Population-based Ensemble Learning with Tree Structures for Classification

by

Benjamin Patrick Evans

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

Victoria University of Wellington
2019

Abstract

Ensemble learning is one of the most powerful extensions for improving upon individual machine learning models. Rather than a single model being used, several models are trained and the predictions combined to make a more informed decision. Such combinations will ideally overcome the shortcomings of any individual member of the ensemble. Most machine learning competition winners feature an ensemble of some sort, and there is also sound theoretical proof to the performance of certain ensembling schemes. The benefits of ensembling are clear in both theory and practice.

Despite the great performance, ensemble learning is not a trivial task. One of the main difficulties is designing appropriate ensembles. For example, how large should an ensemble be? What members should be included in an ensemble? How should these members be weighted? Our first contribution addresses these concerns using a strongly-typed population-based search (genetic programming) to construct well-performing ensembles, where the entire ensemble (members, hyperparameters, structure) is automatically learnt. The proposed method was found, in general, to be significantly better than all base members and commonly used comparison methods trialled.

With automatically designed ensembles, there is a range of applications, such as competition entries, forecasting and state-of-the-art predictions. However, often these applications also require additional preprocessing of the input data. Above the ensemble considers only the original training data, however, in many machine learning scenarios a pipeline is required (for example performing feature selection before classification).

For the second contribution, a novel automated machine learning method is proposed based on ensemble learning. This method uses a random population-based search of appropriate tree structures, and as such is embarrassingly parallel, an important consideration for automated machine learning. The proposed method is able to achieve equivalent or improved results over the current state-of-the-art methods and does so in a fraction of the time (six times as fast).

Finally, while complex ensembles offer great performance, one large limitation is the interpretability of such ensembles. For example, why does a forest of 500 trees predict a particular class for a given instance? In an effort to explain the behaviour of complex models (such as ensembles), several methods have been proposed. However, these approaches tend to suffer at least one of the following limitations: overly complex in the representation, local in their application, limited to particular feature types (i.e. categorical only), or limited to particular algorithms. For our third contribution, a novel model agnostic method for interpreting complex black-box machine learning models is proposed. The method is based on strongly-typed genetic programming and overcomes the aforementioned limitations. Multi-objective optimisation is used to generate a Pareto frontier of simple and explainable models which approximate the behaviour of much more complex methods. We found the resulting representations are far simpler than existing approaches (an important consideration for interpretability) while providing equivalent reconstruction performance.

Overall, this thesis addresses two of the major limitations of existing ensemble learning, i.e. the complex construction process and the black-box models that are often difficult to interpret. A novel application of ensemble learning in the field of automated machine learning is also proposed. All three methods have shown at least equivalent or improved performance than existing methods.

Acknowledgments

This thesis is dedicated to my mother. Thank you for everything.

I would also like to thank my supervisors and the rest of my family for their continued support.

Contents

Chapter 1

Introduction

In this chapter, ensemble learning is introduced and the research objectives and contributions are outlined.

1.1 Ensemble Learning

Ensemble learning is a key tool for improving the performance of machine learning methods. For example, a collection of uncorrelated weak learners which are only slightly better than random selection can be ensembled to produce a strong learner which error approaches zero as the number of learners increases [1]. Ensembling is a powerful tool and most challenge winners, such as Kaggle [2, 3], Netflix competition [4], and ImageNet [5] use ensembling in their final solution.

Ensembling can help overcome errors of individual learners, for example, if one learner is incorrect but the other two are correct with majority voting. The idea of ensembling is not isolated to machine learning. For example, in boxing, scores are based on 3 judges. Taking the most common result from the three judges helps overcome for example a malicious judge, and the result can be considered fairer than if only a single judge was used. A similar method is employed in the legal system with juries. Jurors must agree to a verdict for the defendant. The more serious the crime, the larger

the jury tends to be. Having a jury helps ensure the correct decision is made. The US Electoral College is another example, however, unlike the previous two, this behaves like a weighted ensemble in that not all members have uniform importance. Linking these examples back to machine learning ensembles, we can see that utilising multiple learners (members) can help to increase the belief in a prediction (a vote, verdict, score etc), and should help overcome any of the individual member's shortcomings (i.e. biases). Furthermore, a weighting system can also be used (i.e. with the electoral system) for the ensemble, if certain members are deemed less important than others for whatever reason.

Like with selecting judges for sports, or selecting members of a jury, the selection criteria for an ensemble is important. For example, selecting impartial judges is essential for a fair result. Likewise, a jury should be impartial and representative (although this is not always the case). Consider if a jury was composed of only one family, or all jurors were extremely similar, the result may not be fair. Likewise in ensembles, if the learners are not diverse, the ensemble will not see improved performance over the base members.

There are a number of open issues with ensemble learning which we aim to address in this thesis, and they are outlined in the following section. In particular, we focus on ensemble learning for classification due to the wide range of uses, although most methods would also work with regression either directly or with minimal changes.

1.2 Motivations

While the benefits of ensembling are clear, there remain some limitations which become the motivations for this thesis.

Well designed ensembles often outperform any of the individual members, however, constructing such ensembles can be a complex task. The complexity arises as it is necessary that the members are both accurate and

diverse [6, 7, 8], so care must be taken when constructing ensembles. Selecting appropriate base members is not the only issue, as each of the base members of the ensemble also tends to have a variety of hyperparameters associated (such as k in k-NN, or c in SVM's). Each of the hyperparameters can affect the resulting ensembles in different ways, so simultaneous optimisation of both the base members and hyperparameters is required to learn optimal ensembles. The result is a huge search space, as every combination of base members and all possible combinations of hyperparameters constitutes a potential solution. With N combinations of hyperparameters/members, there are 2^N possible ensembles. Thus, the search space of ensembles is far too large to consider an exhaustive search (the problem is NP-hard), so efficient approximations are required to construct well-performing ensembles.

Secondly, Automated Machine Learning (AutoML) is an emerging field which essentially uses machine learning to learn machine learning pipelines without requiring human input. Many of the current state-of-the-art approaches to AutoML are limited to learning a single pipeline, as the search space is already very large. Above we only considered base members to be learning algorithms, however, with AutoML these learning algorithms can also be accompanied by various preprocessing steps such as dimensionality reduction or feature scaling. However, ensemble learning is known to improve results over a single member [9], so combinations of such pipelines are required. This is a large limitation which potentially limits the growth of AutoML, as many competition winning solutions feature an ensemble of some sort [4, 5, 2, 3]. Introducing ensemble learning into the process of AutoML is difficult again due to the huge search space, and existing approaches lack extensive ensembling support. Furthermore, existing approaches to AutoML are already time-consuming to produce good results, and the methods are sequential by nature so massively parallel searches cannot be executed.

Finally, the interpretability of state-of-the-art ensembles is another cur-

rent limitation. Without understanding how an ensemble works, this could mean ensembles may go unchosen in favour of lower performing alternatives such as decision trees. Simple methods such as decision trees or linear methods are generally interpretable, however, the results are far lower than state-of-the-art methods [10]. In contrast, the results of ensembles are great, however, interpreting the ensembles (and other complex methods) can often be difficult [11]. For example, why did a 500-member ensemble choose to reject a specific applicant yet accept another? As the complexity of state-of-the-art methods increases, understanding why predictions are made is an increasingly important concern [12]. The limitation of interpreting the behaviour of complex ensembles must be addressed for ensembles to be considered a suitable approach in a range of areas (such as healthcare) where explainability is important.

1.3 Goals

The overall goal of this thesis is to investigate population-based methods for ensemble learning and ensemble interpretation for a wide range of classification tasks.

There are three main goals with this work.

1. The first is to utilise genetic programming for automatically constructing well-performing ensembles.

   Genetic Programming [13] is an evolutionary computation technique where solutions are typically represented as tree structures, and appropriate solutions are discovered automatically using Darwinian principles. Due to the automatic discovery of solutions, genetic programming is an ideal candidate for ensemble learning without requiring a priori knowledge. We would like to explore the use of genetic programming for designing hierarchical tree-based ensembles, where the base members and associated hyperparameters are

selected automatically (i.e. without requiring human input). The expectation is a fully automatic procedure which constructs ensembles which can outperform manually constructed ensembles.

2. Secondly, we would like to investigate the usefulness of ensembling in automated machine learning (AutoML).

   An AutoML method capable of searching ensembles of pipelines would be ideal, and as the goal above, we would like to investigate the use of tree-based ensembles for achieving this. Furthermore, we would like to address the limitation of lack of parallelization of existing approaches. The expectation is the introduction of ensembling into AutoML will improve upon the currently limited approaches to AutoML, and a parallelizable method will allow for fast and large-scale optimisation of pipelines.

3. The third goal is to overcome the limitation of the interpretability of complex ensembles.

   We would like to investigate the usefulness of genetic programming as a surrogate model for interpreting complex ensembles and other complex machine learning methods such as deep neural networks. Multi-objective optimisation can be used to control the trade-off between the complexity and the performance of the model. Learning a simpler representation of the complex models would allow the end user to understand why particular reasons are being made, rather than the classifier being treated as a black box. The expectation is the resulting models will be much simpler than existing approaches, while still providing good performance.

1.4 Major Contributions

There are three major contributions corresponding to the three goals in Section 1.3, which are presented as follows.

1. This thesis has shown how automated ensemble learning based on stacked generalisation (stacking) can be achieved using GP without requiring human input. The proposed method alleviates users from model selection, hyperparameter tuning, and ensembling. Appropriate base members and their key hyperparameter are selected automatically (i.e. without requiring human expertise), and the meta-learner is evolved as a tree structure. Methods for caching and automatic pruning are also demonstrated. The proposed method is evaluated on ten datasets, and on seven datasets the method achieved significantly better results than all the comparison methods, and on the remaining three, performed at least as good as the best comparison method. Using the Friedman test paired with Holm post-hoc analysis, the method is significantly better than all compared methods in general (i.e. over all datasets). This work sets the foundation for a fully automated approach to ensemble learning utilising genetic programming.

   This work is currently under review for journal submission.

2. This thesis has shown the importance of ensembling in automated machine learning. A novel ensemble-based automated machine learning method is proposed, which is able to achieve equivalent performance to the current state-of-the-art approaches in a fraction of the time. Furthermore, the proposed method is able to be run entirely in parallel, which helps overcome the limitation of automated machine learning taking too long to generate good pipelines. The usefulness of ensembling for AutoML is demonstrated, and a comparison of state-of-the-art approaches is also presented. The proposed method achieves equivalent or improved results across all the 15 datasets trialled and does so six times faster than the comparison methods.

   This work is currently under review for journal submission.

3. This thesis has shown the usefulness of multiobjective optimisation

for interpreting complex state-of-the-art black-box machine learning models. A novel method is proposed which is applicable to any machine learning model (e.g. complex ensembles, deep neural networks), and gives a global approximation of the behaviour of the model. Evolutionary multi-objective optimisation is used in conjunction with strongly-typed genetic programming to construct accurate, simple and model-agnostic representations of complex black-box estimators. We found the resulting representations are far simpler than existing approaches while providing comparable reconstructive performance. This is demonstrated on a range of datasets, by approximating the knowledge of complex black-box models such as 200 layer neural networks and ensembles of 500 trees, with a single tree.

This work has been published in:

Evans, B., Xue, B., Zhang, M. (2019). What's inside the black box? A genetic programming method for interpreting complex machine learning models. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2019*. ACM.

1.5 Structure

The remainder of the thesis is organised as follows. An overview of machine learning and a review of relevant literature in ensemble learning, automated machine learning and interpretable machine learning is given in Chapter 2. Chapter 3 introduces a novel method for ensemble learning. Chapter 4 investigates the usefulness of ensemble learning for automated machine learning and introduces a novel method for automated machine learning. Chapter 5 introduces a novel method for interpreting complex black-box machine learning models. Conclusions and future work are outlined in Chapter 6.

Chapter 2

Literature Review

2.1 Machine Learning

Machine learning can be defined as systems which automatically learn programs from data [14]. Machine learning has a huge range of applications, such as fraud prevention [15], game playing [16], recommendations [17], healthcare [18], credit scoring [19], forecasting [20], cybersecurity [21], stock trading [22], and advertising [23] to name a few. Machines are able to outperform humans in a range of these tasks [16, 24], and with traditional software development, this would not have been possible as only human-learnt rules would be encoded into the programs. Now, with machine learning, a training set is given to the learning machine, from which it is able to learn its own set of rules, which often outperform human-derived rules.

Machine learning can be broadly split into three main categories: supervised learning, unsupervised learning and reinforcement learning [25]. Other methods such as semi-supervised and transductive learning also exist [26]. The focus of this thesis is on supervised learning only, so this is what is expanded here.

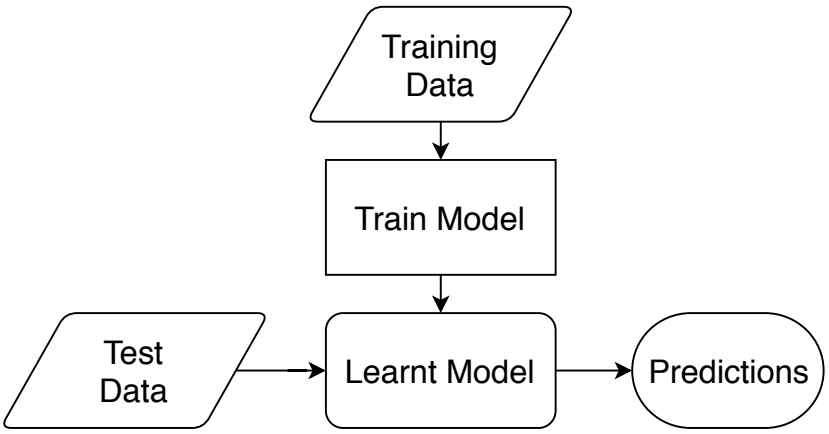


Figure 2.1: Supervised Learning process.

2.1.1 Supervised Learning

Supervised Learning is a subfield of machine learning, where the goal is to map an input to an output. This output can be discrete (classification), or continuous (regression). Here we focus only on classification, but most of the discussion also holds true for regression.

The overall process for supervised learning is summarised in Fig. 2.1.

It is common to split the available data into a training set and a test set. The training set is used to learn a model, and the test set is used to evaluate how well the learnt model performs on unseen data. Having dedicated training/test splits or using cross-fold validation is important to prevent bias when interpreting machine learning results, i.e., ensuring we do not evaluate a learnt model on data which was used to train the model. Overfitting can be categorised by high accuracy on the training set, but low accuracy on the unseen (test) data.

The generalisation ability of a model (the performance on unseen data) is an important characteristic, as such, overfitting to the training set must be avoided [14].

In supervised learning, the result is a model which can be used to make predictions for unseen data.

Classification

Classification is a sub-field of supervised learning, which attempts to assign a class label (a finite categorical output) y to an input vector x, by approximating a function f using labelled training data. More formally, it aims to learn a mapping f, such that $f(x) = y$, where y is from a finite set $y \in \{c_1, c_2.., c_n\}$. The end goal is to be able to predict a class label for some unseen data. Classification is perhaps the most widely used form of data mining tasks with applications such as face recognition [27], medical diagnosis [28] and sentiment analysis [29].

2.2 Evolutionary Computation

Evolutionary computation (EC) is a sub-field of methods in artificial intelligence which take inspiration from patterns seen in nature, such as utilising genetic operators (i.e. with genetic algorithms, genetic programming) or incorporating social behaviour (i.e. with particle swarm optimisation) for a population of candidate solutions. In this work, we focus particularly on the genetic side, so we will only introduce genetic-based methods.

Genetic methods can be seen as a search algorithm, where we are searching for an ideal candidate. A population of candidates (individuals) is used, and they are evolved over generations by utilising genetic operators. Each time operators are applied to the population, a new generation (i.e. an updated population) is formed. Randomness is incorporated using *mutation* of individuals. However, rather than having a purely random search, the search is guided using the idea of fitness, so we can favour breeding fit individuals in an attempt to improve the population. This breeding (combination) of two individuals is known as *crossover*. *Elitism* is used to ensure individuals never get worse in subsequent generations. The fitness for an individual is measured by a fitness function, which is any function which is able to evaluate performance (i.e. accuracy for clas-

sification, or error for regression). This function does not need to be differentiable as gradients are not incorporated, although they can be if desired, see memetic computing [30, 31].

EC techniques can be seen as an approximation to global search, in which they do not focus on a single area of the search space (unlike local search methods), instead, they are able to consider a large portion of the search space. They are able to do so efficiently compared to exhaustive/grid search, as they are only an approximation, meaning that they are more feasible to compute (at the expense of not being able to guarantee the obtained solution is the global optimum). Despite being an approximation, this tends to be strong enough in practice to effectively find high performing candidate solutions.

2.2.1 Genetic Programming

Genetic programming (GP) is an EC technique where individuals are typically represented as tree-like structures [13]. There are several favourable properties of GP over other methods, such as Genetic Algorithms (GAs). These are flexibility in representation (GAs are typically fixed-length bit strings, whereas trees are naturally more flexible), ability to enforce constraints (i.e. tiered structures using strongly-typed GP [32]), and also the increased interpretability (tree-structures tend to be more human readable than bit strings). An example of such a tree structure is given in Fig. 2.2, which can also be represented as the equation $10 * x^2$

Three key components of GP are the function set, the terminal set and the fitness function.

The function set defines the operations of internal nodes (i.e. nodes with children, shown as curved rectangles in Fig. 2.2). An example of a function node could be a node which takes two children a and b, and returns $a + b$. The terminal set defines the leaf nodes of the tree (i.e. nodes without children, shown as circles in Fig. 2.2), typically terminals are the

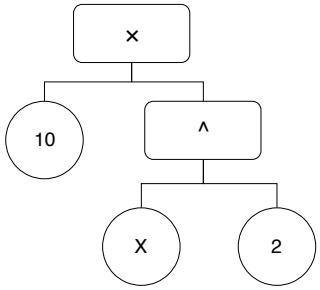


Figure 2.2: Example tree showing how GP represents solutions.

input data (i.e. features), and random constants (e.g. if we are performing regression, a random constant can be important to evolve coefficients). The primitive set refers to both the function and terminal set. The primitive set should aim to satisfy two properties: sufficiency and closure [33]. Sufficiency means some combination of the function and terminal set can solve the problem, however, this is only guaranteeable in problems with sound theoretical understanding, so often a good approximation is acceptable [34]. Closure means "any non-terminal should be able to handle as an argument any data type and value returned from a terminal or non-terminal" [32]. As an example, if divide is included in the function set, and 0 is a possible value, this should be protected to ensure dividing by zero returns an acceptable value (such as 0) to allow the programs to continue running as expected. Other examples are ensuring all inputs/outputs are of a single type (i.e. all integers) or converting types within functions (i.e. treating booleans as integers). Closure is important as both crossover and mutation can affect any node in the tree, so ensuring consistency between the primitives allows this to work.

The fitness function is problem specific and gives a way of evaluating an individual in the population. For example, in classification, the fitness function may be the percentage of correctly classified instances.

GP starts from a randomly generated initial population, and the population is updated via selection, crossover, mutation and reproduction operators in a number of generations until a predefined stopping criterion has

been met. Tournament selection is the most popular selection technique [35]. Crossover is performed by selecting two individuals and choosing a random point (node) from each individual. These nodes are then swapped, producing two new offspring (children). The idea is that combining two well-performing individuals could result in a new child which outperforms either of the parents. Individuals are typically selected with either roulette-wheel selection or tournament selection. Mutation selects a random node in the tree and replaces this branch with a new randomly generated branch. This operator is similar to random search and is used to keep the diversity of the population [36].

Strongly-typed Genetic Programming

Strongly-typed genetic programming (STGP) is a special type of GP first introduced by Montana in [32]. As stated above, with traditional GP, the primitive set should aim to satisfy two properties: sufficiency and closure. For STGP, the closure property is relaxed by enforcing type constraints on both the input and the output of all function nodes, and the output of terminal nodes. When performing crossover and mutation, rather than considering all possible points in the tree and all primitives, this is narrowed down only to those with the same type. For example, a node with type A will never be crossed over with a node with type B. This helps reduce the search space drastically and allows the enforcement of particular constraints within a tree.

STGP has a range of applications, such as unit-testing for software [37, 38], generating robot behaviour for football gameplay [39], and designing trees for image classification [40, 41]. Another example of a strongly-typed architecture for car creation is given in Fig. 2.3. We can see from the figure that a car requires an engine, wheels and a body. Each of these branches could not be crossed over with one another, as this would breach the constraints. For example, a turbo could not be added to the wheels. Likewise, when performing mutation we would only select from relevant types. For

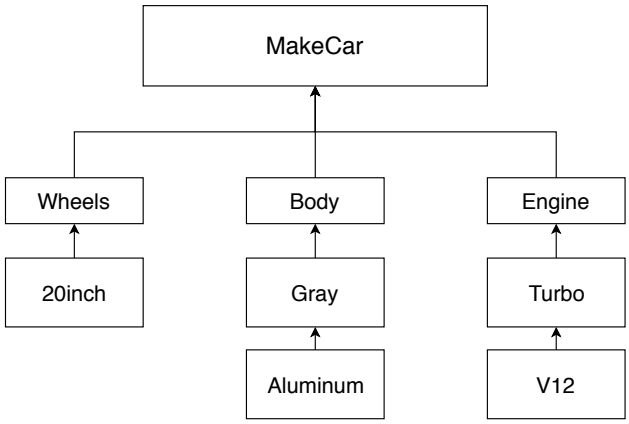


Figure 2.3: An example tree indicating how STGP can be used to enforce constraints.

.

example, the wheels may be mutated to be 19inch, or the body to be made of steel. However, an invalid tree would never be generated by, for example, by mutating the body to be 19inch wheels. Enforcing strong typing means hierarchies can also be implemented, such as a specific function must be applied before another, or a particular terminal can only be input for a given function.

2.3 Ensemble Learning

Ensemble learning refers to combining various base members (methods) to solve a problem, in this case, classification. In fact, this is similar to what is seen in business or politics, where crucial decisions are often made by a group rather than an individual. Ensemble learning can be seen as maximising the likelihood of generating correct results, by aggregating the results from multiple methods. In [42], three reasons are outlined for this:

1. There may not be enough training data to adequately search the hypotheses space for an individual learner, meaning we get several

equally performing hypotheses. Averaging these hypotheses can reduce the risk of selecting the wrong one.

2. Individual learners may get stuck in a local optimum, e.g. when performing gradient descent with neural networks. Combining several methods (assuming they are non-deterministic) can help overcome the local optimum, such as by starting the local search procedure in several different locations.

3. The true function which we aim to approximate may not be in the hypotheses space we are searching with an individual model. Whereas with combinations of these hypotheses, it is possible to generate a new hypothesis outside of the individual search spaces.

Intuitively, we can understand why ensembles have the ability to outperform individual learners by considering Venn diagrams, which is shown in Fig. 2.4.

Another way to think about the usefulness of ensembles is considering the classic bias-variance trade-off [43]. An ideal model is to have low bias and low variance, but with individual models, this is typically impossible. High variance means noisy data points have been modelled (overfitting), whereas high bias means relevant relations have been missed (underfitting). For an individual model, finding the balance between the two (generalising well why still capturing all relevant relationships) can be tricky, which is where ensemble methods can help.

