A Genetic Algorithm Based Model Tree Forest

Werner Van der Merwe
20076223

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Industrial Engineering) in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof. A.P. Engelbrecht

March 2023

Declaration

Date: 15 November 2022

Abstract

Abstract

This thesis presents an ensemble approach that reduces the high variance error exhibited by model trees that comprise multivariate nonlinear models and increases the overall robustness of model trees. The ensemble approach is conceptualised, tuned for, and evaluated against competing regression ensemble models on ten separate benchmarking datasets. The ensemble, referred to as the model tree forest (MTF), incorporates a hybrid genetic algorithm approach to construct structurally optimal polynomial expressions (GASOPE) within the leaf nodes of greedy induced model trees that form the base learners of the ensemble. Bootstrap aggregation, together with the implementation of randomised feature space splits during tree induction, sufficiently decorrelates the base learners within the ensemble. Thereby, the variance error of MTF is reduced compared to that of a single model tree, whilst the favourable low bias error of model trees is retained. The multivariate nonlinear models that predict the output enable MTF to produce approximations of highly nonlinear data. The addition of ensembling methods passively combat overfitting brought forth by the increased model complexity, compared to the previous implementation of GASOPE within a tree structure, which exhibits overfitting in specific cases. MTF produced a similar predictive accuracy to the random forest method and outperformed an artificial feed-forward neural network ensemble, an ensemble of M5 model trees, and ensembled support vector regression models. However, the computational cost of the MTF induction algorithm is up to four orders of magnitude greater than RF.

Opsomming

Hierdie tesis stel 'n groeperingsmetode voor wat die hoë variansie fout van modelbome, met hoër orde uitset funksies, verminder. Die groeperingsmetode verhoog ook die robuustheid van die modelbome. Hierdie groeperingsmetode word na verwys as "model tree forest"(MTF). MTF word volledig beskryf, verstel en evalueer teen ander regressie modelle op tien verskillende datastelle. MTF behels 'n genetiese algoritme wat optimale polinoom funksies binne die blaar nodes van die gierige geïnduseerede modelbome kweek. Die gebruik van "bootstrap aggregation"gesaamentlik met lukraak-bepaalde vertakkings gedurende die kweekingsproses van 'n enkele modelboom, verseker dat die varianse fout van MTF verlaag word. Terselfdetyd word die gemiddelde afwykingsfout van die modelbome, wat reeds baie laag is, laag gehou. Die polinoom funksies wat die skatting akkuraatheid van MTF bepaal, laat toe dat die gesaamentlike model data, wat hoogs nie-liniêr is, goed naboots. Die polinoom funksies help ook om te keer dat die modelbome nie die datastel oorpas nie, wat andersins die geval sou wees as gevolg van die hoë kompleksiteit wat modelbome besit. MTF het gelykstaande resultate gelewer aan 'n lukrake woud, en beter resultate as 'n groepering van neurale netwerke, 'n groepering van M5 model bome en 'n groepering van ondersteunende vektor regressie. Die tydkoste van 'n MTF model kweek is tot vier eksponentiële ordes grootter as 'n lukrake woud.

Acknowledgements

I would like to hereby acknowledge everyone who helped me throughout the past three years in writing this thesis. Professor Andries Engelbrecht, who provided me with guidance. My family and friends for their encouragement. The School for Data Science and Computational Thinking for supporting me. Without them, this would not have been possible. Thank you.

Contents

List of Figures

List of Tables

Nomenclature

Variables and functions

A	intersection of two sets of term-coefficient mappings.
B	union of the exclusion between two sets of term-coefficient mappings.
c	model complexity.
c_s	ensemble pruning factor.
D	set of training instances comprising input features and target values.
$D(\cdot)$	euclidean distance between two vectors.
d	distance value.
$\boldsymbol{e}$	subset of polynomial orders.
f	given order within e.
$f_1, f_2, ..., f_i$	functions defining the output of node i.
g	given feature index from the vector of m inputs.
$g_i(\boldsymbol{x})$	predicted value of model tree i given the input vector $\boldsymbol{x}$.
G	number of generations.
h	uniformly distributed random number.
h_b	bandwidth parameter.
I	set of unique term-coefficient mappings.
i, j, k	generic counter variables/set indices.
$K_1(\cdot), K_2(\cdot)$	given kernel functions.
k_{nn}	number of nearest neighbours for evaluation.
m	number of input features.
N	set defining the children and split of a node.
n	number of instances within a given set.
n_l	number of base learners within an ensemble.
n_t	number of decision trees within an ensemble.
o	degree of the polynomial order for a given function.
P	set comprising a population of individuals.
p	maximum number of terms.

