *Configs* after undergoing crossover and mutation;
   b. $\hat{Y}$ = calculate a surrogate score for each new config in *Temp_Configs* with *Surr_model*;
   c. *Best_config* = config with highest score according to $\hat{Y}$;
   d. *k_pop* = [ ],;
   e. **For** *k* times*:*
      i. *k_cand_sol* = select from Temp_*Configs* using tournament selection based on surrogate scores;
      ii. *k_pop* = *k_pop* ∪ *k_cand_sol*;
   f. *k_pop* = configurations in *k_pop* undergo crossover and mutation*;*
   g. Assess *Best_config* and each configuration from *k_pop* with objective function to obtain objective scores; // see Procedures 3.1-3.4, Chapter 3
   h. *Configs* = *Configs* ∪ *Best_config* ∪ *k_pop*;
   i. Retrain *Surr_model* on *Configs*;

Output: Best configuration according to objective score;

A random forest Regressor, *Surr_model*, is then trained, using *Configs* as features, and *Scores* as the target variable (step 3). The *Configs* are processed as they are for the BO-Auto-PU system, as described in Section 5.1. The configurations in *Configs* are copied to a temporary store *Temp_Configs* to undergo uniform crossover (with candidate solutions selected via tournament selection) and mutation to produce an evolved population of configurations (step 4.a). This process of tournament selection, uniform crossover and mutation is the same as that undertaken by GA-Auto-PU and described in Section 4.1. To briefly restate, tournament selection selects a specified number of candidate solutions (2, in this implementation) from the pool of available candidate solutions and progresses the candidate solution with the highest score value to the next stage of evolution. The process of uniform crossover involves swapping the component (hyperparameter) values of two selected candidate solutions (selected via tournament selection), with a given probability. Mutation involves slightly altering the values of the candidate solution hyperparameters.

The surrogate scores of each just-produced configuration in *Configs* is calculated by *Surr_model* and saved as $\hat{Y}$ (step 4.b), with the configuration with the highest surrogate score saved as *Best_config*

(step 4.c). This selection of the best configuration is inspired by elitism, but note that it is not exactly elitism, as the selection is based on the surrogate score, not the objective score. This surrogate score is, essentially, an estimation of the objective score that would be produced by assessing the candidate solution according to the objective function. Recall from the background information on Bayesian optimisation (see Section 2.3) that this estimation is conducted in order to minimise computational runtime. Creating a probabilistic model allows for an estimation to be calculated based on the previously calculated objective scores of other candidate solutions, thus allowing for a somewhat informed exploration of the search space.

Tournament selection is utilised to select *k* candidate solutions according to their surrogate score (F-measure), which are then added to a population *k_pop* (step 4.d-e). The procedure for tournament selection in this step is the same as the procedure previously described. *k_pop* then undergoes uniform crossover and mutation to produce a newly evolved population (step 4.f). The uniform crossover and mutation procedures are the same as those previously described, with parents selected via tournament selection. *Best_config* and the configurations from *k_pop* are assessed according to the objective function to calculate their objective scores, before the configurations are added to *Configs* and used to update the *surr_model* (steps 4.g-i). The result of this is that *Configs* is ever-growing, meaning that there is no selection pressure on the population. However, an ever-growing population is beneficial in this scenario, as it provides more data for the surrogate model to learn from. The objective function cited in step 4.g is defined in Section 3.2.3. This process (step 5 in Procedure 6.1) is repeated *It_count* times. Finally, the best configuration, according to the objective score, is returned.

Recall that (as in BO in general) the computation of the objective score is much more expensive than the computation of the surrogate score, and hence, at each iteration, just k + 1 configurations (*Best_config* and the *k* configurations in *k_pop*) in the current population have their objective score computed.

Configs are processed as follows for training *Surr_model*: for the base search space, the numeric components of each configuration (*Threshold_1A, Iteration_Count_1A, Threshold_1_B*) are treated as numeric features, the Boolean component (*Flag_1B*) is treated as a binary feature, and the nominal components (*Classifier_1A, Classifier_1B, Classifier_2*) are one-hot encoded, with a binary value

for each potential value of the component, indicating whether or not that value is used by the PU learning method. The resulting instances used as input by the regression algorithm consist of 58 features. For the extended search space, all the previously mentioned components are treated as they are in the base search space. However, we also have the additional spy components, with *Spy_rate* and *Spy_tolerance* treated as numeric features, and the Boolean component "*Spy_flag*" treated as a binary feature. This results in instances consisting of 61 features.

Figure 6.1 shows the hybridisation of EA and BO to create EBO. The figure shows the key components of EA and BO relevant to the EBO system. The EA utilises evolutionary operators (crossover, mutation & elitism), which are incorporated into the EBO system. At each iteration, the EA evaluates the whole population with the expensive objective function, whilst the BO evaluates all candidate solutions with the fast surrogate model to select a single candidate solution to evaluate with the objective function. The EBO, like the BO, uses a surrogate model to assess all individuals at each iteration. However, instead of selecting only a single candidate solution, at each iteration the EBO selects multiple promising candidate solutions which undergo evolutionary operations before being assessed by the objective function. Whilst the EA uses selection based on fitness, the EBO utilises the BO approach of selection based on surrogate score.

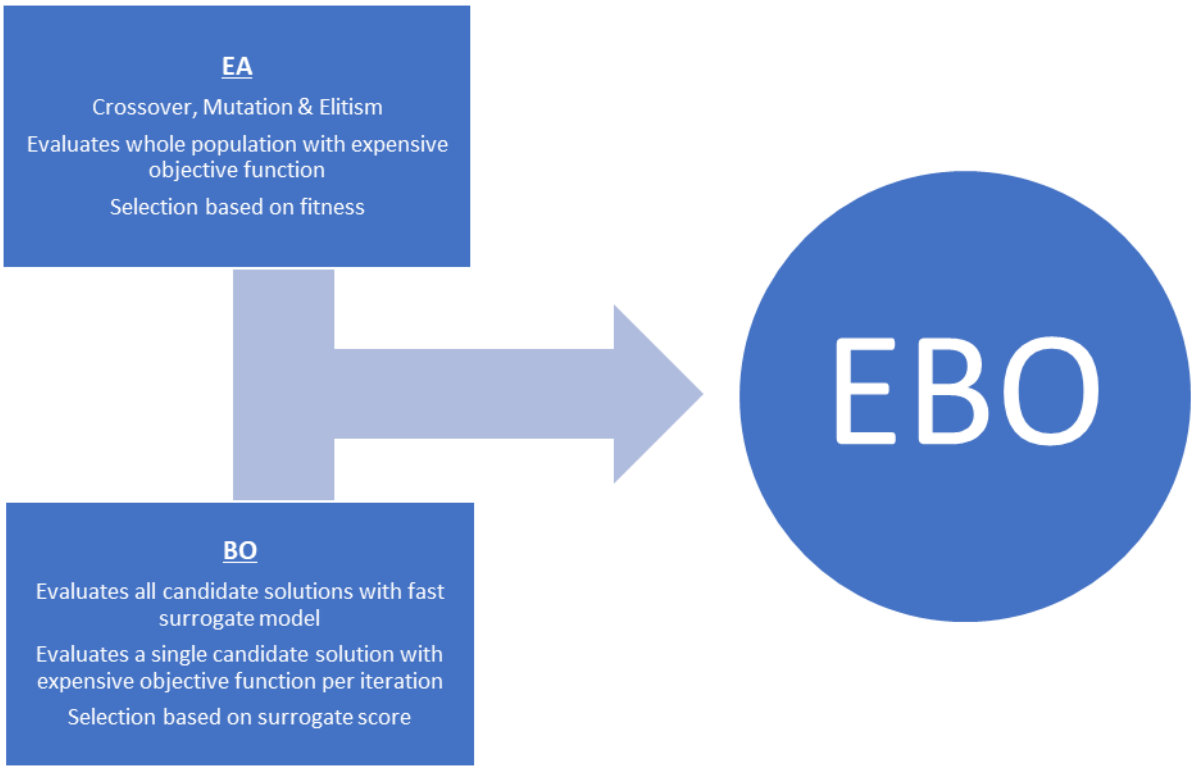


Figure 6.1. EA/BO hybridisation.

6.1.2 The EBO Procedure's Hyperparameters

Table 6.1 shows the hyperparameter settings of the EBO procedure underlying EBO-Auto-PU.

Table 6.1. Hyperparameters of the EBO-Auto-PU system, with their default values.

Hyperparameter	**Value**
#Configs	101
It_count	50
Surr_model	Random Forest Regressor
Crossover probability	0.9
Component crossover probability	0.5
Mutation probability	0.1
Tournament size	2
k	10

Note that the hyperparameters *#Configs*, *It_count* and *Surr_model* in Table 6.1 are essentially the same as the corresponding hyperparameters in Table 5.1 for the BO procedure, and these hyperparameters take the same settings in both tables (i.e. for both the EBO and the BO procedures), in order to make the comparison between EBO-Auto-PU and BO-Auto-PU as fair as possible. In addition, the hyperparameters "Crossover probability", "Component crossover probability", "Mutation probability", and "Tournament size" in Table 6.1 are also essentially the same as the hyperparameters "*Cross_prob*", "*Gene_cross_prob*", "*Mutation_prob*", and "*Tournament_size*" from Table 4.1 in Chapter 4, respectively; and again, these hyperparameters take the same settings in both tables for a fair comparison between EBO-Auto-PU and GA-Auto-PU. The only parameter unique to EBO-Auto-PU is the k parameter, used to determine the number of candidate solutions to be selected with tournament selection to assess according to their objective score.

6.1.3 Computational Efficiency

In the GA-based, BO-based, and the EBO-based Auto-ML systems for PU learning, the running time is by far dominated by the time to evaluate the candidate solutions along the iterations of the search, i.e., the time to learn a PU model and evaluate its F-measure on the training set, for each candidate PU learning method. GA, BO, and the EBO-based methods perform the same number of iterations (50) in our experiments. However, in each generation (iteration) of GA-Auto-PU the GA must learn n PU models, where n is the number of individuals (candidate solutions) in the population, each iteration of BO-Auto-PU needs to learn a single PU model, whilst each iteration of EBO-Auto-PU

needs to learn k+1 candidate solutions, where k is the number of candidate solutions selected via tournament selection to assess according to the objective function (see Section 3.2.3 for full details of this procedure). Learning a PU model can be very computationally expensive, depending not only on the size of the dataset but also on the time complexity of the 3 classification algorithms chosen for Phases 1A, 1B and 2 of the 2-step method, and the number of iterations the classifier is applied in Phase 1A.

All three Auto-ML systems also must perform other steps for generating candidate solutions to be evaluated, but these take in general much less time than the time to evaluate candidate solutions using the objective function (F-measure) as described above. More precisely, at each iteration, the GA and the EBO must perform tournament selection, crossover and mutation, but these are all simple operations, which are much faster than computing the fitness function (learning one PU model for each individual).

Unlike the GA, at each iteration BO and EBO learn a surrogate model, but again, the time for this is much shorter than the time taken to learn a PU model in each iteration of BO. This is because the surrogate model is learned by a relatively fast random forest algorithm using a small dataset of PU algorithm configurations, whilst learning a PU model involves running multiple classifiers (one of them for several iterations in Phase 1A), each classifier can be much slower than a random forest. In addition, each classifier is learned using the training data of the current dataset, which is typically much larger in number of instances than the small dataset of PU method configurations. Regarding the number of features, the training set for learning a PU model has in general more features than the training set for learning the surrogate model in the case of the synthetic datasets; whilst the converse is true in the case of the biomedical datasets – but even for this latter group of datasets, the overall time taken to learn a surrogate model is still much faster than the time to learn a PU model, making BO-Auto-PU much faster than GA-Auto-PU, as discussed in Chapter 5. As EBO-Auto-PU assesses (using the expensive objective function) fewer candidate solutions than GA-Auto-PU, but more than BO-Auto-PU, the EBO-Auto-PU system sits between the GA-Auto-PU and the BO-Auto-PU systems in regard to computational runtime, as is discussed in Section 6.3.

6.2 Experimental Setup

The experimental procedure is explained in detail in Chapter 3. However, to briefly recap, two types of datasets are used in these experiments (biomedical and synthetic), each with 3 versions (varying the % of positive instances hidden in the unlabelled set), thus creating 120 datasets total.

A nested cross-validation procedure is used, with a simple 5-fold cross-validation procedure as the external layer. The internal layer splits the training set into 5 learning and validation sets, which is used to evaluate the candidate solutions.

To compare the performance of the methods, we use the Wilcoxon signed rank test [202], with Holm correction for testing multiple hypotheses [203].

6.2.1 Structure of the Results Section

In the next section, we present experimental results comparing the EBO-Auto-PU system with both search spaces (without and with the Spy method). Firstly, EBO-Auto-PU is compared against GA-Auto-PU, BO-Auto-PU. Secondly, EBO-Auto-PU is compared against the two PU learning baselines. Experiments were conducted on both the real-world biomedical datasets and the synthetic datasets, for three values of δ (the percentage of positives hidden in the unlabelled set): 20%, 40%, and 60%. Each section will report the F-measure results in full and will provide a summary of the precision and recall results. The full precision and recall results (for each dataset) can be found in the Appendix.

For the sake of brevity, the EBO-Auto-PU, BO-Auto-PU, and the GA-Auto-PU systems utilising the base search space (without the Spy method) will be referred to as EBO-1, BO-1 and GA-1 respectively; whilst the systems utilising the extended search space (with the Spy method) will be referred to as EBO-2, BO-2, and GA-2 respectively.

6.3 Results for EBO-Auto-PU

6.3.1 Results comparing EBO-Auto-PU with GA-Auto-PU and BO-Auto-PU PU

In the next section, results comparing the EBO-Auto-PU system with GA-Auto-PU and BO-Auto-PU are presented, beginning with a comparison of EBO-1 with GA-1 and BO-1 (Auto-PU systems with the base search space) on the biomedical datasets in Table 6.2.

Table 6.2. F-measure results of EBO-1 against BO-1 and GA-1 on real-world biomedical datasets.

	20%			δ = 40%			δ = 60%		
Dataset	**EBO-1**	**GA-1**	**BO-1**	**EBO-1**	**GA-1**	**BO-1**	**EBO-1**	**GA-1**	**BO-1**
Alzheimer's	0.629	0.529	0.615	0.587	0.551	0.600	0.540	0.456	0.436
Autism	0.986	0.960	0.967	0.926	0.927	0.956	0.927	0.910	0.863
Breast cancer Coi.	0.966	0.705	0.694	0.952	0.687	0.701	0.615	0.510	0.586
Breast cancer Wis.	0.893	0.954	0.949	0.872	0.932	0.969	0.927	0.906	0.895
Breast cancer mut.	0.672	0.893	0.893	0.667	0.868	0.873	0.862	0.854	0.841
Cervical cancer	0.839	0.828	0.839	0.904	0.903	0.903	0.667	0.714	0.645
Cirrhosis	0.532	0.573	0.545	0.453	0.464	0.529	0.507	0.443	0.489
Dermatology	0.899	0.860	0.872	0.813	0.780	0.905	0.716	0.828	0.725
PI Diabetes	0.654	0.677	0.647	0.661	0.649	0.645	0.634	0.606	0.594
ES Diabetes	0.973	0.958	0.983	0.913	0.895	0.877	0.909	0.930	0.902
Heart Disease	0.833	0.843	0.844	0.800	0.801	0.830	0.774	0.785	0.777
Heart Failure	0.732	0.770	0.753	0.666	0.652	0.605	0.640	0.674	0.704
Hepatitis C	0.925	0.953	0.907	0.835	0.771	0.838	0.667	0.588	0.708
Kidney Disease	1.000	0.976	0.988	0.938	0.988	0.964	0.646	0.754	0.806
Liver Disease	0.827	0.834	0.820	0.819	0.803	0.817	0.717	0.804	0.795
Maternal Risk	0.855	0.476	0.837	0.803	0.812	0.780	0.739	0.735	0.689
Parkinsons	0.929	0.860	0.935	0.894	0.836	0.875	0.707	0.818	0.732
Parkinsons Biom.	0.203	0.476	0.167	0.337	0.265	0.192	0.133	0.233	0.182
Spine	0.933	0.652	0.954	0.932	0.907	0.926	0.775	0.818	0.742
Stroke	0.239	0.474	0.244	0.225	0.255	0.153	0.229	0.255	0.208

Table 6.3 summarises the statistical significance of the results from Table 6.2, as well as the results for precision and recall. In Table 6.3, for each combination of a performance measure (F-measure, precision, recall) and a δ value (δ= 20%, 40%, 60%), the table reports the average (Avg.) rank of EBO-1 vs GA-1 (EBO-1 is the left rank, GA-1 is the right one) and EBO-1 vs BO-1, with the corresponding p-value. The better (lower) avg. rank in each cell is shown in boldface, and significant p-values (smaller than α) are also shown in boldface. For example, in the cell for F-measure, δ = 20%, the average ranks for EBO-1 is 1.48 and BO-1 is 1.52. Hence, EBO-1 was the winner, but the p-value (0.658) was greater than the significant level α (0.05), so this result was not statistically significant.

Table 6.3. Results of Wilcoxon signed-rank tests when comparing EBO-1 against GA-1 and BO-1 regarding F-measure, Precision and Recall, for the 3 δ values.

δ (%)	Compared systems	F-measure			Precision			Recall		
		Avg ranks	p-value	α	Avg ranks	p-value	α	Avg ranks	p-value	α
20%	EBO-1 vs GA-1	1.5 vs 1.5	0.784	0.025	**1.48** vs 1.52	0.446	0.017	1.62 vs **1.38**	0.178	0.025
	EBO-1 vs BO-1	**1.48** vs 1.52	0.658	0.017	**1.45** vs 1.55	0.968	0.025	**1.45** vs 1.55	0.983	0.05
40%	EBO-1 vs GA-1	**1.4** vs 1.6	0.245	0.017	1.55 vs **1.45**	0.828	0.05	**1.42** vs 1.58	0.381	0.017
	EBO-1 vs BO-1	**1.45** vs 1.55	1.000	0.05	1.55 vs **1.45**	0.387	0.017	1.5 vs 1.5	0.557	0.05
60%	EBO-1 vs GA-1	1.55 vs **1.45**	0.312	0.025	1.5 vs 1.5	0.812	0.05	1.55 vs **1.45**	0.388	0.025
	EBO-1 vs BO-1	**1.4** vs 1.6	0.674	0.05	1.62 vs **1.38**	0.184	0.017	**1.38** vs 1.62	0.629	0.05

The results for Table 6.3 show EBO-1 performing best overall, though no results are statistically significant. For F-measure, EBO-1 is outperformed only 1 time, against GA-1 when δ=60%. This was also the case for BO-1, shown in Table 5.3 in Chapter 5, although BO-1 was outperformed by a much larger margin than EBO-1 has been. For precision, EBO-1 performed best for δ=20%, but was outperformed for 40% and 60%, albeit only slightly. For recall, EBO-1 performed best in 3 of 6 cases and drew with BO-1 in 1 case. Thus, EBO-1 was outperformed in only 2 of 6 cases.

Table 6.4. F-measure results of EBO-2 against BO-2 and GA-2 on real-world biomedical datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**EBO-2**	**BO-2**	**GA-2**	**EBO-2**	**BO-2**	**GA-2**	**EBO-2**	**BO-2**	**GA-2**
Alzheimer's	0.559	0.580	0.548	0.597	0.603	0.576	0.581	0.492	0.529
Autism	0.964	0.963	0.982	0.938	0.937	0.940	0.887	0.914	0.927
Breast cancer Coi.	0.967	0.667	0.711	0.952	0.618	0.671	0.923	0.000	0.553
Breast cancer Wis.	0.882	0.959	0.956	0.863	0.942	0.936	0.839	0.889	0.866
Breast cancer mut.	0.666	0.890	0.896	0.655	0.853	0.739	0.587	0.845	0.872
Cervical cancer	0.867	0.867	0.867	0.904	0.867	0.839	0.516	0.839	0.350
Cirrhosis	0.506	0.497	0.446	0.493	0.515	0.397	0.322	0.472	0.204
Dermatology	0.857	0.876	0.901	0.891	0.841	0.896	0.750	0.795	0.692
PI Diabetes	0.668	0.653	0.642	0.665	0.648	0.646	0.607	0.615	0.634
ES Diabetes	0.957	0.954	0.978	0.905	0.891	0.887	0.915	0.912	0.894
Heart Disease	0.826	0.844	0.836	0.804	0.817	0.780	0.747	0.805	0.786
Heart Failure	0.741	0.757	0.751	0.656	0.652	0.670	0.514	0.600	0.671
Hepatitis C	0.907	0.964	0.944	0.907	0.761	0.863	0.689	0.612	0.610
Kidney Disease	0.911	0.976	0.925	0.897	0.976	0.951	0.656	0.789	0.806
Liver Disease	0.832	0.822	0.831	0.800	0.815	0.817	0.748	0.722	0.748
Maternal Risk	0.854	0.847	0.862	0.810	0.786	0.813	0.731	0.729	0.738
Parkinsons	0.914	0.936	0.935	0.850	0.837	0.843	0.720	0.800	0.792
Parkinsons Biom.	0.259	0.286	0.282	0.276	0.000	0.259	0.203	0.000	0.280
Spine	0.942	0.941	0.923	0.920	0.936	0.917	0.802	0.700	0.761
Stroke	0.232	0.256	0.241	0.225	0.255	0.239	0.201	0.233	0.243

Moving on next to a comparison of EBO-2 with GA-2 and BO-2, Table 6.4 presents these results for the biomedical datasets.

Table 6.5 details the statistical significance of the F-measure results shown in Table 6.4 and summarises the results for precision and recall. These results show EBO-2 performing similarly to BO-2 and GA-2, with no statistically significant differences in results, with the exception of precision when δ=60%, with GA-2 and BO-2 outperforming EBO-2 with statistical significance.

Table 6.5. Results of Wilcoxon signed-rank tests when comparing EBO-2 against BO-2 and GA-2 regarding F-measure, Precision and Recall, for the 3 δ values on the biomedical datasets.

δ (%)	Compared systems	F-measure			Precision			Recall		
		Avg ranks	p-value	adj. α	Avg ranks	p-value	adj. α	Avg ranks	p-value	adj. α
20%	EBO-2 vs GA-2	1.62 vs **1.38**	0.658	0.025	1.68 vs **1.32**	0.268	0.05	1.68 vs **1.32**	0.286	0.025
	EBO-2 vs BO-2	**1.38** vs 1.62	0.825	0.05	1.65 vs **1.35**	0.061	0.025	1.5 vs 1.5	0.329	0.05
40%	EBO-2 vs GA-2	**1.35** vs 1.65	0.039	0.025	1.55 vs 1.45	0.927	0.05	**1.42** vs 1.58	0.260	0.05
	EBO-2 vs BO-2	**1.48** vs 1.52	0.220	0.05	1.68 vs 1.32	0.268	0.025	**1.35** vs 1.65	0.177	0.025
60%	EBO-2 vs GA-2	1.5 vs 1.5	0.756	0.025	1.82 vs 1.18	**0.005**	0.025	**1.32** vs 1.68	0.07	0.025
	EBO-2 vs BO-2	1.6 vs **1.4**	0.956	0.05	1.78 vs **1.22**	**0.049**	0.05	**1.32** vs 1.68	0.107	0.05

Table 6.6. F-measure results of EBO-1 against BO-1 and GA-1 on synthetic datasets.