There are various methods for generating ensembles, but the simplest general definition could be represented as $f(f_1(x), f_2(x), ..., f_L(x))$, where f is a function, which combines its inputs, x is the input vector, f_i is the ith base member, and L is the number of base members. The function f could be any range of functions, such as a simple vote, Bayesian averaging, or another classification method (stacking). Of course, each f_i could have been learnt from different overlapping subsets of the data (bagging), or

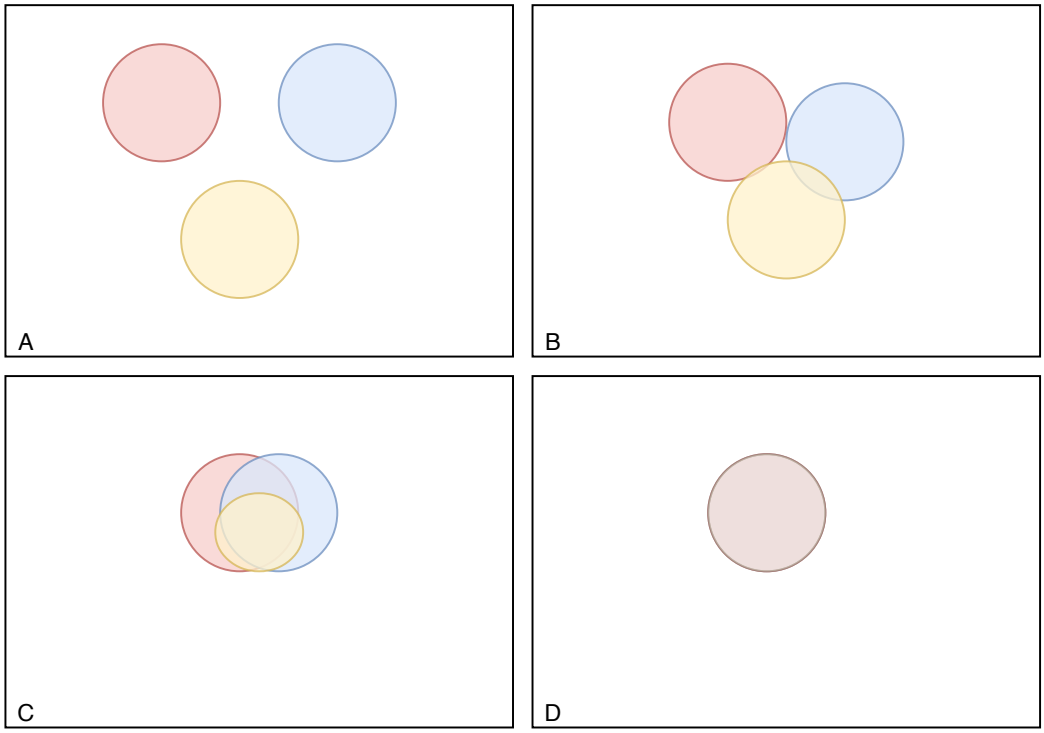


Figure 2.4: An example showing how ensembles have the ability to outperform base members. Each circle represents the errors for a particular base member. If we use a simple majority voting rule to generate an ensemble, we can see if the errors of the base members are completely uncorrelated (A), our voting scheme will result in a perfect result. When the area of the overlapping circle is less than the size of the smallest circle, the ensemble will result in a smaller error than any of the individual members (B). However, if it is the case that the overlapping area is larger than the smallest circle, the ensemble will do worse than the best individual (C). If errors are completely correlated (D), i.e. combining three identical models, then the ensemble will show no benefit over the individual model.

sequentially with different weightings for each of the training instances (boosting). More details can be seen as follows.

2.3.1 Combination Methods

Simple Voting

Perhaps the most basic method of constructing an ensemble for classification is to let each member vote for class "belongingness" [44]. This means if there are L base members, there will be L corresponding predictions, which we can then summarise, and choose based on: the most common vote, priority voting, the most confident vote, or other trivial metrics.

The main downfall with such an approach is that each base member is used, even if they are poor. If a large number of base members are poor, the ensemble will likely be poor, even if there are high performing members present. To account for this, a selection process could be performed before voting, or weighted voting used. However, trivial selection (i.e. selecting best performers) does not necessarily take into account complementary classifiers, so selection becomes difficult as a diverse range of complementary base members is required (essentially becomes feature selection which is NP-hard [45]).

Hansen and Salamon [44] showed for majority voting that if each base member achieves an error better than a random decision, and these errors are independent, then the ensemble error approaches zero as classifiers are added. Conversely, if the errors are worse than a random choice, the error grows as adding classifiers. This can be verified with Equation (2.1), for a particular error rate ϵ, and number of classifiers L, which is the probability that at least $> L/2$ classifiers are wrong (under the binomial distribution). If we assume an ϵ of 0.3 (i.e. 70% accuracy) for all classifiers (ϵ_i for $i \in 1..L$), and 3 classifiers (L=3), an ensemble error is $\epsilon_{ensemble}$ of 0.22. If increasing

L to 10, $\epsilon_{ensemble} = 0.048$.

$$\epsilon_{ensemble} = \sum_{i=\lceil L/2 \rceil}^{L} \binom{L}{i} \epsilon_i (1-\epsilon)^{L-i} \tag{2.1}$$

Note that while this is true in theory, this is rarely true in practice since the errors are not necessarily independent, so it is not reasonable to just keep adding classifiers to decrease the error. An important takeaway here is that the diversity (i.e. base members failing on separate parts of the data), along with the accuracy, is beneficial for ensembles, a belief shared by many researchers ([46, 7, 47]).

Bayesian Averaging

Another approach is based on Bayesian statistics, where Bayes theorem can be used to achieve an ensemble which is optimal in the ensemble space (i.e. no other method could outperform it on average using the same hypothesis space and priors) [42].

Typically, one may model multiple hypotheses and choose the single hypothesis which is most likely based on the overall training data (i.e. maximum a posteriori estimation). However, what we are perhaps more interested in is choosing the prediction with the most confidence for an individual instance (not being limited to a single hypothesis). The difference is the former looks for the most probable individual hypothesis given the entire training data (i.e. $\arg\max_h P(h|D)$), whereas the later looks for the most probable label for an individual instance given the training data (where we can consider multiple hypotheses, shown in Equation 2.2). Mitchell et al. [48] showed it is possible to combine these hypotheses to create a Bayes optimal classifier, where the predictions for each hypothesis are weighted by its posterior probability. The formula for this is given in Equation (2.2), where y is a class, h_i is an individual hypothesis, H is the

set of hypotheses, and D is the training data.

$$\arg\max_{y} \sum_{h_i \in H} P(y|h_i)P(h_i|D) \tag{2.2}$$

The question now becomes, if we are able to generate the optimal combination with Equation (2.2), why would we consider other combiners? The problem is that computing the posterior probability is often unfeasible in practice due to the large search space of H, and the true priors being unknown. These problems are solved with sampling, and estimating the priors (i.e. using uniform priors). However, this means the solution is no longer optimal. Therefore, this tends to only be used when the number of hypotheses is small, or priors are explicitly known.

Stacked Generalisation

Stacked generalisation [49] (stacking) can be seen as an improvement to the basic voting methods above. Rather than using simple votes, the predictions from the classifiers can be seen as new features for the data, and then any classification algorithm used to make a prediction based on these new features. The aim is to reduce both the bias and the variance of using a single model, by first running the data through multiple uncorrelated base learners. This process is shown in Fig. 2.5.

So if the original data was an $N \times M$ matrix, with N instances and M features, this runs into L classification algorithms, and L predictions are produced for each instance. These can be combined to form a new $N \times L$ matrix, treating the predictions as feature values for each instance. Now the new data is fed into the stacked learner (referred to as the meta-learner), and this prediction used as the new ensemble choice. This could also be stacked arbitrarily high by repeating the above process, which has been investigated in [50].

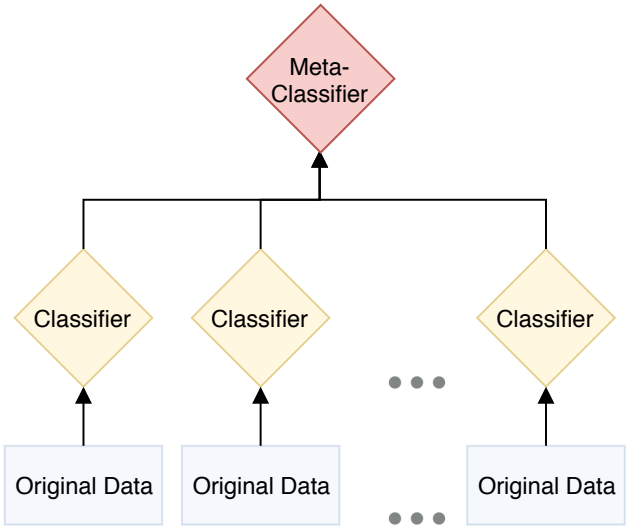


Figure 2.5: Stacking Process.

2.3.2 Perturbation Methods

In the previous section, we discussed how to combine individual methods to create an ensemble. However, these individual learners are often modified to promote the diversity of the methods. There are a number of ways to achieve this, such as training on different subsets of data points, training on different feature subsets, injecting randomness, or modifying input data. The most common ways are discussed in this section.

Bagging

Bootstrap aggregating (bagging) is a method which aims to reduce the variance of models by introducing sampling (with replacement) into the training process. Rather than training a model on a single dataset, several bagged subsets of the training data are made, and one model is trained on each of these subsets, and finally, these models are averaged to make a prediction (in the case of classification, by using plurality voting) [51].

To show how bagging helps reduce the variance of a model, consider

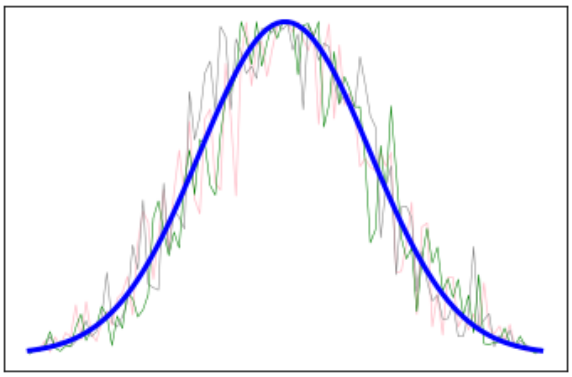


Figure 2.6: An example showing the usefulness of bagging. Assume we are trying to model the normal distribution. The various predictors are shown with the thin lines, as we can see these are overly noisy, where each predictor has high (but uncorrelated) variance from the true function. Averaging these results would help smooth out the function, giving a better approximation of the true function (shown with the thicker blue line).

Fig. 2.6, which shows the individual learners (thin lines) are potentially over-fitting the data, whereas when these results are averaged, a much simpler prediction line can be obtained, which in this case is closer to the true function.

Ideal candidates for bagging are models with high variance and low bias, such as unpruned decision trees. This is the basis for Random Forests [52], which are discussed in more detail later. While the process of bagging is simple (shown in Fig. 2.7), good results have been reported in practice, and Random Forests are one of the most popular classification methods today (despite being introduced over 15 years ago) due to their high accuracy.

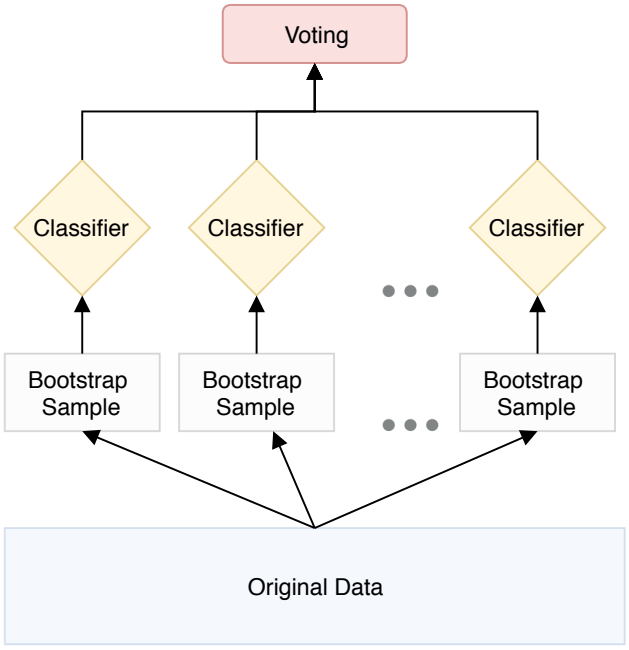


Figure 2.7: General Bagging process.

Random Subspace Method

With bagging, bootstrapped samples of instances can be created. The random subspace method is similar except that rather than sampling instances, features are sampled (sometimes referred to as feature bagging). So for an $N \times M$ matrix, where N is the number of instances and M the number of features, bagging selects from the rows, whereas random subspace selects from the columns.

Again, the goal here is to reduce the variance by reducing the correlation between learners, as each learner is using a different set of features. This has been applied to decision trees in [53] to create a forest (ensemble of trees) and the results significantly outperform an individual tree. Random Forests can also be seen as performing the random subspace method in a way, as when selecting splitting points, they consider only a random subset of all the features.

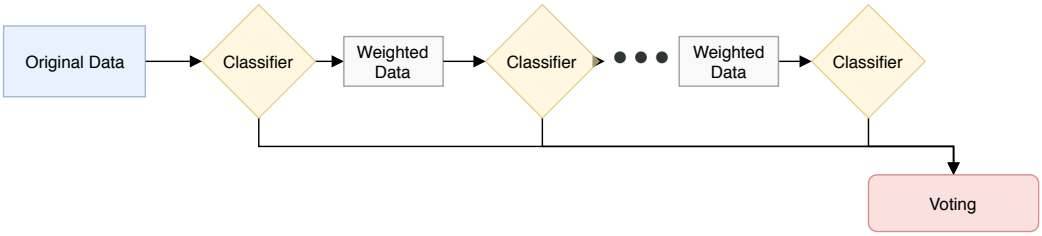


Figure 2.8: General boosting process.

Boosting

While bagging aims to reduce the variance of a model, boosting aims to reduce the bias of a model. This is achieved by iteratively training weak models and assigning higher weights to instances which were misclassified in previous steps to ultimately create a strong model by weighted voting. The general process of this is shown in Fig. 2.8.

Boosting can be applied to any base learners, but a key consideration with boosting is that the base learner is not too strong or the algorithm may overfit (if run for too many iterations), and is not too weak (i.e. should be better than a random vote) or the ensemble may still perform poorly. There are many applications of boosting, but the most common methods are AdaBoost [54], Gradient boosting [55], and more recently XGBoost [56].

Two of the main considerations are: how to weight the instances (i.e. how to place more emphasis on the misclassified points), and how to weight the base learners contribute to the final vote. As an example, we show how AdaBoost solves these two considerations. Binary classification is assumed for simplicity, although AdaBoost is not limited to binary classification only.

For the classifier weights (contribution to the weighted vote), this is based on the performance of the classifier as given in Equation (2.3), which is 0 when the error is 0.5 (the same as a random choice). The error grows exponentially as the error approaches 0, and shrinks exponentially when the error is above 0.5 (i.e. worse than random). ϵ is the misclassification

rate. This gives good performers exponentially higher weights than poor performers.

$$\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t}) \tag{2.3}$$

For the instance weights, we start with uniform weights (i.e. $1/N$) for each instance i and increase the weight if i is misclassified, and decrease the weight if i is correctly classified. This is done using Equation (2.4), where $w_t(i)$ is the weight for instance i at the tth iteration. The denominator is a normalising constant to ensure the weights sum to 1. f_t is the prediction of the previous classifier for the input instance x_i, and y_i is the real class label.

$$w_{t+1}(i) = \frac{w_t \exp(-\alpha_t y_i f_t(x_i))}{\sum w_{t+1}} \tag{2.4}$$

Injecting Randomness

While noise can often be seen as problematic for learners, e.g. by increasing the ease of overfitting, we can actually make use of intentionally injecting noise for ensemble learning. Consider with a single method, if noise is added to the input data, noise is more likely to be modelled directly, but if we have several methods with varying amounts of noise, averaging these results could help "smooth" out the function, giving a better approximation of the true function (assuming unstructured noise).

There are several parts of the learning process we could inject noise to, such as the input data (feature values), the output data (i.e. swapping classes for some data), or the model parameters (i.e. neural network weights). However, the most common method of injecting noise is to add the noise to the input data, as this has been shown to have the best generalisation ability when compared with adding noise to the outputs or to model parameters [57] (at least in neural networks). We will refer to the process of adding noise to the input data as additive noise, and this can be

represented as $\bar{x} = x + noise(x)$, where $\bar{x}$ represents the new feature value, x is the original value, and $noise$ is a noise generating function.

When adding noise, typically Gaussian noise is used (referred to as additive white Gaussian noise) as shown in Equation (2.5), as this is symmetric around the mean, and extreme values are unlikely due to the inclusion of e^{-x^2} term. Furthermore, Gaussian noise turns out to be a good estimator of real noise in many situations [58], although this is not the only noise function which can be used.

$$noise(x) = \frac{e^{-(x-\mu)^2}/2\sigma^2}{\sqrt{2\pi\sigma^2}} \tag{2.5}$$

Now by combining learners which have variable noise, we will ideally get better generalisation ability, as the data we are using to learn our function is more diverse and this diversity will hopefully encapsulate the unseen data. Note that this is similar to data augmentation commonly seen in deep learning ([59, 60]). However, with data augmentation, the goal is just to enlarge the training set for a single model.

The methods outlined above are by no means an exhaustive list of applicable methods for ensemble learning, just a collection of the most common ones. These methods are often combined, so we are not limited to selecting a single method either.

2.3.3 Related Work

In Chapter 3, we focus on utilising GP for evolving ensembles. There are two main ways that GP can be used for ensemble learning. The first (as done in [61, 62]) treats individuals/trees in the population as base members of the ensemble, which is computationally effective as EC techniques natively feature a population of solutions (even when we only care about a single solution). The main issue with using the individual trees as base members of the ensemble is that a large portion of the individuals in EC algorithms are poor performers, which could impact the resulting accuracy

without performing additional selection. To select ensemble members, if there are n individuals in the population, there are 2^n possible ensembles, a number which becomes intractable very quickly for typical population sizes (i.e. 1024), meaning we must use heuristic or approximate search methods (essentially feature selection). For this reason, this is not an ideal approach as feature selection is NP-hard.

The second approach is based on stacking, which appears to be a very promising method of ensemble learning as discussed in Section 2.3.1. Stacking requires a meta-learner to combine the base learners, and GP appears very promising as a meta-learner. The base learners become the terminal set, and the function set is composed of nodes which can combine the predictions, meaning that the evolved tree serves as the meta-learner. Another benefit of utilising GP for the meta-learner is individuals in the initial population tend to be small and grow as the generations increase, which means that the ensemble complexity will increase with the generations, so simple problems will ideally result in a simple ensemble, and the ensemble complexity increases with the problems complexity. We hypothesise that utilising each individual in the population as an ensemble on its own can result in improved performance (at the expense of increased computational time, which we aim to offset with caching), so in this section, we focus on works which treat each individual as an ensemble, rather than the entire population as an ensemble.

Langdon and Buxton [63] worked on fusing classifiers with GP to produce an ensemble which has a better ROC curve than the convex hull of the base members ROC curve (a greater area under the curve). The area under the ROC curve gives an idea of the performance of a classifier in binary classification, the larger the area, the better a method can be considered. The terminals are the output of the classifiers, where the sign (positive or negative) indicates the class, and the value/magnitude the confidence of the prediction. This showed that a rather simple configuration, which utilises arithmetic operators (+, -, *, /) along with some basic

functions (min, max, absolute min, absolute max, if) can produce a powerful GP-based ensemble which outperforms its members for binary classification. This was further extended by the authors in [64], where they used GP to evolve a composite classifier (ensemble) of neural networks, and in [65] where they used GP to combine decision trees and neural networks for drug discovery. All three works showed the benefits of using GP to combine classifiers.

Evolving ensembles of Support Vector Machines (SVMs) with different kernel functions was looked at in [66]. Despite being limited to only SVMs, which would seem to have limited diversity, it was found GP was able to learn a combination of SVMs which outperformed the individual SVM classifiers (even when these individual classifiers were optimised with grid search) perhaps due to the variety of kernel functions.

Khan et al. [67] looked at evolving combinations of classifiers using GP, however, this is limited to only 2 classifiers (Linear discriminant analysis and Mahalanobis distance based classifier). Nevertheless, the results were good and again showed the promise of using GP for combining classifiers. There are two combination approaches trialled: homogeneous and heterogeneous. The heterogeneous approach combines different classifiers using the same training data, whereas the homogeneous approach uses the same classifier on different data and/or feature subsets. The authors found heterogeneous combinations (i.e. utilising different classifiers) outperformed homogeneous combinations, likely due to the improved diversity.

While GP has shown the capability to outperform the base members significantly, in all cases such works are limited to either: binary classes, or limited base members. In Chapter 3, we propose a method which can effectively deal with multi-class classification, and features a broad range of potential base members, with the added benefit of selecting the various hyperparameters of the base members.

2.4 Automated Machine Learning

AutoML aims to automate the data processing, feature processing and model selection steps of a machine learning pipeline, freeing the data scientist to focus on other aspects of the workflow. The goal is to be able to replace the difficult process of selecting components of the pipeline with an automatic approach, where all the user needs to do is specify a dataset and an amount of time to train for, and an appropriate pipeline is returned automatically. In the following section, we discuss several search techniques and discuss their applications to both AutoML and related areas such as hyperparameter optimisation.

2.4.1 Grid Search

A grid search is a global search technique which explores every possible combination of values (i.e. an exhaustive search). For very small search spaces, this is a good choice since the best possible solution is found. However, this becomes unpractical for many real-world problems. For AutoML, a grid search would entail searching over every possible combination of preprocessors and estimators, as well as all of their hyperparameters, a search space which is far too large to consider in practice ($\approx 2.47 * 10^{17}$ potential solutions in [68]).

2.4.2 Random Search

Rather than searching the entire search space as done in a grid search, random search randomly samples from the search space. Bergstra and Bengio [69] found random search to be competitive to grid search for hyperparameter optimisation while being far more computationally efficient. Despite the stochastic nature, good results have also been reported in practice and random search is still a commonly used technique particular with limited computational resources or large scale optimisation.

While this may initially seem surprising, there is also sound probabilistic backing. Often we do not require the single best solution, but a good enough approximation. If we assume we only want to choose the AutoML system which is in the top 5% of all possible AutoML systems for a problem, then it holds true that drawing a single random sample we have a 5% chance of achieving this. However, if we draw n samples, then the probability that at least one of them is in the top 5% becomes $1 - (1 - 0.05)^n$ [70]. The n is often surprisingly low, i.e. to be 90% confident our solution is in the top 5%, we only require $n = 40$ samples.

For AutoML, the range of potential combinations of the components can be arranged as a grid, and this grid sampled uniformly to perform a random search. This has been compared to a Bayesian approach (AutoWEKA discussed below) in [71] and was outperformed in general. In [72], the method based on random search was able to get competitive performance to the genetic programming based method (TPOT, discussed below), however, had a larger variation in the results, and resulted in larger pipelines on average. So at this stage, random search has not been considered a suitable approach to AutoML.

A benefit both grid search and random search have over the methods below is that they are considered "embarrassingly parallel". The entire search process can be parallelised, which is a great benefit with computational power being available, yet time being limited. For example, a search which could take 365 days on a single machine, could take 1 day on 365 machines. With cloud computing, the costs would be equivalent, but the waiting time drastically reduced. For AutoML, this is an important consideration, as evaluation of pipelines can be costly/time consuming, and results are usually required as soon as possible.

2.4.3 Bayesian Approaches

Bayesian optimisation can be seen as a guided improvement to random search. Bayesian optimisation uses a statistical model to model the objective function, and an acquisition function to decide where to sample from next [73]. By using this acquisition function, we are able to make informed decisions, unlike random search. Relating Bayesian methods to exploration and exploitation, exploration is achieved by sampling from areas of high uncertainty, whereas exploitation is achieved by sampling from promising areas (i.e. those given from the acquisition function) [74].