q	given instance.
R^2	fitness function of a given individual.
S	set defining the splitting feature and value.
s	split value.
s_c	specified numeric smoothing constant.
$s(\cdot)$	smoothing function.
$sd(D)$	standard deviation of the target values within set D.
T	set of unique variable-order pairs.
$Var(X, \boldsymbol{y})$	variance of a given model w.r.t the set X and target values $\boldsymbol{y}$.
$\boldsymbol{w}$	vector of weights.
w_m	real-valued polynomial coefficient of the m^{th} input feature.
X	set of instances.
$\boldsymbol{x_i}$	vector of input values of instance i.
y_i	target value of instance i.
y'_i	predicted value of instance i.
Z	set of nodes.
α	fuzzy parameter.
Δ_ϵ	expected error reduction.
ϵ	error criterion.
λ_j	whole-valued polynomial order of the input variable $x_j, j \in \{1, \ldots, m\}$.
$\mu(\cdot)$	Gaussian function.
$\rho(X)$	correlation with respect to the set X.
$\sigma^2(\boldsymbol{y})$	variance over the vector $\boldsymbol{y}$.

Acronyms and abbreviations

AG	additive groves
CART	classification and regression trees
BLMC	baseline linear model comparison
CASP	physicochemical properties of protein tertiary structure dataset
GASOPE	genetic algorithm approach to evolve optimal polynomial expressions
GPMCC	genetic programming approach to extract symbolic rules from data sets with continuous-valued classes
GPR	Gaussian process regression
HTL	hybrid regression tree
LM	Leven-Marquardt
MAE	mean absolute error
MARS	multivariate adaptive regression spline
MSE	mean square error
MT	model tree forest base learner
MTE	fuzzified M5 model tree ensemble
MTF	model tree forest
M5	piecewise linear model tree
M5E	bagged M5 model tree ensemble
NN	artificial feed-forward neural network
NNE	artificial feed-forward neural network ensemble
PLT	partial linear tree
ReLU	rectified linear unit
RF	random forest
RMSE	root mean square error
RMT	random model trees
SGD	stochastic gradient descent
SVR	support vector regression
SVRE	support vector regression ensemble
UCI	University of California at Irvine

Chapter 1

Introduction

Machine learning, regarded as a subset of artificial intelligence, is quite an old research field. Many machine learning algorithms referred to in this thesis and still used to this day were first conceptualised in the late 1990s and the early 2000s [6, 7, 24, 30]. Computers have since been able to produce increased computing power to allow for larger-scale application of these algorithms. Furthermore, the available computing power continues to increase each year. With the increase in computing power comes an increase in interest in machine learning research and applications. The development of more complex models that utilise the increased computing power to better capture highly intricate patterns in data is now possible. However, it is vital that the complexity of a model be finely tuned given the problem at hand because an increase in complexity does not guarantee an increase in performance for all cases.

This chapter is outlined as follows. Section 1.1 provides further motivation for the research into model tree ensembles that comprise higher-order polynomial output functions. Section 1.2 presents the research objectives of this thesis. The main contribution of this thesis is listed in Section 1.3. Finally, Section 1.4 presents the outline of this thesis.

1.1. Motivation

There are two different types of problems in machine learning that predictive models are tasked to solve, namely classification and regression problems. The former focuses on the prediction of a class label given a set of input features. The latter refers to the prediction of a target feature that takes on a numerical value from a set of input features and is the focus of this thesis. A problem observed today is the size of data and the intricacy growing to a point where older predictive models, which were conceptualised long before data reached this point, may struggle to predict the target feature effectively. Moreover, the predictive models available today which have the capability to model these nonlinear relationships between input and target features within large datasets, such as artificial neural networks, suffer from a lack of opaqueness, i.e. the models function as a black box model wherein the rules on which the model bases its prediction on are unknown to the user. For these reasons, research on predictive models which have the capability to model complex relationships between the input and target features within data and provide transparency behind the prediction is a valuable commodity.