Dataset	δ = 20%			δ = 40%			δ = 60%		
	EBO-1	GA-1	BO-1	EBO-1	GA-1	BO-1	EBO-1	GA-1	BO-1
1	0.678	0.661	0.663	0.644	0.718	0.619	0.581	0.603	0.554
2	0.194	0.136	0.128	0.043	0.044	0.046	0.040	0.065	0.061
3	0.770	0.788	0.761	0.685	0.693	0.671	0.594	0.637	0.555
4	0.809	0.831	0.811	0.795	0.818	0.785	0.675	0.674	0.627
5	0.439	0.618	0.651	0.420	0.616	0.496	0.292	0.609	0.514
6	0.714	0.759	0.763	0.677	0.769	0.676	0.605	0.684	0.600
7	0.565	0.520	0.598	0.545	0.515	0.478	0.487	0.478	0.465
8	0.508	0.525	0.512	0.400	0.477	0.473	0.332	0.381	0.391
9	0.023	0.111	0.043	0.076	0.080	0.051	0.133	0.146	0.087
10	0.973	0.903	0.918	0.959	0.872	0.868	0.907	0.742	0.756
11	0.635	0.604	0.568	0.576	0.567	0.575	0.535	0.531	0.531
12	0.752	0.674	0.671	0.673	0.666	0.683	0.607	0.609	0.574
13	0.665	0.644	0.662	0.610	0.623	0.603	0.539	0.516	0.511
14	0.909	0.975	0.978	0.878	0.962	0.969	0.790	0.925	0.900
15	0.580	0.601	0.635	0.558	0.593	0.588	0.462	0.519	0.493
16	0.546	0.477	0.432	0.451	0.388	0.333	0.372	0.301	0.282
17	0.372	0.347	0.389	0.297	0.496	0.302	0.129	0.412	0.218
18	0.532	0.559	0.502	0.469	0.389	0.436	0.413	0.326	0.245
19	0.500	0.472	0.423	0.324	0.468	0.426	0.273	0.381	0.406
20	0.661	0.705	0.696	0.633	0.692	0.670	0.532	0.625	0.622

Moving now to the synthetic datasets, Table 6.6 presents the results for the three systems, with the statistical significance test results presented in Table 6.7.

Table 6.7. Results of Wilcoxon signed-rank tests when comparing EBO-1 against GA-1 and BO-1 regarding F-measure, Precision and Recall, for the 3 δ values on the synthetic datasets.

δ (%)	Compared systems	F-measure			Precision			Recall		
		Avg ranks	p-value	α	Avg ranks	p-value	α	Avg ranks	p-value	α
20%	EBO-1 vs GA-1	1.5 vs 1.5	0.869	0.05	1.5 vs 1.5	0.898	0.05	1.5 vs 1.5	0.729	0.05
	EBO-1 vs BO-1	1.5 vs 1.5	0.784	0.025	1.55 vs **1.45**	0.216	0.025	**1.25** vs 1.75	0.053	0.025
40%	EBO-1 vs GA-1	1.7 vs **1.3**	0.076	0.025	**1.35** vs 1.65	0.076	0.025	1.8 vs **1.2**	**0.001**	0.025
	EBO-1 vs BO-1	**1.45** vs 1.55	0.985	0.05	1.55 vs **1.45**	0.349	0.05	**1.45** vs 1.55	0.845	0.05
60%	EBO-1 vs GA-1	1.65 vs **1.35**	0.083	0.025	1.5 vs 1.5	0.898	0.05	1.85 vs **1.15**	**0.001**	0.025
	EBO-1 vs BO-1	**1.4** vs 1.6	0.927	0.05	1.85 vs **1.15**	0.002	0.025	**1.32** vs 1.68	0.126	0.05

Table 6.7 summarises the statistical significance of the results from Table 6.6, as well as the results for precision and recall. The results shown in this table are mixed. For F-measure, EBO-1 performed well, winning in 2/6 cases and drawing in 2/6, thus losing in only 2/6. However, none of these differences were statistically significant. For precision, EBO-1 is outperformed by BO-1 with statistical significance when δ=60%, and for recall, EBO-1 is outperformed by GA-1 when δ=40% and 60%. Despite this, as previously mentioned, no statistically significant difference was observed for F-measure. Thus, whilst those loses for precision and recall were statistically significant, they were not significant enough to tip the balance for F-measure.

Considering the results of both Table 6.3 and 6.7, regarding predictive performance, EBO-1 performed well against GA-1 and BO-1, achieving a higher rank than the compared system in more cases than it achieved an inferior rank for F-measure. Considering computational efficiency, recall from the previous chapter that GA-1 took 226.3 minutes on average to run a 5-fold cross-validation procedure per dataset, whilst BO-1 took only 8.4 minutes. EBO-1 took 18.1 minutes, thus performing 2.15 times slower than BO-1, but 12.5 times faster than GA-1. Considering both these factors, it can be argued that EBO-1 is the best performing system when compared against GA-1 and BO-1, achieving high predictive performance and representing a trade-off between GA-1 and BO-1 in regard to computational efficiency.

Table 6.8. F-measure results of EBO-2 against BO-2 and GA-2 on synthetic datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**EBO-2**	**BO-2**	**GA-2**	**EBO-2**	**BO-2**	**GA-2**	**EBO-2**	**BO-2**	**GA-2**
1	0.670	0.649	0.640	0.654	0.586	0.709	0.570	0.578	0.545
2	0.204	0.203	0.176	0.024	0.022	0.105	0.050	0.094	0.111
3	0.766	0.770	0.759	0.707	0.676	0.702	0.582	0.537	0.612
4	0.809	0.831	0.824	0.793	0.795	0.809	0.710	0.622	0.692
5	0.452	0.603	0.612	0.389	0.614	0.571	0.411	0.499	0.559
6	0.709	0.746	0.762	0.652	0.672	0.751	0.622	0.652	0.672
7	0.585	0.495	0.528	0.537	0.456	0.496	0.509	0.416	0.448
8	0.559	0.531	0.571	0.386	0.480	0.484	0.385	0.341	0.390
9	0.068	0.019	0.098	0.041	0.039	0.000	0.114	0.105	0.143
10	0.975	0.900	0.896	0.960	0.854	0.850	0.879	0.756	0.716
11	0.622	0.573	0.574	0.563	0.554	0.579	0.498	0.474	0.525
12	0.734	0.713	0.681	0.676	0.647	0.692	0.609	0.586	0.599
13	0.628	0.667	0.648	0.605	0.578	0.612	0.510	0.533	0.576
14	0.905	0.974	0.977	0.854	0.948	0.966	0.702	0.908	0.934
15	0.602	0.589	0.595	0.536	0.545	0.575	0.486	0.415	0.565
16	0.544	0.507	0.431	0.442	0.298	0.402	0.364	0.320	0.299
17	0.323	0.289	0.384	0.207	0.304	0.470	0.211	0.070	0.382
18	0.517	0.529	0.576	0.467	0.459	0.408	0.397	0.297	0.373
19	0.427	0.444	0.462	0.351	0.405	0.483	0.275	0.361	0.385
20	0.651	0.710	0.701	0.621	0.632	0.664	0.539	0.554	0.594

Table 6.9 details the statistical significance of the F-measure results shown in Table 6.8 and summarises the results for precision and recall. These results differ from the results of Table 6.5, with GA-2 largely outperforming EBO-2 for F-measure and recall, achieving statistical significance in one instance for F-measure, and all instances for recall. The results for precision are more favourable for EBO-2, with EBO-2 performing best in 2/6 cases and achieving equal performance in 2/6 cases.

Table 6.9. Results of Wilcoxon signed-rank tests when comparing EBO-2 against BO-2 and GA-2 regarding F-measure, Precision and Recall, for the 3 δ values on the synthetic datasets.

δ (%)	**Compared systems**	**F-measure**			**Precision**			**Recall**		
		Avg ranks	**p-value**	**α**	**Avg ranks**	**p-value**	**α**	**Avg ranks**	**p-value**	**α**
20%	EBO-2 vs GA-2	1.55 vs **1.45**	0.600	0.05	**1.25** vs 1.75	**0.006**	0.025	1.85 vs **1.15**	**0.00005**	0.025
	EBO-2 vs BO-2	**1.45** vs 1.55	0.409	0.025	1.5 vs 1.5	0.729	0.05	**1.45** vs 1.55	0.368	0.05
40%	EBO-2 vs GA-2	1.75 vs **1.25**	**0.017**	0.025	1.5 vs 1.5	0.729	0.05	1.8 vs **1.2**	**0.0004**	0.025
	EBO-2 vs BO-2	**1.45** vs 1.55	0.990	0.05	1.55 vs **1.45**	0.622	0.025	**1.38** vs 1.62	0.968	0.05
60%	EBO-2 vs GA-2	1.65 vs **1.35**	0.070	0.025	**1.4** vs 1.6	0.498	0.05	1.75 vs **1.25**	**0.0007**	0.025
	EBO-2 vs BO-2	**1.4** vs 1.6	0.261	0.05	1.7 vs **1.3**	0.083	0.025	**1.2** vs 1.8	**0.004**	0.05

Figure 6.2 shows graphically how the average F-measure of each of the Auto-PU systems and the baselines changes over the different values of δ for the biomedical datasets. In order to reduce the number of figures across this chapter, this Figure shows the results for all Auto-ML systems (EBO-1, EBO-2, BO-1, BO-2, GA-1, GA-2) and all baseline PU learning methods (DF-PU and S-EM) investigated in this chapter, but in this current part of the text the analysis is focussed on the results for EBO-Auto-PU, BO-Auto-PU and GA-Auto-PU only. Note that the charted data for BO-1, BO-2, GA-1, GA-2, DF-PU, and S-EM were shown in Chapters 4 and 5 but have been included here for the reader's reference.

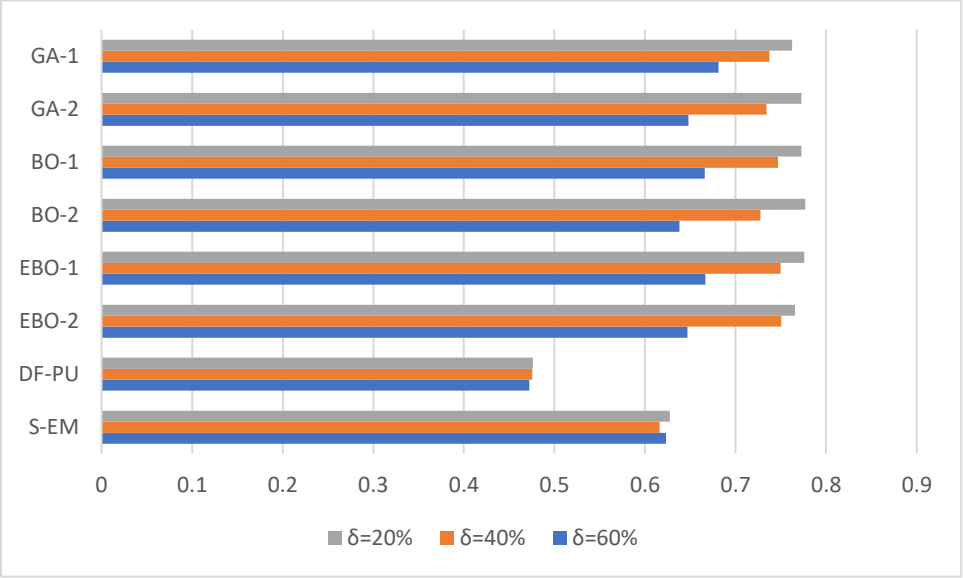


Figure 6.2. Average F-measure results comparison for EBO-1, EBO-2, BO-1, BO-2, GA-1, GA-2, DF-PU and S-EM, across the three values of δ for the biomedical datasets.

As with the results for GA-Auto-PU, discussed in Section 4.3, and the results for BO-Auto-PU discussed in Section 5.3, the performance of EBO-Auto-PU does also decline monotonically with the increase in the value of δ. The decline for δ=60% is sharper than that of GA-Auto-PU and BO-Auto-PU.

Figure 6.3 shows graphically how the average F-measure of each of the systems changes over the different values of δ, for the synthetic datasets. GA-1 appears best performing in regard to average F-measure as it maintains relatively high F-measure values across the values of δ. Whereas, the F-measure values of EBO-1, EBO-2, BO-1, and BO-2 drop sharply as the value of δ increases.

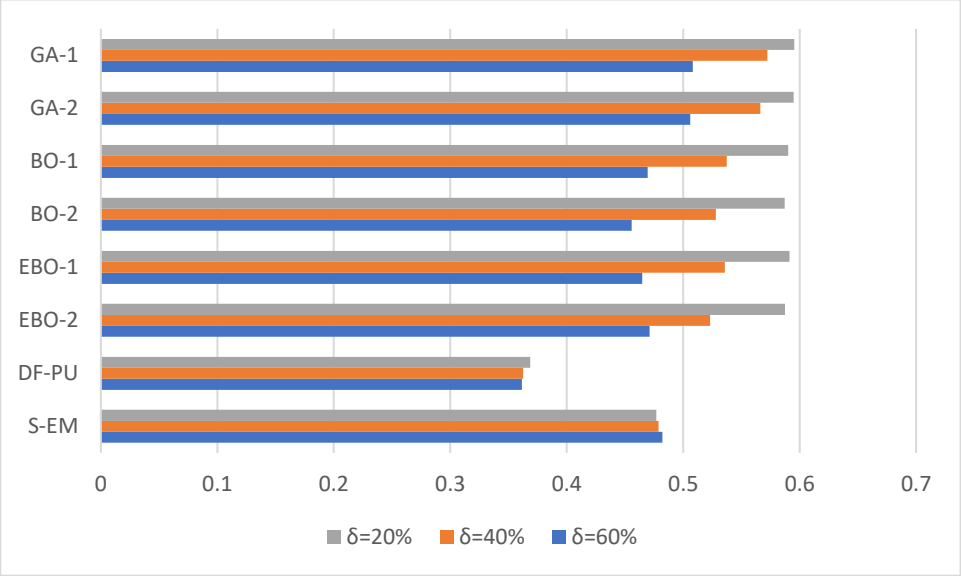


Figure 6.3. Average F-measure results comparison for EBO-1, EBO-2, BO-1, BO-2, GA-1, GA-2, DF-PU and S-EM, across the three values of δ for the synthetic datasets.

In order to further analyse the results, Table 6.10 shows the values of Pearson's linear correlation coefficient between the F-measure values achieved by EBO-1, EBO-2, BO-1, BO-2, GA-1, GA-2, DF-PU, and S-EM and percentages of positive examples in the original dataset, for each δ value, for the biomedical datasets. Again, in order to reduce the number of tables across this chapter, the results for all the aforementioned systems or methods are reported in Table 6.6.

Table 6.10. Linear (Pearson's) correlation coefficient value between the F-measure and the percentage of positive examples in the original dataset (before hiding some positive examples in the unlabelled set) for each combination of a method and a δ value, for the biomedical datasets, for all methods.

Method	δ = 20%	δ = 40%	δ = 60%
EBO-1	0.440	0.469	0.460
EBO-2	0.363	0.361	0.447
BO-1	0.398	0.360	0.498
BO-2	0.339	0.348	0.225
GA-1	0.333	0.385	0.504
GA-2	0.340	0.357	0.580
DF-PU	0.988	0.988	0.988
S-EM	0.646	0.558	0.652

Table 6.10 shows a relatively stable correlation for EBO-1, with a moderate correlation observed for all three δ values, as defined by the categorisation of coefficient values outlined in Section 3.4.3. The values for δ=20% and 40% are somewhat higher than those observed for BO-1 and GA-1, indicating that, even at lower values of δ, of the three systems, the performance of EBO-1 is most closely linked to the percentage of positive instances hidden in the unlabelled set for the biomedical datasets. EBO-2 displays weak correlations for 20% and 40%, but increases to a moderate correlation for 60%, following a similar trend to GA-2.

Table 6.11. Linear (Pearson's) correlation coefficient value between the F-measure and the percentage of positive examples in the original dataset (before hiding some positive examples in the unlabelled set) for each combination of a method and a δ value, for the synthetic datasets, for all methods.

Method	δ = 20%	δ = 40%	δ = 60%
EBO-1	0.651	0.624	0.578
EBO-2	0.627	0.639	0.569
BO-1	0.682	0.710	0.700
BO-2	0.706	0.695	0.659
GA-1	0.712	0.682	0.687
GA-2	0.700	0.702	0.696
DF-PU	0.990	0.990	0.990
S-EM	0.794	0.793	0.776

Table 6.11 shows the values of Pearson's linear correlation coefficient between the F-measure values achieved by the systems and the percentages of positive examples in the original dataset, for each δ value for the synthetic datasets. As was the case for GA-Auto-PU (Section 4.3) and BO-Auto-PU (Section 5.3), the correlation between percentage of positive instances and F-measure is substantially higher for EBO-Auto-PU for the synthetic datasets than it was for the biomedical datasets. However, the correlations are all still moderate, making EBO-1 and EBO-2 the only systems of the six discussed that do not exhibit a high correlation for any of the values of δ. This is at odds with the results of Table 6.10, which showed that EBO-1 had the highest correlation of the three systems. This underscores the importance of considering dataset characteristics and system performance in tandem to gain a comprehensive understanding of the correlation between these factors.

Considering computational efficiency, recall from the previous chapter that GA-1 took 226.3 minutes on average to run a 5-fold cross-validation procedure per dataset, GA-2 took 223.2 minutes, whilst BO-1 took only 8.4 minutes and BO-2 took only 9.8 minutes. EBO-1 took 18.1 minutes, thus performing 2.15 times slower than BO-1, but 12.5 times faster than GA-1. EBO-2 took 20.2 minutes, thus performing 2.06 times slower than BO-2, but 11.05 times faster than GA-2. Considering these factors, it can be argued that EBO-1 is the best performing system when compared against GA-Auto-PU and BO-Auto-PU, achieving high predictive performance and representing a trade-off between GA-Auto-PU and BO-Auto-PU in regard to computational efficiency.

6.3.2 Results comparing EBO-Auto-PU with two baseline PU learning methods

This section details the results achieved by EBO-Auto-PU and two baseline PU learning methods (DF-PU and S-EM, see Section 2.5) when applied to 20 real-world biomedical datasets and 20

synthetic datasets. Note that the results in the EBO-1 and EBO-2 columns in the tables reported in this section are the same as those reported in the previous section, but they are repeated in this section for the reader's convenience.

Table 6.12. F-measure results of EBO-1 and baseline PU learning methods on real-world biomedical datasets.

	δ = 20%			**δ = 40%**			**δ = 60%**		
Dataset	**EBO-1**	**DF-PU**	**S-EM**	**EBO-1**	**DF-PU**	**S-EM**	**EBO-1**	**DF-PU**	**S-EM**
Alzheimer's	0.629	0.195	0.321	0.587	0.194	0.370	0.540	0.171	0.373
Autism	0.986	0.648	0.820	0.926	0.648	0.841	0.927	0.645	0.835
Breast cancer Coi.	0.966	0.697	0.711	0.952	0.711	0.704	0.615	0.697	0.699
Breast cancer Wis.	0.893	0.543	0.898	0.872	0.543	0.903	0.927	0.539	0.904
Breast cancer mut.	0.672	0.489	0.892	0.667	0.489	0.893	0.862	0.485	0.892
Cervical cancer	0.839	0.061	0.054	0.904	0.042	0.053	0.667	0.044	0.046
Cirrhosis	0.532	0.405	0.436	0.453	0.401	0.442	0.507	0.405	0.459
Dermatology	0.899	0.228	0.718	0.813	0.229	0.718	0.716	0.219	0.719
PI Diabetes	0.654	0.516	0.534	0.661	0.516	0.525	0.634	0.515	0.544
ES Diabetes	0.973	0.762	0.792	0.913	0.756	0.859	0.909	0.759	0.793
Heart Disease	0.833	0.705	0.811	0.800	0.705	0.828	0.774	0.702	0.829
Heart Failure	0.732	0.487	0.529	0.666	0.486	0.508	0.640	0.481	0.557
Hepatitis C	0.925	0.176	0.695	0.835	0.171	0.708	0.667	0.160	0.609
Kidney Disease	1.000	0.428	1.000	0.938	0.428	1.000	0.646	0.428	0.951
Liver Disease	0.827	0.834	0.816	0.819	0.832	0.587	0.717	0.834	0.788
Maternal Risk	0.855	0.403	0.454	0.803	0.395	0.433	0.739	0.390	0.438
Parkinsons	0.929	0.856	0.815	0.894	0.860	0.748	0.707	0.860	0.762
Parkinsons Biom.	0.203	0.354	0.333	0.337	0.354	0.261	0.133	0.367	0.331
Spine	0.933	0.652	0.820	0.932	0.652	0.839	0.775	0.652	0.830
Stroke	0.239	0.086	0.102	0.225	0.094	0.102	0.229	0.094	0.102

Table 6.13 summarises the statistical significance of the F-measure results from Table 6.12, as well as the results for precision and recall.

Table 6.13. Results of Wilcoxon signed-rank tests with Holm correction for multiple hypothesis when comparing EBO-1 against S-EM and DF-PU regarding F-measure, Precision and Recall, for the 3 δ values for the biomedical datasets**.**

δ (%)	**Methods compared**	**F-measure**			**Precision**			**Recall**		
		Avg. ranks	**p-value**	**α**	**Avg. ranks**	**p-value**	**α**	**Avg. ranks**	**p-value**	**α**
20%	EBO-1 vs DF-PU	**1.1** vs 1.9	**0.00004**	0.025	**1.05** vs 1.95	**0.00001**	0.025	1.98 vs **1.02**	**0.0001**	0.025
	EBO-1 vs S-EM	**1.18** vs 1.82	**0.004**	0.05	**1.22** vs 1.78	**0.002**	0.05	1.72 vs **1.28**	**0.030**	0.05
40%	EBO-1 vs DF-PU	**1.1** vs 1.9	**0.00001**	0.025	**1.0** vs 2.0	**0.000002**	0.025	1.98 vs **1.02**	**0.0001**	0.025
	EBO-1 vs S-EM	**1.2** vs 1.8	**0.002**	0.05	**1.18** vs 1.82	**0.001**	0.05	1.7 vs **1.3**	**0.048**	0.05
60%	EBO-1 vs DF-PU	**1.1** vs 1.9	**0.00001**	0.025	**1.0** vs 2.0	**0.000002**	0.025	1.95 vs **1.05**	**0.0001**	0.025
	EBO-1 vs S-EM	**1.25** vs 1.75	**0.004**	0.05	**1.18** vs 1.82	**0.001**	0.05	1.65 vs **1.35**	0.076	0.05

Table 6.13 shows EBO-1 substantially outperforming DF-PU and S-EM across all values of δ for F-measure and precision with statistical significance. As was the case for GA-1 (Section 4.3) and BO-1 (Section 5.3), the baseline methods substantially outperform EBO-1 for recall, with DF-PU outperforming EBO-1 in all cases with statistical significance. S-EM achieved statistically significant superiority in all but 1 case for recall. However, the reasons for this are the same as those outlined in Section 4.3, namely that the baselines massively overpredict the positive class, thus resulting in high recall but low precision.

Table 6.14. F-measure results of EBO-2 and two baseline PU learning methods on real-world biomedical datasets.