Auto-WEKA [71, 75] is a Bayesian approach to AutoML, and can be seen as the pioneering AutoML system. This works on the WEKA software [76]. Pipeline design is treated as an optimisation process, where even the models selected are just treated as hyperparameters, and the process is dubbed combined algorithm selection and hyperparameter optimisation (CASH). CASH can be formally defined as in Equation 2.6.

$$A^*\lambda^* \in \underset{A^{(j)}\in\mathcal{A},\lambda\in\Lambda^{(j)}}{\operatorname{argmin}} \frac{1}{k}\sum_{i=1}^{k}\mathcal{L}(A_\lambda^{(j)}, D_{\text{train}}^{(i)}, D_{\text{valid}}^{(i)}) \tag{2.6}$$

which is minimising the loss of k-fold cross-validation, where D is the original training data, A is a learning algorithm, λ the hyperparameters. The goal is to choose the method and parameters with optimal generalization performance. For optimisation, Sequential Model-based Algorithm Configuration (SMAC) was found to perform the best. Auto-WEKA can select a base classifier, a feature selector, and appropriate hyperparameters automatically. Ensembling can occur if Auto-WEKA chooses a combination method as the base classifier, in which case up to 5 classifiers can come under this (limited to a single layer of ensembling, i.e. combination methods cannot be stacked). No data preprocessing is included in the pipeline, instead, if an algorithm does not apply, it is excluded from the search space.

auto-sklearn [68] is similar Auto-WEKA, in that it uses Bayesian optimi-

sation (SMAC) to minimise the CASH problem (Equation 2.6). However, Auto-sklearn uses scikit-learn (sklearn) [77]. Pipelines are composed of up to three data preprocessors, one feature preprocessor, and one classifier. Data preprocessors can include re-scaling features, imputing missing values, one-hot encoding for categorical features and class balancing. Combination ensembles are not directly included as classifiers in auto-sklearn, instead, ensembles are constructed using a greedy ensemble construction method (iteratively add the pipeline which maximises the ensemble performance on a validation set). The size of the ensemble defaults to 50, but can be specified by the user. Uniform weights are used, but the selection method is with replacement, effectively introducing a weighted system. Meta-learning can also be used, where rather than starting from a random state, auto-sklearn can initialise the search with a solution for a similar previous problem.

2.4.4 Evolutionary Approaches

Evolutionary Computation (EC) is an area of nature-inspired techniques that approximate global search. The search space is effectively (but not exhaustively) explored using mutation and exploited using crossover operations. Mutation operators behave like random search, while crossover operations aim to place more emphasis on good regions of the search space by combining well-performing solutions.

TPOT [78] is a genetic programming (GP) based AutoML system for sklearn. Both the Bayesian methods had fixed shape pipelines. However, in TPOT, pipelines can be variable shaped. For example, two separate feature processors could be performed on the raw data, and then the results combined to create a new dataset. Much of the search space is therefore dedicated to exploring various pipeline structures. To balance the complexity of the pipeline, multi-objective optimisation (NSGA-II) is used. While TPOT does not evolve ensembles of pipelines, a form of stacking

can appear in the automatically constructed pipelines. The way this is achieved is if an estimator appears anywhere besides the final step of the pipeline, the prediction of this estimator is appending as a synthetic feature to the original data which can be used in later steps of the pipeline.

RECIPE [79] is a grammatical-based GP framework for AutoML, again using the sklearn library. The key limitations RECIPE hoped to resolve were to remove the generation of invalid individuals in the evolutionary pipeline, by enforcing a grammatical structure. Enforcing a pipeline structure was a key development, and doing so allowed the expansion of the search space (by including more possible preprocessors and classifiers), as resources were not wasted evaluating invalid individuals. No explicit ensembling is done in RECIPE. It is also worth mentioning, since the publishing of the paper, TPOT has overcome these limitations in their newer versions (and the version used later for comparison).

One downfall of both evolutionary and Bayesian methods is that they are sequential by design. Meaning, while parallelisation can speed up the process (i.e. evaluating individuals in parallel or evaluating each fold in parallel), the overall process is still sequential and thus not embarrassingly parallel, so we are limited in the speedup achievable.

2.4.5 AutoML for Deep Learning

All of the above methods focus on more "traditional" approaches to Machine Learning. However, there are also other methods, such as Auto-Keras (Bayesian) [80] and AdaNet (evolutionary) [81], which focus on AutoML for deep learning based methods only. In this thesis, we do not focus on deep learning approaches (or aim to compare "traditional" vs deep approaches) due to the high computational cost associated with both deep learning and AutoML. For this reason, we do not extend upon this discussion further. However, if ample computational power was available, this work could be extended to incorporate deep learning methods as the

principles are not dependant on particular learning methods.

2.5 Interpretable Machine Learning

Interpreting Machine Learning models is becoming an increasingly important concern as the state-of-the-art approaches increase in complexity.

There is no short history of bias in machine learning techniques. As examples, Sweeney [82] exposed potentially racial bias in the suggestion algorithm used in Google AdSense, and Bolukbasi et al. [83] showed the popular word2vec algorithm can be heavily susceptible to gender bias. This has started the push towards Explainable AI (XAI).

XAI, more specifically, interpretable machine learning (IML), can help observe these biases, to then ideally remedy the biases. The bias can occur from a number of sources, such as the sampling of the training data, uncovering correlative but not causal relationships, or poor selection of feature sets. To uncover these biases, it's important to understand how the model is making particular decisions.

With XAI, the goal is to have the simplest rules possible without sacrificing the performance. Simplicity and performance are often conflicting objectives (motivating the need for XAI, since top performing methods are often complex). With traditional tree-based methods for XAI, such as decision tree construction, complexity is controlled by early stopping or post-pruning. These approaches suffer from limitations, such as, with early stopping (or pre-pruning), a branch is terminated when no reduction in cross-validation error is noted. However, this may be premature since additional splits further down may have reduced the error drastically (i.e. if the feature becomes more informative with the addition of another because of feature interaction). With post pruning, leaves are shrunk by replacing parent nodes with the majority class of the leaf. If no increase in error is seen, this process continues until the error increases for each branch. Alternatively, trees are shrunk in a top-down manner, i.e. with

cost-complexity pruning. Of course a major drawback to both approaches is since the tree was greedily constructed, a poor split in hindsight cannot be undone, so the pruning is limited in its ability, i.e. pruning is not going to find the optimal tree which balances complexity and accuracy, since it first greedily maximises the accuracy, then attempts to greedily reduce the complexity afterwards.

There are several main approaches to IML/XAI, which are briefly introduced along with the limitations here. For a more in-depth discussion, please see [84].

Firstly is exploratory data analysis [85] (EDA). While EDA can help analyse features and attributes of the data, it does not tell us anything about the model being used. For this reason, we do not consider EDA as part of IML/XAI. Rather, a preprocessing step for data explanation (and not model explanation). Likewise, feature selection can also be considered as IML [86]. However, again here we consider this a potential preprocessing step only.

Next is to use explainable models directly, with methods such as decision trees, linear models, or simple classification rules. While it is true that these models can offer high interpretability, the performance is drastically lower than the current state-of-the-art methods in ML (e.g. neural networks, random forests and boosting). Often, this drop in predictive ability practical is too large to consider for the increased interpretability. For this reason, simple models on their own are not an ideal approach to IML.

Sparse models are another approach. However, while sparsity does simplify models, they are still not necessarily interpretable. For example consider applying an L1 penalty to a deep neural network, despite having zeroed out some weights. This resulting model is still far from interpretable.

For deep learning methods, there are various explanation methods which can be used, such as sensitivity analysis with partial derivatives, heat

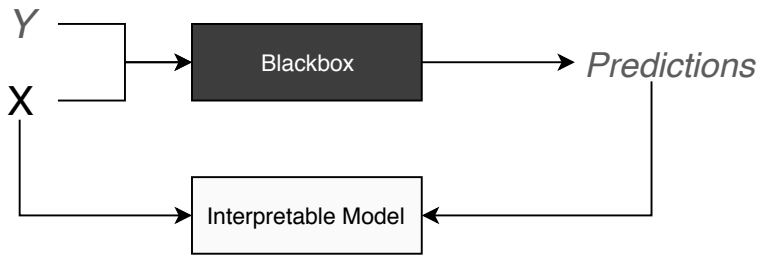


Figure 2.9: Model extraction process. The black-box model uses the original y labels for training, whereas the interpretable model uses the *predictions* from the black-box model.

maps of activation's, layer-wise methods [87], or deconvolutional networks [88]. The limitation with such methods is that they are only relevant to deep learning models (not arbitrary black-box models), and also are often local (applicable to a single prediction only) in their application.

There are other local model agnostic approaches to IML, such as LIME [89], which give information about particular predictions, but not on the global behaviour of a system. Local explanations can be useful but should be paired with a global approach to provide a fuller understanding.

A global model agnostic approach to IML is *model extraction* [90] (also called *mimic models* [91] or global surrogate models [92]). One of the trade-offs of using interpretable models was a drop in predictive performance, so one solution is to utilise two models - one black-box (complex) models for predictions, and a secondary simple model for describing the black-box model. This is referred to as model extraction [90]. Rather than the secondary model being trained on the original outputs, the secondary model is trained on the predictions from the black-box model. This process is shown in Fig. 2.9. This is the approach we take in this work, due to the fact that the method is applicable to any black-box method and makes no further assumptions (such as gradient-based, or ability to apply sparsity).

It should also be mentioned that it is not always the case that interpretability is important, i.e. to prevent "gaming the system" [91], or in well-studied problems [93]. A model extraction approach means the stan-

dard models used and work-flows can remain the same, however, in cases where interpretability is required, a secondary model can be utilised to gather additional insights to the complex model.

2.5.1 Model Extraction

One of the early works in the area was [94] which used decision trees to approximate complex black-box models. Similar work was done in [90]. Both utilise decision trees as a simple method for approximating more complex models. However, an issue is ensuring these decision trees remain simple themselves, so operations such as early stopping or pruning become essential. Furthermore, due to the greedy construction of the trees, they may not be the best approximator of the more complex black-box methods. Other methods such as logistic regression can also be used, and these are compared in Section 5.3.

Bayesian Rule lists [95, 96] are an approach to IML which aim to achieve a good balance between complexity and accuracy, by using Bayesian optimisation to generate a set of "*if...then...*" statements which can be used for prediction. The idea is that these resulting rules are simple and easily interpretable.

Model compression was proposed in [97] (and expanded in [98]), where the authors use a neural network to compress large complex ensembles (often with thousands of base members) by training the smaller neural networks on the predictions of the ensemble. While these are not directly related to interpretable machine learning (as the neural network learnt is not necessarily interpretable), the concept is similar, and these works showed the simpler model can often achieve similar error rates to the larger, more complex ensembles, so this is promising for model extraction methods.

2.6 Chapter Summary

An overview of machine learning and relevant sub-areas such as ensemble learning and evolutionary computation was given. Related work was also reviewed for the three main contributions.

Ensemble learning is a powerful concept with a wealth of benefits, however, construction of effective ensembles can be a complex task. GP has been shown to be successful for constructing ensembles, however, current approaches were all limited to either binary classification, a limited number of base members, or had the inability to perform simultaneous classifier selection and hyperparameter selection.

For Automated Machine Learning, there has been little focus on ensembling despite the range of success reported in classification tasks. Introducing advanced ensembling schemes such as stacking could significantly improve upon individual pipelines. Current methods for AutoML are also sequential by nature, and as AutoML can be seen as a costly procedure, this severely limits the potential applications.

Finally, interpreting state-of-the-art machine learning models (such as complex ensembles) still remains a complex task. Existing approaches tend to suffer from at least one of the following limitations: applicable only to specific models, inability to deal with a mixture of feature types, local explanations only, or overly complex in their global representation.

Chapter 3

Automated Ensemble Learning and Parameter Selection with GP

Classification is an important task in a vast variety of areas, but in many areas, there is no adequate knowledge available to select a suitable method(s). It can be daunting for users (particularly those with limited machine learning experience) to develop appropriate models. The task is complex as there are often assumptions which need to be carefully considered before selecting an algorithm (such as whether the function we are trying to approximate is plagued with local optima, whether models are scale invariant, whether the data is linearly separable etc.), and even once an "ideal" algorithm is selected, this then needs to be tuned by selecting appropriate hyperparameters. In addition, the method could still likely be further improved by combining this with others (known as ensembling) [42]. Alleviating users from these tasks is the main motivation for the work.

3.1 Chapter Introduction

In this chapter, a novel method for automated ensemble learning based on stacked generalisation is proposed. The method is capable of automatically selecting appropriate base members (classifiers) without needing to

specify any *prior* information about the model or the data by utilising genetic programming. Key parameters for classifiers in the ensemble are also automatically determined, i.e. removing the need to manually specify these parameters. The work is a step towards "automated machine learning" (AutoML) for ensembles, and aims to solve the following problems: removing the need for model selection, selecting key parameters for the models, and finding combinations (ensembles) of models which perform well automatically.

Particularly, the focus is on ensembles for classification (predicting categorical outputs), as this is one of the most common tasks in machine learning (although the method is general and could be extended to regression problems as well). The specific goals of this chapter are to develop a novel algorithm which can:

- Automatically select appropriate base members of an ensemble (or even a single member if this performs well),
- Minimise the generalisation error of these members by utilising the stacked generalisation,
- Select key hyperparameter values of the base members, and
- Allow a user to interpret the rules of the ensemble.

3.2 The Proposed Method

A genetic programming based method is proposed which automatically evolves heterogeneous ensembles of classification algorithms. A heterogeneous approach is chosen for two reasons, firstly, improved performance [67], and secondly, removing the need to choose an appropriate classifier with the homogeneous approach. The method not only evolves suitable base members of the ensemble (removing the need for model selection)

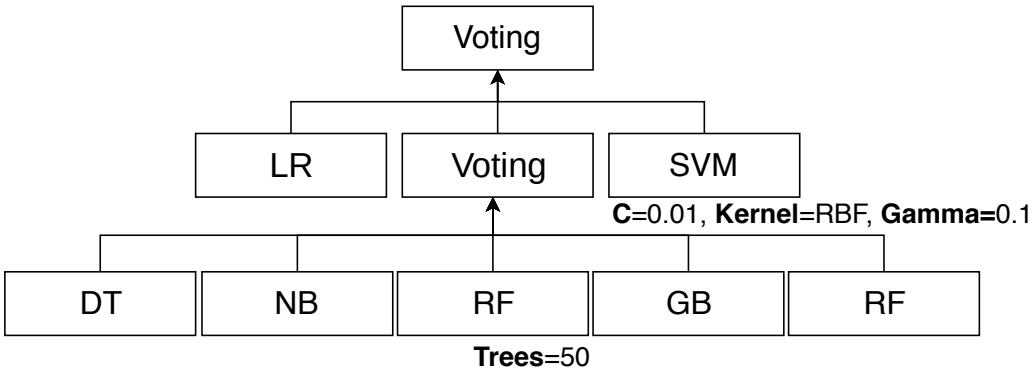


Figure 3.1: An example tree showing the proposed structure. Seven base members are used by combining two voting functions. Parameters evolved are shown as text below the classification node, such as "Trees" for the number of trees in the random forest, or "C" indicating the penalty term for the SVM. In cases with no parameters shown (i.e. no text under the node), the default parameters were used.

but also selects key hyperparameters of the methods (removing the need for model tuning).

3.2.1 Overall Structure

Ensembles are represented as tree structures, where the leaves (terminal nodes) represent base members and tunable parameters, and the internal nodes compose the meta-learner. Each tree serves as a stacked ensemble, where the output of the tree is the prediction of the ensemble as a whole, and the output of any non-root node can be considered as an intermediary prediction (i.e. a sub-ensemble). This representation means that we can utilise GP to explore the search space of potential trees since the number of potential trees makes it unfeasible to perform an exhaustive search. An example tree structure is given in Fig. 3.1.

3.2.2 Representation

To evolve individuals (trees), what constitutes a valid tree needs to be defined. In this chapter, Strongly-typed GP [32] is used, which allows placing of additional restrictions on the trees. For example, this can ensure we do not try and set a parameter value which does not exist for a particular classifier.

Terminal Set

The terminal set is composed of various classification algorithms which serve as the potential ensemble members (i.e. base learners). It is worth noting these classification algorithms can also serve as the root node, so a tree can be composed of only a single member. Doing so ensures that if a single method achieves the highest achievable fitness, then we do not need to bother evolving an overly complex ensemble. As terminals, these algorithms take their default values (as specified in the scikit-learn library). To allow parameter tuning, algorithms with parameters are also specified in the function set (discussed below), as technically, in this case, the parameters would be the leaf nodes and the classification algorithm a function node.

The classification algorithms used are shown in Table 3.1, where ensemble methods are also used as terminals here (so we can have an ensemble of ensembles). This is why Random Forests, Gradient Boosting and AdaBoost were used in conjunction with the "basic" methods. There are two benefits of including ensembles as terminals. Firstly, this allows the automated selection of an existing ensemble method and appropriate hyperparameters. This removes the need for a user to determine an appropriate ensemble technique as this requires machine learning knowledge, which is not always present. Secondly, in some complex situations, a combination of these ensemble methods is often better than using only a single one. This method will allow for the automatic generation of an

Table 3.1: The various classification algorithms used, and the hyperparameters which we evolve.

Model	Parameter	Range	Description
	C	[0.01, 0.1, 1.0, 10.0, 100.0]	Penalty term
SVM	Kernel	["rbf", "linear", "poly"]	The type of kernel to use
	Gamma	[0.01, 0.1, 1.0, 10.0, 100.0]	Kernel coefficient
Logistic Regression	Penalty	["l1", "l2"]	The norm of penalty function
K-Nearest Neighbour	K	[1,2,...,50]	Number of neighbours
Naive Bayes	*N/A*		
Decision Tree	Split	["best", "random"]	Split at best, or at best of a random subset
Random Forest			
Adaboost	Trees	[10,11,..,100]	The number of base estimators to use
Gradient Boosted Trees			

ensemble with appropriate complexity. A description of each algorithm is given in Section 3.3.

The various parameter values given in Table 3.1 are also used as terminals, with strongly typed genetic programming [32] used to ensure the algorithms only use the appropriate parameters (i.e. for K-Nearest Neighbour we are not going to pass in a kernel function).

The training data is also considered as a terminal, so various copies of the training data are passed to the evaluation nodes. This configuration allows for easy expansion to include data sampling techniques or operating on specific feature subsets if desired in future developments.

Function Set

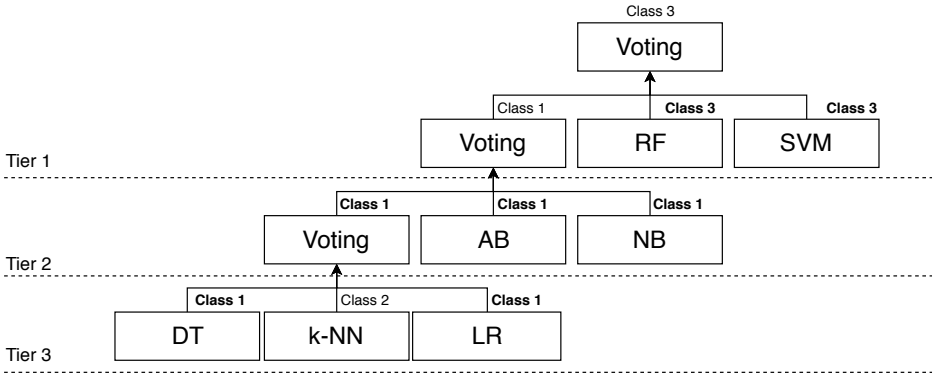
The main component of the function set is plurality voting. Plurality voting differs from majority voting in that the most common (mode) vote is used, even if this does not achieve a majority. This is best shown with multi-class classification. For example, if there are 5 predictors, and the class predictions are (1, 4, 2, 1, 3), 1 forms a plurality, but does not achieve a majority. So, in this case, 1 is used as the prediction. In cases when there is a tie, the first class to achieve the plurality is chosen (from the left).

Voting functions can be stacked arbitrarily high, creating a non-linear combination of the predictors. Of course, the classification algorithms can also be used as inputs to the voting functions. The number of inputs can be either 3, or 5. Any fewer inputs would be unnecessary, as a single input could be composed of only that node instead, and two inputs would likely disagree and offer little benefit (compared to having three members, where the chances of any 2 agreeing are much higher). Any larger number of inputs could instead be composed of various stacked voting functions (i.e. 7 inputs made with a 5 and 3 stacked, as shown in Figure 3.1).

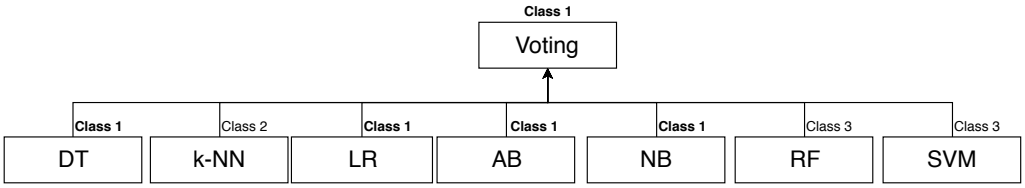
The classification algorithms with tunable parameters (as shown in Table 3.1) are all included as function nodes in the tree, where the inputs are the hyperparameters. Both voting nodes and classification nodes output predictions for the input data, which means they are able to be crossed over and still result in valid trees.

3.2.3 Voting Strategy

As discussed in the previous section, plurality voters can be stacked. This is much more powerful than simply flattening all the predictors and performing a plurality vote, as this effectively introduces a weighted scheme for the predictors. Rather than voters having a weight as specified by a coefficient, the relative position of a predictor in the tree reflects the weighting. A predictor which occurs higher in the tree (closer to the root) has a higher implicit weighting than one which is lower in the tree. The reason for this is simple, for the vote of a predictor to count, it must be the most common vote at every node on the the path to the root - the longer this path (the more steps to the root), the lower the weight. An example of this is given in Fig. 3.2.



(a) Weighted (tree) plurality voting



(b) Flattened plurality voting

Figure 3.2: A comparison showing the weighting strategy. We can see when flattened (b), Class 1 achieves the majority. However, with the tree structure (a.), Class 3 is chosen. The tiered system used in the voting scheme introduces an implicit weighting scheme, where Tier 1 (in this case the Random Forest and Support Vector Machine) have a higher weight than any of the other predictors.

3.2.4 Fitness Function

The fitness function defines how to evaluate individuals. In this work, the larger the fitness value, the better the individual (ensemble).

The evaluation measure is designed to adequately deal with both binary and multi-class classification tasks. The datasets are also not guaranteed to be evenly distributed between the two classes, so the measure should be relatively robust to uneven class distributions. Because of these considerations, the weighted F-measure (F1 score) is chosen as the fitness function. This is briefly outlined in the following section.

Weighted F-Measure

The formula for computing the weighted F-measure is given in Equation (3.1), where C represents the classes, and $|c|$ the number of instances in class c.

$$F1 = (\sum_{c \in C} |c| \times \frac{2 \times precision \times recall}{precision + recall}) / \sum_{c \in C} |c| \tag{3.1}$$

In the multiclass case, the computation for precision and recall are computed independently for each class. While in certain situations it can be more important to predict one class correctly (e.g. when predicting fatalities), in this thesis we assume equal importance's between classes and thus use the weighted F-measure. This metric could also be trivially substituted to any other sensible performance metric (such as balanced accuracy).

3.2.5 Tree Simplification

GP can often produce bloated trees. Here, since plurality voting is used, the most logical method of reducing bloat is to introduce pruning where an internal voting node is reduced to a leaf node if a majority of the children are made up by the same classifier (with the same parameters). For example, if there are five children, and three of them are logistic regression, the other two children will never be used regardless of their predictions. An example of a tree before and after pruning is given in Figure 3.3. Of course, there are many possible cases in which this can happen, i.e. when we must solve this for multiple stacked layers as shown in Figure 3.3, the pseudo-code for this is given in Algorithm 1. Pruning happens every 5 generations on the entire population (running too frequently would increase the training time, and could also oversimplify the trees for breeding), and then on the best resulting individual of the final generation.

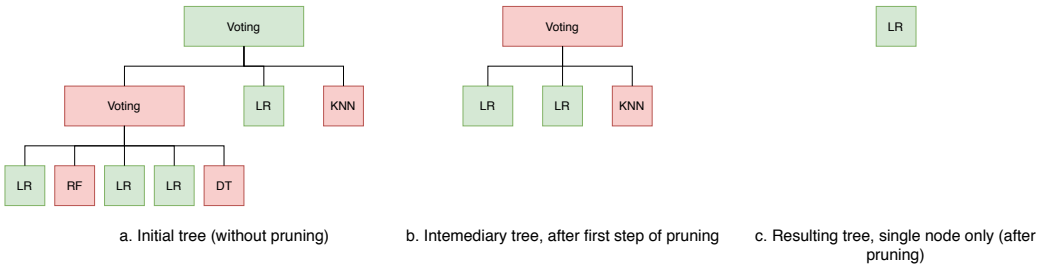


Figure 3.3: An example showing the effect of pruning on tree sizes. Red nodes show irrelevant nodes, as the green nodes form a majority, meaning the red nodes can be removed as they are having no effect on the predictions. The intermediary step is visualised in b, which shows that pruning can function through multiple redundant layers and that the result of step (a) can be further pruned.