To address the abovementioned problem, Engelbrecht and Potgieter conceptualised a genetic programming approach to extract symbolic rules (GPMCC) from regression problems [23].

A genetic algorithm approach to construct optimal polynomial expressions (GASOPE) was implemented during induction to evolve the output functions of GPMCC [22]. The advantages of GPMCC are the ability to adequately model the nonlinear relationships between the input and target features in data, whilst the set of rules on which the model is based are visible. However, GPMCC was outperformed by a competing model tree on data with linear relationships between the input and target features [23]. GPMCC is a model tree that comprises higher-order polynomial functions and therefore is likely to produce poor approximations of data with strictly linear relationships between input and target features. Furthermore, the computational cost of the induction of GPMCC is significant due to the genetic programming approach. Therefore, the application of GPMCC within an ensemble is not pursued, and the benefits of the use of GPMCC within an ensemble remain unknown. For decision trees, ensemble techniques significantly increase the robustness of the collective model [6, 7]. Robustness refers to the ability of the model to produce good performance on training data and unseen data.

The majority of research on model tree ensembles is only focused on the M5 model tree, a model tree which utilises linear output functions to piecewise model nonlinear relationships between input and output features [1, 21, 25]. The benefit of linear output functions is the reduced computational cost of their induction, which in turn allows for multiple model trees to be grown within an ensemble. Through the use of model trees as base learners, it was found that ensemble models were more robust than a single base learner [1, 21, 25]. For a complex regression model, such as GPMCC, robustness is a valuable aspect which may bring increased performance. Therefore, it is beneficial to investigate the feasibility of the application of ensemble techniques to model trees that comprise robust higher-order polynomial output functions to retain the increased accuracy.

This thesis utilises the increased computational power available today, to redesign the application of GASOPE within a model tree to allow for the use of GASOPE within an ensemble. Through the use of an ensemble of model trees that comprise higher-order output functions, the observed variance error is reduced while the favourable low bias error that model trees offer is retained.

The ensemble of model trees that comprise higher-order output functions proposed in this thesis is referred to as a model tree forest (MTF). For an MTF to function to the best of its ability, it is essential that the induction algorithm be optimised through hyperparameter tuning. Several questions arise when a decision tree is used as the base learner of an ensemble: What is the optimal size of the ensemble? How large should trees be allowed to grow? How can overfitting be prevented? Which type of data best suits the application of MTF? These questions are all answered through the empirical investigation and evaluation of model trees on a set of benchmarking datasets.

1.2. Research objectives

The main goal of this thesis is to thoroughly research, design and implement a model tree that comprises higher-order polynomial output functions, within an ensemble approach. To achieve the aforementioned goal, the following research objectives are defined:

- Provide an overview of existing decision tree and ensemble methods as well as how the ensembling of decision trees addresses the bias-variance tradeoff.
- Investigate the use of model trees that comprise higher-order output functions in the currently available literature as well as identify the ensemble methods used for model trees and the implications thereof.
- Redesign the original GASOPE with the improved computational power available today. As well as implement GASOPE to evolve nonlinear output functions for a model tree and ultimately an ensemble of model trees.
- Design and optimise the induction algorithm of computationally cost-effective model trees with higher-order polynomial output functions within an ensemble.
- Statistically test and compare the performance of MTF against other state-of-the-art regression models on a set of benchmarking datasets.

1.3. Contributions

The main contributions of this thesis are:

1. A redesigned implementation of GASOPE which utilises the increased computational power now available.
2. The design and optimisation of decorrelated, bootstrap aggregated model trees that comprise higher-order polynomial output functions on a set of benchmarking datasets.
3. The empirical evaluation and statistical testing of MTF against other state-of-the-art regression models on a set of benchmarking datasets.
4. The finding that MTF can produce a competitive performance on both linear as well as nonlinear problems, given that the maximum polynomial order hyperparameter is properly tuned for the problem at hand. Furthermore, it is unfavourable in both predictive accuracy and computation cost for the maximum polynomial to exceed three.
5. Large datasets favour the use of deeper model trees within an ensemble, whereas small data favours the use of shallow model trees within an ensemble.
6. The finding that MTF produces similar predictive accuracy to a random forest, but outperforms ensembled neural networks, ensembled support vector regression, and an M5 model tree ensemble on the majority of ten benchmarking datasets.