	δ = 20%			**δ = 40%**			**δ = 60%**		
Dataset	**EBO-2**	**DF-PU**	**S-EM**	**EBO-2**	**DF-PU**	**S-EM**	**EBO-2**	**DF-PU**	**S-EM**
Alzheimer's	0.559	0.195	0.321	0.597	0.194	0.370	0.581	0.171	0.373
Autism	0.964	0.648	0.820	0.938	0.648	0.841	0.887	0.645	0.835
Breast cancer Coi.	0.967	0.697	0.711	0.952	0.711	0.704	0.923	0.697	0.699
Breast cancer Wis.	0.882	0.543	0.898	0.863	0.543	0.903	0.839	0.539	0.904
Breast cancer mut.	0.666	0.489	0.892	0.655	0.489	0.893	0.587	0.485	0.892
Cervical cancer	0.867	0.061	0.054	0.904	0.042	0.053	0.516	0.044	0.046
Cirrhosis	0.506	0.405	0.436	0.493	0.401	0.442	0.322	0.405	0.459
Dermatology	0.857	0.228	0.718	0.891	0.229	0.718	0.750	0.219	0.719
PI Diabetes	0.668	0.516	0.534	0.665	0.516	0.525	0.607	0.515	0.544
ES Diabetes	0.957	0.762	0.792	0.905	0.756	0.859	0.915	0.759	0.793
Heart Disease	0.826	0.705	0.811	0.804	0.705	0.828	0.747	0.702	0.829
Heart Failure	0.741	0.487	0.529	0.656	0.486	0.508	0.514	0.481	0.557
Hepatitis C	0.907	0.176	0.695	0.907	0.171	0.708	0.689	0.160	0.609
Kidney Disease	0.911	0.428	1.000	0.897	0.428	1.000	0.656	0.428	0.951
Liver Disease	0.832	0.834	0.816	0.800	0.832	0.587	0.748	0.834	0.788
Maternal Risk	0.854	0.403	0.454	0.810	0.395	0.433	0.731	0.390	0.438
Parkinsons	0.914	0.856	0.815	0.850	0.860	0.748	0.720	0.860	0.762
Parkinsons Biom.	0.259	0.354	0.333	0.276	0.354	0.261	0.203	0.367	0.331
Spine	0.942	0.652	0.820	0.920	0.652	0.839	0.802	0.652	0.830
Stroke	0.232	0.086	0.102	0.225	0.094	0.102	0.201	0.094	0.102

Table 6.14 reports the results of EBO-2 compared with the baseline methods on the synthetic datasets, whilst Table 6.15 summarises the statistical significance test results. As has been the case with the previously discussed methods, EBO-2 largely outperforms the baselines for F-measure and precision across all values of δ, with statistical significance in all but one case. As has been previously discussed, the baselines overpredict the positive class, achieving high recall but at a substantial loss to precision.

Table 6.15. Results of Wilcoxon signed-rank tests with Holm correction for multiple hypothesis when comparing EBO-2 against S-EM and DF-PU regarding F-measure, Precision and Recall, for the 3 δ values for the biomedical datasets.

δ (%)	Methods compared	F-measure			Precision			Recall		
		Avg. ranks	p-value	α	Avg. ranks	p-value	α	Avg. ranks	p-value	α
20%	EBO-2 vs DF-PU	**1.15** vs 1.85	**0.0001**	0.025	**1.0** vs 2.0	**0.000002**	0.025	2.0 vs **1.0**	**0.000002**	0.025
	EBO-2 vs S-EM	**1.175** vs 1.825	**0.003**	0.05	**1.2** vs 1.8	**0.0001**	0.05	1.75 vs **1.25**	**0.009**	0.05
40%	EBO-2 vs DF-PU	**1.15** vs 1.85	**0.00004**	0.025	**1.05** vs 1.95	**0.0000004**	0.025	1.98 vs **1.02**	**0.0001**	0.025
	EBO-2 vs S-EM	**1.15** vs 1.85	**0.0001**	0.05	**1.25** vs 1.75	**0.0002**	0.05	1.75 vs **1.25**	**0.036**	0.05
60%	EBO-2 vs DF-PU	**1.25** vs 1.75	**0.002**	0.025	**1.15** vs 1.85	**0.000003**	0.025	1.95 vs **1.05**	**0.00001**	0.025
	EBO-2 vs S-EM	**1.5** vs **1.5**	0.546	0.05	**1.32** vs 1.68	**0.033**	0.05	1.8 vs **1.2**	**0.002**	0.05

Moving next to the synthetic datasets, Table 6.16 reports the results of EBO-1 compared with the baseline methods.

Table 6.16. F-measure results of EBO-Auto-PU with base search space and baseline PU learning methods on synthetic datasets.

Dataset	δ = 20%			δ = 40%			δ = 60%		
	EBO-1	DF-PU	S-EM	EBO-1	DF-PU	S-EM	EBO-1	DF-PU	S-EM
1	0.678	0.484	0.616	0.644	0.484	0.602	0.581	0.483	0.613
2	0.194	0.125	0.194	0.043	0.120	0.130	0.040	0.112	0.120
3	0.770	0.552	0.589	0.685	0.552	0.587	0.594	0.552	0.600
4	0.809	0.454	0.644	0.795	0.416	0.633	0.675	0.417	0.630
5	0.439	0.357	0.402	0.420	0.356	0.436	0.292	0.357	0.465
6	0.714	0.403	0.477	0.677	0.402	0.525	0.605	0.402	0.582
7	0.565	0.285	0.433	0.545	0.283	0.462	0.487	0.282	0.451
8	0.508	0.326	0.468	0.400	0.326	0.457	0.332	0.326	0.439
9	0.023	0.035	0.099	0.076	0.000	0.044	0.133	0.000	0.120
10	0.973	0.233	0.612	0.959	0.234	0.627	0.907	0.233	0.663
11	0.635	0.491	0.505	0.576	0.491	0.520	0.535	0.490	0.517
12	0.752	0.397	0.550	0.673	0.397	0.567	0.607	0.394	0.586
13	0.665	0.500	0.551	0.610	0.460	0.556	0.539	0.456	0.549
14	0.909	0.529	0.817	0.878	0.529	0.840	0.790	0.529	0.873
15	0.580	0.387	0.423	0.558	0.387	0.425	0.462	0.385	0.422
16	0.546	0.239	0.414	0.451	0.239	0.401	0.372	0.240	0.299
17	0.372	0.214	0.262	0.297	0.214	0.281	0.129	0.214	0.267
18	0.532	0.372	0.444	0.469	0.373	0.433	0.413	0.373	0.422
19	0.500	0.378	0.426	0.324	0.378	0.429	0.273	0.376	0.413
20	0.661	0.610	0.615	0.633	0.610	0.620	0.532	0.610	0.613

Table 6.17. Results of Wilcoxon signed-rank tests with Holm correction for multiple hypothesis when comparing EBO-1 against S-EM and DF-PU regarding F-measure, Precision and Recall, for the 3 δ values for the synthetic datasets.

δ (%)	Methods compared	F-measure			Precision			Recall		
		Avg. ranks	p-value	α	Avg. ranks	p-value	α	Avg. ranks	p-value	α
20%	EBO-1 vs DF-PU	**1.05** vs 1.95	**0.000004**	0.025	**1.05** vs 1.95	**0.000004**	0.025	1.8 vs **1.2**	**0.005**	0.025
	EBO-1 vs S-EM	**1.1** vs 1.9	**0.00005**	0.05	**1.1** vs 1.9	**0.00002**	0.05	1.52 vs **1.48**	0.520	0.05
40%	EBO-1 vs DF-PU	**1.1** vs 1.9	**0.00005**	0.025	**1.05** vs 1.95	**0.00001**	0.025	1.9 vs **1.1**	**0.00001**	0.025
	EBO-1 vs S-EM	**1.2** vs 1.8	**0.017**	0.05	**1.1** vs 1.9	**0.00001**	0.05	1.9 vs **1.1**	**0.00001**	0.05
60%	EBO-1 vs DF-PU	**1.25** vs 1.75	**0.015**	0.025	**1.15** vs 1.85	**0.000004**	0.025	1.9 vs **1.1**	**0.00001**	0.025
	EBO-1 vs S-EM	1.55 vs **1.45**	0.452	0.05	**1.35** vs 1.65	**0.033**	0.05	1.9 vs **1.1**	**0.0001**	0.05

Table 6.17 summarises the statistical significance of the F-measure results from Table 6.16, as well as the results for precision and recall. The results follow a largely similar trend to those reported in Table 6.13, with EBO-1 outperforming the baseline methods for F-measure and precision across all values of δ with statistical significance, except in the case of EBO-1 vs S-EM when δ=60%, where S-EM outperformed EBO-1 for F-measure. Recall that this was also the case for BO-1 when compared with S-EM on the synthetic datasets. The performance of the baseline methods in regard to recall follow the same trend as the results of the previous sections, with the baseline methods outperforming BO-1 with statistical significance. However, this is due to the large overprediction of the positive class as previously discussed.

Moving on to EBO-2, Table 6.18 reports the results of EBO-2 compared with the baseline methods on the synthetic datasets, whilst Table 6.19 summarises the results of the statistical significance tests. EBO-2 again largely outperforms the baseline methods regarding F-measure and precision for all values of δ, with the exception of S-EM for F-measure when δ=60%. The difference in performance is statistically significant in all cases when EBO-2 outperforms the baselines, with the exception of S-EM when δ=40% for F-measure. For recall, as expected based on previous results, the baseline methods outperform EBO-2 for all values of δ with statistical significance.

Table 6.18. F-measure results of EBO-Auto-PU with extended search space and two baseline PU learning methods on synthetic datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**EBO-2**	**DF-PU**	**S-EM**	**EBO-2**	**DF-PU**	**S-EM**	**EBO-2**	**DF-PU**	**S-EM**
1	0.670	0.484	0.616	0.654	0.484	0.602	0.570	0.483	0.613
2	0.204	0.125	0.194	0.024	0.120	0.130	0.050	0.112	0.120
3	0.766	0.552	0.589	0.707	0.552	0.587	0.582	0.552	0.600
4	0.809	0.454	0.644	0.793	0.416	0.633	0.710	0.417	0.630
5	0.452	0.357	0.402	0.389	0.356	0.436	0.411	0.357	0.465
6	0.709	0.403	0.477	0.652	0.402	0.525	0.622	0.402	0.582
7	0.585	0.285	0.433	0.537	0.283	0.462	0.509	0.282	0.451
8	0.559	0.326	0.468	0.386	0.326	0.457	0.385	0.326	0.439
9	0.068	0.035	0.099	0.041	0.000	0.044	0.114	0.000	0.120
10	0.975	0.233	0.612	0.960	0.234	0.627	0.879	0.233	0.663
11	0.622	0.491	0.505	0.563	0.491	0.520	0.498	0.490	0.517
12	0.734	0.397	0.550	0.676	0.397	0.567	0.609	0.394	0.586
13	0.628	0.500	0.551	0.605	0.460	0.556	0.510	0.456	0.549
14	0.905	0.529	0.817	0.854	0.529	0.840	0.702	0.529	0.873
15	0.602	0.387	0.423	0.536	0.387	0.425	0.486	0.385	0.422
16	0.544	0.239	0.414	0.442	0.239	0.401	0.364	0.240	0.299
17	0.323	0.214	0.262	0.207	0.214	0.281	0.211	0.214	0.267
18	0.517	0.372	0.444	0.467	0.373	0.433	0.397	0.373	0.422
19	0.427	0.378	0.426	0.351	0.378	0.429	0.275	0.376	0.413
20	0.651	0.610	0.615	0.621	0.610	0.620	0.539	0.610	0.613

Table 6.19. Results of Wilcoxon signed-rank tests with Holm correction for multiple hypothesis when comparing EBO-2 against S-EM and DF-PU regarding F-measure, Precision and Recall, for the 3 δ values for the synthetic datasets.

δ (%)	**Methods compared**	**F-measure**			**Precision**			**Recall**		
		Avg. ranks	**p-value**	**α**	**Avg. ranks**	**p-value**	**α**	**Avg. ranks**	**p-value**	**α**
20%	EBO-2 vs DF-PU	**1.0** vs 2.0	**0.000002**	0.025	**1.0** vs 2.0	**0.000002**	0.025	1.95 vs **1.05**	**0.00001**	0.05
	EBO-2 vs S-EM	**1.05** vs 1.95	**0.00001**	0.05	**1.05** vs 1.95	**0.000004**	0.05	1.95 vs **1.05**	**0.000004**	0.025
40%	EBO-2 vs DF-PU	**1.15** vs 1.85	**0.0002**	0.025	**1.05** vs 1.95	**0.000004**	0.025	1.9 vs **1.1**	**0.00001**	0.05
	EBO-2 vs S-EM	**1.3** vs 1.7	0.083	0.05	**1.1** vs 1.9	**0.00001**	0.05	1.98 vs **1.02**	**0.0001**	0.025
60%	EBO-2 vs DF-PU	**1.2** vs 1.8	**0.004**	0.025	**1.05** vs 1.95	**0.000004**	0.025	1.95 vs **1.05**	**0.00001**	0.025
	EBO-2 vs S-EM	1.65 vs **1.35**	0.522	0.05	**1.2** vs 1.8	**0.006**	0.05	1.9 vs **1.1**	**0.00005**	0.05

6.4 The PU Learning Algorithm's Hyperparameter Values Most Frequently Selected by EBO-Auto-PU

In this section we report the optimised PU learning algorithm's hyperparameter values which were most frequently selected by EBO-Auto-PU utilising the base search space (EBO-1) and the extended search space (EBO-2). We report the selection frequency, baseline frequency, and their difference. The selection frequency of a PU learning algorithm's hyperparameter value is calculated as the ratio

of the number of times that value was used in the optimised PU learning algorithm returned by EBO-Auto-PU over the total number of EBO-Auto-PU runs, which is 300 per type of dataset (biomedical or synthetic), considering 20 datasets times 3 values of δ times 5 runs of an EBO-Auto-PU version per dataset, due to the use of 5-fold cross-validation. The baseline frequency is the expected selection frequency of a hyperparameter value if all values of that hyperparameter were randomly selected for use in a PU learning algorithm. I.e., it is calculated by simply dividing 1 (one) by the number of candidate values for that hyperparameter. The difference between these two frequencies is simply the selection frequency minus the baseline frequency.

As mentioned in previous chapters, little has been written on the topic of suitable algorithm configuration for PU learning, and no guidelines exist in the literature. By analysing the most frequently selected hyperparameter values in our experiments, we can begin to understand which PU learning algorithm configurations perform well. This information could prove useful for future research into improving the performance of PU learning algorithms.

Throughout this section, the term "classifier" is used to refer to a classification algorithm (rather than a classification model learned by an algorithm), unless explicitly mentioned otherwise.

6.4.1 The Hyperparameter Values Most Frequently Selected by EBO-1

Table 6.20. Selection frequency of hyperparameter values by EBO-1 for the biomedical datasets.

Hyperparameter	Most selected value	Selection Freq. (%)	Baseline Freq. (%)	Diff. (%)
Phase 1A Iteration count	2	19.67	10.00	9.67
Phase 1A RN Threshold	0.4	13.67	10.00	3.67
Phase 1A Classifier	Logistic reg.	14.00	5.56	8.44
Phase 1B Flag	TRUE	59.67	50.00	9.67
Phase 1 B RN Threshold	0.4	18.00	10.00	8.00
Phase 1B Classifier	SVM	11.67	5.56	6.11
Phase 2 Classifier	Random forest	9.33	5.5%	3.77

Table 6.20 reports the most frequently selected values of the hyperparameters of the optimised PU learning algorithms returned by all runs of EBO-1 on the biomedical datasets. Table 6.21 reports the most frequently selected values of the hyperparameters of the optimised PU learning algorithms returned by all runs of EBO-1 on the synthetic datasets. Starting with the classifiers, Logistic Regression was selected as the most frequent value for Phase 1A classifier for both the biomedical

and the synthetic datasets. This is unsurprising, given the results of the previous two chapters looking at the most frequently selected hyperparameters by GA-Auto-PU and BO-Auto-PU, as logistic regression has occurred frequently as the most selected value for the classifier parameters. As per the discussion of the previous chapters, this supports the hypothesis that the *Phase 1A Classifier* most frequently selected values are adhering to the assumptions of separability and smoothness (see Section 2.5). Logistic regression was also selected most frequently as *Phase 1B Classifier* for the synthetic datasets, whilst support vector machine (SVM) was selected most frequently for the biomedical datasets. SVM has also occurred with relative frequency in the previous results sections, being the most frequently selected *Phase 1B Classifier* for GA-2 on the biomedical datasets, and BO-1 for the synthetic datasets. Several proposed PU learning methods have been based on the SVM classifier [6,15,126,140,172,208,209,210], and these results support SVM as a good choice for a component of a two-step PU learning algorithm. Regarding *Phase 2 Classifier*, the most frequently selected value by EBO-1 for the biomedical datasets was random forest, and for the synthetic datasets was deep forest. Both of these are powerful classifiers that have, like logistic regression, appeared frequently as the most selected classifiers.

Table 6.21. Selection frequency of hyperparameter values by EBO-1 for the synthetic datasets.

Hyperparameter	Most selected value	Selection Freq. (%)	Baseline Freq. (%)	Diff. (%)
Phase 1A Iteration count	2	17.00	10.00	7.00
Phase 1A RN Threshold	0.1	15.67	10.00	5.67
Phase 1A Classifier	Logistic reg.	11.00	5.56	5.44
Phase 1B Flag	TRUE	55.48	50.00	5.48
Phase 1 B RN Threshold	0.25	13.00	10.00	3.00
Phase 1B Classifier	Logistic reg.	8.67	5.56	3.11
Phase 2 Classifier	Deep forest	12.67	5.56	7.11

As with the previous chapters, a more in-depth discussion of the *Phase 1A Iteration Count* parameter has been conducted which has, again, shown moderate to strong correlations between the average most frequently selected value of the *Phase 1A Iteration Count* parameter and the percentage of positive instances in the full unaltered dataset. The Pearson's correlation coefficient values for the biomedical datasets are -0.656, -0.688, and -0.696 when δ = 20%, 40%, and 60% respectively. The Pearson's correlation coefficient values for the synthetic datasets are -0.640, -0.708, and -0.684 when δ = 20%, 40%, and 60% respectively.

6.4.2 The Hyperparameter Values Most Frequently Selected by EBO-2

Table 6.22. Selection frequency of hyperparameter values by EBO-2 for the biomedical datasets.

Hyperparameter	Most selected value	Selection Freq. (%)	Baseline Freq. (%)	Diff. (%)
Phase 1A Iteration count	2	22.33	10.00	12.33
Phase 1A RN Threshold	0.15	18.33	10.00	8.33
Phase 1A Classifier	LDA	10.33	5.56	4.77
Phase 1B Flag	TRUE	56.67	50.00	6.67
Phase 1 B RN Threshold	0.2	14.67	10.00	4.67
Phase 1B Classifier	Logistic reg.	9.00	5.56	3.44
Spy rate	0.1	24.00	5.56	18.44
Spy tolerance	0.01	13.88	10.00	3.88
Spy flag	False	65.67	50.00	15.67
Phase 2 Classifier	Deep forest	10.67	5.56	5.11

Table 6.22 reports the most frequently selected values of the hyperparameters of the optimised PU learning algorithms returned by all runs of EBO-2 on the synthetic datasets. Starting with the classifiers, the *Phase 1A Classifier* most frequently selected values were linear discriminant analysis (LDA) and Gaussian naïve Bayes (NB) for the biomedical and synthetic datasets respectively. Throughout this work, a trend has emerged of favouring linear classifiers for the *Phase 1A Classifier* hyperparameter, with the classifiers LDA, Gaussian NB, Bernoulli NB, and logistic regression being selected in almost every case, with the exception that random forest was selected most frequently by GA-2 on the biomedical datasets. From these results, it can be argued that the best choice of a classifier for the *Phase 1A Classifier* parameter is a linear classifier, adhering to the assumptions of separability and smoothness noted in Section 2.5. Such cannot be argued for the *Phase 1B Classifier*, as little cohesion has emerged regarding the most suitable classifier for this hyperparameter. However, for *Phase 2 Classifier*, a favourite has emerged with the deep forest classifier occurring 6/12 times. Deep forest is a powerful classifier utilised in our baseline method, DF-PU [129]. However, as it is a relatively recently proposed classifier, it has not yet been widely used in the machine learning field. These results indicate that, at least in the area of PU learning, the deep forest classifier is a promising candidate tool.

Table 6.23. Selection frequency of hyperparameter values by EBO-2 for the synthetic datasets.

Hyperparameter	Most selected value	Selection Freq. (%)	Baseline Freq. (%)	Diff. (%)
Phase 1A Iteration count	2	19.00	10.00	9.00
Phase 1A RN Threshold	0.1	20.00	10.00	10.00
Phase 1A Classifier	Gaussian NB	17.67	5.56	12.11
Phase 1B Flag	TRUE	53.67	50.00	3.67
Phase 1 B RN Threshold	0.3	12.67	10.00	2.67
Phase 1B Classifier	Logistic reg.	12.67	5.56	7.11
Spy rate	0.15	18.33	5.56	12.77
Spy tolerance	0.07	16.07	10.00	6.07
Spy flag	False	52.67	50.00	2.67
Phase 2 Classifier	Deep forest	20.33	5.56	14.77

Regarding spy results, *Spy flag* was, again, set to False in the majority of cases, albeit with a much smaller margin than all previous results. The trend seen throughout this work is of candidate solutions that do not utilise the spy method have been heavily favoured over those that do. The spy method is common throughout the PU learning literature, and many extensions and modifications of the method have been proposed [126,163,211,212,213]. However, these results indicate that it may not be as effective as the frequency of its use would suggest. This further justifies the need for Auto-ML systems such as those proposed throughout this work as, simply based on a literature review of the PU learning literature, one would be forgiven for assuming that spy-based methods are the most effective PU learning systems, given the frequency of their use. However, based on these results, it is clear that this is not the case.

Moving on to the discussion of the *Phase 1A Iteration Count* hyperparameter, 2 was the most frequently selected value for both the biomedical and synthetic datasets by EBO-2. This was also the case for both types of datasets for EBO-1, and BO-2. Interestingly, 2, 4, and 1 are the only values to appear as the most frequently selected throughout this work. This is interesting, given that they are relatively low values (in the context of the other available values $\{1 \dots 10\}$. PU learning datasets often exhibit large degrees of class imbalance, and the datasets used in this work are no exception. However, it has been shown throughout this work that the average selected value of this parameter exhibits a moderate to strong correlation to the percentage of positive instances in the full, unaltered datasets. The Pearson's correlation coefficient values for the biomedical datasets for EBO-2 follow this same trend, -0.680, -0.710, and -0.687 when δ = 20%, 40%, and 60% respectively. For the synthetic datasets, -0.721, -0.705, and -0.646 when δ = 20%, 40%, and 60% respectively. It could be

argued that this is shown somewhat in the most frequently selected values. Recalling the dataset characteristics, the biomedical datasets have a much higher percentage of positive instances in the full, unaltered dataset than the synthetic datasets, as the synthetic datasets were constrained to keeping the distribution at 50% positive instances at most. No such constraint was applied to the biomedical datasets as they are real-world datasets, and the dataset with the highest percentage of positive instances has a percentage of 75.38%. GA-1 and GA-2 both selected 1 most frequently for the biomedical datasets, whilst GA-1, GA-2, and EBO-1 selected 4 most frequently for the synthetic datasets. Based on these results, and the correlations shown throughout this work, it can be argued that when assembling a two-step PU learning algorithm, one should consider the class imbalance present when deciding upon the iteration count to apply.