3.2.6 Efficiency Improvements

GP can be a costly procedure, especially when trees are expensive to evaluate. In this case, to evaluate a tree, the various models in the tree must be trained, and predictions made for each instance, then finally the fitness must be computed for every tree in the population by comparing the predictions to the real class labels. To offset some of this cost, we introduce surrogate models, caching, concurrency and automatic disabling of slow learners.

Surrogate Models On large datasets (defined as those with a volume greater than 50,000), e.g. musk and hill-valley, only 20% of the training data is used for the evolutionary process. However, in the final generation, the entire training dataset is used. This allows us to compute approximations of good ensembles throughout the evolution, and only use the entire training dataset on the final population to speed up the overall procedure.

Caching To improve the training time and prevent needing to retrain the models a caching scheme was implemented. The model and model's

Algorithm 1: Pruning pseudo code

```
Function prune(tree) is
    if leaf node then
        return leaf
    end
    children = prune(subtree) for each subtree in tree
    if children has majority then
        return majority
    end
    return tree with updated children
end
```

inputs are hashed as a key, and the learnt model stored as a value. This means for a given input, the classification algorithms only need to be run at most once in the evolutionary process. As only the learnt model needs to be saved, we assume sufficient computational memory as the amount of memory required should not scale with the size of the datasets, with the exception of k-NN in which case caching may need to be disabled for "big data" tasks if memory is an issue. We use a least recently used cache, so we remove models which only appear in earlier generations.

Concurrency When evaluating trees, there is no need to do so sequentially as there is no interaction between trees. For further performance improvements, we evaluate trees in parallel using multiprocessing when possible. This may have the effect of missing certain nodes in the cache (if two of the same nodes are being executed in parallel) but we found this to be worth the trade-off compared to maintaining a perfectly shared cache, which has extra overhead such as locking.

Disabling Slow Learners Training classification methods can often be slow, and when this needs to be done repeatedly for various parameter

settings, mixed with the time taken for evolution, this can become unfeasible. For example, with SVMs with non-linear kernels, the complexity is approximately $O(N^2 \times M)$ where N is the training size, and M the number of features. When dealing with multiple classes, SVMs become even slower. For this reason, on large datasets (German, hill-valley, musk, and glass due to a large number of classes) SVMs are disabled for efficiency.

3.2.7 Guided Crossover

It is commonly believed that constructing ensembles which focus on diversity as well as the accuracy of base members can create better ensembles than focusing on accuracy alone ([46, 99, 7]). When using GP, selection of base members is performed automatically through crossover and mutation operations, however, we would like to see if this can be improved by explicitly focusing on crossing over diverse points in the trees.

In strongly typed GP, the typical crossover operator used (which we refer to as uniform crossover) is a single point crossover which takes two individuals and produces two new offspring by combining these individuals. How this works is a random type which is shared between the two individuals is selected, and from there a random node with this type is selected from each individual. As this is single point crossover, the individuals directly swap the subtrees (from the selected nodes), to produce two new children and since these two are of the same type (from step 1), no structure has been broken. An example is given in Figure 3.4 for typical GP.

This works well in the general case of GP, however, this does not directly consider diversity. For example, we may cross over two points from two separate trees, however, the two points may be functionally very similar (i.e. make the same predictions), resulting in no improvement. Ideally, we would explore diverse crossover points, as this could place additional pressure on the evolutionary process to select points which perform dif-

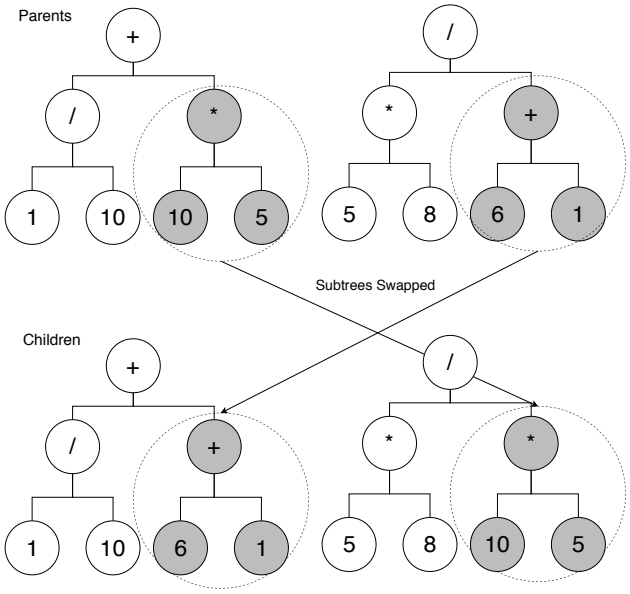


Figure 3.4: An example of singlepoint crossover.

ferently to the current node. With this in mind, a new "guided crossover" operation is tested, which does just this (which we refer to as diversity preserving crossover).

To achieve this, we use the diversity measure Pairwise Failure Crediting (PFC) proposed in [100]. In [100] PFC was used to compute a second objective for the evolutionary process, as the entire population was being used as the ensemble. However, here we propose to instead integrate this into the crossover operation, as individual trees are used as an ensemble instead. PFC gives a diversity measure between two points. To compute the PFC for two points a and b, a failure pattern (represented as a bit string of 1s for success and 0s for failures) is constructed for each possible crossover point, and the hamming distance is used for computing the distance between two points. This distance is then divided by the total sum of failures between the two individuals, as to prefer solutions which achieve higher accuracy. This is shown in Equation 3.2, where a_i means the predicted class from point a for instance i, and y_i means the real class

label for instance i.

$$pfc(a,b) = \frac{\sum_{i=1}^{N} \begin{cases} 1 & a_i \neq b_i \\ 0 & else \end{cases}}{\sum_{i=1}^{N} \sum_{p \in a,b} \begin{cases} 1 & p_i \neq y_i \\ 0 & else \end{cases}} \tag{3.2}$$

So the worst case is a PFC of 0, which means the two individuals fail on the exact same points (shown in Table 3.3), and the best case is a 1 which means they fail on completely separate points (shown in Table 3.2). All other cases fall between [0, 1]. In [100] when there were no errors (i.e. the summation used as the denominator was 0), a PFC of 0 was used, however, here we instead treat this as 1, as we do not want to penalise classifiers which achieve the highest accuracy (by giving them no chance of being crossed over).

Table 3.2: An example of the ideal case for PFC, when PFC=1. The black indicates a failure (misclassification), and we can see the two individuals fail on seperate parts of the data (diverse).

	Failure String									
Individual 1:						■	■	■		
Individual 2:		■	■							

Table 3.3: An example of the worst case for PFC, when PFC=0. The two individuals fail on the same data points (not diverse).

	Failure String									
Individual 1:		■	■							
Individual 2:		■	■							

For implementation, the selection criteria remain the same (in this case tournament selection). However, now when choosing crossover points

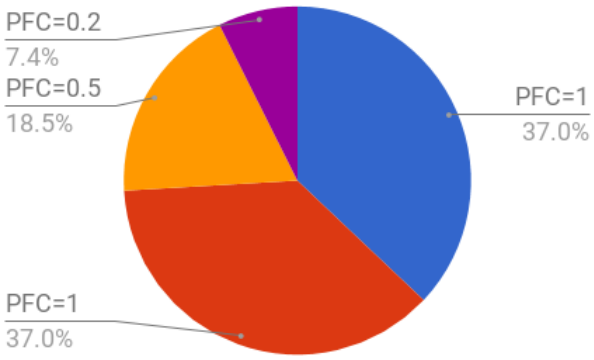


Figure 3.5: An example showing the weighted crossover probabilities. Here we have five potential crossover points, with PFC values of 1, 1, 0.5, 0 and 0.2. Note: since Point 4 has a PFC of 0, this does not appear on the chart.

from the two selected trees, if the type chosen is either an evaluation node or a voting node, the probability of selecting a crossover point (a, b) is weighted by the diversity of the point (i.e. the PFC value achieved) in relation all potential crossover points P. That is, the probability of choosing a crossover point (a, b) from a set of possible points P is given in Equation 3.3.

$$p(a, b) = \text{pfc}(a, b) / \sum_{j \in P} \text{pfc}(j) \tag{3.3}$$

So the larger the diversity (the higher the PFC), the higher probability of being selected, as shown in Figure 3.5. We can see if a particular point has a diversity of 0 (i.e. no diversity), this will have zero chance of being selected for crossover (although this can still be affected by mutation).

Other nodes in the tree (i.e. parameter nodes) are crossed over as usual. It's important to reemphasise here that a relatively large mutation rate of 0.2 was used, which means even though we favour the nodes/crossover points that perform well, random permutations can still happen to the tree to try and prevent getting stuck in local optima.

	Testing Accuracy		Training Time (m)	
Dataset	Uniform	DPC	Uniform	DPC
Iris	0.975 ± 0.033	0.975 ± 0.033	0.61 ± 0.28	6.35 ± 3.49
Balance	0.991 ± 0.009	0.990 ± 0.009	4.2 ± 1.49	4.13 ± 2.34
Glass	0.922 ± 0.071	0.926 ± 0.067	7.59 ± 7.61	65.08 ± 54.79
Wine	0.989 ± 0.012	0.989 ± 0.012	3.63 ± 6.48	15.05 ± 12.11
German	0.954 ± 0.050	0.950 ± 0.057	54.65 ± 38.21	233.77 ± 74.23
Wdbc	0.989 ± 0.013	0.989 ± 0.014	9.31 ± 3.06	12.05 ± 3.34
Sonar	0.937 ± 0.090	0.940 ± 0.090	18.79 ± 23.64	24.99 ± 35.29
Ionopshere	0.981 ± 0.027	0.979 ± 0.026	11.99 ± 10.28	10.67 ± 11.85

To evaluate the crossover, we test on several datasets from Section 3.4 and compare uniform and diversity preserving crossover. We exclude Hill Valley, Musk, and German due to the high computational cost associated with the custom crossover. The results are shown in Section 3.2.7. Somewhat interestingly, we find there was no statistical difference in the average testing accuracy between the proposed method with uniform crossover, and the proposed method with the new diversity preserving crossover.

On simple datasets (such as Iris or Wine), achieving equivalent results make sense as few errors will be made, meaning the weightings will follow a uniform (or near uniform) distribution (i.e. $p(i) = \frac{1}{|P|}, \forall i \in P$), resulting in equal chances for each potential crossover point, just as done with uniform crossover.

However, on harder datasets (i.e. Sonar, Balance, German), this should not happen as many points will have $p(i) \ll 1$. This shows that when utilising GP with standard crossover, the resulting ensembles will necessarily be diverse since we compute fitness using the entire ensemble. For example, if we require diverse classifiers to achieve an accurate ensemble, if we have similar classifiers, this ensemble will be given a low fitness and thus lower chance of survival throughout the generations (survival of the fittest).

For these reasons, and since standard crossover achieves drastically lower training time, we believe standard crossover with uniform weightings to be enough to generate diverse ensembles when paired with an appropriate fitness function. Therefore, in the following sections, we use the standard crossover only.

3.3 Experiment Design

3.3.1 Experiment Details

To evaluate the effectiveness of the proposed method, 10-fold cross-validation was used to compute the test accuracy on a range of datasets from the UCI repository. This was then tested against various comparison methods, using the same folds to allow for a fair comparison. Stochastic methods, such as the proposed method and neural networks, were run 30 times and the results averaged for each fold to minimise the effect of the random seed.

In this section, experiment configurations are presented, including comparison methods, datasets and parameter settings.

All of the GP code was implemented in Python3 using the DEAP library [101] as a base for the evolutionary process, scikit-learn [102] (sklearn) was used for the various base classification and comparison methods. The code was run on the Oracle Grid Engine.

3.3.2 Datasets

Ten datasets were chosen to offer a broad range of difficulty, with a varying number of instances, features and classes, and from a variety of domains. These datasets were chosen to ensure the proposed method is applicable to a wide range of applications. The datasets are summarised in Table 3.4.

Table 3.4: Summary of the Datasets used, all datasets were retrieved from the UCI repository. Datasets are ordered based on the number of features.

Dataset	# Features	# Instances	# Classes	Domain
Iris	4	150	3	Life
Balance	4	625	3	Social
Glass	10	214	7	Physical
Wine	13	178	3	Physical
German	20	1000	2	Financial
Wisconsin Breast Cancer (WDBC)	32	569	2	Life
Ionosphere	34	351	2	Physical
Sonar	60	208	2	Physical
Hill Valley	101	606	2	Physical
Musk1	168	476	2	Physical

3.3.3 Significance Tests

To test whether there was a significant difference in the results, firstly an unpaired Welch t-test [103] was used with a significance level of 5% ($\alpha = 0.05$). This was chosen as we can not assume equal variance between methods.

To achieve more general claims, and reduce the impact of Type-I errors on the results, we also perform Friedman testing from [104] to compare differences across the classifiers. That is, h_0=All classifiers perform the same and the differences are only from randomness. Next conclusions about the differences are drawn by following up with Holm post-hoc analysis. This follows the suggestions in [105] for comparing multiple classifiers. The average rankings used in the computation are given in Table 3.7.

3.3.4 Comparison Methods

Base members

For comparison, all base members were tested. It is expected that the method would perform at least as good as the best base member, and ideally better than this. The base members are: Support vector machines (SVMs) [106], Logistic Regression, K-Nearest Neighbours (k-NN), Naive Bayes, Decision Trees (DTs), Random Forests [52] , AdaBoost [54] and Gradient Boosted Machines [55].

Other Ensemble Methods

While Random Forests, AdaBoost and Gradient Boosted Machines are all ensemble methods, since they were used as base-learners, it is not fair to only compare to these three ensembles. Instead, three new ensemble methods were added for comparison.

Voting Classifier. To verify the proposed method is better than simply selecting a plurality vote from the base members, this was included as a comparison method. Achieving better results than the voting classifier shows the proposed method is able to find more useful combinations than performing no selection and utilising all members.

Bagged Neural Network. Neural Networks are perhaps one of the most widely used techniques in machine learning, due to their high reported performance across many domains [107]. The key idea is to try and approximate a mapping $x \mapsto y$, by learning a set of weights for a pre-specified architecture which minimises a loss function. One problem with neural networks is they can be sensitive to the initial weights used, which can be solved by starting at several different points. This is done here with bagging, where several networks are trained each starting with a different set

Table 3.5: Summary of parameter settings used for proposed method.

Parameter	Value
Generations	50
Population Size	100
Selection	Tournament (Size=7)
Creation	Ramped half-and-half
Max Height	17
Crossover Rate	0.7
Reproduction Rate	0.1
Mutation Rate	0.2

of weights, on different bagged subsets of the data, and the networks then vote for a class.

Extremely Randomised Trees. Extremely randomised trees are an extension of random forests, with additional randomness introduced in the splitting criteria. This method was proposed in [108] and was shown to outperform random forest on a range of tasks trialled.

3.3.5 Parameter Settings

For all base members, the default values as specified by sklearn were utilised. For the proposed method, common parameter settings as used in literature were chosen. These are summarised in Table 3.5. A low population size of 100 was used to save computational cost, a larger population size is likely to result in improved performance at the expense of increased training time.

3.4 Results and Discussion

3.4.1 Overall Results

Table 3.6: Comparison results on the UCI datasets. Results are presented as mean ± standard deviation. (=) indicates no statistical difference between the row and the proposed method, (-) indicates the row did significantly worse, and (+) indicates the row did significantly better than the proposed method (however this did not happen on any datasets trialled).

	Iris	Ionosphere	Sonar	Balance	Glass
Proposed Method	**0.975±0.032**	**0.981±0.027**	**0.937±0.089**	**0.991±0.009**	**0.922±0.071**
SVM	0.973±0.044 (=)	0.923±0.051(-)	0.557±0.100(-)	0.864±0.057(-)	0.625±0.104(-)
Logistic Regression	0.961±0.032 (=)	0.855±0.056(-)	0.757±0.055(-)	0.834±0.047(-)	0.568±0.111(-)
k-NN	0.967±0.045 (=)	0.831±0.071(-)	0.815±0.086(-)	0.818±0.069(-)	0.661±0.105(-)
Naive Bayes	0.952±0.068 (=)	0.888±0.061(-)	0.650±0.132(-)	0.873±0.056(-)	0.435±0.150(-)
Decision Tree	0.945±0.048 (=)	0.891±0.038(-)	0.716±0.096(-)	0.793±0.077(-)	0.677±0.047(-)
Random Forest	0.947±0.046 (=)	0.927±0.037(-)	0.771±0.064(-)	0.818±0.068(-)	0.738±0.038(-)
Adaboost	0.947±0.051 (=)	0.922±0.051(-)	0.805±0.100(-)	0.927±0.026(-)	0.350±0.107(-)
Gradient Boosting	0.953±0.043 (=)	0.933±0.042(-)	0.814±0.077(-)	0.860±0.068(-)	0.743±0.045(-)
Bagged NN	0.975±0.030 (=)	0.915±0.044(-)	0.796±0.072(-)	0.931±0.040(-)	0.193±0.033(-)
Extra Trees	0.950±0.044 (=)	0.934±0.027(-)	0.813±0.047(-)	0.817±0.066(-)	0.740±0.054(-)
Voting Classifier	0.948±0.049 (=)	0.937±0.040(-)	0.807±0.084(-)	0.854±0.062(-)	0.700±0.076(-)

	Wdbc	Wine	German	Hill-Valley	Musk
Proposed Method	**0.989±0.013**	**0.989±0.012**	**0.954±0.050**	**0.974±0.014**	**0.972±0.031**
SVM	0.487±0.067(-)	0.326±0.123(-)	0.649±0.069(-)	0.444±0.059(-)	0.412±0.074(-)
Logistic Regression	0.956±0.024(-)	0.955±0.048(=)	0.743±0.053(-)	0.968±0.013(=)	0.857±0.037(-)
k-NN	0.934±0.041(-)	0.690±0.120(-)	0.669±0.041(-)	0.542±0.041(-)	0.878±0.041(-)
Naive Bayes	0.941±0.024(-)	0.972±0.028(=)	0.730±0.060(-)	0.449±0.039(-)	0.720±0.058(-)
Decision Tree	0.920±0.034(-)	0.929±0.055(-)	0.708±0.034(-)	0.554±0.026(-)	0.781±0.056(-)
Random Forest	0.955±0.022(-)	0.973±0.030(=)	0.712±0.042(-)	0.559±0.023(-)	0.859±0.031(-)
Adaboost	0.958±0.022(-)	0.916±0.053(-)	0.744±0.039(-)	0.504±0.031(-)	0.886±0.032(-)
Gradient Boosting	0.963±0.025(-)	0.954±0.042(-)	0.754±0.046(-)	0.548±0.029(-)	0.896±0.036(-)
Bagged NN	0.888±0.043(-)	0.279±0.024(-)	0.758±0.044(-)	0.559±0.015(-)	0.873±0.034(-)
Extra Trees	0.960±0.019(-)	0.970±0.035(-)	0.715±0.040(-)	0.571±0.027(-)	0.897±0.025(-)
Voting Classifier	0.967±0.019(-)	0.970±0.023(=)	0.755±0.048(-)	0.631±0.033(-)	0.901±0.036(-)

Table 3.6 gives a summary of the results across all datasets trialled, on

every dataset, the proposed method performs much better than (or equivalent to) all comparison methods. The proposed method is shown in bold as the first row, the base members are shown below this (separated by a line), and the comparison ensemble methods (i.e. those not used as base members) are in the final segment.

On very simple datasets such as Iris, we see equivalent performance to the basic methods. This is because all methods do extremely well on this task, due to the simple nature, and there is little room for improvement. Analysing some of the evolved programs from the Iris task, we can see often we just select a single base member, as shown in Figure 3.6a.

On more complex datasets, such as Glass, German, and Musk, we see extreme improvement (p-values of essentially 0) over all other methods trialled. For example with the German dataset, the comparison ensemble methods (Extremely randomised trees and the voting classifier) achieve f1-scores < 0.75, whereas the proposed method is able to achieve 0.92. One interesting point here is the bagged neural network performs very poorly (achieving only 0.19 f1-score), which could be due to the problem which we aim to solve, that hyperparameters can be difficult to choose, and the default settings often perform poorly. Similar results were seen with the Wine data set and bagged neural networks, but on all other tasks, the neural network performed relatively well.

3.4.2 Statistical Tests

The proposed method is compared with several comparison methods for each dataset. To draw conclusions about the effectiveness of the proposed method in general, we perform the Friedman test [104] to compare differences across the classifiers. The average rankings used in the computation are given in Table 3.7.

At $\alpha = 0.05$, $p << \alpha$, which means h_0 is rejected and one can conclude there is a significant difference in the classifiers that is not attributed to ran-

Table 3.7: Results from the Friedman test and Holm post-hoc analysis. The higher the ranking, the better.

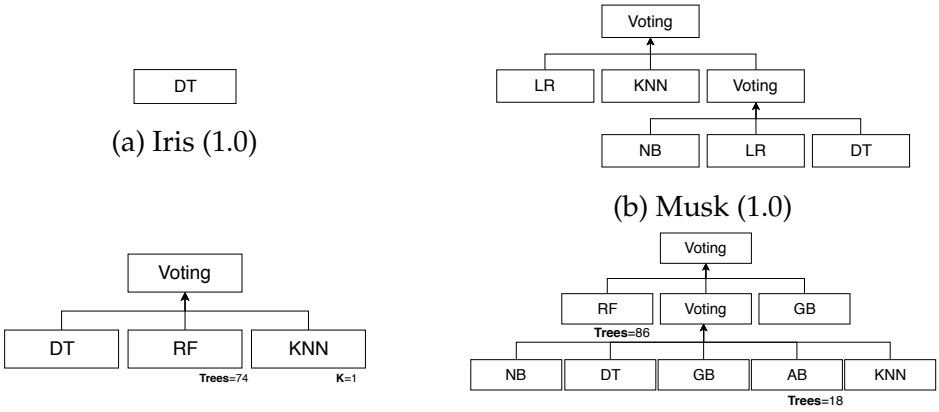
Method	**Friedman Rank**	**Adjusted P (3dp)**
Proposed	**7.41**	**(Control)**
Voting Classifer	5.43	0.047
Gradient Boosting	5.15	0.047
Extremely Randomised Tree	4.81	0.027
Bagged Neural Network	3.85	0.002
Random Forest	3.81	0.002
Logistic Regression	3.66	0.001
AdaBoost	3.63	0.001
k Nearest Neighbours	3.13	$<$0.001
Naive Bayes	2.98	$<$0.001
Support Vector Machines	2.29	$<$0.001
Decision Trees	2.23	$<$0.001

domness alone. To compare this difference between the classifiers, we perform the Holms step-down procedure [109] using the proposed method as the control group. For all classifiers, the resulting $p < \alpha$, so one can conclude the proposed method outperforms all other methods trialled. The resulting p values are given in Table 3.7.

3.5 Further Analysis

To see how the proposed method achieves such good results, an example program is analysed from each of these difficult tasks. We choose relatively simple ensembles for clarity and discussion, however, some other evolved ensembles are much more complex (i.e. deeper) than those shown.

A top performing Musk ensemble is shown in Figure 3.6b. Interest-



(a) Iris (1.0)

(b) Musk (1.0)

(c) German (0.99)

(d) Glass (1.0)

Figure 3.6: Examples of evolved ensembles, validation f1-score (fitness) presented in brackets.

ingly, Logistic Regression appears twice in the tree despite only being ranked 4th on this dataset. However, when combined with other classifiers, the results are far better than the more accurate classifiers alone. This shows that only choosing the most accurate classifiers as base members is not always an ideal choice.

An evolved German ensemble is shown in Figure 3.6c, which shows three methods were chosen here, Random Forests, Decision Trees, and k-NN. Both Random Forests and k-NN had parameter selection performed automatically, whereas the decision tree used the default values. For k-NN, a k value of 1 was selected, meaning we just use the class of the closest neighbour for classification. In the usual case, using this predicted class would be negatively affected by outliers in the data, however, for the ensemble to predict this class, at least one of the other two methods must also predict this class. For the random forest, a relatively large number of trees (74) is used. It is interesting to see that even though k-NN performed the worst out of all the methods on its own, it was able to combine with the two tree-based methods to achieve results far higher than any of the individual methods.

Figure 3.6d shows an evolved Glass ensemble. This features two layers of voting, and a wide variety of base learners used. Looking at the results for this dataset (Table 3.6), the tree-based methods (particularly gradient boosting) tend to achieve the highest score, and these are used throughout the evolved ensemble. They are also supplemented with other methods which perform poorly in isolation (such as Naive Bayes, and AdaBoost even though AdaBoost is tree based), however, when combined with other methods make for a diverse and accurate ensemble.

It is noticed interesting results with Hill-Valley, logistic regression did remarkably well, achieving an f1-score of 0.968, which is significantly higher than even the other ensemble methods such as Random Forests or Gradient Boosting. The proposed method still performs equivalently (or slightly better), and in all cases uses Logistic Regression in the best performing ensemble, either as a standalone node, or part of a more complex ensemble. In [110], they determined logistic regression was likely to outperform tree induction in cases with "smaller training-set sizes and where the classes cannot be separated well", which is especially true when it is "too difficult to distinguish from the noise to allow identification of the correct relationship". Although this particular dataset was not used in [110], we do note that this dataset has had noise added to make the task more complex, so this follows the conclusions drawn in that work and could help explain why logistic regression outperforms the other methods by such a large margin.

3.6 Chapter Conclusions

In this chapter, a novel GP-based method was proposed for automated ensemble learning which utilises GP as the meta-learner in stacked generalisation. The proposed method was able to outperform a range of ensemble methods and base learners, on a range of tasks ranging from simple to complex. The method was shown to generalise well to a variety

of tasks from different domains and outperform all comparison methods using Friedman statistical significance tests with Holms post-hoc analysis. Not only was the method able to achieve the best results, but it also removes the burden of manual model selection and parameter tuning. This work set the foundation for many future developments, such as a fully automated ensemble learning pipeline, and shows the promise of GP for automated machine learning.

Here, all classification algorithms used the raw/original training data as input. Often, in real-world situations, a pipeline is needed. We would like to incorporate a pipeline into the method, so preprocessing steps such as imputing missing data, scaling data, and feature selection could be incorporated below the classification nodes in the tree. This has been looked at with GP in [111], however, the focus was not explicitly on ensemble learning. In the next chapter, we will further develop this idea.

Chapter 4

Efficient Ensemble-based Automated Machine Learning

With the huge rise in demand for machine learning applications, the supply of machine learning talent is unable to keep up. Like in many sectors seen before, this motivated the push towards automation, and the field of automated machine learning (AutoML) has emerged. AutoML can be seen as using machine learning to learn machine learning tasks, without requiring the end user to have extensive (or any) ML knowledge. It is important to mention the goal of AutoML is not to replace ML engineers/data scientists, but rather to assist them in their jobs. For example, irrigation sprinklers did not replace farmers, it merely freed them from the burden of watering crops. Likewise, AutoML automates the process of preprocessing, parameter tuning and model selection. However, other tasks such as analysis, presentation, and decision making are left to the ML expert(s).

4.1 Chapter Introduction

In this chapter, we propose a novel ensemble-based approach to AutoML, which can help overcome some limitations of current state-of-the-art automated classification algorithms. The importance of AutoML is apparent,

and in recent years, several AutoML systems have been proposed from a variety of optimisation backgrounds [71, 68, 79]. However, a major limitation of all existing methods is the large computational cost associated with them, and the inability to parallelise the entire process. One of the main factors affecting the adoption of AutoML is the prohibitive cost/time of running an AutoML system. Therefore, the specific objectives of this work are to:

- Develop a novel, efficient, and parallelisable method of AutoML,
- Investigate the usefulness of ensembling for improving results in AutoML, and
- Provide a comparison and analysis of the current state of AutoML.

We focus particularly on classification tasks in this thesis but note that most methods can work with regression tasks either directly or with minimal changes.

4.2 The Proposed Method

In this section, a novel ensemble method based on random search is proposed (named AutoVoter). Despite the sound work on the usefulness of random search for hyperparameter optimisation in [69], little has been done in this area for AutoML. Perhaps because in [71], the Bayesian approach was found to outperform a basic random search, and [72] found the variation in the random pipelines was higher, and the pipelines grew too large. In this chapter, we look to re-emphasise that random search can and should be considered as a suitable alternative to both Bayesian and evolutionary approaches for AutoML due to the ability to horizontally scale. Placing constraints on the architecture prevent the pipelines from growing too large (addressing the concerns in [72]), and ensembling

improves the performance and can help reduce the variation in the results (addressing both [71] and [72]).

4.2.1 Architecture

With AutoML, one key consideration is the large size of the search space. Using ensembles exponentially inflates this search space, as now we must consider all combinations of possible pipelines (if there are n individual pipelines, there are 2^n combinations of these pipelines). In order to combat the growth of this search space, the pipelines are limited to be composed of at most three components, a data processor, a feature processor and an estimator. This is, of course, a simplification of potential machine learning pipelines, however, we show this to work well on a range of tasks in Section 4.3. To avoid confusion, we call these "pipeline stumps" (shown in Fig. 4.1), these pipeline stumps can then be combined in a tree structure to form a final pipeline (i.e. a pipeline can be composed of several stumps).

Figure 4.1: The structure of a pipeline stump. A stump is composed of at most three items. Dotted rectangles indicate an optional component. For example, a stump could be only an estimator, a data processor and an estimator, a feature processor and an estimator, or all three components.

In order to ensemble/combine pipeline stumps, we proposed a novel stacked voting approach for AutoML, where pipelines are presented as tree structures, where stumps and voting nodes are searched simultaneously. A pipeline must be composed of at least one pipeline stump but can be composed of many by combining pipeline stumps with plurality voting. Voting nodes can also be stacked, allowing for more sophisticated ensembles than a standard 1-layer majority voting scheme (as well as automatically selecting the appropriate base members). An example of a

Pipeline is given in Fig. 4.2. For each of the nodes, hyperparameters are also selected.

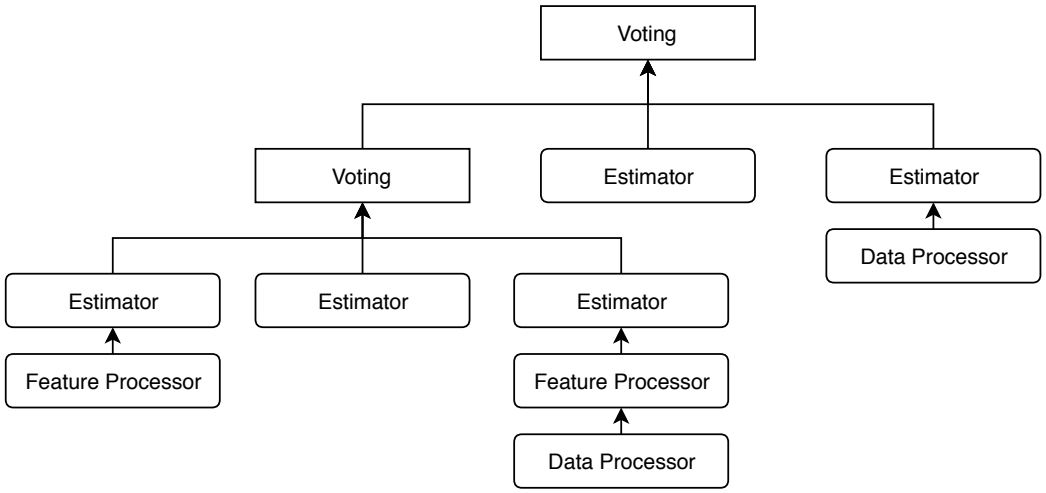


Figure 4.2: An example pipeline. Voting combines three pipeline stumps. The input to a voting node can also be another voting node (as shown), so we can stack voters. Each of the nodes can also have a set of optional hyperparameters (not pictured).

Components

There are four main components which build up the structure of AutoVoter. The estimators, the feature processors, the data processors and the voters. In this case, an estimator is a classification algorithm. The classification algorithm can also take a range of custom hyperparameters, or use the default values as specified by sklearn. Table 4.1 summarises the estimators included. Feature processors are limited to dimensionality reduction methods, such as feature selection and principal components analysis. Feature processors could also be trivially extended to include feature construction methods. The methods included are summarised in Table 4.1. Again, each feature processor has a set of hyperparameters associated with it. Data processing operates on the raw data, and here this involves vari-

ous methods for scaling the data. A voting node can take 3 or 5 estimators (or other voting nodes) as inputs and returns the plurality (mode) vote from the inputs. If no plurality is found (or a tie), the first class predicted is returned.

In cases where there is missing data, an imputation step is forced prior to the running the search which replaces any missing values with the most common value for that feature.

Table 4.1: Summary of the components included in AutoVoter

	Algorithm	**Parameters**
Classifiers	Naive Bayes, Decision Tree	*NA*
	Linear SVM, Logistic Regression	C: np.logspace(-3, 2, 6) Penalty: L1, L2
	K-Nearest Neighbour	K: 1 - 50
	Random Forest, Adaboost	Num Estimators: 10 - 150
	XGBoost	Num Estimators: 10 - 150 Booster: gbtree, gblinear, dart Depth: 2-8 LR: [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.5]
Feature Processors	PCA	Percentage of components: 5-100%
	Mutual Info, Chi-Squared, ANOVA F-value	Percentage of features: 5-100%
	Decision Tree, Extra Trees, Random Forest	Threshold: 0.5, 0.75, 1.0, 1.25, 2, 2.5
	LinearSVC, Logistic Regression (with L1)	C: np.logspace(-3, 2, 6)
Data Processors	Standard Scaler, Robust Scaler, 0..1 Scaler	*NA*
Voting	Voting Classifier	3 or 5 Classifiers/Voters

4.2.2 Ensembling

While it has already been shown that guided approaches to pipeline search can outperform random searches in certain cases, this does not necessarily hold true with ensembling. For the guided search spaces, considering

ensembles enlarges the search space too drastically to find good solutions (as the exploitation becomes somewhat ineffective). In contrast, we show a random search can sample effectively from this solution space to provide equivalent results in a fraction of the time. It is surprising how little emphasis has been placed on emsembling in AutoML systems, considering the wealth of support in both theory and practice for ensembles [42, 112].

The support for ensembles is limited in the current state-of-the-art methods to AutoML, falling much behind the advancements in ensemble learning for classification. Without placing a focus on ensembling pipelines, it will be difficult to construct any state-of-the-art performing models (such as those that win Kaggle competitions). In TPOT, ensembling can only appear as cascading, where a synthetic feature is appending to the feature set if a classifier appears anywhere besides the final step in the pipeline. In auto-sklearn, pipelines are greedily constructed through a separate optimisation process, where the Bayesian optimisation does not optimise the ensembles. In AutoWEKA, ensembles are limited to having at most 5 classifiers (a number far smaller than typical classification ensemble sizes). Here we place an explicit focus on stacked generalisation [49] for constructing ensembles, which we find to outperform the simpler cascading approach of TPOT, the greedy approach of auto-sklearn, and the restricted approach in AutoWEKA. The proposed approach overcomes these weaknesses as it is not limited to any fixed/predefined number of base members, and allows search over the base members and potential ensembles simultaneously.

Evaluation

The tree construction process is run for the specified amount of time, resulting in a set of trees X. In order to select the best resulting tree x from X, an internal 10-fold cross-fold validation is used on the training set, where we are trying to maximise the weighted f1-measure (although any metric can be specified). The individual x which achieves the highest f1 score is

selected.

4.2.3 Search Algorithm

In this chapter, we utilised random search for sampling tree-based pipelines. One obvious question is why did we not use GP for this as was done in the previous chapter? There are two main reasons for this.

Firstly, we wanted to show that with the introduction of ensembles a randomised approach can be competitive to the current state-of-the-art guided searches (including those already based on GP). The reason for this is the parallel nature of the random search, which becomes an important concern for automated machine learning due to the costly procedure (constructing and evaluating thousands of machine learning pipelines).

Secondly, the huge search space of the ensembled pipelines may limit the effectiveness of the exploitation of GP, meaning the ability to parallelise the entire search is lost for little extra gain. This comes down to the classical discussion of exploration vs exploitation in search, and how evolutionary methods should achieve these traits [113, 114]. Achieving an ideal balance is difficult [115], and with a huge search space, exploration is particularly important to prevent premature convergence in a local optima [116].

To achieve this exploration, mutation is required. However, a mutation rate which is high begins to behave like a random search, yet has the downside that it is still sequential by nature. Contrarily, a mutation rate which is too low will not effectively explore the search space and may result in poor local optima. It should also be noted TPOT has a high mutation rate of 0.9 by default, which is remarkably high compared to some other common suggestions for GP (more in line with 0.1).

It can be argued that crossover also explores the search space [117], but with GP since the children are the result of combining two parent trees, the children are unlikely to be drastically different (a lack of diversity

from crossover in GP showed in [118]). Instead, crossover performs more like exploitation (this is in contrast to GAs or fixed length representations where the crossover is more like exploration [119]), particularly when the space of possible trees is huge. Eshelman [120] state "increased exploitation by selection leads to decreased exploration by crossover" [114], as such GP placing this additional stress on crossover will have a negative effect on the exploration ability. With such a large search space, this may negatively impact the resulting accuracy (or at the very least not improve the result) due to focusing on locally optimal trees, i.e., in 90% of the GP runs in [121] a single tree was a root ancestor of every tree in the final population.

An in-depth comparison of exploration/exploitation in GP vs random search could entail an entire thesis on its own, so, for now, we leave this as hypothetical only, and show that random search with ensemble learning is competitive to the existing guided approaches. Utilising GP as the search mechanism is unlikely to perform worse than random search, but has the downside that the entire process cannot be parallelised (only each generation). We will leave this further investigation for future research.

4.3 Expirement Design

4.3.1 Expirement Details

We notice in many of the existing AutoML works, the comparison is done on a per dataset basis, and conclusions drawn based on this. This is not an ideal way of comparing AutoML systems, as we are much more interested in the general performance of the method, not the results on individual datasets (not to mention the inflated type-1 errors if significance tests are performed per dataset). Instead, it would be ideal to compare the methods generally over a range of tasks. Therefore, an in-depth comparison is performed using the Friedman test on AutoVoter and the 4 comparison

methods introduced in Chapter 2.

10-fold cross-validation was used to produce an average test accuracy for each of the datasets. For each fold k, the AutoML methods were fed the remaining $k-1$ folds as training data, and the left out fold as test data. No systems saw the leave out fold until after a final pipeline was learnt. When datasets had a dedicated train: test split, these were combined to facilitate cross-fold validation. 10-fold validation was used for two reasons: to give a large enough number of runs that the stochastic nature of the methods does not affect the results, and to get an average result across disjoint sets of test data. It would have been preferable to run say 10x10-fold cross-validation to get a larger sample, however, generally speaking, AutoML is a costly procedure, so this was not a viable option (this would require 1,000 days of CPU time in total).

4.3.2 Datasets

17 datasets from the UCI repository [122] were chosen, these were the datasets identified as "hard" (i.e. an AUC and F-measure ≤ 0.8) in [123] after an extensive trial of 54 different classifier and feature selection combinations, on 129 total datasets. Pittsburgh Bridges was excluded as this is not strictly speaking a classification dataset. Primary Tutor was also excluded as this was problematic for one comparison method, generating invalid results. These datasets were chosen since simple datasets, such as Iris, do not show the benefits of AutoML as many methods are able to achieve high performance somewhat trivially. The datasets feature a mixture of complete and missing data, continuous and categorical features, and a varying number of features, classes and instances. The datasets are summarised in Table 4.2.

The only preprocessing done to these datasets is to remove any unique ID columns, and replace missing values with NaNs. If a particular method could not deal with categorical features, these were treated as suggested

Table 4.2: Summary table of all 15 datasets used for comparison. Ordered in terms of descending difficulty (from [123]).

Name	Missing	Feature Types	Instances	Features	Classes
Post-Operative	Yes	Categorical, Numeric	90	8	3
Planning	No	Numeric	182	13	2
Dresses Sales	Yes	Categorical	501	13	2
Statlog Australia	Yes	Categorical, Numeric	690	14	2
Heart Disease Switzerland	Yes	Numeric	123	13	5
Heart Disease VA	Yes	Numeric	200	13	5
Congressional Voting	Yes	Categorical	435	16	2
Thoracic Surgery	No	Categorical, Numeric	470	17	2
Haberman Survival	No	Numeric	306	3	2
Contraceptive Method	No	Categorical, Numeric	1473	9	3
Breast Cancer	Yes	Categorical	286	9	2
Titanic	No	Categorical, Numeric	1309	9	2
SPECT	No	Categorical	267	22	2
Breast Cancer Wisc (Prog)	Yes	Numeric	198	34	2
Statlog German	No	Categorical	1000	20	2

by the original developers (for TPOT that is one-hot-encoding, and for auto-sklearn converting to integers and specifying the categorical columns as a parameter). We note that in one case oversampling was required, this was to ensure RECIPE worked as expected on post-operative.

4.3.3 Significance Tests

To compare AutoML methods, the average result for each dataset for each classifier was used to perform the Friedman test, with Nemenyi post-hoc analysis to provide an in-depth comparison between the methods in general, as suggested in [105].

4.3.4 Comparison Methods

All methods outlined in Chapter 2 were compared, along with AutoVoter. All methods were allowed to run for 3 hours, with the exception of AutoVoter, which was only run for 30 minutes ($< 1/5$th of the time). For RECIPE, this was run for approximately 3 hours as the current release did not have support for overall runtime. All methods were set to optimise the weighted f1-measure. The goal was to determine whether AutoVoter could achieve competitive results in a fraction of the time. One consideration is if the extra time could cause overfitting. However, with AutoML, generally speaking, the longer the time allocated, the better the results will be. This is due to the fact all methods have internal measures in place to prevent overfitting (such as complexity control and cross-fold validation), as well as 3 hours being a relatively low amount of time for AutoML methods, so the risk of overfitting by running the methods for longer is low.

The methods were the most recent stable releases at the time of conducting this research, which were: TPOT (0.9.3), auto-sklearn (0.4), Auto-WEKA (2.6) and RECIPE (unversioned, but most recent commit: #44af4d1). To minimise the effect of parallelism from the different systems (for example Java vs Python based), we only run on one core. We note that all methods can benefit from parallelism, but ours particularly, since the GP and Bayesian methods are sequential by nature.

It should also be pointed out that AutoVoter, TPOT, RECIPE and Auto-Sklearn all optimise over similar base members (feature selectors, estimators etc) (those available in scikit-learn), whereas AutoWEKA uses the base members available in WEKA (WEKA essentially offers a superset of the methods in sklearn). AutoVoter includes no additional methods to any of the comparisons, to allow for a fairer comparison rather than optimising over different/improved base members. Methods such as Auto-Keras were not compared, as these offer completely different base members (deep learning methods).

4.4 Results and Discussion

4.4.1 Overall Results

The results on all 15 datasets are summarised in Table 4.3. The proposed method is the column "AutoVoter". We can see in 8/15 cases, AutoVoter achieved the highest accuracy, despite only being allocated $\approx 16.6\%$ of the running time when compared to the other methods (30 minutes vs 3 hours). It is also interesting to note that based on the reported results here, 4 of the datasets (congress, titanic, spect, and statlog) would no longer be considered hard by the definition (an AUC and F-measure ≤ 0.8) in [123]. Each method achieved the top result on at least one dataset, with TPOT achieving the highest result on $6/15$, auto-sklearn the best on $2/16$ and RECIPE and AutoWEKA both the best on $1/15$ datasets. We also note this addresses the concerns in [72], that random search had too large of a variation in the results. Here we can see the standard deviation is on par with the comparison methods, and in fact smaller in some cases, due to the ensembling providing more robust predictions.

4.4.2 Statistical Tests

The resulting p-values from the Nemenyi significance testing are given in Table 4.5, the corresponding rankings (the higher the better) were: AutoVoter: 7.39, TPOT: 5.89, auto-sklearn: 5.20, AutoWEKA: 4.62, RECIPE: 2.89 (also presented in Table 4.4).

We can see in general, there is no drastic difference in the methods in terms of resulting f1-scores (indicated by the majority of the squares being red). The exception being the proposed method outperforms both RECIPE and AutoWEKA in general, and TPOT outperforming RECIPE in general. This further supports our claim that random search can be considered as a contender for AutoML, especially when an emphasis is placed on constructing good ensembles.

Table 4.3: Average testing results from all 15 datasets. Results presented as mean ± standard deviation.

	AutoVoter	TPOT	RECIPE	AutoWEKA	AutoSklearn
Post-Operative	**0.582 ± 0.193**	0.482 ± 0.169	0.540 ± 0.253	0.568 ± 0.207	0.535 ± 0.194
Planning	**0.651 ± 0.114**	**0.651 ± 0.162**	0.531 ± 0.276	0.613 ± 0.161	0.602 ± 0.185
Dresses Sales	0.573 ± 0.094	0.614 ± 0.082	0.376 ± 0.127	**0.627 ± 0.078**	0.572 ± 0.117
Statlog Australia	0.868 ± 0.037	0.850 ± 0.038	0.848 ± 0.045	0.860 ± 0.038	**0.870 ± 0.054**
Heart Disease Switzerland	**0.342 ± 0.136**	0.252 ± 0.146	0.182 ± 0.082	0.239 ± 0.200	0.258 ± 0.098
Heart Disease VA	0.341 ± 0.129	**0.351 ± 0.135**	0.199 ± 0.094	0.243 ± 0.100	0.232 ± 0.126
Congressional Voting	**0.961 ± 0.027**	**0.961 ± 0.024**	0.314 ± 0.151	0.958 ± 0.030	0.954 ± 0.027
Thoracic Surgery	0.776 ± 0.086	**0.802 ± 0.077**	0.719 ± 0.104	0.759 ± 0.084	0.762 ± 0.068
Haberman Survival	0.697 ± 0.104	0.638 ± 0.186	**0.741 ± 0.129**	0.691 ± 0.084	0.719 ± 0.107
Contraceptive Method	0.562 ± 0.030	**0.564 ± 0.027**	0.542 ± 0.027	0.538 ± 0.041	0.537 ± 0.022
Breast Cancer	0.684 ± 0.074	0.676 ± 0.067	0.422 ± 0.271	0.699 ± 0.061	**0.725 ± 0.062**
Titanic	**0.823 ± 0.032**	0.822 ± 0.040	0.756 ± 0.061	0.803 ± 0.031	0.819 ± 0.035
SPECT	**0.819 ± 0.052**	0.802 ± 0.045	0.778 ± 0.045	0.801 ± 0.070	0.816 ± 0.048
Breast Cancer Wisc (Prog)	**0.788 ± 0.130**	0.729 ± 0.166	0.711 ± 0.234	0.723 ± 0.171	0.730 ± 0.123
Statlog German	**0.760 ± 0.041**	0.736 ± 0.056	0.733 ± 0.029	0.697 ± 0.040	0.724 ± 0.046

Table 4.4: Resulting rank from the Friedman test. The higher the better.

Method	Friedman Rank
AutoVoter	**7.39**
TPOT	5.89
AutoSklearn	5.20
AutoWEKA	4.62
RECIPE	2.89

4.5 Further Analysis and Recommendations

By looking at the results, we are able to draw some general conclusions. There are no major differences in terms of resulting accuracy, so we attempt to make the selection of an appropriate AutoML method easier by

Table 4.5: Resulting p-values from Nemenyi post-hoc analysis. Grey indicates a comparison to self, so no value is present. Red indicates no statistically significant difference. Light green indicates statistical significance at $\alpha = 0.05$, and dark green significance at $\alpha = 0.01$.