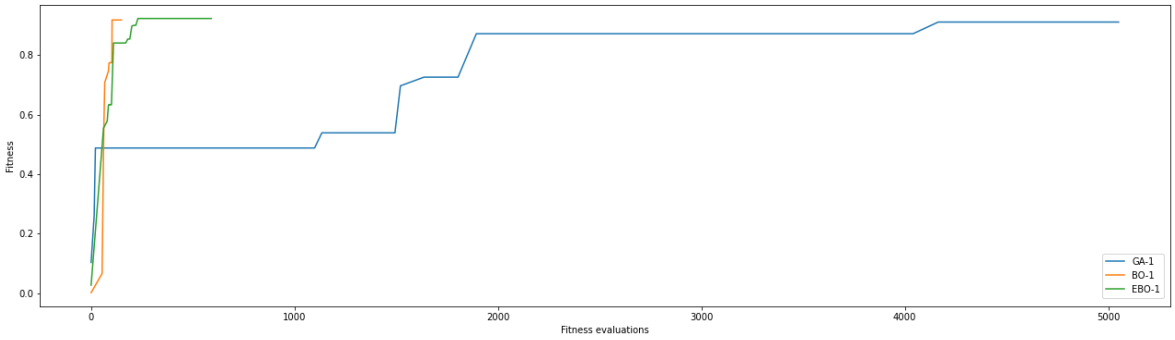
6.5 Comparing the Auto-PU Systems' Learning Rates

In this section, the learning rates of the three Auto-ML systems are compared. That is, Figures 6.4 to 6.6 display the fitness value of each individual (candidate solution) whose fitness is evaluated by the objective function, for GA-1, BO-1, and EBO-1 for three datasets. First, the 'Kidney Disease' dataset, as it was on this dataset that most methods achieved the highest F-measure. Second, the 'Parkinsons Biom.' dataset, as it was on this dataset that most methods achieved the lowest F-measure. Finally, the 'PI Diabetes' dataset, as this dataset generally elicited mid-level performance for all methods. These datasets' main characteristics are shown in Table 3.1.

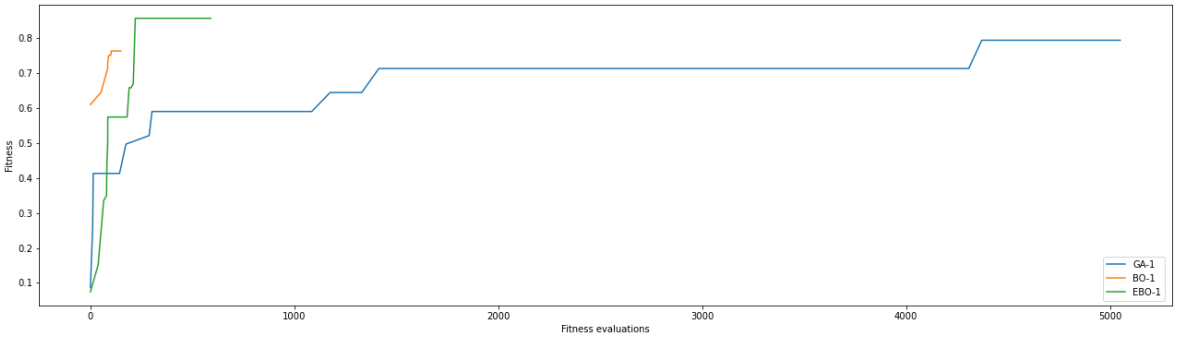
This analysis focusses only on this sample of three biomedical datasets. Given the large number of datasets used in this work, providing graphs and analysis for all would be impractical. As the biomedical datasets are arguably more important than the synthetic datasets due to their real-world applications, we sampled only the biomedical datasets for this analysis. Furthermore, we have selected only the base-search space implementations for this analysis, given that the distinction between base vs extended search space is not critical.

The x axis on the graphs shows the number of fitness evaluations, whilst the y axis shows the best fitness observed at that point in the optimisation procedure. The number of fitness evaluations is being used as a proxy for the 'cost' of each Auto-PU system, so that a direct comparison can be made between the systems. This is more accurate than simply using the number of iterations as an

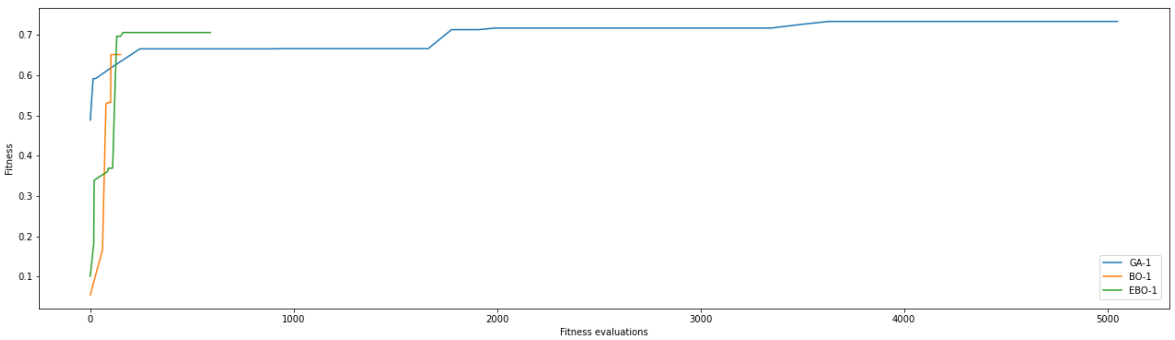
iteration for the GA-based system would be much slower to complete than an iteration for the BO-based system. The fitness evaluation procedure is by far the biggest bottleneck in the three Auto-PU systems, so whilst each system does perform additional steps (such as the evolution procedures of the GA, or the surrogate model training of the BO), the time taken for these steps is relatively small in the context of the fitness evaluation. This is discussed in more detail in Section 5.1.3. GA-Auto-PU performs by far the most fitness evaluations (101 individuals × 50 generations = 5050), whilst BO-Auto-PU performs the least (101 candidate solutions in the first iteration, then 1 × 49 iterations = 150). EBO-Auto-PU sits between the two (101 candidate solutions in the first iterations, then 11 × 49 iterations = 640).



(a). δ = 20%.

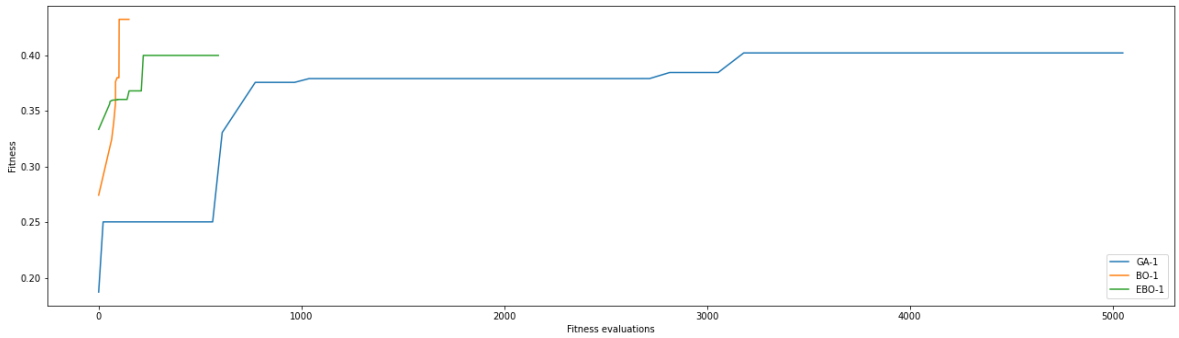


(b). δ = 40%.

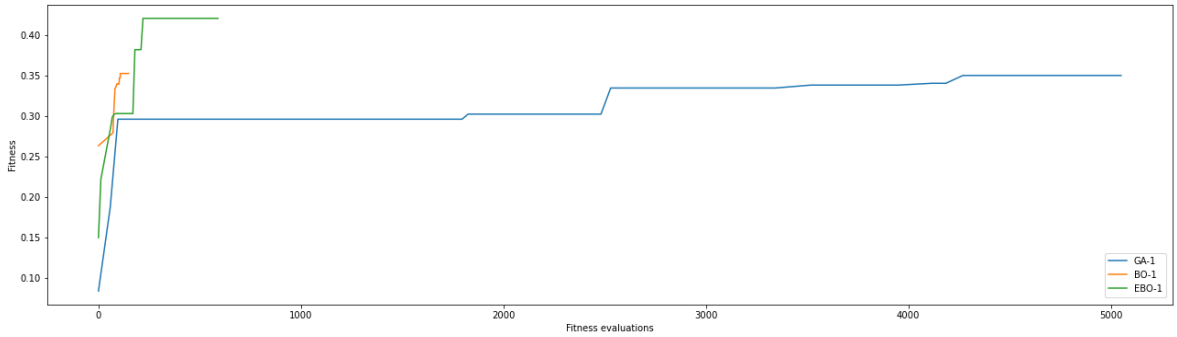


(c). δ = 60%.

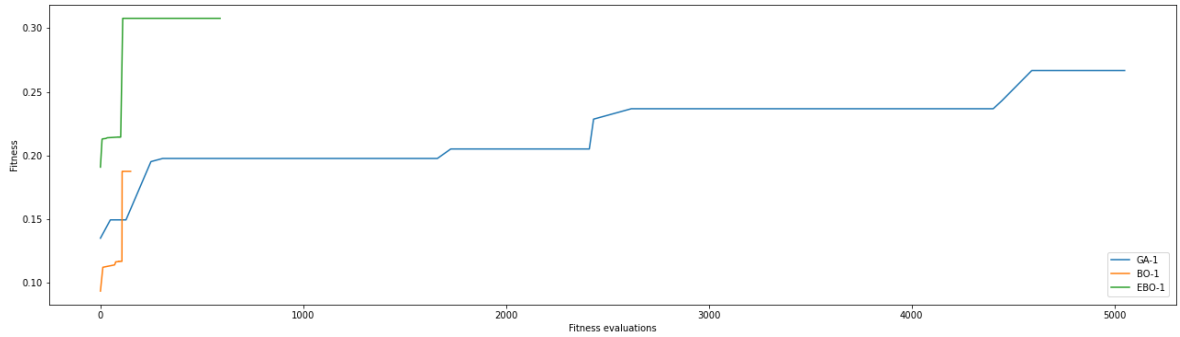
Figure 6.4. Learning rates of the Auto-PU systems on Kidney Disease dataset, varying the δ value.



(a). δ = 20%.

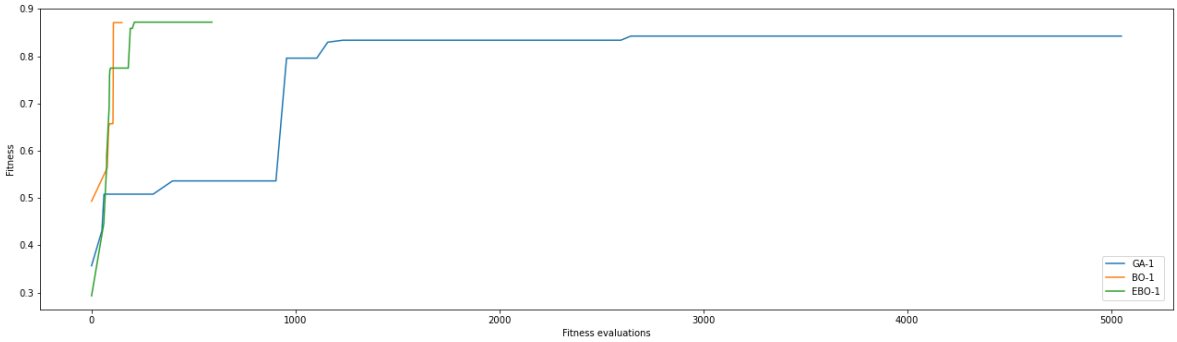


(b). δ = 40%.

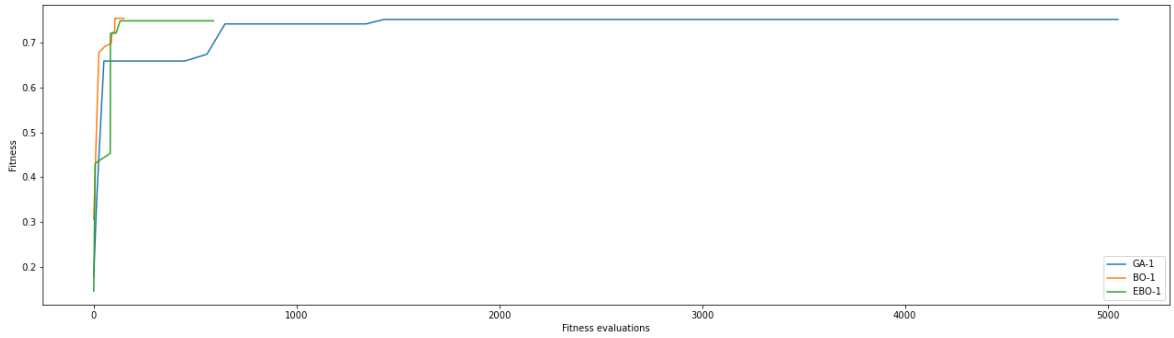


(c). δ = 60%.

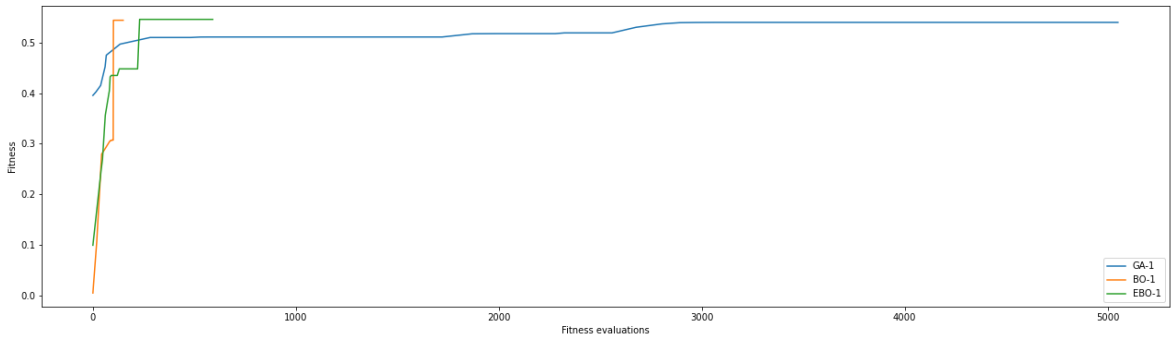
Figure 6.5. Learning rates of the Auto-PU systems on Parkinson's Biom. dataset, varying the δ value.



(a). δ = 20%.



(b). δ = 40%.



(c). δ = 60%.

Figure 6.6. Learning rates of the Auto-PU systems on PI Diabetes dataset, varying the δ value.

While GA-1 demonstrates a conventional learning rate pattern for genetic algorithms, characterized by initial large jumps in performance and subsequent convergence, BO-1 deviates from this trend by exhibiting early convergence with only 1 or 2 substantial performance jumps. It could be argued that this early convergence is likely due to BO-1 converging to a local optima. However, in 6 out of the

9 cases it achieves a higher fitness than GA-1 at its final fitness evaluation, which is much sooner than the final fitness evaluation of GA-1 (which performs a much greater number of fitness evaluations). This supports claims in Chapter 5 that BO-1 was generally a more efficient optimiser than GA-1.

EBO-1 sits somewhere between GA-1 and BO-1 regarding their learning rate trend, with fewer jumps in performance than GA-1 but more than BO-1.

Overall, GA-1 achieves the highest fitness at the final fitness evaluation in 1 out of the 9 analysed cases, BO-1 in 2 out of 9 cases, and EBO-1 in 6 out of 9 cases; and in general GA-1 needs a much larger number of fitness evaluations in order to achieve fitness values competitive with BO-1 and EBO-1.

6.6 Summary

Overall, considering the comparison between the EBO-based systems and the PU learning baselines, both EBO-1 and EBO-2 outperformed the baselines in general, with statistical significance regarding F-measure and precision in several cases. S-EM did, however, show a large increase in predictive performance for δ=60%, as has been shown and discussed in the previous chapters.

Regarding the comparison with the GA-based and BO-based systems, the experimental results indicate that EBO-1 performs favourably in terms of predictive performance when compared to GA-1, and BO-1. Although statistical significance is not observed in most cases, EBO-1 consistently outperforms or performs on par with the compared systems for F-measure, precision, and recall. EBO-2 performs similarly to BO-2 and GA-2 in terms of F-measure, precision, and recall, with some exceptions. EBO-2 achieves the best average rank in the majority of cases, outperforming GA-2 in precision at δ=40% with statistical significance. However, the results indicate that GA-2 generally outperforms EBO-2 in terms of F-measure and recall, with statistical significance observed in multiple instances. Conversely, EBO-2 exhibits better precision performance in some cases. Regarding computational efficiency, the EBO-based systems strike a balance between the GA and BO-based systems in terms of computational efficiency. Overall, considering both predictive performance and computational efficiency, EBO-1 emerges as the preferred system compared to the other methods evaluated in this research.

To summarise the data regarding the most frequently selected hyperparameter values, the results indicate that linear classifiers, such as Logistic Regression, LDA, and Gaussian NB, are favoured as Phase 1A Classifiers in PU learning algorithms, supporting the hypothesis that the selection of this hyperparameter adheres to the assumptions of separability and smoothness that are fundamental to the two-step PU learning framework. The Phase 1B Classifier does not exhibit a clear preferred choice across datasets, indicating that the selection of this hyperparameter is highly dataset specific. For Phase 2 Classifier, the deep forest classifier shows promise in both biomedical and synthetic datasets, suggesting that this relatively recently proposed classifier is a powerful tool in the context of PU learning. The results also show that low values of the Phase 1A *Iteration Count* hyperparameter are commonly selected and exhibit a correlation with the percentage of positive instances in the datasets. Thus, when designing a two-step PU learning algorithm, one should consider the class distribution when setting this value. Additionally, the Spy method is, surprisingly, not frequently utilized, despite its prevalence in PU learning literature. These findings emphasize the importance of automated systems, like the proposed Auto-ML systems, in selecting suitable hyperparameters for PU learning algorithms based on dataset characteristics and performance correlations.

At the start of this chapter 2 research questions were posed to evaluate EBO-Auto-PU. Firstly, does EBO-Auto-PU present a good trade-off, in regard to computational efficiency, between GA-Auto-PU and BO-Auto-PU? And, secondly, does EBO-Auto-PU achieve good predictive performance compared with GA-Auto-PU? To answer the first question, EBO-Auto-PU does present a good trade-off in regard to computational efficiency, performing 2.06 - 2.15 times slower than BO-Auto-PU, but 11.05-12.5 times faster than GA-Auto-PU. It can be argued that this is a good trade-off, given that 11.05-12.5 is, arguably, a substantial improvement upon GA-Auto-PU, whilst 2.06-2.15 is not a substantial decline in performance compared with BO-Auto-PU. In response to the second question, considering EBO-1 and EBO-2 separately, the answers are yes and no respectively. The aim was to improve upon the performance of GA-Auto-PU, which, despite lacking statistical significance, EBO-1 did against GA-1. EBO-1 achieved a superior rank against both GA-1 and BO-1, thus representing an improvement in performance. Whilst these results are not statistically significant, they are an improvement. Statistical significance is not the sole indicator of improvement, it is a commonly used measure to determine whether observed differences are likely due to chance

or a genuine effect. However, it does not discount the possibility of meaningful trends or improvements that fall short of statistical significance. In this context, the results of EBO-1, even without statistical significance, demonstrate a consistent trend of outperforming both GA-1 and BO-1. Considering EBO-2, it cannot be argued that this system improved upon GA-2 in regard to predictive performance. So, to evaluate both EBO-1 and EBO-2 in regard to the research questions posed, it can be said that both achieve the goal of the first question, whilst EBO-1 achieves the goal set by the second. Thus, to conclude, EBO-1 has obtained overall the best predictive performance of the Auto-PU systems evaluated in this work.

Chapter 7

Conclusions

Positive-Unlabelled (PU) learning is an under-explored area of machine learning that has potential to aid challenging learning tasks, where fully labelled data is impractical or impossible to obtain. This learning paradigm occurs frequently, as discussed in Chapters 1 and 2, and naturally occurs in areas of great importance such as medical diagnosis, gene function prediction, and cyber security.

However, the field is challenging. Until recently, guidance regarding specifically how to evaluate PU learning classifiers was limited. Standard evaluation metrics cannot be calculated when using genuine PU data, and the right metric to use for evaluation depends heavily on the learning task. Furthermore, many PU learning methods have been proposed in the literature, with little guidance regarding algorithm construction.

In this work, these issues have been approached. Whilst there are limitations to the solutions presented, it is hoped that the contributions aid the literature of PU learning and provide guidance to future researchers. This chapter outlines the contributions made (Section 7.1) and future research directions (Section 7.2).

7.1 Summary of Contributions

In the Introduction chapter of this thesis, its primary contributions were briefly outlined. In this section, these contributions will be discussed in more detail.

7.1.1 A Framework for Evaluating the Predictive Performance of PU Learning Algorithms

Evaluation of PU learning algorithms is non-trivial, as was discussed in Section 2.5, given that the true class labels of all instances are not defined. Thus, metrics such as the true positive, false positive, true negative, and false negative counts cannot be accuracy calculated. These values are the foundation of most of the popular evaluation metrics in the field of standard classification, thus a challenge is presented.

In our previous work [20], a literature review was conducted to establish the most frequently used evaluation metrics and the most frequently used type of dataset. By type of dataset, we refer to either genuine PU data, or PN (Positive-Negative) data that has been engineered to a PU dataset. Guidelines for evaluation were established, determining that newly proposed PU learning algorithms should be evaluated on engineered PU datasets with varying percentages of positive instances hidden in the unlabelled set. F-measure, precision, and recall should all be reported, given that either precision or recall may be important depending on the application. When evaluating on genuine PU data, if using standard evaluation metrics, it should be noted that these metrics are simply an estimation, rather than accurate calculations. However, before evaluating on genuine PU data, it is important to first gain an understanding of the models' performance by evaluating on engineered PU data as discussed. In addition, to aid in the evaluation of PU learning algorithms, benchmarking datasets have been made publicly available[6].

7.1.2 An Auto-ML Framework for PU Learning

Chapter 3 of this thesis details the Auto-ML framework used for developing the Auto-PU systems presented in this work. This was provided in detail so that other researchers can utilise it for development of Auto-ML systems specific to PU learning.

The framework features a flexible search space structure, allowing researchers to incorporate various algorithmic components of PU learning, such as different classifiers and more discrete values for the numeric hyperparameters. Furthermore, the search spaces could be adapted to consider

[6] https://github.com/jds39/Unlabelled-Datasets/

continuous values, rather than the discrete values currently used. However, the discrete values allow for a more controlled analysis of the correlation between some of the most frequently selected hyperparameter values and the predictive performance of the Auto-PU systems, such as was conducted in this work. The framework also presents a clearly defined objective function that can be used for the evaluation of two-step PU learning methods.

7.1.3 The Proposed Auto-PU Systems

The Auto-PU systems proposed were the primary contributions of this work. All three of these systems utilise the Auto-ML framework outlined in Section 7.1.2. Each of the Auto-PU systems had two versions, with a base or extended search space, and each version was evaluated in two separate experiments involving 20 biomedical datasets and 20 synthetic datasets.

GA-Auto-PU was the first Auto-ML system specific to PU learning. It achieved statistically significant better performance against TPOT, a state-of-the-art Auto-ML system for binary classification, and against two strong baseline PU learning methods. However, regarding efficiency, GA-Auto-PU is expensive to run, averaging 226.3 and 223.2 minutes to run a 5-fold cross-validation procedure per dataset, when utilising the base search space and the extended search space respectively. To improve upon the computational efficiency, BO-Auto-PU was proposed.

BO-Auto-PU is a Bayesian optimisation (BO)-based Auto-ML system that achieved the goal of improving the computational efficiency of GA-Auto-PU, averaging 8.4 and 9.8 minutes to run a 5-fold cross-validation procedure per dataset, when using the base search space and the extended search space respectively, being 23-27 times faster than GA-Auto-PU. However, this was achieved at a small loss to predictive performance. It was hypothesised that this loss in predictive performance could be due to a lack of population diversity by the BO-based system in comparison to the GA. GA-Auto-PU introduces diversity through the use of evolutionary operators, but no such diversity is introduced by BO-Auto-PU. Therefore, it follows that an improved approach could strike a trade-off between the two systems in regard to computational efficiency and population diversity.