	AutoVoter	TPOT	RECIPE	AutoWEKA	AutoSklearn
AutoVoter		0.553	0.001	0.044	0.182
TPOT	0.553		0.023	0.684	0.900
RECIPE	0.001	0.023		0.416	0.142
AutoWEKA	0.044	0.684	0.416		0.900
AutoSklearn	0.182	0.900	0.142	0.900	

highlighting the benefits of each method in this section.

AutoVoter is fast and can be trivially extended to be massively parallelisable, as random search is "embarrassingly parallel". With the rise in cheap cloud compute power, we foresee this becoming an important aspect for AutoML. Despite being fast, the results are still equivalent to or better than the comparison methods in general.

TPOT and AutoWEKA are the easiest for an end user to install and begin using, due to the support for a range of systems natively (i.e. MacOs, Windows, and Linux), and easy access (TPOT fully installable via pip, and AutoWEKA prepackaged as a .jar). Auto-Sklearn only supports Linux systems. RECIPE is runnable assuming Python2 and a C compiler. Once installed, all methods are relatively trivial to use.

TPOT, AutoWEKA and RECIPE can all export the generated pipelines directly to code. We foresee this being an important ability in AutoML systems, as the resulting model should be able to be further adjusted by an expert if desired. Both auto-sklearn and AutoVoter do not currently offer this option, as the ensembling functionality makes these less interpretable (although AutoVoter can be extended to do this, and resulting trees can be visualised already).

AutoWEKA is the only method which features a Graphical User Inter-

face (GUI). As one of the goals of AutoML is to make ML more accessible, having a GUI could be very beneficial, particularly for users without programming knowledge.

RECIPE allows for a user to customise the search space by specifying a grammar file. This is powerful for advanced users and gives a good deal of control over the search space.

Auto-Sklearn is the only method which features a sophisticated initialisation procedure, which takes into account past datasets to initialise in a sensible manner (meta-learning). While this was disabled here to prevent information leak (as the meta-learning likely involved datasets used here), this could show benefit for novel datasets. This has also been explored with TPOT [124] but is not yet included in the stable release.

From our results, we have shown all methods to perform well on a variety of domains. The search function of choice had little impact on the resulting accuracies. To continue progress in the area, we suggest a focus shift to highly parallelisable/scalable tools, and to expand the AutoML userbase we suggest a specific focus on both the usability and availability of the tools. Placing more focus on ensembling should also be considered if the goal is to achieve high accuracy.

4.6 Chapter Conclusions

A novel method for AutoML was proposed, which was able to achieve competitive (and in some cases improved) results when compared to the current state-of-the-art approaches in a fraction of the time (trained $6\times$ faster). While the comparisons were all run on a single core for a fair comparison, the speedup becomes even more drastic when considering multiple cores (as the method is embarrassingly parallel). We showed these results on 15 datasets and used the Friedman test for checking statistical significance. We can conclude that ensembling provides a key future direction for AutoML, as do massively parallel methods. Random grid search

has been previously dismissed in [71], as auto-sklearn outperformed this in the trials, however, we have shown here this changes when placing restrictions on the search space and focusing on ensembling the pipeline stumps. Likewise, ensembling addressed the concerns of a large variance of random search in [72], as shown by the results in Table 4.3.

A limitation which still persists is the interpretation of the results of the ensembles. For example, why does the ensemble in Chapter 3 or the pipelines in this chapter predict a particular class? In the next chapter, we address the concern of interpreting complex ensembles.

Chapter 5

Interpreting Complex Ensemble Structures with GP

A common criticism of modern machine learning techniques (such as complex ensembles) is the lack of interpretability/explainability. This could serve as a barrier to the wide-scale adoption of ensembles, as users are more likely to deploy a model if they can understand why particular decisions are being made. This is discussed for business in [125], for health care in [126], and more broadly in the recently deployed general data protection regulation [127] (GDPR) in the European Union, which prohibits "solely automated decisions, including profiling, which have a legal or similarly significant effect on them".

5.1 Chapter Introduction

In this chapter, a new multi-objective GP-based method for interpretable machine learning (IML) is proposed to overcome the limitations outlined in Chapter 2. The specific objectives are to:

- Propose a simple, human-readable tree structure which can be used for reconstructing complex predictions and overcome the limitations

of existing tree-based methods, such as greedy construction or need for pruning,

- Simultaneously maximise the reconstruction ability while minimising the complexity of the trees,
- Generate a frontier of trade-off solutions for user selection, and
- Finally, to evaluate the method against current state-of-the-art approaches on various datasets.

Specifically, the goal of this method is to reconstruct how a complex black-box method performs classification, rather than directly maximising classification accuracy. The proposed method is applicable to any black-box (arbitrarily complex) classifier, such as state-of-the-art ensembles, and makes no further assumptions about these models (such as gradient-based, or ability to apply sparsity).

5.2 The Proposed Method

In this section, a novel tree-based method is proposed for IML. Rather than the greedy construction seen with typical tree-based methods (such as CART), the proposed method uses GP to approximate an optimal tree. The tree construction process aims to maximise the reconstruction ability (mimic the predictions of a complex black-box) while minimising the complexity of the trees. The resulting trees are often far simpler while providing equivalent reconstruction ability to current approaches.

To overcome the limitations outlined in Section 5.1, we use NSGA-II [128], paired with strongly-typed GP (STGP) [32], to evolve decision tree-like structures, which simultaneously balance the complexity and the accuracy of the trees, by approximating a global search of the potential trees. An approximation is required as a global search would not be possible for any real datasets as the tree construction process is NP-complete. Hence,

the goal is to outperform greedy methods, while still being computationally feasible (at the expense of accepting near-optimal trees). Another benefit of such an approach is that rather than producing a single tree, a Pareto front of non-dominated trees is produced, which is particularly important for XAI since the user can select a tree by visualising the trade-off between the complexity and accuracy (examples of such visualisations are given in Section 5.5).

5.2.1 Overall Algorithm

The overall training algorithm is shown in Fig. 5.1. The black-box classifier is trained once only on the original data (x and y values), then the evolutionary process is performed based on the resulting predictions ($\hat{y}$) from this black-box model (for a total 50 generations). The evolutionary algorithm never sees the original labels (y), as this is instead attempting to recreate the predicted labels $\hat{y}$. At the end of the evolutionary run, the result is a set of Pareto optimal models/trees which approximate the complex black-box model. Only the model with the highest reconstructive ability is used here (i.e. the largest $f1$).

5.2.2 Multi-objective Optimisation

Evolutionary Multi-objective Optimisation (EMO) is used for generating a Pareto front of models, as shown in Fig. 5.1. When selecting individuals, the non-dominated sorting from NSGA-II [128] is first used to rank the individuals.

With a single objective, ranking for selection in GP is relatively straightforward. The rank is based only on this single objective/fitness value, i.e. the better the fitness the better the rank. However, when considering two (or more) objectives, this becomes less trivial as now the various objectives must be considered. The traditional approach to dealing with multiple objectives is scalarization, i.e., converting the problem to a single-

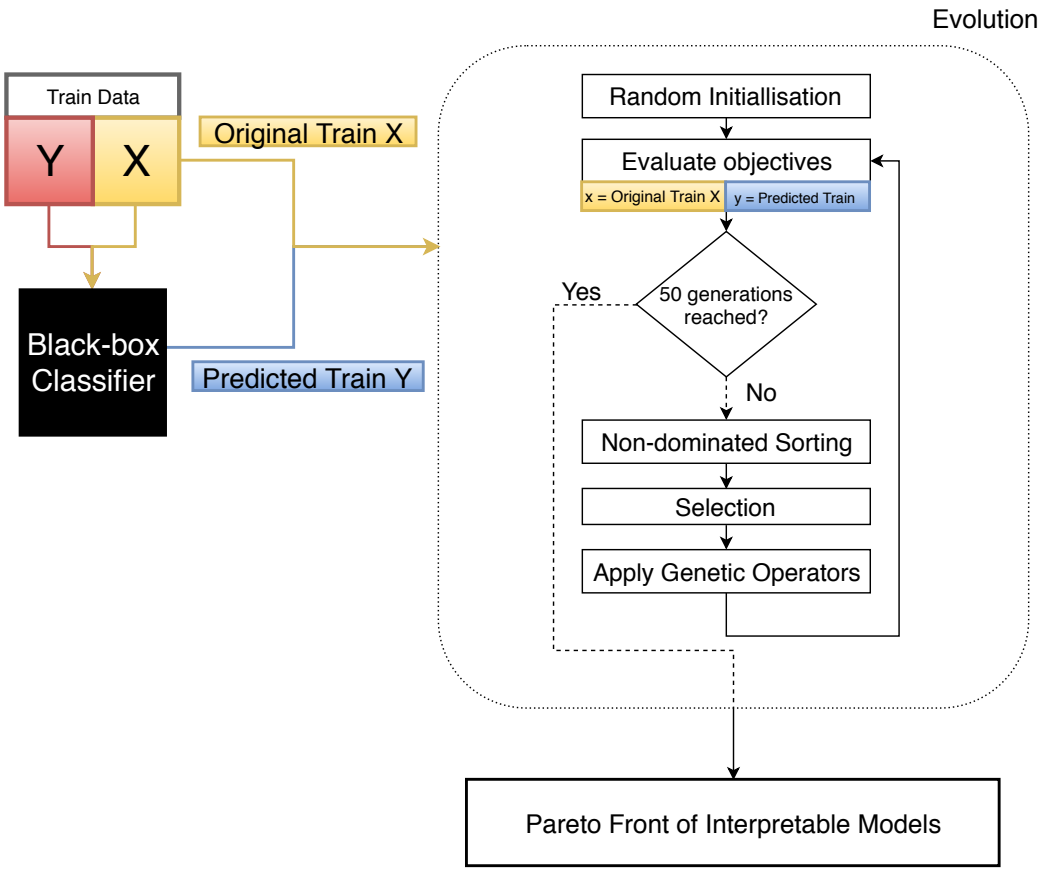


Figure 5.1: Evolutionary Training Process.

objective one (e.g. with a weighted sum or penalty term). However, such approaches require a priori information which is not always available, and often result in sub-optimal solutions (i.e. those which do not lie on the true Pareto front) due to conflicting objectives, and furthermore, do not give a good insight into the optimisation problem [129].

Instead, the preferred approach is Pareto-based optimisation. Pareto-based optimisation can simultaneously optimise conflicting objectives, by using the idea of Pareto dominance to generate a frontier of non-dominated solutions. Non-dominated means there is no other solution in the population which improves upon any of the objectives of the solution with-

out sacrificing another. There are various ways for ranking individuals in Pareto-based multi-objective optimisation, depending on the algorithm used. NSGA-II [128] and SPEA2 [130] are two of the most popular algorithms for EMO. Here NSGA-II was chosen due to the proven track record, however, additional algorithms could be explored in the future. With NSGA-II, the rank is based on the non-dominated rank in the population, and then if solutions are of the same rank, the solution which is in a less populous region is preferred [128].

Once the solutions have been ranked, the rest of the evolutionary process remains the same as with standard optimisation, however, an approximation of the true Pareto-front can be acquired from the final generation (rather than just a single best solution).

5.2.3 Objective Functions

The two objectives optimised are the reconstruction ability (maximisation) and the complexity (minimisation). In this case, the complexity is measured as the number of splitting points in a tree. The reconstruction ability is measured as an internal (i.e. on the training set only) 3-fold ($K = 3$) cross-validation. Meaning the two objectives are:

Objective 1:

$$maximise \frac{1}{K} \sum_{i=1}^{K} f1(predict(fold(i)), black - box\_predict(fold(i)) \quad (5.1)$$

where f1 is the weighted f1-score (i.e. weighted by the number of instances per class c)

$$f1(predicted, real) = (\sum_{c \in C} |c| \times \frac{2 \times precision \times recall}{precision + recall}) / \sum_{c \in C} |c| \quad (5.2)$$

Objective 2:

$$minimise \sum split\_points \quad (5.3)$$

Utilising the average reconstruction ability of multiple folds helps create more robust trees, in that a small change to the dataset will not result in a drastic change in the resulting tree. This is in contrast to decision-tree learners which can change drastically with small changes in input. Robustness is an important concept in XAI, as if the learnt rules change drastically with the training data, this can diminish faith in the learnt rules as it implies the rules were very specific to the training data (overfit) rather than generalising well to the new/unseen cases.

5.2.4 Representation

The evolved trees are decision tree-like, meaning the internal (function) nodes are splitting points (binary for numeric data, and a branch for each category in categorical data), and the output of such nodes are probability vectors for each class. An example of such a tree is shown in Fig. 5.2. Numeric splitting points are treated as typed terminals (so values will only be crossed over between trees for a given feature, and not between features), and these values are sampled uniformly from feature ranges.

As the output of all nodes is a probability vector, multiclass classification is supported directly. There are several other ways GP has been used for multi-class classification in the literature, but the output of probability vectors makes fewer assumptions (e.g. numeric outputs with class boundaries [131] assumes an ordering of classes), has better run time than others (e.g. a tree for each class [132], the run time scales with the number of classes), and also is the most interpretable due to the avoidance of complex numerical expressions or multiple trees, while also following closer to that of decision trees.

With decision trees, the input is at the root and the output at a leaf. Traditionally with GP, the input is at the leaves (terminals), and the output is at the root. To get around this, data is passed in at the leaves but only functions are returned until the root node (only one branch will end up re-

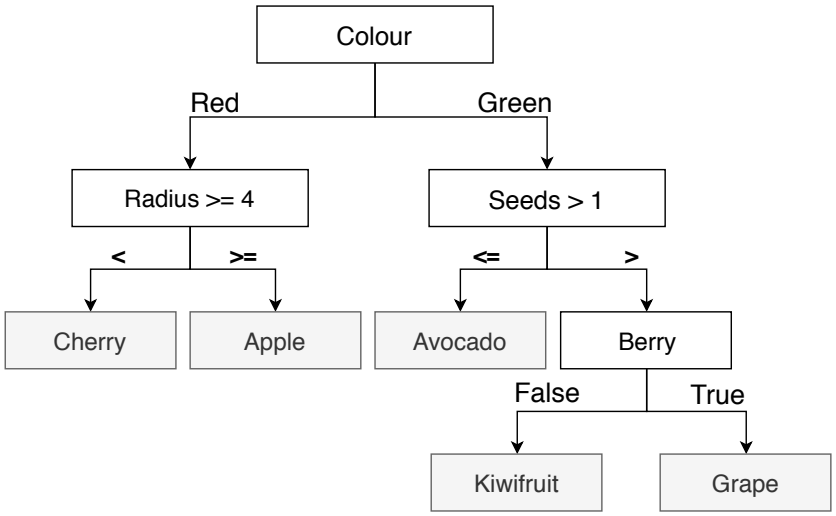
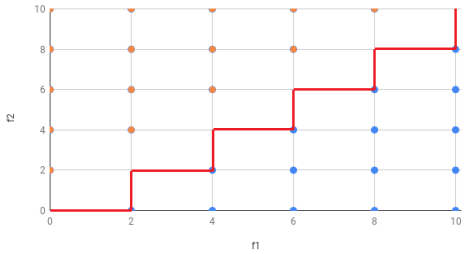


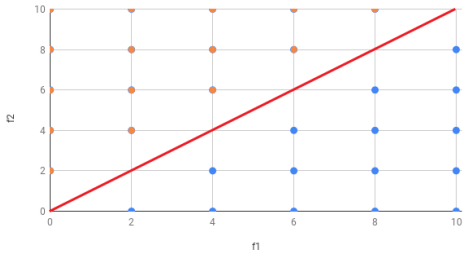
Figure 5.2: An example tree for fruit classification demonstrating the structure. Each node has a probability vector associated with it, but this is excluded for readability. The grey nodes indicate the predicted class (i.e. the class which had the highest probability).

turning values for a given input), and then the functions are executed once we are at the root. For visualisation and use-cases, the two can be treated uniformly, but the details are given for clarity (i.e. in the visualisations in Section 5.5 class values are leaf nodes, but they are merely the majority class predicted for this branch, rather than being an evolvable node).

The trees are also designed to have the ability to construct features implicitly, from the observation that some simple patterns are not easily approximable in decision trees due to the axis-parallel splits of decision trees. Consider the simple (artificial) dataset with two features $f1, f2$, and two classes $0, 1$, where the condition is *if* $f1 \geq f2$ *then class* 1 *else class* 0. With decision trees, the resulting trees would be needlessly complex, as shown in Fig. 5.3. We use subtrees to construct features, then check if these constructed features are greater than or equal to zero, in addition to the standard binary splitting nodes discussed above. This encapsulates many checks for mathematical relations, i.e. $f1 - f2 \geq 0$ is equivalent



(a) Decision Tree splitting criteria.



(b) GP Splitting criteria using constructed features

Figure 5.3: A comparison of the splitting points when using axis-parralel methods (decision trees) and oblique methods (GP with constructed features). Oblique methods can result in simpler splits as shown here.

to $f1 \geq f2$ from above. Constructed features can be combined using the standard mathematical operations (+, -, *, protected /). Again, this can be important for XAI due to the dramatically simpler rules learnt and utilising the complexity as the secondary objective prevents these constructed features from becoming overly complex.

5.3 Experiment Design

5.3.1 Experiment Details

The aim is to reconstruct the predictions of complex black-box models, and evaluate how well the proposed method is able to achieve this. In order to ensure the method is agnostic to the black-box, we chose to use three of the most common and high performing black-box models, which are random forests, gradient boosting, and a deep neural network (with 200 hidden layers with 200 neurons in each layer) all implemented in h2o [133]. 500 trees are used for random forests and gradient boosting, and the remaining hyperparameters are kept as default.

The aim is to see how well a single tree (or in the case of comparisons, a

simple model) is able to reconstruct the predictions of 500 ensembles trees or a deep network with 200 hidden layers. It is important to note these black-box methods were not finely tuned and the relative performance of each is not of importance in this paper, as we are *not* trying to compare black-box methods (rather the ability to reconstruct their predictions).

5.3.2 Datasets

30 datasets comprising some of the most run datasets from the OpenML repository [134] were used for comparison. These datasets are from a variety of domains and have a varying number of features (with both categorical and numeric), classes, and instances. The datasets offer a broad range to ensure generalisability of the proposed method. These datasets are summarised in Table 5.1.

5.3.3 Comparison Methods

Current state-of-the-art approaches to model extraction were trialled. These were Bayesian rule lists from [96] (as *pysbrl* on pip), an h2o decision tree ([133]), a simplified scikit-learn [102] decision tree [135], and logistic regression with $\ell 1$-regularization (from scikit-learn). For the scikit-learn methods, unfortunately, these do not natively support categorical features, so a one-hot-encoding is needed to be applied prior. For Bayesian rule lists, they currently only support discrete features, so as is commonly done, multi-interval discretization as proposed in [136] was first applied. No preprocessing was required for the h2o trees. Unfortunately, to our knowledge there are no current approaches in literature to multi-objective XAI, so we only compare the best resulting solutions rather than comparing Pareto frontiers/hypervolumes as is more commonly done with multi-objective optimisation algorithms.

5.3.4 Parameter Settings

The evolutionary search was run for 50 generations, with 100 trees in each generation. A larger population size could be utilised, and the results would likely improve at the expense of an increased runtime. Trees were limited to a maximum height of 17. A crossover rate of 0.8 was used, and a mutation rate of 0.2. Top performing individuals are never lost, as the $\mu + \lambda$ algorithm [137] is used, with both values set to the population size (i.e. keep the 100 best).

5.3.5 Evaluation Measures

Classification Performance For measuring performance, we use a weighted f1-measure where the inputs are the predicted labels and the black-box labels (rather than the real class labels). This was chosen as we assume each class is equally important, and wish to have a valid measure for both binary and multi-class classification. This measure can be roughly interpreted as "how well are we able to reconstruct the predictions of the black-box classifier for each class". For presentation, we scale this to the range $0 \ldots 100$, rather than $0 \ldots 1$.

Complexity Measuring complexity across classifiers can be a complex task, however, here thankfully since each of the comparison methods is somewhat similar in representation, there is a natural definition of complexity. We define complexity as the number of splitting points in a tree. If a constructed feature is used as a splitting point in the proposed method, this counts as multiple splits, i.e. $f1 + f1 \leq 0$, would be a complexity of 2, rather than 1, to provide a fair comparison. For Bayesian rule lists, the complexity is the number of rules + the number of conjunctions ($\wedge$'s) in these rules, i.e. *if* $f1 = 2 \wedge f2 = 0$ *then* ... counts as 2. The number of rules in logistic regression is measured as the number of non-zero coefficients. Therefore, for all methods, the minimum complexity is 0 (i.e.

predict majority class, no rules learnt), and the maximum is ∞.

5.3.6 Significance Tests

To compute whether the difference in reconstruction ability across datasets for each method was statistically significant, we used Friedman tests paired with Nemenyi post-hoc analysis as suggested in [105] as general performance is the main focus. Significance tests on each dataset were not used for the reasons described in [138], i.e. violations in the conclusions due to the increased probability of type-I errors, as well as the Friedman test making fewer assumptions.

To show the average reconstruction ability for each method, for each dataset, we present the average testing accuracy of a 10-fold cross-validation procedure, where each method gets the same train: test sets. The averages are also across the 3 black-box methods (so for each method, 30 runs are executed for each dataset, with the same random samples used across methods). The goal is for the extraction methods to be invariant to the black-box model used, hence the averaging.

5.4 Results and Discussion

5.4.1 Overall Results

The results are shown in Table 5.2, on the two measures of concern: reconstruction ability and complexity.

As we are interested in achieving the best reconstruction ability, while also achieving the simplest representation, we compute the number of times a method was dominated across datasets. Here, the definition of non-dominated is no other method achieves either a simpler representation with the same (or improved) reconstruction ability, i.e., fewer dominations is a good sign.

Table 5.1: Summary of dataset characteristics

Dataset	Numeric Features	Categorical Features	Classes	Instances
analcatdata	70	0	4	841
autoUniv-au7-500	8	4	5	500
balance-scale	4	0	3	625
blood-transfusion	4	0	2	748
climate-model	20	0	2	540
cmc	2	7	3	1473
credit-g	7	13	2	1000
diabetes	8	0	2	768
eeg-eye-state	14	0	2	14980
GesturePhase	32	0	5	9873
hill-valley	100	0	2	1212
ilpd	9	1	2	583
iris	4	0	3	150
kc1	21	0	2	2109
kc2	21	0	2	522
kr-vs-kp	0	36	2	3196
monks-problems-1	0	6	2	556
monks-problems-2	0	6	2	601
ozone-level-8hr	72	0	2	2534
pc1	21	0	2	1109
phoneme	5	0	2	5404
qsar-biodeg	41	0	2	1055
spambase	57	0	2	4601
splice	0	60	3	3190
steel-plates-fault	33	0	2	1941
tic-tac-toe	0	9	2	958
vehicle	18	0	4	846
wall-robot-navigation	24	0	4	5456
waveform-5000	40	0	3	5000
wdbc	30	0	2	569

Table 5.2: Summary of the results. The average testing performance is presented.

	Black-box Test Accuracy			Test Reconstruction Ability					Model Complexity				
	RF	**GB**	**DL**	**GP**	**BRL**	**SDT**	**DT**	**LR**	**GP**	**BRL**	**SDT**	**DT**	**LR**
Analcatdata_Authorship	99.40	98.70	99.80	83.21	92.33	92.3	91.71	98.81	5	17	6	59	122
Autouniv-Au7-500	47.50	44.52	38.30	55.69	36.83	49.49	50.64	46.57	7	1	18	163	250
Balance-Scale	82.70	86.40	96.90	72.9	73.06	77.6	81.44	80.68	6	9	13	127	12
Blood-Transfusion	66.70	73.70	70.50	81.65	79.24	86.34	87.71	78.73	4	5	8	35	4
Climate-Model	86.90	85.30	88.20	88.44	92.47	90.66	89.72	91.09	3	3	6	22	13
Cmc	54.30	54.20	45.90	60.58	51.84	58.6	68.62	67.81	6	7	11	123	150
Credit-G	73.60	75.70	72.40	76.61	57.3	76.92	77.12	80.19	7	2	8	46	76
Diabetes	74.20	73.40	71.00	77.5	79.56	78.72	78.15	79.26	4	9	9	39	8
Eeg-Eye-State	93.10	87.50	78.90	54.26	77.65	71.86	74.3	49.28	4	120	8	50	14
GesturePhase	67.00	62.80	60.00	47.51	53.16	55.94	61.35	47.14	5	81	10	270	23
Hill-Valley	35.70	52.70	64.10	80.57	76.68	84.96	82.26	81.4	4	3	18	27	100
Ilpd	65.40	66.60	70.00	67.2	51.29	63.98	70.13	67.19	4	1	8	32	118
Iris	93.30	94.64	98.30	98.93	97.37	98.46	96.89	95.4	3	3	3	15	10
Kc1	82.30	84.30	82.70	85.19	83.98	85.8	86.14	85.34	4	10	5	39	21
Kc2	77.80	80.06	86.10	86.79	86.38	87.33	85.96	85.7	3	5	6	27	18
Kr-Vs-Kp	98.80	99.40	99.00	94.22	92.25	96.62	94.48	96.8	6	15	8	16	57
Monks-Problems-1	99.80	98.90	99.10	86.63	72.44	92.9	86.34	72.44	7	2	15	14	10
Monks-Problems-2	92.30	97.10	99.80	55.11	52.85	86.3	89.93	52.42	10	0	28	43	10
Ozone-Level-8Hr	93.70	93.20	93.60	93.44	93.76	94.82	94.52	95.55	3	11	5	36	61
Pc1	91.90	92.80	91.50	92.95	93.26	93.45	94.6	92.46	4	5	6	26	20
Phoneme	91.10	88.50	90.71	81.77	85.02	82.17	88.36	76.96	6	35	6	47	5
Qsar-Biodeg	87.00	86.60	84.70	81.14	86.52	85.81	87.1	89.59	4	18	8	38	31
Spambase	95.20	95.40	93.90	81.59	94.27	89.27	92.42	94.65	4	45	5	33	54
Splice	97.30	96.20	95.10	82.09	89.66	93.62	95.47	96.99	6	24	10	100	449
Steel-Plates-Fault	99.70	94.50	99.80	88.21	99.67	99.78	99.8	99.74	5	7	6	7	20
Tic-Tac-Toe	98.80	97.40	97.60	73.07	61.3	91.1	92.25	97.83	6	3	18	33	27
Vehicle	75.20	77.30	84.30	61.14	66.36	72.65	77.9	80.59	6	17	15	112	72
Wall-Robot-Navigation	99.20	99.69	92.50	78.63	96.23	95.33	97.4	68.53	5	58	8	79	96
Waveform-5000	85.30	83.80	83.20	72.64	79.48	76.91	83.51	92.5	6	71	8	168	117
Wdbc	95.40	94.40	98.70	95.21	95.13	94.14	94.2	95.48	4	8	4	17	11

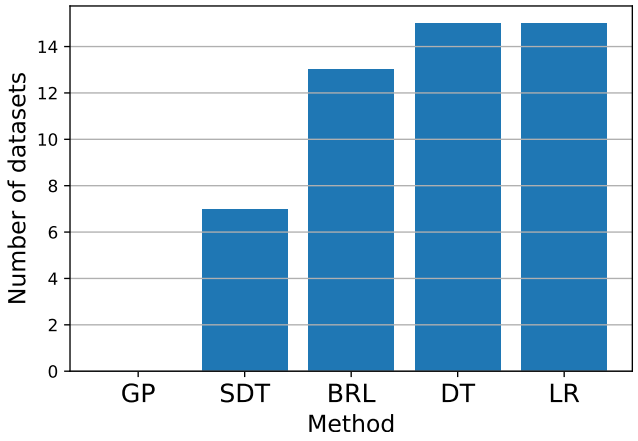


Figure 5.4: Number of datasets on which a method was dominated by other methods (lower the better).

The number of times dominated was chosen as the comparison methods present only a single solution, therefore we can not compare hypervolumes of resulting frontiers (despite the proposed method returning a frontier), yet we still wish to compare the two objectives without making assumptions about the importance of either (i.e. without computing a scalar value). As the proposed method returns a frontier of solutions, only the resulting solution with the highest reconstruction ability (as measured by the fitness function on the training data, not on the unseen data), was used to represent the performance. This result is shown in Fig. 5.4.

GP (the proposed method) was not dominated on any of the datasets. One caveat is that analysing the dominated counts alone is not a comprehensive indicator of performance, since if the simplest possible model was used (i.e. just use the majority class, which would give a complexity of 0 as no rules were learnt), then this would never be dominated. Likewise, using the black-box model itself would achieve maximal reconstruction ability, and even though the complexity would be far greater this would still never be dominated. Another argument could be that since GP was the

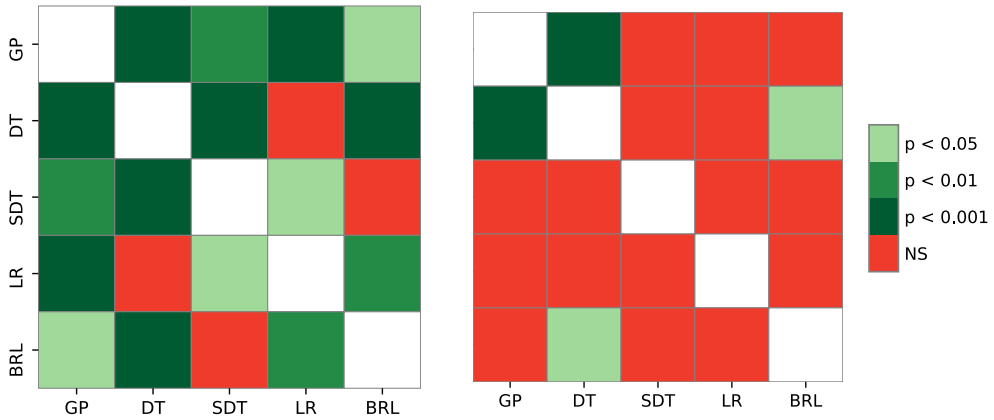
only method which simultaneously balanced these objectives, this measure can be seen as biased towards the proposed. This is true, but also shows the ability/usefulness of population-based techniques such as GP as they can effectively optimise multiple objectives simultaneously. This shows that multi-objective optimisation is a good choice for IML, as the objectives were optimised better than the existing approaches (in terms of dominance).

5.4.2 Statistical Tests

In addition to the dominated counts, the relative performance of the methods must also be considered. Friedman testing paired with Nemenyi post-hoc analysis is performed to compare whether the difference in the resulting accuracies was statistically significant across datasets. The resulting p-values are visualised in Fig. 5.5, where the only statistically significant differences in reconstruction ability are between the proposed method and the standard decision tree, and Bayesian rule lists and the decision tree. For complexity, the proposed method was significantly simpler than all comparisons methods.

To analyse the results on specific results, a per dataset breakdown of the reconstruction ability vs complexity is given in Fig. 5.6, where the ideal position is the top left of the chart (i.e. minimal complexity, maximal reconstruction ability).

From this, we are able to conclude the proposed GP-based method consistently produces compact rules, while achieving statistically equivalent accuracy to the more complex approaches. The one exception to this was the decision tree, which was, however, on average $15\times$ more complex than the proposed approach.



(a) Complexity

(b) Reconstruction Ability

Figure 5.5: Statistical significance testing. Resulting p-values from Friedman test w/ Nemenyi post-hoc correction. Red indicates no significant difference. Green indicates a difference (darker=more significant).

5.5 Further Analysis

To give further insights into the resulting extracted models, a comparison is given on the hill-valley dataset across the methods. The hill-valley dataset was chosen as this is the most complex (i.e. lowest resulting test accuracy on the black-box models), and the deep learning reconstructions are used. While we would like to present all results for all methods, this would compromise 900 diagrams per method. Logistic regression is not presented, as this only a list of up-to 100 coefficients for the features, and 1 value for the intercept. These comparisons are shown in Fig. 5.7.

From the examples (Fig. 5.7), we can see that the resulting tree from the proposed method and the Bayesian rule list are by far the simplest of the compared methods, condensing the approximate knowledge from a 200 layer neural network into small human readable form. A similar trend is seen across the datasets when comparing the complexities. In this case, the Bayesian rule-list actually just predicts 1 class, so is overly simplistic (shown by the differences in the f-measures). To get an idea of what GP

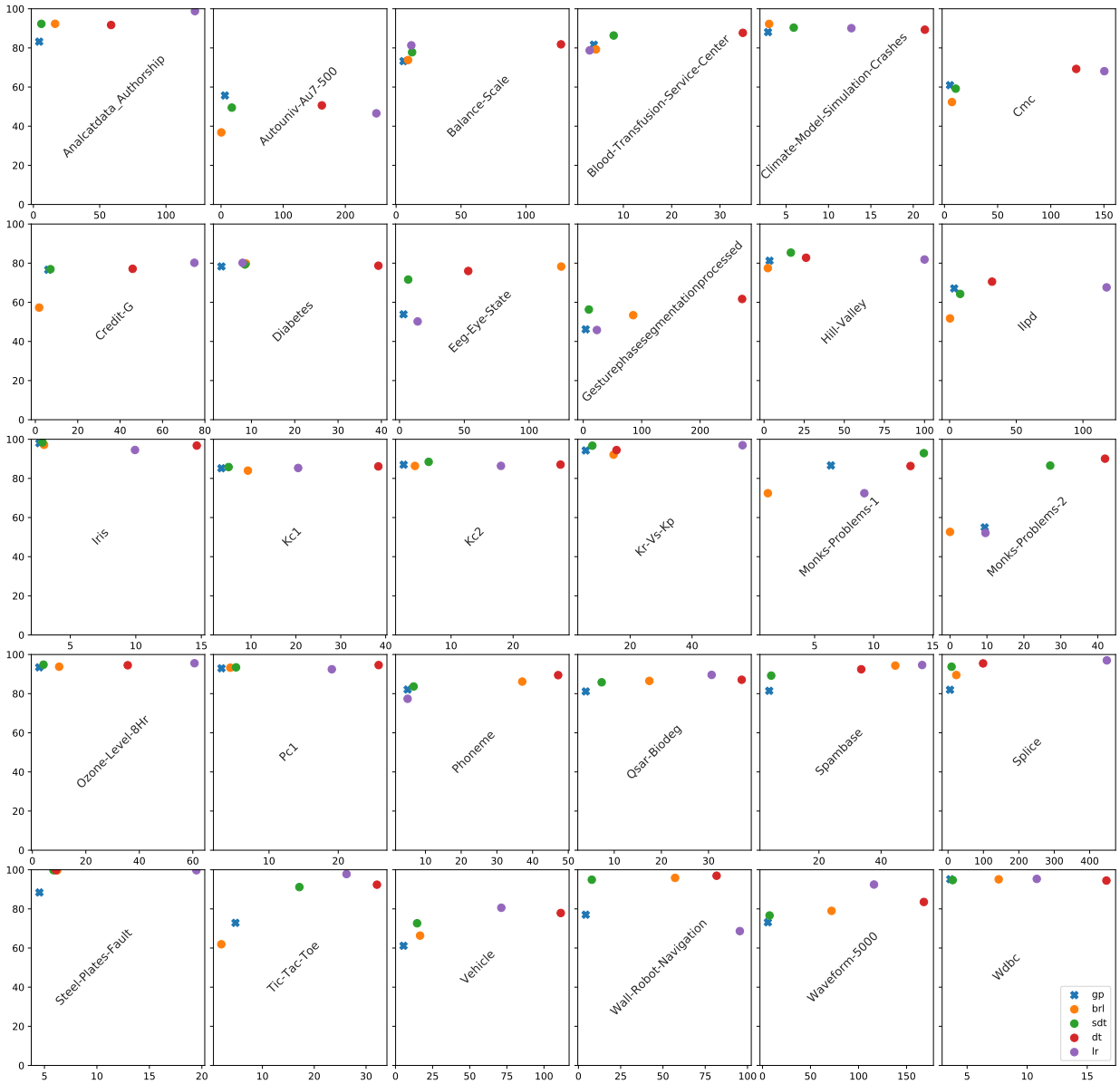
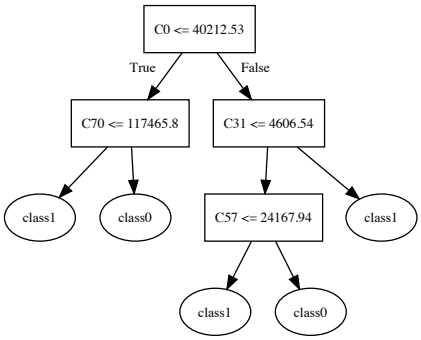
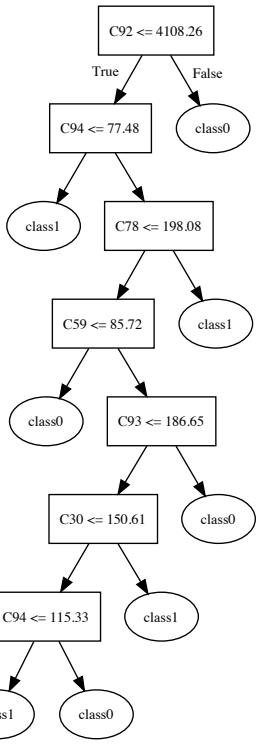


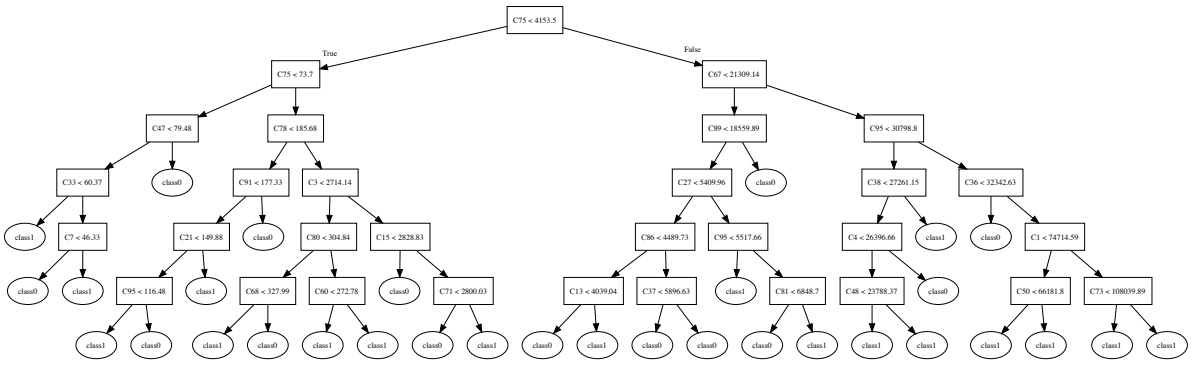
Figure 5.6: Testing reconstruction ability (y) vs Complexity (x). The ideal position is the top left corner of the graph.



(a) Proposed ($f_1 = 0.757$)



(b) Simplified Decision Tree ($f_1 = 0.759$)



(c) Decision Tree ($f_1 = 0.790$)

IF (X43 = 3) AND (X78 = 3) THEN: Class1
ELSE IF (X62 = 0) THEN: Class1
ELSE DEFAULT: Class1

(d) Bayesian Rule List ($f_1 = 0.704$)

Figure 5.7: A comparison of the resulting models on the hill-valley dataset, attempting to recreate the 200 layer deep neural network predictions. The testing reconstruction ability is given in brackets next to the method name.

has found, the evolved tree can be consulted (Fig. 5.7a). The evolved tree is attempting to split the data based on four features (or points) in the dataset.

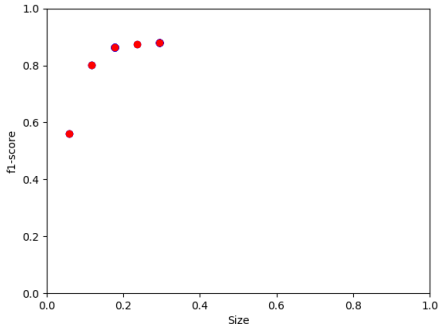
This makes sense when we consider the hill-valley dataset, which "when plotted in order (from 1 through 100) as the Y coordinate, the points will create either a Hill (a 'bump' in the terrain) or a Valley (a 'dip' in the terrain)" [122]. We can see the tree is checking the first point, and comparing to the point at 30% (i.e. the $30th$ feature), or the point at 70%, where the tree is trying to distinguish between classes by finding the common points for the hills/valleys and checking if these are high or low relative to the training data (e.g. a high point at the start, a low point at 30%, then a high a high point at 57% indicates a valley based on this tree).

Across the board, the datasets which were most difficult to reconstruct the predictions on were: Autouniv-Au7-500, Eeg-Eye-State, Gesturephasesegmentationprocessed (GesturePhase), and Monks-Problems-2.

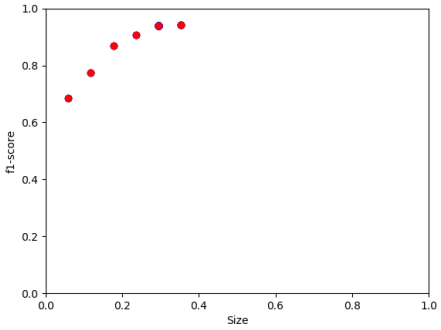
Two of these datasets (Autouniv-Au7-500, GesturePhase) have 5 classes. One explanation here is that as the number of classes in a dataset increases, as does the complexity necessarily with tree-based methods. For example, if we have 100 classes, we therefore require 100 leaf nodes to have a predictive branch for each class. This presents a potential area of future research, as the size of the trees could negate the explainability as the number of classes grows. Here the pressure for small trees was perhaps too strong, and this requirement would need to be relaxed in the case of a high number of classes.

Monks-Problems-2 is entirely categorical features. In the proposed method, a categorical node has a branch for each feature - this potentially overfits to the training data, and combining categorical features into a single branch should be considered in future work (for reference, this is done in the decision tree method, where we can see a significant improvement in reconstructive ability, and this is consistent across the datasets with all categorical features).

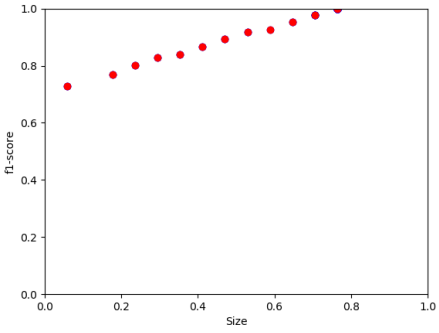
For eeg-eye-state, the data is sequential/time-series. The proposed method is not optimised/designed for such datasets, so this explains the lower performance.



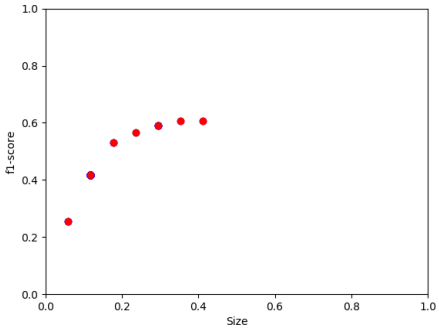
(a) Analcatdata_Authorship



(b) Kr-Vs-Kp



(c) Monks-Problems-1



(d) Vehicle

Figure 5.8: Resulting Pareto Frontiers.

To highlight another benefit the proposed method has over the existing IML approaches, a Pareto front is given in Fig. 5.8. This shows a resulting front for four of the datasets, but similar fronts are available for all datasets. In all cases, the model with the highest reconstruction ability was chosen, however, even simpler models could be used from the front if desired. Likewise, if models were overly simple, any restrictions on the

height of evolved trees could be relaxed.

5.6 Chapter Conclusions

In this chapter, a novel model agnostic method for XAI was proposed which utilises model extraction. Multi-objective GP is used to learn a simple and interpretable representation of a complex black-box model, which is often able to effectively reproduce the black-box's predictions. This new method was compared to existing approaches to model extraction and was found to offer drastically simpler models, with statistically equivalent test accuracy. The method is also able to handle categorical and continuous features natively, unlike some existing approaches such as Bayesian rule lists. To our best knowledge, this is the first utilisation of multi-objective optimisation in XAI, and follows the suggestions in [139] that "a multi-objective approach based on Pareto dominance would be more suitable to sufficiently address this trade-off" (between accuracy and interpretability). We also believe this is the first application of GP for model extraction (i.e. training on the predictions of a black-box model rather than the original labels) and shows a promising direction for future developments.

Chapter 6

Conclusions

The overall goal of this thesis was to investigate population-based methods for ensemble learning and ensemble interpretation on a range of classification tasks. The goal has been achieved by developing novel methods capable of performing fully automated ensemble learning (and pipeline design), and also for generating simple structures which can be used to help understand complex ensembles.

6.1 Major Conclusions

Relating to the three initial goals, the contributions are split into three sections here.

Conclusion 1: Genetic programming can be utilised for automatically constructing well-performing ensembles.

In Chapter 3, a novel GP-based method was proposed for automated ensemble learning, where GP was utilised as the meta-learner for stacked generalisation. The method alleviates the user from model selection, hyperparameter tuning, and ensembling for classification tasks. Appropriate base members and their key hyperparameters are tuned automatically, as well as the hierarchical structure of the ensemble. The proposed method outperformed a range of methods (and the individual base mem-

bers) across a broad range of datasets.

We discovered GP was an ideal approach to such a task, as the tree-based structures very naturally lend themselves to a hierarchical ensemble. Furthermore, utilising strongly-typed GP allowed us to perform hyperparameter selection automatically.

The result is a method capable of constructing a fully automatic ensemble, which selects from a range of base members and hyperparameters automatically, without requiring human expertise or prior knowledge on the problems or learning algorithms.

Conclusion 2: Random population-based ensemble search can achieve performance similar to the state-of-the-art AutoML methods in a fraction of the time.

In Chapter 4 a novel method for AutoML was proposed, which was able to achieve competitive (and in some cases improved) results when compared to the current state-of-the-art approaches in a fraction of the time (trained $6\times$ as fast). Furthermore, the proposed method is able to be run entirely in parallel, which helps to overcome the limitation of automated machine learning taking too long to generate good pipelines. While the comparisons were all run on a single core for a fair comparison, the speedup becomes even more drastic when considering multiple cores.

In this case, the proposed method was based on a strongly-typed random search but utilised a tree-based structure extremely similar to Genetic Programming. From the results, we are able to conclude that ensembling provides a key future direction for AutoML. Ensembling is useful for ensuring the robustness of the resulting pipelines, which had been a problem with previous randomised approaches. Furthermore, ensembling meant well-performing individuals were able to be found in a fraction of the time. While this may seem counter-intuitive as the search space is larger, the increase in performance of even an average ensemble was worth the growth of the search space.

Conclusion 3: Multi-objective GP can be used for generating simple

and interpretable representations for complex black-box methods by simultaneously optimising the reconstruction ability and complexity.

In Chapter 5, a novel method for interpreting state-of-the-art complex black-box machine learning models was proposed. The method is agnostic to the model used, meaning any black-box models can be interpreted (such as complex ensembles or deep neural networks). Evolutionary multiobjective optimisation is used in conjunction with strongly-typed genetic programming to learn a simple and interpretable approximation of a complex black-box models behaviour, which is often able to effectively reconstruct the black-box's predictions.

This new method was compared to existing approaches for model extraction and was found to offer drastically simpler models while providing comparable reconstructive performance. This was demonstrated on a range of datasets, by approximating the knowledge of complex black-box models such as 200 layer neural networks and ensembles of 500 trees, with a single tree. The method is also able to handle categorical and continuous features natively, unlike some existing approaches such as Bayesian rule lists. To the best of our knowledge, this was the first application of multiobjective GP for global model interpretation for black-box explainability.

Both GP and multiobjective optimisation are ideal candidates for discovering/interpreting the inner-workings of complex black-box methods such as state-of-the-art ensembles.

6.2 Additional Findings

As well as the initial goals we set out, there was also some additional knowledge we gained in the process.

Firstly, the diversity achieved in ensembles constructed automatically with genetic programming was strong enough on its own when using a suitable fitness function (such as a weighted f1-measure). Diverse ensembles were generated without requiring extra evolutionary pressure. We

evaluated the use of a special crossover which weighted crossover points based on the diversity/information gained and found the results to be equivalent to standard (uniform weighted) crossover. This shows the randomness involved in the selection procedure may be enough of a pressure to construct diverse ensembles assuming sensible parameter settings (i.e. a relatively high mutation rate).