EBO-Auto-PU was proposed based on a new hybrid approach between BO and evolutionary computation. This optimisation procedure is a contribution in itself and is discussed in Section 7.1.5. EBO-Auto-PU achieved the aim of striking a trade-off between GA-Auto-PU and BO-Auto-PU in

regard to computational efficiency, performing 2.06–2.15 times slower than BO-Auto-PU, but 11.05–12.50 times faster than GA-Auto-PU. This is a substantial improvement on GA-Auto-PU's runtime, arguably without being a substantial increase on BO-Auto-PU's runtime.

Regarding predictive performance, EBO-Auto-PU utilising the base search space consistently outperformed or performed on par with GA-Auto-PU and BO-Auto-PU. Whilst statistical significance was not achieved, the system still exhibited superior performance and a good trade-off in computational efficiency, and thus EBO-Auto-PU emerged as the preferred of the three Auto-PU systems proposed in this work. The results for EBO-Auto-PU utilising the extended search space were not as good, with GA-Auto-PU performing best overall when utilising the extended search space. A possible reason as to why is due to the much larger number of possible candidate solutions in the extended search space, compared with the base search space. GA-Auto-PU performs a global search and evaluates many candidate solutions throughout the run. As BO-Auto-PU and EBO-Auto-PU employ a surrogate model, they are susceptible to becoming trapped in local optima, as discussed in Section 2.3. It is possible that the extended search space requires a more extensive search, as conducted by GA-Auto-PU.

It is worth noting that all three proposed Auto-PU systems achieved statistically significantly better F-measure results than two baseline PU learning methods (the S-EM ("Spy") method and deep forest for PU learning), and as such they are useful contributions to the PU learning area. The only situation where the Auto-PU systems struggled against the baselines in regard to F-measure was when the percentage of positive examples hidden in the unlabelled set (as described in Section 3.3) was set to $\delta = 60\%$. Specifically, S-EM outperformed BO-1, BO-2, EBO-1, and EBO-2 on the synthetic datasets when $\delta = 60\%$, although these results were not statistically significant. GA-1 and GA-2 both outperformed S-EM in this scenario. It can, therefore, be argued that when presented with a challenging PU learning task where the majority of positive instances are included in the unlabelled set, the GA-based systems are most suitable.

Given the analysis of the results of the Auto-PU systems, certain conclusions can be drawn regarding which system to use in which scenario. For challenging learning tasks, the GA-based systems are arguably the best choice of the three systems due to the robust global search and its ability to perform well when a high number of positive instances are hidden in the unlabelled set. If

fast execution time is an important criterion, the BO-based systems are the best choice given that they are the best system in regard to computational runtime, and predictive performance is not significantly decreased in comparison to the other two systems. The EBO-based systems are a good trade-off between the two other systems, and it achieves the best predictive performance overall when utilising the base search space. Thus, EBO-Auto-PU with the base search space can be considered overall the best Auto-ML system among the six versions of Auto-PU systems in this thesis (two versions for each of the three types of Auto-PU systems).

Regarding the base versus the extended search space, the results throughout this thesis have frequently shown a preference for not utilising the "Spy" approach to PU learning. As discussed, this is somewhat surprising given the prevalence of the Spy approach in the PU learning literature. However, the results arguably lead to the conclusion that, for the Auto-PU systems, the base search space is superior given that the only addition of the extended search space is the Spy components, and these are not frequently selected by the Auto-PU systems.

7.1.4 Analysis of Frequently Selected PU Learning Algorithm Components

In Chapters 4, 5, and 6 of this work, an analysis of the PU learning algorithm components most frequently selected by the Auto-PU systems was conducted. From this analysis, guidelines can be established regarding PU learning algorithm design for datasets with specific characteristics.

To summarise this analysis, starting with classifiers, linear classifiers were favoured as the Phase 1A Classifier, which adheres to the assumptions of separability and smoothness that underly the two-step PU learning framework. The most popular selection for the Phase 2 Classifier was the deep forest classifier, a relatively recently proposed classifier that serves as the classifier used in a baseline method, DF-PU.

Extra analysis was conducted on the Phase 1A *Iteration Count* hyperparameter, analysing the correlation between the values selected and the class distributions of the datasets. This analysis revealed a reasonable degree of correlation between the iteration count hyperparameter and the class distribution. Thus, when designing a two-step PU learning algorithm, one should consider the class distribution when setting this value.

Regarding the "Spy" approach to PU learning (see Section 2.5), determined by the Spy Flag hyperparameter, as mentioned earlier, it was, surprisingly, frequently not utilised in the optimised PU learning algorithms. Considering that the use of this Spy approach greatly increases the size of the search space of the proposed Auto-PU systems, the use of this approach is not recommended for these systems.

7.1.5 Evolutionary Bayesian Optimisation (EBO)

Due to the observed results of the GA-based and the BO-based systems in regard to predictive performance and computational efficiency, a hybrid approach between the two was developed. The GA-based system exhibited high predictive performance but low computational efficiency, whereas the BO-based system did not perform as well in regard to predictive performance but was much more computationally efficient. As mentioned earlier, it was hypothesised that the decline in predictive performance by the BO-based system was due to a lack of population diversity, and that if population diversity could be increased, predictive performance would improve. However, the aim was for this to be done such that computational efficiency did not substantially suffer.

This hybrid was achieved through the development of EBO, utilising evolutionary operators and a population to introduce diversity, whilst employing a surrogate model to ensure computational efficiency. Full details of this approach can be found in Section 6.2 and results utilising EBO in the EBO-Auto-PU system are found in Section 6.3. The development of EBO is a contribution not just to the PU learning literature through EBO-Auto-PU, but the field of Auto-ML optimisers as a whole.

7.2 Future Research Directions

In this section, future research directions are outlined to develop upon the contributions made in this work.

7.2.1 Experiments with More Datasets

Although each of the Auto-PU systems proposed in this work achieved their specified goals, the difference in performance between the systems was in general not statistically significant. To

improve the analysis, further experiments can be done with more datasets to increase the sample size and conduct a more rigorous experimentation. Furthermore, this work focussed on datasets in the biomedical domain, but other domains relevant to PU learning can be explored.

7.2.2 Further Comparisons Against Other Baseline PU Learning Methods

In this work, the proposed Auto-PU systems were compared against two strong baseline PU learning methods. However, for a more rigorous evaluation of the systems, comparisons against other baseline PU learning methods could be conducted. As PU learning is a growing field, there is little doubt that many new strong methods will be developed in the future. It is encouraged that the Auto-PU systems should be compared against these proposed new methods to continue to establish their competitiveness in the expanding environment of PU learning algorithms.

7.2.3 Alternative Search Spaces

Chapter 3 of this thesis proposed two search spaces to be explored with the Auto-PU systems, named as the base search space and the extended search space. The base search space defines the PU learning landscape as a discrete subset of algorithm components that constitute a two-step PU learning method. The extended search space adds spy components into the base search space, allowing the development of two-step PU learning methods that utilise the spy approach. However, other search spaces could be defined, either as an extension to the existing search spaces, such as allowing continuous rather than discrete values or additional discrete values to those already defined, or entirely new search spaces, such as explorations of the biased approach to PU learning.

7.2.4 Optimising the Hyperparameters of the Auto-ML Systems

Throughout this thesis, the Auto-PU systems have used their default hyperparameter settings, based on settings often used in the EA or BO literature. That is, although each Auto-ML system optimised the hyperparameter settings of a two-step PU learning method, there was no attempt to optimise the hyperparameter settings of the Auto-ML systems themselves, which would be a new (meta)-level of

optimisation. Hence, a natural but challenging approach for future research would be to optimise the hyperparameter settings of the Auto-ML systems themselves. This can be achieved e.g., through adaption or self-adaptation of hyperparameter settings during the evolutionary search, in the case of EAs [79], or an equivalent approach in the case of BO.

7.2.5 Developing New Multi-Objective Auto-ML Systems

The Auto-PU systems proposed in this thesis focus on predictive performance (more specifically the F-measure) as their single optimisation objective. However, a growing area of interest within Auto-ML is the computational efficiency of optimised classification pipelines. As such, a future research direction could be to introduce a new version of the Auto-PU systems that optimise for both predictive performance and computational efficiency of the produced PU learning algorithms. Such a system could to some extent favour PU learning algorithms that utilise efficient, fast classifiers (e.g., naïve Bayes) over complex, slow classifiers (e.g., multilayer perceptron); if this preference would have little or no impact on the predictive performance of the produced PU learning algorithms.

References

[1] Witten, I.H., Frank, E., Hall, M.A. and Pal, C.J., 2005. *Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.

[2] Tan, P.N., Steinbach, M. and Kumar, V., 2016. *Introduction to Data Mining*. Pearson Education India.

[3] Bekker, J. and Davis, J., 2020. Learning from Positive and Unlabeled Data: A Survey. *Machine Learning*, *109*(4), pp.719-760.

[4] Elkan, C. and Noto, K., 2008. Learning Classifiers from Only Positive and Unlabeled Data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.213-220.

[5] Li, X. and Liu, B., 2003. Learning to Classify Texts Using Positive and Unlabeled Data. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence.* 3, pp.587-592.

[6] Zheng, Y., Peng, H., Zhang, X., et al., 2019. DDI-PULearn: A Positive-Unlabeled Learning Method for Large-Scale Prediction of Drug-Drug Interactions. *BMC Bioinformatics*, *20*(19), pp.1-12.

[7] Škunca, N., Roberts, R.J. and Steffen, M., 2017. Evaluating Computational Gene Ontology Annotations. In *The Gene Ontology Handbook*, pp.97-109.

[8] Yao, Q., Wang, M., Chen, Y., et al., 2018. Taking Human Out of Learning Applications: A Survey on Automated Machine Learning. *arXiv preprint arXiv:1810.13306*.

[9] He, X., Zhao, K. and Chu, X., 2021. AutoML: A Survey of the State-of-the-Art. *Knowledge-based Systems, 212*, pp.1-27.

[11] Li, F., Zhang, Y., Purcell, A.W., et al., 2019. Positive-Unlabelled Learning of Glycosylation Sites in the Human Proteome. *BMC Bioinformatics*, *20*, pp.1-17.

[12] Chapel, L., Alaya, M.Z. and Gasso, G., 2020. Partial Optimal Transport with Applications on Positive-Unlabeled Learning. *Advances in Neural Information Processing Systems*, *33*, pp.2903-2913.

[13] Jang, J., Gu, G.H., Noh, J., et al., 2020. Structure-based Synthesizability Prediction of Crystals Using Partially Supervised Learning. *Journal of the American Chemical Society*, *142*(44), pp.18836-18843.

[14] Gao, Y., Shi, B., Dong, B., et al., 2021. Tax Evasion Detection with FBNE-PU Algorithm Based on PnCGCN and PU Learning. *IEEE Transactions on Knowledge and Data Engineering*, *35*(1), pp.931-944.

[15] Li, Z., Hu, L., Tang, Z. and Zhao, C., 2021. Predicting HIV-1 Protease Cleavage Sites with Positive-Unlabeled Learning. *BMC Bioniformatics*, *23*, pp.1-18.

[16] Jin, H., Li, C., Xiao, J., et al., 2022. Detecting Arbitrage on Ethereum Through Feature Fusion and Positive-Unlabeled Learning. *IEEE Journal on Selected Areas in Communications*, *40*(12), pp.3660-3671.

[17] Qachfar, F.Z., Verma, R.M. and Mukherjee, A., 2022. Leveraging Synthetic Data and PU Learning for Phishing Email Detection. In *Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy*, pp.29-40.

[18] Chen, S., Qiu, Y., Li, J., et al., 2023. Precision Marketing for Financial Industry using a PU-Learning Recommendation Method. *Journal of Business Research*, *160*, pp.1-13.

[19] Shunxiang, Z., Aoqiang, Z., Guangli, Z., et al., 2023. Building Fake Review Detection Model Based on Sentiment Intensity and PU Learning. *IEEE Transactions on Neural Networks and Learning Systems*, pp.1-14.

[20] Saunders, J.D. and Freitas, A., 2022. Evaluating the Predictive Performance of Positive-Unlabelled Classifiers: A Brief Critical Review and Practical Recommendations for Improvement. *ACM SIGKDD Explorations Newsletter, 24*(2), pp.5-11.

[21] Saunders, J.D. and Freitas, A.A., 2022. GA-Auto-PU: a Genetic Algorithm-based Automated Machine Learning system for Positive-Unlabeled learning. In *Proceedings of the 2022 Genetic and Evolutionary Computation Conference Companion*, pp.288-291. ACM Press.

[22] Saunders, J.D and Freitas A. A., 2022. Evaluating a New Genetic Algorithm for Automated Machine Learning in Positive-Unlabelled learning. In *Proceedings of the 15th International Conference on Artificial Evolution (EA 2022), Lecture Notes in Computer Science*, *Vol. 14091*, 42-57. Springer.

[23] Russell, S.J., & Norvig, P. 2010. *Artificial Intelligence a Modern Approach*. Pearson Education, Inc.

[24] Hossin, M. and Sulaiman, M.N., 2015. A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, *5*(2), pp.1-11.

[25] Botchkarev, A., 2018. Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology. *arXiv preprint arXiv:1809.03006*.

[26] Gu, Q., Zhu, L. and Cai, Z., 2009. Evaluation Measures of the Classification Performance of Imbalanced Data Sets. In *International Symposium on Intelligence Computation and Applications*, pp.461-471. Springer.

[27] Fawcett, T., 2006. An Introduction to ROC Analysis. *Pattern Recognition Letters*, *27*(8), pp.861-874.

[28] Sokolova, M. and Lapalme, G., 2009. A Systematic Analysis of Performance Measures for Classification Tasks. *Information Processing & Management*, *45*(4), pp.427-437.

[29] Wang, Z. and Bovik, A.C., 2009. Mean Squared Error: Love it or Leave it? A New Look at Signal Fidelity Measures. *IEEE Signal Processing Magazine*, *26*(1), pp.98-117.

[30] Chai, T. and Draxler, R.R., 2014. Root Mean Square Error (RMSE) or Mean Absolute Error (MAE)?–Arguments Against Avoiding RMSE in the Literature. *Geoscientific Model Development*, *7*(3), pp.1247-1250.

[31] Pedregosa, F., Varoquaux, G., Gramfort, A., et al., 2011. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, pp.2825-2830.

[32] Murphy, K. P. 2012. *Machine Learning: A Probabilistic Perspective*. MIT press.

[33] Hastie, T., Tibshirani, R., and Friedman, J., 2009. *The Elements of Statistical Learning*. Springer

[34] Cortes, C. and Vapnik, V., 1995. Support-Vector Networks. *Machine Learning*, *20*(3), pp.273-297.

[35] Priyanka and Kumar, D., 2020. Decision Tree Classifier: A Detailed Survey. *International Journal of Information and Decision Sciences*, *12*(3), pp.246-269.

[36] Breiman, L., 2001. Random Forests. *Machine Learning*, *45*(1), pp.5-32.

[37] Biau, G. and Scornet, E., 2016. A Random Forest Guided Tour. *Test*, *25*(2), pp.197-227.

[38] Geurts, P., Ernst, D., and Wehenkel, L., 2006. Extremely Randomized Trees, *Machine learning*, *63*, pp.3-42.

[39] Breiman, L., 1996. Bagging Predictors. *Machine Learning, 24*(2), pp.123-140.

[40] Guryanov, A., 2019. Histogram-based Algorithm for Building Gradient Boosting Ensembles of Piecewise Linear Decision Trees. In *International Conference on Analysis of Images, Social Networks and Texts*, pp.39-50. Springer.

[41] Zhou, Z.H. and Feng, J., 2019. Deep Forest. *National Science Review*, *6*(1), pp.74-86.

[42] Pal, S.K. and Mitra, S., 1992. *Multilayer Perceptron, Fuzzy Sets, Classification.* IEEE.

[43] Minsky, M.L. and Papert, S.A., 1969. *Perceptrons*. MIT Press.

[44] Abiodun, O.I., Jantan, A., Omolara, A.E., et al., 2018. State-of-the-Art in Artificial Neural Network Applications: A Survey. *Heliyon*, *4*(11), pp.1-41.

[45] Bartz-Beielstein, T., Branke, J., Mehnen, J. and Mersmann, O., 2014. Evolutionary Algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *4*(3), pp.178-195.

[46] Antonio, L.M. and Coello, C.A.C., 2017. Coevolutionary Multiobjective Evolutionary Algorithms: Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation*, *22*(6), pp.851-865.

[47] Al-Sahaf, H., Bi, Y., Chen, Q., et al., 2019. A Survey on Evolutionary Machine Learning. *Journal of the Royal Society of New Zealand*, *49*(2), pp.205-228.

[48] Maier, H.R., Razavi, S., Kapelan, Z., et al., 2019. Introductory Overview: Optimization Using Evolutionary Algorithms and Other Metaheuristics. *Environmental Modelling & Software*, *114*, pp.195-213.

[49] Slowik, A. and Kwasnicka, H., 2020. Evolutionary Algorithms and Their Applications to Engineering Problems. *Neural Computing and Applications*, *32*, pp.12363-12379.

[50] Liang, J., Ban, X., Yu, K., et al., 2022. A survey on Evolutionary Constrained Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation*, *27*(2), pp.201-221.

[51] Zhan, Z.H., Li, J.Y. and Zhang, J., 2022. Evolutionary Deep Learning: A Survey. *Neurocomputing*, *483*, pp.42-58.

[52] Cheng, S., Ma, L., Lu, H., et al., 2021. Evolutionary Computation for Solving Search-based Data Analytics Problems. *Artificial Intelligence Review*, *54*, pp.1321-1348.

[53] Tian, Y., Si, L., Zhang, X., et al., 2021. Evolutionary Large-Scale Multi-Objective Optimization: A Survey. *ACM Computing Surveys*, *54*(8), pp.1-34.

[54] Fister, I., Iglesias, A., Galvez, A. and Fister, D., 2020. Parallel Differential Evolution with Variable Population Size for Global Optimization. In *International Workshop on Soft Computing Models in Industrial and Environmental Applications*, pp.89-99.

[55] Zhang, J., Zhan, Z.H., Lin, Y., et al., 2011. Evolutionary Computation Meets Machine Learning: A Survey. *IEEE Computational Intelligence Magazine*, *6*(4), pp.68-75.

[56] Lambora, A., Gupta, K. and Chopra, K., 2019. Genetic Algorithm – A Literature Review. In *International Conference on Machine Learning, Big Data, Cloud and Parallel Computing* pp.380-384. IEEE.

[57] Telikani, A., Tahmassebi, A., Banzhaf, W. and Gandomi, A.H., 2021. Evolutionary Machine Learning: A Survey. *ACM Computing Surveys*, *54*(8), pp.1-35.

[58] Li, N., Ma, L., Yu, G., et al., 2022. Survey on Evolutionary Deep Learning: Principles, Algorithms, Applications and Open Issues. *ACM Computing Surveys*, pp.1-34.

[59] Eiben, A.E. and Smith, J., 2015. From Evolutionary Computation to the Evolution of Things. *Nature*, *521*(7553), pp.476-482.

[60] Chugh, T., Sindhya, K., Hakanen, J. and Miettinen, K., 2019. A Survey on Handling Computationally Expensive Multiobjective Optimization Problems with Evolutionary Algorithms. *Soft Computing*, *23*, pp.3137-3166.

[61] Vu, T.M., Probst, C., Epstein, J.M., et al., 2019. Toward Inverse Generative Social Science Using Multi-Objective Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1356-1363.

[62] Surry, P.D., and Radcliffe, N.J., 1996. Inoculation to Initialise Evolutionary Search. In *Selected Papers from The Society for the Study of Artificial Intelligence and Simulation of Behaviour Workshop on Evolutionary Computing,* pp.269-285.

[63] Kazimipour, B., Li, X. and Qin, A.K., 2014. A Review of Population Initialization Techniques for Evolutionary Algorithms. In *IEEE Congress on Evolutionary Computation,* pp.2585-2592. IEEE.

[64] Kora, P. and Yadlapalli, P., 2017. Crossover Operators in Genetic Algorithms: A Review. *International Journal of Computer Applications*, *162*(10), pp.34-36.

[65] Malik, A., 2019. A Study of Genetic Algorithm and Crossover Techniques. *International Journal of Computer Science and Mobile Computing*, *8*(3), pp.335-344.

[66] Sudholt, D., 2020. The Benefits of Population Diversity in Evolutionary Algorithms: a Survey of Rigorous Runtime Analyses. *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pp.359-404.

[67] Jain, B.J., Pohlheim, H. and Wegener, J., 2001. On Termination Criteria of Evolutionary Algorithms. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation,* pp.768-768.

[68] Ghoreishi, S.N., Clausen, A. and Joergensen, B.N., 2017. Termination Criteria in Evolutionary Algorithms: A Survey. In *International Joint Conference on Computational Intelligence,* pp.373-384.

[69] Poli, R., Langdon, W.B., McPhee, N.F. and Koza, J.R., 2008. *A Field Guide to Genetic Programming*. Lulu.com.

[70] Sivanandam S., Deepa S., 2008. Genetic Programming. In *Introduction to Genetic Algorithms*. Springer.

[71] Liang, J. and Xue, Y., 2021. Multi-Objective Memetic Algorithms with Tree-based Genetic Programming and Local Search for Symbolic Regression. *Neural Processing Letters*, pp.1-23.

[72] Ahvanooey, M.T., Li, Q., Wu, M. and Wang, S., 2019. A Survey of Genetic Programming and Its Applications. ACM *Transactions on Interactive Intelligent Systems*, *13*(4), pp.1765-1794.

[73] Pappa, G.L. and Freitas, A.A., 2009. Evolving Rule Induction Algorithms with Multi-Objective Grammar-based Genetic Programming. *Knowledge and Information Systems, 19*(3), pp.283-309.

[74] de Sá, A.G., Pinto, W.J.G., Oliveira, L.O.V. and Pappa, G.L., 2017. RECIPE: A Grammar-based Framework for Automatically Evolving Classification Pipelines. In *European Conference on Genetic Programming,* pp.246-261. Springer.

[75] Veiga, R.V., Barbosa, H.J., Bernardino, H.S., et al., 2018. Multiobjective Grammar-based Genetic Programming Applied to the Study of Asthma and Allergy Epidemiology. *BMC Bioinformatics*, *19*(1), pp.1-16.

[76] Nguyen, T.H. and Tettamanzi, A.G., 2020. Using Grammar-based Genetic Programming for Mining Disjointness Axioms Involving Complex Class Expressions. In *International Conference on Conceptual Structures,* pp. 18-32.

[77] Aleti, A. and Moser, I., 2016. A Systematic Literature Review of Adaptive Parameter Control Methods for Evolutionary Algorithms. *ACM Computing Surveys*, *49*(3), pp.1-35.

[78] Pinel, F., Danoy, G. and Bouvry, P., 2012. Evolutionary Algorithm Parameter Tuning with Sensitivity Analysis. In *Security and Intelligent Information Systems: International Joint Conferences*, pp. 204-216. Springer.

[79] Eiben, A.E. and Smith, J.E., 2015. *Introduction to Evolutionary Computing*. Springer.

[80] Črepinšek, M., Liu, S.H. and Mernik, M., 2013. Exploration and Exploitation in Evolutionary Algorithms: A Survey. *ACM Computing Surveys*, *45*(3), pp.1-33.

[81] M. Pelikan, D.E. Goldberg, E. Cantú-Paz, et al. 1999. BOA: The Bayesian Optimization Algorithm. In *Proceedings of The Genetic and Evolutionary Computation Conference*. pp.525-532.

[82] B. Shahiari, K. Swersky, Z. Wang, et al. 2015. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, *104*(1), pp.148-175.

[83] J. Snoek, H. Larochelle, and R.P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems, 2,* pp.2951-2959.

[84] P.I. Frazier. 2018. A Tutorial on Bayesian Optimization. *arXiv preprint arXiv:1807.02811*

[85] J. Van Hoof & J. Vanschoren. 2021. Hyperboost: Hyperparameter Optimization by Gradient Boosting Surrogate Models. *arXiv preprint arXiv:2101.02289*

[86] De Ath, G., Everson, R.M., Rahat, A.A. and Fieldsend, J.E., 2021. Greed is Good: Exploration and Exploitation Trade-Offs in Bayesian Optimisation. *ACM Transactions on Evolutionary Learning and Optimization*, *1*(1), pp.1-22.

[87] Močkus, J., 1975. On Bayesian Methods for Seeking the Extremum. In *Optimization Techniques IFIP Technical Conference,* pp.400-404.

[88] Yang, L. and Shami, A., 2020. On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice. *Neurocomputing*, *415*, pp.295-316.

[89] Brazdil, P., Carrier, C.G., Soares, C. and Vilalta, R., 2008. *Metalearning: Applications to Data mining*. Springer Science & Business Media.

[90] Thornton, C., Hutter, F., Hoos, H.H. and Leyton-Brown, K., 2013. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* pp.847-855.

[91] Zöller, M.A. and Huber, M.F., 2021. Benchmark and Survey of Automated Machine Learning Frameworks. *Journal of Artificial Intelligence Research*, pp.409-472.

[92] Bergstra, J. and Bengio, Y., 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, *13*(2), pp.281-305.

[93] LaValle, S.M., Branicky, M.S. and Lindemann, S.R., 2004. On the Relationship Between Classical Grid Search and Probabilistic Roadmaps. *The International Journal of Robotics Research*, *23*(7-8), pp.673-692.

[94] Syarif, I., Prugel-Bennett, A. and Wills, G., 2016. SVM Parameter Optimization Using Grid Search and Genetic Algorithm to Improve Classification Performance. *Telkomnika*, *14*(4), p.1502-1509.

[95] Liashchynskyi, P. and Liashchynskyi, P., 2019. Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. *arXiv preprint arXiv:1912.06059*.

[96] Liu, H., Simonyan, K. and Yang, Y., 2018. Darts: Differentiable Architecture Search. *arXiv preprint arXiv:1806.09055*.

[97] Zoph, B. and Le, Q.V., 2016. Neural Architecture Search with Reinforcement Learning. *arXiv preprint arXiv:1611.01578*.

[98] Freitas, A.A., 2002. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer Science & Business Media.

[99] Freitas, A.A., 2010. A Review of Evolutionary Algorithms for Data Mining. *Data Mining and Knowledge Discovery Handbook*, pp.371-400.

[100] Olson, R.S., Bartley, N., Urbanowicz, R.J. and Moore, J.H., 2016. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp.485-492.

[101] Velez, D.R., White, B.C., Motsinger, A.A., et al., 2007. A Balanced Accuracy Function for Epistasis Modeling in Imbalanced Datasets Using Multifactor Dimensionality Reduction. *Genetic Epidemiology: The Official Publication of the International Genetic Epidemiology Society*, *31*(4), pp.306-315.

[102] Gijsbers, P., Vanschoren, J. and Olson, R.S., 2018. Layered TPOT: Speeding Up Tree-based Pipeline Optimization. *arXiv preprint arXiv:1801.06007*.

[103] Xavier-Júnior, J.C., Freitas, A.A., Ludermir, T.B., et al., 2020. An Evolutionary Algorithm for Automated Machine Learning Focusing on Classifier Ensembles: An Improved Algorithm and Extended Results. *Theoretical Computer Science*, *805*, pp.1-18.

[104] Zhou, Z.H., 2012. *Ensemble Methods: Foundations and Algorithms*. CRC press.

[105] Real, E., Liang, C., So, D. and Le, Q., 2020. AutoML-Zero: Evolving Machine Learning Algorithms from Scratch. In *International Conference on Machine Learning*, pp.8007-8019.

[106] Nair, V. and Hinton, G.E., 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning,* pp.807-814.

[107] Bengio, Y., Léonard, N. and Courville, A., 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv preprint arXiv:1308.3432*.

[108] Guha, R., Ao, W., Kelly, S., et al., 2023. MOAZ: A Multi-Objective AutoML-Zero Framework. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp.485-492.

[109] Eibe, F., Hall, M.A. and Witten, I.H., 2016. *The WEKA Workbench. Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers.

[110] Kotthoff, L., Thornton, C., Hoos, H.H., et al., 2017. Auto-WEKA 2.0: Automatic Model Selection and Hyperparameter Optimization in WEKA. In *Journal of Machine Learning Research, 18*(25), pp.1-5.

[111] Feurer, M., Klein, A., Eggensperger, K., et al., 2019. Auto-Sklearn: Efficient and Robust Automated Machine Learning. In *Automated Machine Learning,* pp.113-134.

[112] Kenny, A., Ray, T., Limmer, S., et al., 2023. Hybridizing TPOT with Bayesian Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp.502-510.

[113] Fabris, F. and Freitas, A.A., 2019. Analysing the Overfit of the Auto-Sklearn Automated Machine Learning Tool. In *Machine Learning, Optimization, and Data Science: 5th International Conference,* pp.508-520.

[114] Niu, G., Plessis, M.C.D., Sakai, T., et al., 2016. Theoretical Comparisons of Positive-Unlabeled Learning Against Positive-Negative Learning. *arXiv preprint arXiv:1603.03130*.

[115] Nikdelfaz, O. and Jalili, S., 2018. Disease Genes Prediction by HMM Based PU-Learning Using Gene Expression Profiles. *Journal of Biomedical Informatics*, *81*, pp.102-111.

[116] Vasighizaker, A. and Jalili, S. 2018. C-PUGP: A Cluster-based Positive Unlabelled Learning Method for Disease Gene Prediction and Prioritisation. *Computational Biology and Chemistry*, *76*, pp.23-31.

[117] Yang, P., Li, X., Mei, K., et al. 2012. Positive-Unlabelled Learning for Disease Gene Identification. *Bioinformatics, 28*(20), pp.2640-2647.

[118] Liu, L. and Peng, T., 2014. Clustering-based Method for Positive and Unlabelled Text Categorization Enhanced by Improved TFIDF. *Journal of Information Science and Engineering, 30*, pp.1463-1481.

[119] Ke, T., Yang, B., Zhen, L., et al. 2012. Building High-Performance Classifiers Using Positive and Unlabelled Examples for Text. *International Symposium on Neural Networks*, pp.187-195.

[120] Liu, B., Yu, P., and Li, X. 2002. Partially Supervised Classification of Text Documents. *International Conference on Machine Learning, 2*(485), pp.387-394.

[121] Zhang, Y., Li, L., Zhou, J., et al. 2017. Poster: A PU Learning Based System for Potential Malicious URL Detection. *Proceedings of the ACM Conference on Computer and Communications Security*, pp.2599-2601.

[122] Luo, Y., Cheng, S., Liu, C., et al. 2018. PU Learning in Payload-based Web Anomaly Detection. *Proceedings of the Third International Conference on Security of Smart Cities, Industrial Control System and Communications*, pp.1-5.

[123] Van Engelen, J.E. and Hoos, H.H., 2020. A Survey on Semi-Supervised Learning. *Machine Learning*, *109*(2), pp.373-440.

[124] Jaskie, K. and Spanias, A., 2019. Positive and Unlabeled Learning Algorithms and Applications: A Survey. In *10th International Conference on Information, Intelligence, Systems and Applications,* pp.1-8.

[125] Bekker, J., Robberechts, P. and Davis, J., 2019. Beyond the Selected Completely at Random Assumption for Learning from Positive and Unlabeled Data. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp.71-85.

[126] B. Liu, Y. Dai, X. Li, et al. 2003. Building Text Classifiers Using Positive and Unlabeled Examples. In *The Third International Conference on Data Mining*, pp.179-186.

[127] Wang, Y., Zhang, Y. and Liu, B., 2017. Sentiment Lexicon Expansion Based on Neural PU Learning, Double Dictionary Lookup, and Polarity Association. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing,* pp.553-563.

[128] Yu, H., Han, J. and Chang, K.C.C., 2002. PEBL: Positive Example Based Learning for Web Page Classification Using SVM. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* pp.239-248.

[129] Zeng, X., Zhong, Y., Lin, W. and Zou, Q., 2020. Predicting Disease-Associated Circular RNAs Using Deep Forests Combined with Positive-Unlabeled Learning Methods. *Briefings in Bioinformatics*, *21*(4), pp.1425-1436.

[130] Li, X.L., Zhang, L., Liu, B. and Ng, S.K., 2010. Distributional Similarity vs. PU Learning for Entity Set Expansion. In *Proceedings of the ACL 2010 Conference Short Papers*, pp.359-364.

[131] Xia, R., Hu, X., Lu, J., et al., 2013. Instance Selection and Instance Weighting for Cross-Domain Sentiment Classification via PU Learning. In *Twenty-Third International Joint Conference on Artificial Intelligence,* pp.2176-2182.

[132] Ke, T., Jing, L., Lv, H., et al., 2018. Global and Local Learning from Positive and Unlabeled Examples. *Applied Intelligence*, *48*, pp.2373-2392.

[133] Zhang, J., Wang, Z., Meng, J., et al., 2018. Boosting Positive and Unlabeled Learning for Anomaly Detection with Multi-Features. *IEEE Transactions on Multimedia*, *21*(5), pp.1332-1344.

[134] Schrunner, S., Geiger, B.C., Zernig, A. and Kern, R., 2020. A Generative Semi-Supervised Classifier for Datasets with Unknown Classes. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pp.1066-1074.

[135] Liu, B., Liu, Z. and Xiao, Y., 2021. A New Dictionary-based Positive and Unlabeled Learning Method. *Applied Intelligence*, pp.1-15.

[136] He, Y., Li, X., Zhang, M., et al., 2023. A Novel Observation Points-Based Positive-Unlabeled Learning Algorithm. *Chinese Association for Artificial Intelligence Transactions on Intelligence Technology*, pp.1-19.

[137] Sellamanickam, S., Garg, P. and Selvaraj, S.K., 2011. A Pairwise Ranking Based Approach to Learning with Positive and Unlabeled Examples. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management,* pp.663-672.

[138] Claesen, M., De Smet, F., Suykens, J. & De Moor, B., 2015. A Robust Ensemble Approach to Learn from Positive and Unlabeled Data Using SVM Base Models. *Neurocomputing*, *160*, pp.73-84.

[139] Lee, W.S. and Liu, B., 2003. Learning with Positive and Unlabeled Examples Using Weighted Logistic Regression. In *Proceedings of the International Conference on Machine Learning, 3*, pp.448-455.

[140] Ke, T., Lv, H., Sun, M. and Zhang, L., 2018. A Biased Least Squares Support Vector Machine Based on Mahalanobis Distance for PU Learning. *Physica A: Statistical Mechanics and its Applications*, *509*, pp.422-438.

[141] Elkan, C., 2001, August. The Foundations of Cost-Sensitive Learning. In *International Joint Conference on Artificial Intelligence, 17*(1), pp.973-978.

[142] Denis, F., Laurent, A., Gilleron, R., et al, 2003. Text Classification and Co-Training from Positive and Unlabeled Examples. In *Proceedings of the International Conference on Machine Learning 2003 workshop: The Continuum from Labeled to Unlabeled Data*, pp.80-87.

[143] Calvo, B., Larrañaga, P., and Lozano, J., 2007. Learning Bayesian Classifiers from Positive and Unlabelled Examples. *Pattern Recognition Letters, 28*(16), pp.2375-2384.

[144] Japkowicz, N. and Shah, M., 2011. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press.

[145] Bekker, J. and Davis, J., 2018. Estimating the Class Prior in Positive and Unlabeled Data Through Decision Tree Induction. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, 32*(1), pp.2712-2719.

[146] Du Plessis, M.C. and Sugiyama, M., 2014. Class Prior Estimation from Positive and Unlabeled Data. *IEICE Transaction on Information and Systems, 97*(5), pp.1358-1362.

[147] Basile, T., Di Mauro, N., Esposito, F., et al. 2018. Density Estimators for Positive-Unlabelled Learning. In *Proceedings of the International Workshop on New Frontiers in Mining Complex Patterns,* pp.49-64.

[148] Bekker, J., and Davis, J., 2017. Positive and Unlabelled Relational Classification Through Label Frequency Estimation. In *Proceedings of the International Conference on Inductive Logic Programming*, pp.16-30.

[149] Chaudhari, S., and Shevade, S., 2012. Learning from Positive and Unlabelled Examples Using Maximum Margin Clustering. In *Proceedings of the International Conference on Neural Information Processing*, pp.465-473.

[150] Chiaroni, F., Rahal, M., Hueber, N., et al. 2018. Learning with a Generative Adversarial Network from a Positive Unlabeled Dataset for Image Classification. In *Proceedings of the 25th IEEE International Conference on Image Processing*, pp.1368-1372.

[151] Denis, F., Gilleron, R., and Letouzey, F., 2005. Learning from Positive and Unlabeled Examples. *Theoretical Computer Science*, pp.70-83.

[152] Fung, C., Yu, J., Lu, H., et al. 2006. Text Classification Without Negative Examples Revisit. *IEEE Transactions on Knowledge and Data Engineering, 18*(1), pp.6-20.

[153] Gan, H., Zhang, Y., and Song, Q., 2017. Bayesian Belief Network for Positive Unlabeled Learning with Uncertainty. *Pattern Recognition Letters, 90*, pp.28-35.

[154] He, F., Liu, T., Webb, G.I. and Tao, D., 2018. Instance-Dependent PU Learning by Bayesian Optimal Relabeling. *arXiv preprint arXiv:1808.02180*.

[155] He, J., Zhang, Y., Li, X., & Wang, Y, 2010. Naive Bayes Classifier for Positive Unlabeled Learning with Uncertainty. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pp.361–372.

[156] Hou, M., Chaib-draa, B., Li, C., et al. 2018. Generative Adversarial Positive-Unlabeled Learning. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence,* pp.2255-2261.

[157] Ienco, D., and Pensa, R., 2016. Positive and Unlabeled Learning in Categorical Data. *Neurocomputing*, *196*, pp.113-124.

[158] Kato, M., Teshima, T. and Honda, J., 2019. Learning from Positive and Unlabeled Data with a Selection Bias. *Proceedings of the International Conference on Learning Representations*, pp.1-17.

[159] Lan, W., Wang, J., Li, M., et al. 2016. Predicting Drug-Target Interaction Using Positive-Unlabeled Learning. *Neurocomputing, 206*, pp.50-57.

[160] Li, W., Guo, Q. and Elkan, C., 2010. A Positive and Unlabeled Learning Algorithm for One-Class Classification of Remote-Sensing Data. *IEEE Transactions on Geoscience and Remote Sensing, 49*(2), pp.717-725.

[161] Li, X.L. and Liu, B., 2005. Learning from Positive and Unlabeled Examples with Different Data Distributions. In *Proceedings of the European Conference on Machine Learning*, pp.218-229.

[162] Li, X., Liu, B. and Ng, S.K., 2007. Learning to Identify Unexpected Instances in the Test Set. In *Proceedings of the International Joint Conference on Artificial Intelligence, 7*, pp.2802-2807.

[163] Li, X.L., Yu, P.S., Liu, B. and Ng, S.K., 2009. Positive Unlabeled Learning for Data Stream Classification. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pp.259-270.

[164] Liang, C., Zhang, Y., Shi, P. and Hu, Z., 2012. Learning Very Fast Decision Tree from Uncertain Data Streams with Positive and Unlabeled Samples. *Information Sciences, 213*, pp.50-67.

[165] Mordelet, F. and Vert, J.P., 2014. A Bagging SVM to Learn from Positive and Unlabeled Examples. *Pattern Recognition Letters, 37*, pp.201-209.

[166] Nguyen, M.N., Li, X.L. and Ng, S.K., 2011. Positive Unlabeled Learning for Time Series Classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, 2*, pp.1421-1426.

[167] Peng, T., Zuo, W. and He, F., 2008. SVM Based Adaptive Learning Method for Text Classification from Positive and Unlabeled Documents. *Knowledge and Information Systems*, *16*(3), pp.281-301.

[168] Qin, X., Zhang, Y., Li, C. and Li, X., 2013. Learning from Data Streams with only Positive and Unlabeled Data. *Journal of Intelligent Information Systems, 40*(3), pp.405-430.

[169] Xu, Z., Qi, Z. and Zhang, J., 2014. Learning with Positive and Unlabeled Examples Using Biased Twin Support Vector Machine. *Neural Computing and Applications, 25*(6), pp.1303-1311.

[170] Yang, P., Ormerod, J.T., Liu, W., et al., 2018. AdaSampling for Positive-Unlabeled and Label Noise Learning with Bioinformatics Applications. *IEEE Transactions on Cybernetics, 49*(5), pp.1932-1943.

[171] Yu, H., 2005. Single-Class Classification with Mapping Convergence. *Machine Learning, 61*(1), pp.49-69.

[172] Zhang, Y., Ju, X. and Tian, Y., 2014. Nonparallel Hyperplane Support Vector Machine for PU Learning. In *Proceedings of the 10th International Conference on Natural Computation*, pp.703-708.

[173] Zhang, D. and Lee, W.S., 2005. A Simple Probabilistic Approach to Learning from Positive and Unlabeled Examples. In *Proceedings of the 5th Annual UK Workshop on Computational Intelligence*, pp.83-87.

[174] Zhang, B. and Zuo, W., 2009. Reliable Negative Extracting Based on kNN for Learning from Positive and Unlabeled Examples. *Journal of Computers, 4*(1), pp.94-101.

[175] Zhou, K., Gui-Rong, X., Yang, Q., et al. 2010. Learning with Positive and Unlabelled Examples Using Topic-Sensitive PLSA. *IEEE Transactions on Knowledge and Data Engineering, 22*(1), pp.46-58.

[176] Zhou, J.T., Pan, S.J., Mao, Q. and Tsang, I.W., 2012. Multi-View Positive and Unlabeled Learning. In *Proceedings of the Asian Conference on Machine Learning*, pp.555-570.

[177] Abd Elrahman, S.M. and Abraham, A., 2013. A Review of Class Imbalance Problem. *Journal of Network and Innovative Computing*, pp.332-340.

[178] Ali, H., Salleh, M.M., Saedudin, R., et al., 2019. Imbalance Class Problems in Data Mining: A Review. *Indonesian Journal of Electrical Engineering and Computer Science*, *14*(3), pp.1560-1571.

[179] Fung, G.P.C., Yu, J.X., Lu, H. and Yu, P.S., 2005. Text Classification Without Negative Examples Revisit. *IEEE Transactions on Knowledge and Data Engineering*, *18*(1), pp.6-20.

[180] Zhang, J., Yu, P.S. and Zhou, Z.H., 2014. Meta-Path Based Multi-Network Collective Link Prediction. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.1286-1295.

[181] Shirude, S.B. and Kolhe, S.R., 2016. Classifying Library Resources in Library Recommender Agent Using PU Learning Approach. In *International Conference on Data Mining and Advanced Computing*, pp.79-83.

[182] Zheng, H., Yu, H., Hao, Y., et al., 2021. Distantly Supervised Named Entity Recognition with Spy-PU Algorithm. In *IEEE 2nd International Conference on Pattern Recognition and Machine Learning,* pp.56-63.

[183] Basile, T., Di Mauro, N., Esposito, F., et al., 2017. Density Estimators for Positive-Unlabeled Learning. In *International Workshop on New Frontiers in Mining Complex Patterns*, pp.49–64.

[184] Asuncion, A., & Newman, D., 2007. UCI Machine Learning Repository. *https://archive.ics.uci.edu/*.

[185] Marcus, D.S., Fotenos, A.F., Csernansky, J.G., et al., 2010. Open Access Series of Imaging Studies: Longitudinal MRI Data in Nondemented and Demented Older Adults. *Journal of Cognitive Neuroscience*, *22*(12), pp.2677-2684.

[186] Pereira, B., Chin, S.F., Rueda, O.M., et al., 2016. The Somatic Mutation Profiles of 2,433 Breast Cancers Refine Their Genomic and Transcriptomic Landscapes. *Nature Communications*, *7*(1), pp.1-16.

[187] Fleming, T.R. and Harrington, D.P., 1991. *Counting Processes and Survival Analysis*. John Wiley and Sons Inc.

[188] Islam, M.F., Ferdousi, R., Rahman, S. and Bushra, H.Y., 2020. Likelihood Prediction of Diabetes at Early Stage Using Data Mining Techniques. In *Computer Vision and Machine Intelligence in Medical Image Analysis*, pp.113-125.

[189] Chicco, D. and Jurman, G., 2020. Machine Learning can Predict Survival of Patients with Heart Failure from Serum Creatinine and Ejection Fraction Alone. *BMC Medical Informatics and Decision Making*, *20*(1), pp.1-16.

[190] Hlavnička, J., Čmejla, R., Tykalová, T., et al., 2017. Automated Analysis of Connected Speech Reveals Early Biomarkers of Parkinson's Disease in Patients with Rapid Eye Movement Sleep Behaviour Disorder. *Scientific reports*, *7*(1), pp.1-13.

[191] Emon, M.U., Keya, M.S., Meghla, T.I., et al., 2020. Performance Analysis of Machine Learning Approaches in Stroke Prediction. In *4th International Conference on Electronics, Communication and Aerospace Technology*, pp.1464-1469.

[192] Shinkins, B., Nicholson, B.D., Primrose, J., et al., 2017. The Diagnostic Accuracy of a Single CEA Blood Test in Detecting Colorectal Cancer Recurrence: Results from the FACS Trial. *Public Library of Science One*, *12*(3), pp.1-11.

[193] Beach, T.G. and Adler, C.H., 2018. Importance of Low Diagnostic Accuracy for Early Parkinson's Disease. *Movement Disorders*, *33*(10), pp.1551-1554.

[194] De Bruyn, G. and Graviss, E.A., 2001. A Systematic Review of the Diagnostic Accuracy of Physical Examination for the Detection of Cirrhosis. *BMC Medical Informatics and Decision Making*, *1*(1), pp.1-11.

[195] Palmedo, H., Bucerius, J., Joe, A., et al., 2006. Integrated PET/CT in Differentiated thyroid Cancer: Diagnostic Accuracy and Impact on Patient Management. *Journal of Nuclear Medicine*, *47*(4), pp.616-624.

[196] Nerad, E., Lahaye, M.J., Maas, M., et al., 2016. Diagnostic Accuracy of CT for Local Staging of Colon Cancer: a Systematic Review and Meta-analysis. *American Journal of Roentgenology*, *207*(5), pp.984-995.

[197] Zhang, Y. and Ren, H., 2017. Meta-analysis of Diagnostic Accuracy of Magnetic Resonance Imaging and Mammography for Breast Cancer. *Journal of Cancer Research and Therapeutics*, *13*(5), pp.862-868.

[198] Jaskie, K. and Spanias, A., 2022. Positive Unlabeled Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *16*(1), pp.2-152.

[199] Arjannikov, T. and Tzanetakis, G., 2021. An Empirical Investigation of PU Learning for Predicting Length of Stay. In *IEEE 9th International Conference on Healthcare Informatics,* pp.41-47.

[200] Dey, S., Bose, A., Chakraborty, P., et al., 2021. Impact of Clinical and Genomic Factors on SARS-COV2 Disease Severity. *medRxiv*.