Secondly, we found caching to be a simple solution which results in a drastic speedup when function nodes are costly to evaluate. In this case, nodes were running classification algorithms. Hashing the model and storing the results resulted in a drastic speedup over recomputing the models each time.

Thirdly, we found for AutoML the difference in search algorithms used was not drastic. For example, there were no significant differences between the existing Bayesian and Genetic Programming methods in terms of performance.

Finally, we found multi-objective optimisation to be a very natural and important application for interpretable machine learning. Population-based methods such as GP which are able to simultaneously optimise multiple objectives are an important future direction for explainability and interpretability of resulting models. While the initial goal was just to utilise GP for interpreting a black-box model, introducing multiobjective optimisation and having a Pareto-front of such explanations was very powerful, and XAI should be considered a key application of multiobjective optimisation.

6.3 Future Work

In this section, we outline future work/recommendations for further avenues of research.

Ensemble Construction

In Chapter 3 we used the predicted class as the output for the various

classification nodes. However, many algorithms can also give the probability of the class, which could also be incorporated. For example, if a tie occurs (a disagreement in classifiers about the real class), the classifier which was more certain in its prediction could be preferred, i.e. if we were doing binary classification, and one classifier was 56% certain an instance belonged to class A, however, another classifier was 98% sure the instance belonged to class B, then we should prefer the latter classifier. There are two reasons this was not included here, firstly, not all algorithms give probabilities (or if they do it can be costly to compute, i.e. with SVMs), secondly, for algorithms which do output probabilities, these would need to be well calibrated. As an example, with Boosted trees, probabilities close to the extremes (i.e. 0 and 1) are rare, whereas, with methods such as Naive Bayes, the probabilities are pushed towards 0 or 1 [140]. This calibration is non-trivial and would be required to be performed consistently throughout the evolutionary process to avoid penalising more conservative models. Further work could examine if this additional computation would be worth the cost. Another approach would be to limit the ensemble to classification ensembles which optimise in similar fashions, i.e. by minimising the log loss, for example by having an ensemble of neural networks each with different architectures, however, this reduces the diversity of the ensemble as there is an additional assumption that all base members must be optimising a similar loss function.

In Chapter 3 we used GP for parameter selection as the number of parameters we were trying to tune was relatively small (i.e. each classifier only had a couple of parameters, and the range of these was not huge), however, GP is not an ideal candidate for large-scale parameter selection. The main limitation GP suffers from here is that neighbouring parameters of well-performing values are not necessarily explored, for example, if we find 10 neighbours performs well for k-NN, we do not necessarily try 9 or 11 neighbours which may also perform well, and potentially better. Other methods such as particle swarm optimisation, or Bayesian inference could

be incorporated with GP to further improve this, where GP evolves the structure of the ensemble, and PSO or Bayesian inference is then used to tune the parameters in the trees either throughout the evolutionary process or on the final resulting ensemble. This integration deserves further exploration.

AutoML

In Chapter 4 a novel randomised ensemble-based approach to AutoML was proposed (AutoVoter). While we showed equivalent performance in a fraction of the time to existing approaches, the next stage would be to expand the other approaches (such as TPOT) to feature an ensemble learning process natively, to see if an additional performance gain could be achieved. Alternatively, rather than the random search used, we could expand our own method into a Bayesian and a GP based method, and perform in-depth comparisons between the three approaches when utilising the exact same search space. Currently, we did not notice any significant difference between the search algorithm used, so this comparison would help to gain a deeper understanding of how the methods differ in their exploration and exploitation of the AutoML search space.

Similar to the above, we could also expand the method to have more expressive pipelines. In the proposed method we limited pipeline stumps to be composed of at most 3 components. This was done in order to combat the exponential growth of the search space when introducing ensembles, however, experiments could be performed to see about increasing this size to allow for a broader range of pipelines.

We would also have liked to provide a comprehensive comparison between the randomised approach and GP. Future work could look to understand in what cases random search becomes competitive, and when GP should be preferred.

Finally, we showed that AutoVoter was competitive in a fraction of the time to the current state-of-the-art approaches. We tested this with 30 minutes for the proposed method, and 3 hours for the comparison methods.

However, a large scale in-depth comparison would be ideal to see how the various methods improve over time, for example running all methods for upwards of 24 hours, with continuous comparisons throughout the learning phase. This was not attempted here due to the large computational cost associated, but as computational power becomes more affordable, this may be a possibility for future analysis.

Interpreting Complex Models

In Chapter 5 a novel method for interpreting black-box models was presented. There are several areas of potential improvement.

Firstly, can the recreation ability of the proposed method be improved without sacrificing simplicity by considering local search techniques for splitting points? Currently, the splitting points are sampled uniformly from the feature ranges and evolution used to select good points. Once the structure has been found with GP, these points could be adjusted dynamically with a local search method. Similar to the suggestions for Contribution 1, GP is not necessarily ideal for exploring parameter values, and the same holds true for the splitting points. Exploring regions of well-performing splitting points could further improve the results.

Next, currently for categorical features, a new branch is constructed for each category. This potentially results in overfitting to the training data, and at the very least results in overly complex trees when the number of categories is high. Pruning the trees by grouping nodes with similar subtrees could be investigated, as this may result in improved performance for datasets with a large number of categorical features.

With the proposed method, only the original training data is used for learning the interpretable model. However, since the real labels are never seen by the evolutionary process, this dataset could be heavily augmented to produce an essentially infinite amount of training data as the black-box serves as the labeller. Once the black-box has learnt a model, we could use the black-box's predictions as labels for the new augmented data. This has been explored in [94], where they refer to the black-box as an "or-

acle". Likewise, semi-supervised learning tasks could benefit from this approach. All of the labelled training data could be used to learn the black-box, then the unlabelled training instances used the same as the augmented data above. This could be particularly useful for branches which only have a few instances, to increase the likelihood of predicting the correct class.

In Chapter 5 a basic complexity measure is used for evaluating the simplicity of the models. However, if the goal is human interpretation, it would be ideal to conduct a blind large scale user study on the resulting models. Finally, related to the previous point, it is possible to guide the evolution of the models based on the human feedback, we foresee these human-in-the-loop/people-centric AI type systems being important for XAI, and this is something that could be incorporated into the evolutionary process here by modifying the complexity measure to instead be user determined.

Bibliography

[1] Y. Freund, R. E. Schapire *et al.*, "Experiments with a new boosting algorithm," in *Icml*, vol. 96. Citeseer, 1996, pp. 148–156.

[2] T. Hoch, "An ensemble learning approach for the kaggle taxi travel time prediction challenge." in *DC@ PKDD/ECML*, 2015.

[3] A. Puurula, J. Read, and A. Bifet, "Kaggle lshtc4 winning solution," *arXiv preprint arXiv:1405.0546*, 2014.

[4] R. M. Bell and Y. Koren, "Lessons from the netflix prize challenge," *Acm Sigkdd Explorations Newsletter*, vol. 9, no. 2, pp. 75–79, 2007.

[5] X. Zeng, W. Ouyang, J. Yan, H. Li, T. Xiao, K. Wang, Y. Liu, Y. Zhou, B. Yang, Z. Wang *et al.*, "Crafting gbd-net for object detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 9, pp. 2109–2123, 2018.

[6] G. Brown, J. Wyatt, R. Harris, and X. Yao, "Diversity creation methods: a survey and categorisation," *Information Fusion*, vol. 6, no. 1, pp. 5–20, 2005.

[7] A. Chandra and X. Yao, "Divace: Diverse and accurate ensemble learning algorithm," in *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 2004, pp. 619–625.

[8] L. I. Kuncheva, "That elusive diversity in classifier ensembles," in *Iberian conference on pattern recognition and image analysis*. Springer, 2003, pp. 1126–1138.

[9] S. Džeroski and B. Ženko, "Is combining classifiers with stacking better than selecting the best one?" *Machine learning*, vol. 54, no. 3, pp. 255–273, 2004.

[10] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3133–3181, 2014.

[11] Y. Lou, R. Caruana, and J. Gehrke, "Intelligible models for classification and regression," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 150–158.

[12] J. Burrell, "How the machine thinks: Understanding opacity in machine learning algorithms," *Big Data & Society*, vol. 3, no. 1, 2016.

[13] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and computing*, vol. 4, no. 2, pp. 87–112, 1994.

[14] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[15] S. B. E. Raj and A. A. Portia, "Analysis on credit card fraud detection methods," in *2011 International Conference on Computer, Communication and Electrical Technology (ICCCET)*. IEEE, 2011, pp. 152–156.

[16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[17] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir *et al.*, "Wide & deep learning for recommender systems," in *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 2016, pp. 7–10.

[18] M. Chen, Y. Hao, K. Hwang, L. Wang, and L. Wang, "Disease prediction by machine learning over big data from healthcare communities," *Ieee Access*, vol. 5, pp. 8869–8879, 2017.

[19] T. Harris, "Credit scoring using the clustered support vector machine," *Expert Systems with Applications*, vol. 42, no. 2, pp. 741–750, 2015.

[20] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, 2015, pp. 802–810.

[21] S. Dua and X. Du, *Data mining and machine learning in cybersecurity*. Auerbach Publications, 2016.

[22] J. Patel, S. Shah, P. Thakkar, and K. Kotecha, "Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques," *Expert Systems with Applications*, vol. 42, no. 1, pp. 259–268, 2015.

[23] S. Zhai, K.-h. Chang, R. Zhang, and Z. M. Zhang, "Deepintent: Learning attentions for online advertising with recurrent neural networks," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 2016, pp. 1295–1304.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[25] S. Marsland, *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC, 2011.

[26] T. O. Ayodele, "Types of machine learning algorithms," in *New advances in machine learning*. InTech, 2010.

[27] J. Wang, C. Lu, M. Wang, P. Li, S. Yan, and X. Hu, "Robust face recognition via adaptive sparse representation," *IEEE transactions on cybernetics*, vol. 44, no. 12, pp. 2368–2378, 2014.

[28] G. Deshpande, P. Wang, D. Rangaprakash, and B. Wilamowski, "Fully connected cascade artificial neural network architecture for attention deficit hyperactivity disorder classification from functional magnetic resonance imaging data," *IEEE transactions on cybernetics*, vol. 45, no. 12, pp. 2668–2679, 2015.

[29] K. Schouten, O. van der Weijde, F. Frasincar, and R. Dekker, "Supervised and unsupervised aspect category detection for sentiment analysis with co-occurrence data," *IEEE transactions on cybernetics*, vol. 48, no. 4, pp. 1263–1275, 2018.

[30] Y.-S. Ong, M. H. Lim, and X. Chen, "Memetic computationpast, present & future [research frontier]," *IEEE Computational Intelligence Magazine*, vol. 5, no. 2, pp. 24–31, 2010.

[31] F. Neri and C. Cotta, "Memetic algorithms and memetic computing optimization: A literature review," *Swarm and Evolutionary Computation*, vol. 2, pp. 1–14, 2012.

[32] D. J. Montana, "Strongly typed genetic programming," *Evolutionary computation*, vol. 3, no. 2, pp. 199–230, 1995.

[33] J. R. Koza, *Genetic programming II, automatic discovery of reusable subprograms*. MIT Press, Cambridge, MA, 1992.

[34] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming*. Lulu. com, 2008.

[35] A. Shukla, H. M. Pandey, and D. Mehrotra, "Comparative review of selection techniques in genetic algorithm," in *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*. IEEE, 2015, pp. 515–519.

[36] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2000, vol. 1.

[37] S. Wappler and J. Wegener, "Evolutionary unit testing of object-oriented software using strongly-typed genetic programming," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM, 2006, pp. 1925–1932.

[38] J. C. B. Ribeiro, "Search-based test case generation for object-oriented java software using strongly-typed genetic programming," in *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*. ACM, 2008, pp. 1819–1822.

[39] S. Luke, C. Hohn, J. Farris, G. Jackson, and J. Hendler, "Co-evolving soccer softbot team coordination with genetic programming," in *Robot Soccer World Cup*. Springer, 1997, pp. 398–411.

[40] H. Al-Sahaf, A. Song, K. Neshatian, and M. Zhang, "Two-tier genetic programming: Towards raw pixel-based image classification," *Expert Systems with Applications*, vol. 39, no. 16, pp. 12 291–12 301, 2012.

[41] B. Evans, H. Al-Sahaf, B. Xue, and M. Zhang, "Evolutionary deep learning: A genetic programming approach to image classification," in *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018, pp. 1–6.

[42] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*. Springer, 2000, pp. 1–15.

[43] L. Breiman, "Bias, variance, and arcing classifiers," 1996.

[44] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, 1990.

[45] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[46] P. Melville and R. J. Mooney, "Diverse ensembles for active learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, pp. 584–591.

[47] U. Bhowan, M. Johnston, M. Zhang, and X. Yao, "Evolving diverse ensembles using genetic programming for classification with unbalanced data," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 3, pp. 368–386, 2013.

[48] T. M. Mitchell *et al.*, "Machine learning. wcb," 1997.

[49] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[50] M. Michailidis, "Investigating machine learning methods in recommender systems," Ph.D. dissertation, UCL (University College London), 2017.

[51] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[52] ——, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[53] I. Barandiaran, "The random subspace method for constructing decision forests," *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[54] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[55] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[56] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016, pp. 785–794.

[57] G. An, "The effects of adding noise during backpropagation training on a generalization performance," *Neural computation*, vol. 8, no. 3, pp. 643–674, 1996.

[58] "Lecture 4: Noise notes," http://web.mit.edu/6.02/www/f2010/handouts/lectures/L4-notes.pdf, September 2010.

[59] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[60] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *arXiv preprint arXiv:1611.03530*, 2016.

[61] M. Zhang and W. Smart, "Using gaussian distribution to construct fitness functions in genetic programming for multiclass object classification," *Pattern Recognition Letters*, vol. 27, no. 11, pp. 1266–1274, 2006.

[62] C. Downey, M. Zhang, and J. Liu, "Parallel linear genetic programming for multi-class classification," *Genetic Programming and Evolvable Machines*, vol. 13, no. 3, pp. 275–304, 2012.

[63] W. B. Langdon and B. F. Buxton, "Genetic programming for combining classifiers," in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 2001, pp. 66–73.

[64] W. B. Langdon, S. J. Barrett, and B. F. Buxton, "Genetic programming for combining neural networks for drug discovery," in *Soft Computing and Industry*. Springer, 2002, pp. 597–608.

[65] ——, "Combining decision trees and neural networks for drug discovery," in *Genetic Programming*, J. A. Foster, E. Lutton, J. Miller, C. Ryan, and A. Tettamanzi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 60–70.

[66] A. Majid, A. Khan, and A. M. Mirza, "Combination of support vector machines using genetic programming," *International Journal of Hybrid Intelligent Systems*, vol. 3, no. 2, pp. 109–125, 2006.

[67] A. Khan, A. Majid, and A. M. Mirza, "Combination and optimization of classifiers in gender classification using genetic programming," *International Journal of Knowledge-based and Intelligent Engineering Systems*, vol. 9, no. 1, pp. 1–11, 2005.

[68] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 2962–2970.

[69] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[70] A. Zheng, "Evaluating machine learning models," *Dosegljivo: https://www. oreilly. com/ideas/evaluating-machinelearning-models/page/5/hyperparameter-tuning.[Dostopano: 20. 8. 2017]*, vol. 40, 2015.

[71] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "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*. ACM, 2013, pp. 847–855.

[72] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore, "Evaluation of a tree-based pipeline optimization tool for automating data science," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM, 2016, pp. 485–492.

[73] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[74] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[75] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 826–830, 2017.

[76] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[77] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[78] R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, J. H. Moore *et al.*, "Automating biomedical data science through tree-based pipeline optimization," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2016, pp. 123–137.

[79] A. G. de Sá, W. J. G. Pinto, L. O. V. Oliveira, and G. L. Pappa, "Recipe: a grammar-based framework for automatically evolving classification pipelines," in *European Conference on Genetic Programming*. Springer, 2017, pp. 246–261.

[80] H. Jin, Q. Song, and X. Hu, "Efficient neural architecture search with network morphism," *arXiv preprint arXiv:1806.10282*, 2018.

[81] C. Weill, J. Gonzalvo, V. Kuznetsov, S. Yang, S. Yak, H. Mazzawi, E. Hotaj, G. Jerfel, V. Macko, M. Mohri, and C. Cortes, "Adanet: Fast and flexible automl with learning guarantees," https://github.com/tensorflow/adanet, 2018.

[82] L. Sweeney, "Discrimination in online ad delivery," *Commun. ACM*, vol. 56, no. 5, pp. 44–54, May 2013.

[83] T. Bolukbasi, K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai, "Man is to computer programmer as woman is to homemaker? debiasing word embeddings," in *Advances in Neural Information Processing Systems*, 2016, pp. 4349–4357.

[84] F. K. Došilović, M. Brčić, and N. Hlupić, "Explainable artificial intelligence: A survey," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2018, pp. 0210–0215.

[85] J. W. Tukey, *Exploratory data analysis*. Reading, Mass., 1977, vol. 2.

[86] A. Vellido, J. D. Martín-Guerrero, and P. J. Lisboa, "Making machine learning models interpretable." in *ESANN*, vol. 12. Citeseer, 2012, pp. 163–172.

[87] W. Samek, T. Wiegand, and K.-R. Müller, "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models," *arXiv preprint arXiv:1708.08296*, 2017.

[88] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, June 2010, pp. 2528–2535.

[89] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, 2016, pp. 1135–1144.

[90] O. Bastani, C. Kim, and H. Bastani, "Interpretability via model extraction," *arXiv preprint arXiv:1706.09773*, 2017.

[91] B. Kim and F. Doshi-Velez, "Icml 2017 tutorial on interpretable machine learning." [Online]. Available: http://people.csail.mit.edu/beenkim/icml_tutorial.html

[92] C. Molnar, "Interpretable machine learning," *A Guide for Making Black Box Models Explainable*, 2018.

[93] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," *arXiv preprint arXiv:1702.08608*, 2017.

[94] M. Craven and J. W. Shavlik, "Extracting tree-structured representations of trained networks," in *Advances in neural information processing systems*, 1996, pp. 24–30.

[95] B. Letham, C. Rudin, T. H. McCormick, D. Madigan *et al.*, "Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model," *The Annals of Applied Statistics*, vol. 9, no. 3, pp. 1350–1371, 2015.

[96] H. Yang, C. Rudin, and M. Seltzer, "Scalable bayesian rule lists," *arXiv preprint arXiv:1602.08610*, 2016.

[97] C. Bucilu, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 535–541.

[98] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[99] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine learning*, vol. 51, no. 2, pp. 181–207, 2003.

[100] A. Chandra and X. Yao, "Ensemble learning using multi-objective evolutionary algorithms," *Journal of Mathematical Modelling and Algorithms*, vol. 5, no. 4, pp. 417–445, 2006.

[101] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, 2012.

[102] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[103] G. D. Ruxton, "The unequal variance t-test is an underused alternative to student's t-test and the mann–whitney u test," *Behavioral Ecology*, vol. 17, no. 4, pp. 688–690, 2006.

[104] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the american statistical association*, vol. 32, no. 200, pp. 675–701, 1937.

[105] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[106] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[107] B. Widrow, D. E. Rumelhart, and M. A. Lehr, "Neural networks: applications in industry, business and science," *Communications of the ACM*, vol. 37, no. 3, pp. 93–106, 1994.

[108] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

[109] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian journal of statistics*, pp. 65–70, 1979.

[110] C. Perlich, F. Provost, and J. S. Simonoff, "Tree induction vs. logistic regression: A learning-curve analysis," *Journal of Machine Learning Research*, vol. 4, no. Jun, pp. 211–255, 2003.

[111] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore, "Evaluation of a tree-based pipeline optimization tool for automating data science," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. ACM, 2016, pp. 485–492.

[112] R. M. Bell, Y. Koren, and C. Volinsky, "All together now: A perspective on the netflix prize," *Chance*, vol. 23, no. 1, pp. 24–29, 2010.

[113] M. Črepinšek, S.-H. Liu, and M. Mernik, "Exploration and exploitation in evolutionary algorithms: A survey," *ACM Computing Surveys (CSUR)*, vol. 45, no. 3, pp. 35:1–35:33, 2013.

[114] A. E. Eiben and C. A. Schippers, "On evolutionary exploration and exploitation," *Fundamenta Informaticae*, vol. 35, no. 1-4, pp. 35–50, 1998.

[115] L. Hansheng and K. Lishan, "Balance between exploration and exploitation in genetic search," *Wuhan University Journal of Natural Sciences*, vol. 4, no. 1, pp. 28–32, 1999.

[116] D. Karaboga and B. Basturk, "On the performance of artificial bee colony (abc) algorithm," *Applied soft computing*, vol. 8, no. 1, pp. 687–697, 2008.

[117] W. M. Spears, "Crossover or mutation?" in *Foundations of genetic algorithms*. Elsevier, 1993, vol. 2, pp. 221–237.

[118] W. B. Langdon, *Genetic programming and data structures: genetic programming+ data structures= automatic programming!* Springer Science & Business Media, 2012, vol. 1.

[119] K. A. De Jong and W. M. Spears, "A formal analysis of the role of multi-point crossover in genetic algorithms," *Annals of mathematics and Artificial intelligence*, vol. 5, no. 1, pp. 1–26, 1992.

[120] L. J. Eshelman, "Biases in the crossover landscape," *ICGA'89*, pp. 10–19, 1989.

[121] N. F. McPhee and N. J. Hopper, "Analysis of genetic diversity through population history," in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*. Morgan Kaufmann Publishers Inc., 1999, pp. 1112–1120.

[122] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[123] C. Luan and G. Dong, "Experimental identification of hard data sets for classification and feature selection methods with insights on method selection," *arXiv preprint arXiv:1703.08283*, 2017.

[124] R. S. Olson and J. H. Moore, "Identifying and harnessing the building blocks of machine learning pipelines for sensible initialization of a data science automation tool," *arXiv preprint arXiv:1607.08878*, 2016.

[125] I. Bose and R. K. Mahapatra, "Business data mininga machine learning perspective," *Information & management*, vol. 39, no. 3, pp. 211–225, 2001.

[126] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1721–1730.

[127] G. D. P. Regulation, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46," *Official Journal of the European Union (OJ)*, vol. 59, no. 1-88, p. 294, 2016.

[128] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[129] Y. Jin and B. Sendhoff, "Pareto-based multiobjective machine learning: An overview and case studies," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 3, pp. 397–415, 2008.

[130] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," *TIK-report*, vol. 103, 2001.

[131] M. Zhang and W. Smart, "Multiclass object classification using genetic programming," in *Workshops on Applications of Evolutionary Computation*. Springer, 2004, pp. 369–378.

[132] T. Loveard and V. Ciesielski, "Representing classification problems in genetic programming," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 2. IEEE, 2001, pp. 1070–1077.

[133] T. H. team, *h2o: Python Interface for H2O*, 2015, python package version 3.1.0.99999. [Online]. Available: http://www.h2o.ai

[134] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, "Openml: Networked science in machine learning," *SIGKDD Explorations*, vol. 15, no. 2, pp. 49–60, 2013. [Online]. Available: http://doi.acm.org/10.1145/2641190.2641198

[135] T. Madl, "Sklearn interpretable tree," Feb 2018. [Online]. Available: https://github.com/tmadl/sklearn-interpretable-tree

[136] U. Fayyad and K. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *IJCAI*, 1993, pp. 1022–1029.

[137] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural computing*, vol. 1, no. 1, pp. 3–52, 2002.

[138] J. P. Shaffer, "Multiple hypothesis testing," *Annual review of psychology*, vol. 46, no. 1, pp. 561–584, 1995.

[139] H. K. Dam, T. Tran, and A. Ghose, "Explainable software analytics," in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*. ACM, 2018, pp. 53–56.

[140] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *Proceedings of the 22nd International Conference on Machine Learning*. ACM, 2005, pp. 625–632.