[201] Zhao, X., Tanaka, T., Kong, W., et al., 2018. Epileptic Focus Localization Based on iEEG by Using Positive Unlabeled (PU) Learning. In *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference,* pp. 493-497.

[202] Wilcoxon, F., Katti, S.K. and Wilcox, R.A., 1963. Critical Values and Probability Levels for the Wilcoxon Rank Sum Test and the Wilcoxon Signed Rank Test. *Selected Tables in Mathematical Statistics, 1*, pp.171-259.

[203] Demšar, J., 2006. Statistical Comparisons of Classifiers Over Multiple Data Sets. *The Journal of Machine Learning Research*, *7*, pp.1-30.

[204] De Veaux, R. D., Velleman, P. F., & Bock, D. E., 2021. *Stats: Data and Models (5th ed.).* Pearson.

[205] Schober, P., Boer, C. and Schwarte, L.A., 2018. Correlation Coefficients: Appropriate Use and Interpretation. *Anesthesia & Analgesia*, *126*(5), pp.1763-1768.

[206] Halvari, T., Nurminen, J.K. and Mikkonen, T., 2020. Testing the Robustness of AutoML Systems. *arXiv preprint arXiv:2005.02649*.

[207] Saunders, J.D. and Freitas, A.A., 2024. Automated Machine Learning for Positive-Unlabelled Learning. *arXiv preprint arXiv:2401.06452*.

[208] Li, H., Liu, B., Mukherjee, A. and Shao, J., 2014. Spotting Fake Reviews Using Positive-Unlabeled Learning. *Computación y Sistemas*, *18*(3), pp.467-475.

[209] Yang, P., Li, X., Chua, H.N., et al., 2014. Ensemble Positive Unlabeled Learning for Disease Gene Identification. *Public Library of Science One*, *9*(5), pp.1-11.

[210] Zhang, Z., Ke, T., Deng, N. and Tan, J., 2014. Biased P-norm Support Vector Machine for PU Learning. *Neurocomputing*, *136*, pp.256-261.

[211] Ren, Y., Ji, D. and Zhang, H., 2014. Positive Unlabeled Learning for Deceptive Reviews Detection. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing,* pp.488-498.

[212] Li, T., Lyu, J. and Fan, W., 2019. Semi-supervised Self-training Positive and Unlabeled Learning Based on New Spy Technology. *Journal of Computer Applications*, *39*(10), pp.2822-2828.

[213] Chen, D., Tantai, X., Chang, X., et al., 2022. Weakly Supervised Anomaly Detection Based on Two-Step Cyclic Iterative PU Learning Strategy. *Neural Processing Letters*, *54*(5), pp.4409-4426.

[214] Tamura, S.I. and Tateishi, M., 1997. Capabilities of a Four-Layered Feedforward Neural Network: Four Layers Versus Three. *IEEE Transactions on Neural Networks, 8*(2), pp.251-255.

[215] Sartori, M.A. and Antsaklis, P.J., 1991. A Simple Method to Derive Bounds on the Size and to Train Multilayer Neural Networks. *IEEE Transactions on Neural Networks*, *2*(4), pp.467-471.

[216] Ke, J. and Liu, X., 2008. Empirical Analysis of Optimal Hidden Neurons in Neural Network Modeling for Stock Prediction. In *IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, *2*, pp.828-832.

[217] Trenn, S., 2008. Multilayer Perceptrons: Approximation Order and Necessary Number of Hidden Units. *IEEE Transactions on Neural Networks*, *19*(5), pp.836-844.

[218] Heaton, J., 2008. *Introduction to Neural Networks with Java*. Heaton Research, Inc.

[219] Shibata, K. and Ikeda, Y., 2009. Effect of Number of Hidden Neurons on Learning in Large-scale Layered Neural Networks. In *International Conference on Control, Automation, and Systems-Society of Instrument and Control Engineers*, pp.5008-5013.

Appendix A

Precision & Recall Results

Chapters 4 – 6 of this thesis present tables of results showing the detailed F-measure values and a summary of the precision and recall for each method. This section gives the detailed precision and recall values to complement these results, starting with precision (Tables A.1 through A.6), followed by the recall results (Tables A.7 through A.12).

Table A.1. Precision results of the Auto-PU systems with base search space on real-world biomedical datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**EBO-1**	**BO-1**	**GA-1**	**EBO-1**	**BO-1**	**GA-1**	**EBO-1**	**BO-1**	**GA-1**
Alzheimer's	0.688	0.600	0.600	0.595	0.656	0.613	0.680	0.706	0.684
Autism	0.986	0.978	0.971	0.954	0.985	0.941	0.941	0.948	0.946
Breast cancer Coi.	0.990	0.625	0.653	0.962	0.658	0.672	0.800	0.829	0.684
Breast cancer Wis.	0.815	0.985	0.975	0.833	0.990	0.930	0.960	0.983	0.953
Breast cancer mut.	0.608	0.819	0.813	0.714	0.824	0.813	0.851	0.826	0.830
Cervical cancer	0.929	0.929	1.000	1.000	1.000	1.000	0.688	0.714	0.909
Cirrhosis	0.451	0.479	0.490	0.409	0.447	0.402	0.522	0.516	0.423
Dermatology	0.976	0.891	0.889	0.860	0.915	0.750	0.879	0.906	0.923
PI Diabetes	0.586	0.580	0.594	0.559	0.557	0.541	0.625	0.629	0.583
ES Diabetes	0.978	0.984	0.962	0.975	0.981	0.978	0.962	0.964	0.969
Heart Disease	0.792	0.782	0.796	0.800	0.830	0.822	0.828	0.805	0.808
Heart Failure	0.770	0.778	0.740	0.701	0.629	0.670	0.724	0.759	0.763
Hepatitis C	0.980	0.942	1.000	0.915	0.898	0.925	0.714	0.850	0.862
Kidney Disease	1.000	1.000	1.000	1.000	1.000	1.000	0.955	1.000	1.000
Liver Disease	0.720	0.718	0.715	0.726	0.718	0.727	0.751	0.742	0.735
Maternal Risk	0.840	0.852	0.312	0.820	0.821	0.870	0.743	0.818	0.816
Parkinsons	0.920	0.899	0.754	0.848	0.872	0.857	0.965	0.909	0.923
Parkinsons Biom.	0.207	0.167	0.313	0.246	0.227	0.237	0.200	0.200	0.233
Spine	0.939	0.947	0.484	0.951	0.938	0.942	0.951	0.939	0.924
Stroke	0.167	0.173	0.310	0.189	0.156	0.200	0.192	0.150	0.185

Table A.2. Precision results of the Auto-PU systems with base search space on synthetic datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**EBO-1**	**BO-1**	**GA-1**	**EBO-1**	**BO-1**	**GA-1**	**EBO-1**	**BO-1**	**GA-1**
1	0.648	0.633	0.641	0.729	0.711	0.631	0.63	0.714	0.433
2	0.111	0.071	0.075	0.069	0.083	0.024	0.056	0.086	0.036
3	0.731	0.793	0.739	0.709	0.698	0.614	0.618	0.680	0.612
4	0.857	0.824	0.871	0.842	0.831	0.705	0.734	0.839	0.631
5	0.373	0.595	0.581	0.476	0.583	0.519	0.332	0.433	0.586
6	0.697	0.845	0.734	0.845	0.841	0.738	0.746	0.869	0.923
7	0.426	0.653	0.381	0.516	0.448	0.375	0.464	0.449	0.650
8	0.347	0.724	0.361	0.450	0.537	0.374	0.368	0.395	0.403
9	0.013	0.024	0.067	0.060	0.100	0.045	0.109	0.118	0.080
10	0.962	0.981	0.846	0.919	0.931	0.973	0.877	0.908	0.933
11	0.511	0.497	0.479	0.549	0.548	0.553	0.512	0.586	0.448
12	0.697	0.578	0.616	0.671	0.680	0.624	0.610	0.795	0.914
13	0.561	0.572	0.548	0.652	0.643	0.614	0.582	0.532	0.359
14	0.962	0.958	1.000	0.899	0.985	0.927	0.804	0.958	0.906
15	0.749	0.552	0.767	0.535	0.564	0.440	0.441	0.514	0.368
16	0.382	0.792	0.332	0.474	0.347	0.244	0.395	0.510	0.178
17	0.240	0.369	0.224	0.240	0.244	0.980	0.104	0.295	0.270
18	0.375	0.361	0.393	0.526	0.495	0.429	0.463	0.308	0.227
19	0.375	0.324	0.353	0.255	0.333	0.330	0.219	0.357	0.243
20	0.520	0.642	0.554	0.562	0.593	0.656	0.468	0.570	0.510

Table A.3. Precision results of the Auto-PU systems with extended search space on real-world biomedical datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**EBO-2**	**BO-2**	**GA-2**	**EBO-2**	**BO-2**	**GA-2**	**EBO-2**	**BO-2**	**GA-2**
Alzheimer's	0.633	0.645	0.571	0.690	0.629	0.679	0.750	0.593	0.600
Autism	0.978	0.985	0.986	0.949	0.962	0.930	0.874	0.953	0.941
Breast cancer Coi.	0.981	0.678	0.624	0.966	0.644	0.657	0.937	0.000	0.867
Breast cancer Wis.	0.840	0.923	0.990	0.824	0.961	0.938	0.799	0.957	0.955
Breast cancer mut.	0.590	0.845	0.818	0.692	0.822	0.619	0.597	0.836	0.867
Cervical cancer	1.000	1.000	1.000	1.000	1.000	0.929	0.356	0.929	0.304
Cirrhosis	0.421	0.415	0.456	0.468	0.440	0.375	0.319	0.536	0.370
Dermatology	0.907	0.951	0.953	0.932	0.925	0.896	0.825	0.943	0.900
PI Diabetes	0.597	0.586	0.614	0.629	0.546	0.585	0.564	0.586	0.641
ES Diabetes	0.968	0.965	0.990	0.917	0.978	0.985	0.924	0.937	0.967
Heart Disease	0.793	0.807	0.783	0.746	0.822	0.824	0.710	0.810	0.804
Heart Failure	0.753	0.787	0.763	0.678	0.694	0.697	0.570	0.689	0.831
Hepatitis C	0.942	0.981	0.981	0.942	0.972	0.957	0.912	0.897	0.962
Kidney Disease	0.872	1.000	1.000	0.886	1.000	1.000	1.000	1.000	1.000
Liver Disease	0.726	0.716	0.721	0.701	0.730	0.728	0.679	0.790	0.757
Maternal Risk	0.830	0.830	0.853	0.777	0.853	0.864	0.700	0.802	0.811
Parkinsons	0.890	0.927	0.899	0.831	0.874	0.848	0.706	0.920	0.935
Parkinsons Biom.	0.292	0.273	0.244	0.286	0.000	0.200	0.207	0.000	0.350
Spine	0.972	0.978	0.926	0.920	0.946	0.957	0.747	0.954	0.924
Stroke	0.161	0.178	0.166	0.156	0.191	0.166	0.139	0.171	0.185

Table A.4. Precision results of the Auto-PU systems with extended search space on synthetic datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**EBO-2**	**BO-2**	**GA-2**	**EBO-2**	**BO-2**	**GA-2**	**EBO-2**	**BO-2**	**GA-2**
1	0.710	0.685	0.604	0.700	0.738	0.620	0.621	0.704	0.446
2	0.189	0.187	0.102	0.045	0.036	0.057	0.118	0.182	0.060
3	0.777	0.780	0.730	0.719	0.695	0.840	0.592	0.701	0.457
4	0.812	0.883	0.743	0.796	0.875	0.772	0.724	0.782	0.767
5	0.468	0.625	0.471	0.401	0.612	0.708	0.423	0.524	0.777
6	0.767	0.830	0.703	0.695	0.763	0.655	0.663	0.860	0.558
7	0.556	0.467	0.397	0.514	0.471	0.548	0.480	0.396	0.300
8	0.588	0.562	0.651	0.405	0.442	0.347	0.401	0.407	0.272
9	0.051	0.011	0.052	0.029	0.045	0.347	0.079	0.107	0.085
10	0.989	0.940	0.851	0.988	0.892	0.740	0.902	0.877	0.712
11	0.601	0.561	0.432	0.545	0.579	0.470	0.470	0.505	0.360
12	0.804	0.772	0.526	0.713	0.738	0.785	0.647	0.778	0.481
13	0.615	0.650	0.719	0.589	0.591	0.468	0.492	0.577	0.475
14	0.906	0.975	0.978	0.853	0.948	1.000	0.706	0.972	0.938
15	0.700	0.672	0.554	0.621	0.531	0.921	0.568	0.453	0.431
16	0.718	0.675	0.275	0.620	0.388	0.324	0.514	0.470	0.178
17	0.352	0.317	0.268	0.233	0.228	0.311	0.243	0.143	0.300
18	0.593	0.550	0.532	0.533	0.54	0.261	0.404	0.306	0.480
19	0.383	0.402	0.324	0.316	0.31	0.426	0.253	0.410	0.792
20	0.615	0.686	0.610	0.590	0.579	0.555	0.513	0.579	0.721

Table A.5. Precision results of the baseline methods on real-world biomedical datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**DF-PU**	**S-EM**	**TPOT**	**DF-PU**	**S-EM**	**TPOT**	**DF-PU**	**S-EM**	**TPOT**
Alzheimer's	0.108	0.195	0.654	0.108	0.237	0.647	0.096	0.244	0.219
Autism	0.481	0.858	0.971	0.481	0.808	0.985	0.479	0.815	0.930
Breast cancer Coi.	0.544	0.552	0.611	0.552	0.548	0.539	0.544	0.569	0.615
Breast cancer Wis.	0.373	0.904	0.990	0.373	0.905	0.869	0.370	0.922	0.941
Breast cancer mut.	0.324	0.812	0.822	0.324	0.811	0.809	0.322	0.811	0.829
Cervical cancer	0.032	0.028	0.846	0.021	0.027	0.000	0.023	0.024	0.000
Cirrhosis	0.255	0.286	0.458	0.253	0.291	0.500	0.255	0.303	0.283
Dermatology	0.130	0.566	0.892	0.130	0.566	0.795	0.125	0.575	0.638
PI Diabetes	0.348	0.364	0.628	0.348	0.356	0.540	0.347	0.398	0.582
ES Diabetes	0.617	0.658	0.974	0.612	0.770	0.733	0.614	0.658	0.962
Heart Disease	0.545	0.732	0.806	0.545	0.752	0.831	0.543	0.794	0.787
Heart Failure	0.322	0.367	0.653	0.321	0.344	0.712	0.323	0.424	0.527
Hepatitis C	0.096	0.661	0.938	0.094	0.702	0.843	0.087	0.593	0.704
Kidney Disease	0.272	1.000	1.000	0.272	1.000	0.607	0.272	1.000	0.957
Liver Disease	0.715	0.729	0.726	0.715	0.695	0.574	0.715	0.708	0.735
Maternal Risk	0.257	0.293	0.895	0.252	0.277	0.808	0.248	0.280	0.804
Parkinsons	0.753	0.875	0.842	0.754	0.929	0.717	0.754	0.914	0.921
Parkinsons Biom.	0.219	0.250	0.241	0.219	0.194	0.227	0.227	0.209	0.125
Spine	0.484	0.694	0.966	0.484	0.758	0.992	0.484	0.731	0.910
Stroke	0.045	0.054	0.163	0.050	0.054	0.778	0.050	0.054	0.181

Table A.6. Precision results of the baseline methods on synthetic datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**DF-PU**	**S-EM**	**TPOT**	**DF-PU**	**S-EM**	**TPOT**	**DF-PU**	**S-EM**	**TPOT**
1	0.320	0.512	0.574	0.320	0.518	0.426	0.319	0.601	0.374
2	0.068	0.121	0.700	0.065	0.095	0.318	0.060	0.162	0.080
3	0.382	0.421	0.988	0.382	0.425	0.779	0.382	0.451	0.766
4	0.294	0.511	0.670	0.263	0.503	0.485	0.264	0.538	0.381
5	0.218	0.253	0.797	0.218	0.287	0.612	0.219	0.320	0.508
6	0.255	0.317	0.847	0.263	0.367	0.742	0.263	0.473	0.669
7	0.167	0.301	0.438	0.167	0.347	0.253	0.166	0.351	0.249
8	0.198	0.337	0.855	0.246	0.338	0.731	0.246	0.375	0.580
9	0.018	0.069	0.300	0.000	0.032	0.000	0.000	0.143	0.000
10	0.133	0.450	0.953	0.134	0.466	0.730	0.134	0.509	0.727
11	0.334	0.339	0.782	0.329	0.359	0.661	0.329	0.363	0.642
12	0.248	0.418	0.738	0.248	0.444	0.674	0.246	0.488	0.657
13	0.334	0.436	0.443	0.299	0.449	0.419	0.297	0.452	0.376
14	0.363	0.700	0.931	0.360	0.747	0.848	0.360	0.811	0.711
15	0.240	0.278	1.000	0.241	0.283	0.818	0.239	0.285	0.800
16	0.193	0.318	0.993	0.144	0.351	0.664	0.145	0.439	0.274
17	0.122	0.155	0.444	0.120	0.175	0.600	0.120	0.231	0.462
18	0.230	0.314	0.243	0.280	0.324	0.179	0.280	0.327	0.159
19	0.233	0.299	0.482	0.259	0.310	0.329	0.257	0.336	0.273
20	0.439	0.453	0.456	0.439	0.460	0.391	0.439	0.464	0.356

Table A.7. Recall results of the Auto-PU systems with base search space on real-world biomedical datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**EBO-1**	**BO-1**	**GA-1**	**EBO-1**	**BO-1**	**GA-1**	**EBO-1**	**BO-1**	**GA-1**
Alzheimer's	0.579	0.632	0.474	0.579	0.553	0.500	0.447	0.316	0.342
Autism	0.986	0.957	0.950	0.899	0.928	0.914	0.914	0.791	0.878
Breast cancer Coi.	0.943	0.781	0.766	0.943	0.750	0.703	0.500	0.453	0.406
Breast cancer Wis.	0.987	0.915	0.934	0.915	0.948	0.934	0.896	0.821	0.863
Breast cancer mut.	0.750	0.983	0.991	0.625	0.928	0.930	0.874	0.856	0.880
Cervical cancer	0.765	0.765	0.706	0.824	0.824	0.824	0.647	0.588	0.588
Cirrhosis	0.648	0.634	0.690	0.507	0.648	0.549	0.493	0.465	0.465
Dermatology	0.833	0.854	0.833	0.771	0.896	0.813	0.604	0.604	0.750
PI Diabetes	0.739	0.731	0.787	0.810	0.765	0.810	0.642	0.563	0.631
ES Diabetes	0.969	0.981	0.953	0.859	0.794	0.825	0.863	0.847	0.894
Heart Disease	0.879	0.915	0.897	0.800	0.830	0.782	0.727	0.752	0.764
Heart Failure	0.698	0.729	0.802	0.635	0.583	0.635	0.573	0.656	0.604
Hepatitis C	0.875	0.875	0.911	0.768	0.786	0.661	0.625	0.607	0.446
Kidney Disease	1.000	0.977	0.953	0.884	0.930	0.977	0.488	0.674	0.605
Liver Disease	0.971	0.954	1.000	0.940	0.947	0.896	0.686	0.855	0.889
Maternal Risk	0.871	0.824	1.000	0.787	0.743	0.761	0.735	0.596	0.669
Parkinsons	0.939	0.973	1.000	0.946	0.878	0.816	0.558	0.612	0.735
Parkinsons Biom.	0.200	0.167	1.000	0.533	0.167	0.300	0.100	0.167	0.233
Spine	0.927	0.960	1.000	0.913	0.913	0.873	0.653	0.613	0.733
Stroke	0.422	0.411	1.000	0.278	0.150	0.350	0.283	0.339	0.411

Table A.8. Recall results of the Auto-PU systems with base search space on synthetic datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**EBO-1**	**BO-1**	**GA-1**	**EBO-1**	**BO-1**	**GA-1**	**EBO-1**	**BO-1**	**GA-1**
1	0.712	0.695	0.683	0.576	0.547	0.831	0.539	0.453	0.996
2	0.794	0.635	0.746	0.032	0.032	0.349	0.032	0.048	0.444
3	0.814	0.731	0.844	0.663	0.646	0.793	0.572	0.468	0.664
4	0.766	0.798	0.793	0.753	0.745	0.973	0.625	0.500	0.723
5	0.532	0.719	0.66	0.376	0.432	0.754	0.261	0.632	0.634
6	0.731	0.696	0.787	0.565	0.565	0.803	0.509	0.459	0.544
7	0.836	0.552	0.816	0.577	0.512	0.821	0.512	0.483	0.378
8	0.947	0.396	0.956	0.360	0.422	0.658	0.302	0.387	0.360
9	0.103	0.207	0.345	0.103	0.034	0.414	0.172	0.069	0.862
10	0.984	0.863	0.967	1.000	0.813	0.791	0.940	0.648	0.615
11	0.838	0.662	0.816	0.606	0.604	0.582	0.560	0.486	0.650
12	0.817	0.801	0.742	0.675	0.685	0.715	0.605	0.449	0.457
13	0.816	0.787	0.782	0.574	0.569	0.632	0.502	0.493	0.924
14	0.862	0.998	0.951	0.857	0.953	1.000	0.776	0.848	0.946
15	0.474	0.746	0.495	0.584	0.613	0.907	0.486	0.474	0.884
16	0.953	0.297	0.844	0.430	0.320	0.938	0.352	0.195	0.992
17	0.828	0.411	0.768	0.391	0.397	0.331	0.172	0.172	0.868
18	0.915	0.822	0.966	0.424	0.390	0.356	0.373	0.203	0.576
19	0.752	0.611	0.711	0.443	0.591	0.805	0.362	0.470	0.872
20	0.909	0.759	0.972	0.726	0.769	0.731	0.617	0.685	0.805

Table A.9. Recall results of the Auto-PU systems with extended search space on real-world biomedical datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**EBO-2**	**BO-2**	**GA-2**	**EBO-2**	**BO-2**	**GA-2**	**EBO-2**	**BO-2**	**GA-2**
Alzheimer's	0.500	0.526	0.526	0.526	0.579	0.500	0.474	0.421	0.474
Autism	0.950	0.942	0.978	0.928	0.914	0.950	0.899	0.878	0.914
Breast cancer Coi.	0.953	0.656	0.828	0.939	0.594	0.688	0.910	0.000	0.406
Breast cancer Wis.	0.928	0.998	0.925	0.906	0.925	0.934	0.882	0.830	0.792
Breast cancer mut.	0.766	0.938	0.989	0.563	0.887	0.917	0.578	0.854	0.878
Cervical cancer	0.765	0.765	0.765	0.824	0.765	0.765	0.941	0.765	0.412
Cirrhosis	0.634	0.620	0.437	0.521	0.620	0.423	0.324	0.423	0.141
Dermatology	0.813	0.813	0.854	0.854	0.771	0.896	0.688	0.688	0.563
PI Diabetes	0.757	0.739	0.672	0.709	0.799	0.720	0.657	0.646	0.627
ES Diabetes	0.947	0.944	0.966	0.894	0.819	0.806	0.906	0.888	0.831
Heart Disease	0.861	0.885	0.897	0.873	0.812	0.739	0.788	0.800	0.770
Heart Failure	0.729	0.729	0.740	0.635	0.615	0.646	0.469	0.531	0.563
Hepatitis C	0.875	0.946	0.911	0.875	0.625	0.786	0.554	0.464	0.446
Kidney Disease	0.953	0.953	0.860	0.907	0.953	0.907	0.488	0.651	0.674
Liver Disease	0.973	0.964	0.981	0.932	0.923	0.930	0.833	0.664	0.739
Maternal Risk	0.879	0.864	0.871	0.846	0.728	0.768	0.765	0.669	0.676
Parkinsons	0.939	0.946	0.973	0.871	0.803	0.837	0.735	0.707	0.687
Parkinsons Biom.	0.233	0.300	0.333	0.267	0.000	0.367	0.200	0.000	0.233
Spine	0.913	0.907	0.920	0.920	0.927	0.880	0.867	0.553	0.647
Stroke	0.417	0.456	0.439	0.406	0.383	0.439	0.361	0.367	0.372

Table A.10. Recall results of the Auto-PU systems with extended search space on synthetic datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**EBO-2**	**BO-2**	**GA-2**	**EBO-2**	**BO-2**	**GA-2**	**EBO-2**	**BO-2**	**GA-2**
1	0.634	0.601	0.679	0.613	0.486	0.827	0.527	0.490	0.700
2	0.222	0.222	0.619	0.016	0.016	0.603	0.032	0.063	0.825
3	0.755	0.759	0.789	0.696	0.657	0.602	0.572	0.435	0.928
4	0.806	0.785	0.923	0.790	0.729	0.848	0.697	0.516	0.630
5	0.437	0.583	0.875	0.379	0.616	0.478	0.399	0.476	0.437
6	0.659	0.677	0.832	0.613	0.600	0.880	0.587	0.525	0.843
7	0.617	0.527	0.791	0.562	0.443	0.453	0.542	0.438	0.891
8	0.533	0.502	0.507	0.369	0.524	0.800	0.369	0.293	0.684
9	0.103	0.069	0.759	0.069	0.034	0.800	0.207	0.103	0.448
10	0.962	0.863	0.945	0.934	0.819	1.000	0.857	0.665	0.720
11	0.645	0.585	0.857	0.582	0.531	0.751	0.529	0.447	0.971
12	0.675	0.664	0.968	0.626	0.575	0.618	0.575	0.470	0.796
13	0.642	0.686	0.591	0.623	0.566	0.882	0.529	0.495	0.730
14	0.904	0.973	0.975	0.855	0.948	0.934	0.698	0.853	0.929
15	0.529	0.524	0.643	0.471	0.560	0.419	0.425	0.383	0.822
16	0.438	0.406	1.000	0.344	0.242	0.531	0.281	0.242	0.914
17	0.298	0.265	0.675	0.185	0.457	0.954	0.185	0.046	0.523
18	0.458	0.508	0.627	0.415	0.398	0.941	0.390	0.288	0.305
19	0.483	0.497	0.812	0.396	0.584	0.557	0.302	0.322	0.255
20	0.690	0.736	0.822	0.655	0.695	0.825	0.569	0.530	0.505

Table A.11. Recall results of the baseline methods on real-world biomedical datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**DF-PU**	**S-EM**	**TPOT**	**DF-PU**	**S-EM**	**TPOT**	**DF-PU**	**S-EM**	**TPOT**
Alzheimer's	1.000	0.895	0.447	0.974	0.842	0.289	0.763	0.789	0.553
Autism	0.993	0.784	0.957	0.993	0.878	0.928	0.986	0.856	0.863
Breast cancer Coi.	0.969	1.000	0.516	1.000	0.984	0.641	0.969	0.906	0.375
Breast cancer Wis.	1.000	0.892	0.906	1.000	0.901	0.967	0.991	0.887	0.524
Breast cancer mut.	0.998	0.989	0.974	0.998	0.993	0.989	0.987	0.991	0.837
Cervical cancer	0.882	0.882	0.647	0.824	0.941	0.000	0.706	0.588	0.000
Cirrhosis	0.986	0.915	0.535	0.972	0.915	0.437	0.986	0.944	0.479
Dermatology	0.958	0.979	0.688	0.938	0.979	0.729	0.896	0.958	0.771
PI Diabetes	0.996	1.000	0.698	0.996	1.000	0.556	0.993	0.862	0.567
ES Diabetes	0.994	0.994	0.928	0.988	0.972	0.438	0.994	0.997	0.716
Heart Disease	1.000	0.909	0.830	1.000	0.921	0.776	0.994	0.867	0.782
Heart Failure	1.000	0.948	0.667	1.000	0.969	0.542	0.948	0.813	0.604
Hepatitis C	0.982	0.732	0.804	0.982	0.714	0.768	0.911	0.625	0.339
Kidney Disease	1.000	1.000	0.977	1.000	1.000	0.791	1.000	0.907	0.512
Liver Disease	1.000	0.928	0.725	0.995	0.507	0.365	1.000	0.889	0.548
Maternal Risk	0.941	1.000	0.787	0.915	1.000	0.728	0.901	1.000	0.544
Parkinsons	0.993	0.762	0.980	1.000	0.626	0.619	1.000	0.653	0.476
Parkinsons Biom.	0.933	0.500	0.233	0.933	0.400	0.167	0.967	0.800	0.100
Spine	1.000	1.000	0.960	1.000	0.940	0.787	1.000	0.960	0.607
Stroke	0.811	1.000	0.328	0.811	1.000	0.117	0.756	1.000	0.150

Table A.12. Recall results of the baseline methods on synthetic datasets.

	δ = 20%			δ = 40%			δ = 60%		
Dataset	**DF-PU**	**S-EM**	**TPOT**	**DF-PU**	**S-EM**	**TPOT**	**DF-PU**	**S-EM**	**TPOT**
1	0.996	0.774	0.483	0.996	0.720	0.393	0.996	0.626	0.363
2	0.667	0.492	0.034	0.794	0.206	0.017	0.794	0.095	0.010
3	0.994	0.978	0.617	0.996	0.953	0.469	0.996	0.895	0.461
4	0.987	0.872	0.936	0.989	0.854	0.696	0.992	0.761	0.547
5	0.992	0.969	0.436	0.964	0.913	0.351	0.967	0.854	0.284
6	0.955	0.960	0.765	0.859	0.925	0.629	0.859	0.755	0.576
7	0.990	0.766	0.692	0.940	0.692	0.418	0.940	0.632	0.407
8	0.938	0.764	0.292	0.484	0.707	0.391	0.484	0.529	0.210
9	0.586	0.172	0.020	0.000	0.069	0.000	0.000	0.103	0.000
10	0.967	0.956	0.755	0.918	0.956	0.574	0.918	0.951	0.566
11	0.928	0.983	0.395	0.966	0.942	0.342	0.966	0.899	0.331
12	0.987	0.804	0.666	0.997	0.782	0.613	0.992	0.734	0.598
13	1.000	0.748	0.911	1.000	0.730	0.868	0.990	0.699	0.782
14	0.975	0.980	0.931	0.998	0.958	0.966	0.998	0.946	0.931
15	0.996	0.877	0.259	0.977	0.858	0.206	0.977	0.812	0.203
16	0.313	0.594	0.339	0.703	0.469	0.243	0.711	0.227	0.098
17	0.841	0.828	0.190	0.993	0.709	0.143	0.993	0.318	0.190
18	0.975	0.763	0.242	0.559	0.653	0.195	0.559	0.593	0.168
19	0.993	0.738	0.360	0.698	0.698	0.242	0.698	0.537	0.198
20	1.000	0.957	0.932	1.000	0.949	0.932	1.000	0.904	0.856

As discussed in the main text of the thesis, regarding precision and recall, the DF-PU method generally achieved the best recall overall, however this was due to a large overpredicition of the positive class and came at a great detriment to precision, thus achieving a low F-measure overall. S-EM also achieved good recall, often outperforming the Auto-PU systems, but this, again, came at a high cost to precision in several cases. The results for TPOT were not comparable to GA-1, being largely outperformed by GA-Auto-PU with statistical significance in most cases for both precision and recall.

The Auto-PU systems generally performed best in regard to precision, often outperforming the baseline methods with statistical significance. Statistical significance was rarely observed between the different versions of the Auto-PU systems themselves.

A detailed analysis of the precision and recall results, including details of statistical significance testing and the best method for each comparison, can be found in the main text of this thesis in Chapters 4-6.

Appendix B

Auto-PU-NAS (Neural Architecture Search)

An additional approach to those presented in this thesis that was investigated was an Auto-PU system that performed a neural architecture search for two-step PU learning algorithms. That is, the classifier parameters of the Auto-PU system were replaced with multilayer perceptrons (MLPs), with the hyperparameters of the MLP included in the search space.

More precisely, recall that in the proposed Auto-PU systems, a candidate solution includes (among other PU learning algorithm hyperparameters) three "classifier" hyperparameters, namely *Classifier_1A*, *Classifier_1B* and *Classifier_2*; and each of these three hyperparameters takes as a value the name of one out of a list of 18 predefined candidate classification algorithms. In the Auto-PU-NAS system described in this Appendix, each of those 3 classifier hyperparameters was replaced by a list of 12 hyperparameters of a multilayer perceptron (MLP) algorithm, where each of those 12 hyperparameters takes a value among a predefined list of candidate values (shown next). All the other hyperparameters (i.e., other than the "classifier" hyperparameters) in the solution encoding used by GA-Auto-PU, BO-Auto-PU and EBO-Auto-PU were kept without modification in Auto-PU-NAS. Hence, the total number of PU learning algorithm hyperparameters being optimised by the Auto-PU-NAS, for the base search space, is: $4 + (12 \times 3) = 40$, i.e., the four "non-classifier" hyperparameters of GA-Auto-PU, BO-Auto-PU and EBO-Auto-PU plus 3 times the 12 hyperparameters of the MLP algorithm.

The parameters of the MLP that were tuned were as follows:

- Hidden layer count: { 1, 2, 3, 4, 5 }
- Hidden layer formula: { 5 candidate formulas (see below) }
- Activation function: { identity, logistic, tanh, ReLU }
- L2 regularisation: { True, False }
- Alpha: { 0.00001, 0.0001, 0.001, 0.01, 0.1 }
- Learning rate init: { 0.00001, 0.0001, 0.001, 0.01, 0.1 }
- Early stopping: { True, False }
- Number of iterations no change (early stopping tolerance): { 10, 15, 20, 25, 30 }
- Batch size: { 32, 64, 128, 256 }
- Epochs : { 50, 250, 500, 750, 1000 }
- Tolerance: { 0.0001, 0.0005, 0.001 }
- Validation fraction: { 0.1, 0.15, 0.2 }

These hyperparmeters (with the exception of hidden layer formula) correlate to those of the same name given in the Sklearn documentation [31] for the MLP classifier[7]. The hidden layer formula is used to determine the number of neurons in each hidden layer. As this is highly dependent on the number of input features, using a formula seems a better approach than simply setting a fixed value. The candidate formulas are described next.

Terms

$\#N_h$: The number of hidden neurons for a given layer h.

m: The number of attributes of the data

n: The number of inputs for the layer.

l_i: The value of i for the ith hidden layer. E.g., 2 for the second hidden layer.

o: The number of outputs (1 for binary classification).

[7] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier

Formula 1 for number of neurons in a hidden layer (Equation B.1)

$$\#N_h = \left(\frac{m}{2}\right) + 3 \qquad \text{(B. 1)}$$

This formula was developed by Tamura & Tateishi [214] for 4-layer networks to reduce number of parameters to learn in comparison to a formula previously given by Sartori & Antsaklis [215] when n is large.

Formula 2 for number of neurons in a hidden layer (Equation B.2)

$$\#N_h = \frac{m + \sqrt{n}}{l_i} \qquad \text{(B. 2)}$$

This formula was determined by manually tuning networks on several use cases and generalising for the best performing networks [216].

Formula 3 for number of neurons in a hidden layer (Equation B.3)

$$\#N_h = \frac{n + o - 1}{2} \qquad \text{(B. 3)}$$

For this work, this simplifies just to $\frac{n}{2}$. So, each layer has half the neurons of the previous layer.

Formula 4 for number of neurons in a hidden layer (Equation B.4)

$$\#N_h = n \times \left(\frac{2}{3}\right) + o \qquad \text{(B. 4)}$$

Formula 5 for number of neurons in a hidden layer (Equation B.5)

$$\#N_h = \sqrt{n \times o} \qquad \text{(B. 5)}$$

For the applications in this work, this simplifies just to $\sqrt{m_i}$.

The size of the search space for just the MLP hyperparameters is thus:

$$5 \times 5 \times 4 \times 2 \times 5 \times 5 \times 2 \times 5 \times 4 \times 5 \times 3 \times 3 = 9{,}000{,}000$$

Evolutionary Bayesian optimisation was used as the optimiser for the Auto-PU-NAS system, as it was arguably the best performing of the three approaches used throughout this thesis. Due to the computational expense of the MLP classifier, experiments were conducted utilising the base search space only. With the extra hyperparameters included, the size of the search space is thus:

$$10 \times 10 \times 9{,}000{,}000 \times 10 \times 9{,}000{,}000 \times 2 \times 9{,}000{,}000$$
$$= 1{,}458{,}000{,}000{,}000{,}000{,}000{,}000{,}000 \ (1.458\text{E}24)$$

which is 1.25E17 times larger than the original search space, where the value of 9,000,000 is the size of the search space of just the MLP hyperparameters.

The results are given in Tables B.1 to B.6 for both the biomedical and synthetic datasets. The full comparison with all three Auto-PU systems makes the table too large to display here, so only the results for EBO-1 have been included for reference.

Table B.1. F-measure results of Auto-PU-NAS compared with EBO-1 on real-world biomedical datasets.

	δ = 20%		**δ = 40%**		**δ = 60%**	
Dataset	**NAS**	**EBO-1**	**NAS**	**EBO-1**	**NAS**	**EBO-1**
Alzheimer's	0.120	0.629	0.093	0.587	0.080	0.540
Autism	0.343	0.986	0.311	0.926	0.281	0.927
Breast cancer Coi.	0.436	0.966	0.404	0.952	0.338	0.615
Breast cancer Wis.	0.722	0.893	0.698	0.872	0.668	0.927
Breast cancer mut.	0.385	0.672	0.269	0.667	0.238	0.862
Cervical cancer	0.000	0.839	0.000	0.904	0.000	0.667
Cirrhosis	0.282	0.532	0.190	0.453	0.121	0.507
Dermatology	0.179	0.899	0.073	0.813	0.000	0.716
PI Diabetes	0.129	0.654	0.110	0.661	0.091	0.634
ES Diabetes	0.751	0.973	0.663	0.913	0.611	0.909
Heart Disease	0.355	0.833	0.297	0.800	0.188	0.774
Heart Failure	0.306	0.732	0.246	0.666	0.211	0.640
Hepatitis C	0.487	0.925	0.462	0.835	0.456	0.667
Kidney Disease	0.400	1.000	0.364	0.938	0.295	0.646
Liver Disease	0.394	0.827	0.344	0.819	0.268	0.717
Maternal Risk	0.453	0.855	0.405	0.803	0.380	0.739
Parkinsons	0.532	0.929	0.406	0.894	0.392	0.707
Parkinsons Biom.	0.123	0.203	0.091	0.337	0.000	0.133
Spine	0.611	0.933	0.480	0.932	0.437	0.775
Stroke	0.020	0.239	0.010	0.225	0.010	0.229

Table B.2. F-measure results of Auto-PU-NAS compared with EBO-1 on synthetic datasets.

	δ = 20%		δ = 40%		δ = 60%	
Dataset	**NAS**	**EBO-1**	**NAS**	**EBO-1**	**NAS**	**EBO-1**
1	0.434	0.678	0.402	0.644	0.370	0.581
2	0.084	0.194	0.063	0.043	0.034	0.040
3	0.565	0.770	0.549	0.685	0.532	0.594
4	0.565	0.809	0.536	0.795	0.527	0.675
5	0.234	0.439	0.216	0.420	0.191	0.292
6	0.369	0.714	0.339	0.677	0.312	0.605
7	0.338	0.565	0.297	0.545	0.242	0.487
8	0.541	0.508	0.535	0.400	0.483	0.332
9	0.031	0.023	0.000	0.076	0.000	0.133
10	0.649	0.973	0.620	0.959	0.587	0.907
11	0.453	0.635	0.437	0.576	0.412	0.535
12	0.577	0.752	0.558	0.673	0.507	0.607
13	0.447	0.665	0.416	0.610	0.392	0.539
14	0.624	0.909	0.576	0.878	0.555	0.790
15	0.367	0.580	0.359	0.558	0.340	0.462
16	0.271	0.546	0.222	0.451	0.187	0.372
17	0.290	0.372	0.278	0.297	0.258	0.129
18	0.459	0.532	0.371	0.469	0.330	0.413
19	0.399	0.500	0.355	0.324	0.324	0.273
20	0.521	0.661	0.466	0.633	0.425	0.532

Table B.3. Precision results of Auto-PU-NAS compared with EBO-1 on real-world biomedical datasets.

	δ = 20%		δ = 40%		δ = 60%	
Dataset	**NAS**	**EBO-1**	**NAS**	**EBO-1**	**NAS**	**EBO-1**
Alzheimer's	0.080	0.688	0.063	0.595	0.054	0.680
Autism	0.410	0.986	0.362	0.954	0.330	0.941
Breast cancer Coi.	0.333	0.990	0.308	0.962	0.262	0.800
Breast cancer Wis.	0.775	0.815	0.764	0.833	0.755	0.960
Breast cancer mut.	0.467	0.608	0.350	0.714	0.324	0.851
Cervical cancer	0.000	0.929	0.000	1.000	0.000	0.688
Cirrhosis	0.282	0.451	0.184	0.409	0.115	0.522
Dermatology	0.625	0.976	0.286	0.860	0.000	0.879
PI Diabetes	0.488	0.586	0.415	0.559	0.350	0.625
ES Diabetes	0.856	0.978	0.834	0.975	0.802	0.962
Heart Disease	0.621	0.792	0.579	0.800	0.417	0.828
Heart Failure	0.679	0.770	0.577	0.701	0.481	0.724
Hepatitis C	0.864	0.980	0.818	0.915	0.783	0.714
Kidney Disease	0.519	1.000	0.522	1.000	0.500	0.955
Liver Disease	0.676	0.720	0.628	0.726	0.517	0.751
Maternal Risk	0.580	0.840	0.551	0.820	0.503	0.743
Parkinsons	0.817	0.920	0.745	0.848	0.750	0.965
Parkinsons Biom.	0.114	0.207	0.083	0.246	0.000	0.200
Spine	0.615	0.939	0.528	0.951	0.492	0.951
Stroke	0.118	0.167	0.056	0.189	0.050	0.192

Table B.4. Precision results of Auto-PU-NAS compared with EBO-1 on synthetic datasets.

	δ = 20%		δ = 40%		δ = 60%	
Dataset	**NAS**	**EBO-1**	**NAS**	**EBO-1**	**NAS**	**EBO-1**
1	0.551	0.648	0.516	0.729	0.493	0.630
2	0.089	0.111	0.063	0.069	0.038	0.056
3	0.677	0.731	0.661	0.709	0.642	0.618
4	0.645	0.857	0.632	0.842	0.621	0.734
5	0.269	0.373	0.248	0.476	0.225	0.332
6	0.550	0.697	0.551	0.845	0.515	0.746
7	0.399	0.426	0.359	0.516	0.288	0.464
8	0.600	0.347	0.603	0.450	0.523	0.368
9	0.020	0.013	0.000	0.060	0.000	0.109
10	0.590	0.962	0.551	0.919	0.529	0.877
11	0.496	0.511	0.495	0.549	0.461	0.512
12	0.692	0.697	0.663	0.671	0.636	0.610
13	0.463	0.561	0.438	0.652	0.407	0.582
14	0.842	0.962	0.819	0.899	0.781	0.804
15	0.443	0.749	0.425	0.535	0.403	0.441
16	0.380	0.382	0.344	0.474	0.315	0.395
17	0.311	0.240	0.278	0.240	0.258	0.104
18	0.540	0.375	0.424	0.526	0.372	0.463
19	0.367	0.375	0.326	0.255	0.302	0.219
20	0.532	0.520	0.500	0.562	0.467	0.468

Table B.5. Recall results of Auto-PU-NAS compared with EBO-1 on real-world biomedical datasets.

	δ = 20%		δ = 40%		δ = 60%	
Dataset	**NAS**	**EBO-1**	**NAS**	**EBO-1**	**NAS**	**EBO-1**
Alzheimer's	0.237	0.579	0.184	0.579	0.158	0.447
Autism	0.295	0.986	0.273	0.899	0.245	0.914
Breast cancer Coi.	0.632	0.943	0.585	0.943	0.476	0.5
Breast cancer Wis.	0.675	0.987	0.643	0.915	0.599	0.896
Breast cancer mut.	0.328	0.750	0.219	0.625	0.188	0.874
Cervical cancer	0.000	0.765	0.000	0.824	0.000	0.647
Cirrhosis	0.282	0.648	0.197	0.507	0.127	0.493
Dermatology	0.104	0.833	0.042	0.771	0.000	0.604
PI Diabetes	0.075	0.739	0.063	0.810	0.052	0.642
ES Diabetes	0.669	0.969	0.550	0.859	0.494	0.863
Heart Disease	0.248	0.879	0.200	0.800	0.121	0.727
Heart Failure	0.198	0.698	0.156	0.635	0.135	0.573
Hepatitis C	0.339	0.875	0.321	0.768	0.321	0.625
Kidney Disease	0.326	1.000	0.279	0.884	0.209	0.488
Liver Disease	0.278	0.971	0.237	0.940	0.181	0.686
Maternal Risk	0.371	0.871	0.320	0.787	0.305	0.735
Parkinsons	0.395	0.939	0.279	0.946	0.265	0.558
Parkinsons Biom.	0.133	0.200	0.100	0.533	0.000	0.1
Spine	0.607	0.927	0.440	0.913	0.393	0.653
Stroke	0.011	0.422	0.006	0.278	0.006	0.283

Table B.6. Recall results of Auto-PU-NAS compared with EBO-1 on synthetic datasets.

	δ = 20%		δ = 40%		δ = 60%	
Dataset	**NAS**	**EBO-1**	**NAS**	**EBO-1**	**NAS**	**EBO-1**
1	0.358	0.712	0.329	0.576	0.296	0.539
2	0.079	0.794	0.063	0.032	0.032	0.032
3	0.485	0.814	0.470	0.663	0.455	0.572
4	0.503	0.766	0.465	0.753	0.457	0.625
5	0.207	0.532	0.192	0.376	0.166	0.261
6	0.277	0.731	0.245	0.565	0.224	0.509
7	0.294	0.836	0.254	0.577	0.209	0.512
8	0.493	0.947	0.480	0.360	0.449	0.302
9	0.069	0.103	0.000	0.103	0.000	0.172
10	0.720	0.984	0.709	1.000	0.659	0.94
11	0.418	0.838	0.391	0.606	0.372	0.56
12	0.495	0.817	0.481	0.675	0.422	0.605
13	0.431	0.816	0.397	0.574	0.377	0.502
14	0.496	0.862	0.445	0.857	0.430	0.776
15	0.313	0.474	0.311	0.584	0.294	0.486
16	0.211	0.953	0.164	0.430	0.133	0.352
17	0.272	0.828	0.278	0.391	0.258	0.172
18	0.398	0.915	0.331	0.424	0.297	0.373
19	0.436	0.752	0.389	0.443	0.349	0.362
20	0.510	0.909	0.437	0.726	0.391	0.617

As is evident from these results, the Auto-PU-NAS system performed very poorly, and thus the decision was made not to utilise the system for further experimentation. The primary hypothesis for the reasons as to why the NAS system performed so poorly is the size of the search space. The search space grew extraordinarily large with the addition of the NAS hyperparameters, it is likely that the optimiser was simply not able to cope with the vast number of candidate solutions and thus was unable to find solutions close to those found by the other Auto-PU systems.