1.4. Thesis outline

The remainder of this thesis is structured as follows:

- **Chapter 2** discusses the required information to thoroughly understand the implementation of decision trees and how predictive models address the bias-various tradeoff. Furthermore, existing implementations of model trees and model tree ensembles are discussed together with their shortcomings.
- **Chapter 3** presents and discusses the redesigned version of GASOPE and the statistical tests used to evaluate the performance of the redesigned GASOPE against other state-of-the-art regression techniques.
- **Chapter 4** discusses the entire induction algorithm of MTF and motivates the use of decorrelated model trees within a bootstrap aggregated ensemble.
- **Chapter 5** discusses the experimental process used to both tune and compare the performance of MTF against a selection of ensemble models.
- **Chapter 6** presents and discusses the tuning and comparison results for MTF when compared to the ensemble models on a set of benchmarking datasets. Additionally, statistical tests are performed on the results.
- **Chapter 7** concludes the thesis with a summary of each chapter's findings and conclusions. This chapter also provides avenues for possible future work.

The following appendices are provided to supplement the results summarised in the chapters:

- **Appendix A** lists the optimisation results in full of the maximum polynomial order hyperparameter for each database.
- **Appendix B** lists the optimisation results in full of the maximum tree depth hyperparameter for each dataset.
- **Appendix C** lists the optimisation results in full for the ensemble size hyperparameter for each dataset.

Chapter 2

Literature review

This chapter presents the background of model trees and the ensemble methods required to implement the model tree forest presented in the following chapters. There is no single conceptualised model tree, but many different implementations of the idea of a model tree. This is because any predictive machine learning model can be incorporated within the leaf nodes of a model tree. Model trees are a type of decision tree more commonly used for regression problems. Hence, decision trees are first discussed in this chapter. An understanding of the concept of the bias-variance tradeoff is necessary to motivate the use of decision trees in ensembles.

This chapter is structured as follows: Section 2.1 provides a background of decision trees and discusses how decision trees are induced. The bias-variance tradeoff and two ensemble methods, *bagging* and *boosting*, are introduced and discussed in Section 2.2. Section 2.3 presents the differences between a model tree and other decision trees and a general discussion on the performance of the piecewise linear model tree (M5). Two different model trees, i.e. a hybrid regression tree (HTL) and a genetic program for mining continuous classes, which incorporate nonlinear models at their leaf nodes, are discussed in Section 2.4. Finally, a decision tree ensemble and two model tree ensembles are introduced in Section 2.5.

2.1. Decision trees

This section introduces the concept of decision trees in the context of machine learning. Section 2.1.1 discusses decision trees in the application as predictive models for classification and regression problems. Section 2.1.2 introduces the methodology of classification and regression trees (CART), conceptualised by Breiman et al. [5]. Finally, Section 2.1.3 discusses the induction process employed by CART to induce regression or classification trees.

2.1.1. Introduction to decision trees

Decision trees are tree-like predictive models used in machine learning for classification and regression problems. Decision trees are induced on labelled datasets to model the relationship between input and output variables. The output variable represents the target value that the decision tree predicts. Classification and regression applications of decision trees are characterised by their output variable type:

- classification trees are used to predict a categorical output, while

- regression trees predict a numerical output.

Categorical features are qualitative because they take on labelled values, and numerical features are quantitative because they take on real values.

Decision trees were inspired by the *divide-and-conquer* paradigm, which entails the recursive breaking of an intricate problem down into smaller sub-problems which each can be solved more simply. A decision tree organises data recursively through a hierarchical representation of tests on input variables, The hierarchical representation gives the predictive model the appearance of a tree data structure. Each decision tree houses a series of tests in the nodes of the tree structure. Data is divided in each test through branch-like splits that each represents a different outcome. Alternatively, decision trees are expressed as a collection of if-else statements to model decision boundaries in data.

Before the conception of decision trees, statistical techniques used for classification problems were developed with small datasets in mind [5]. Furthermore, the variables were all of the same types and the data was often assumed to be homogeneous, i.e. the same relationships between these variables held throughout the entirety of the decision space. For that reason, simple models that focused only on a few parameters to model the influence of a wide range of factors made up the majority of conceptualised models at the time [5]. The application of these simple models to larger datasets, with the assumption that the data retains its homogeneous structure, may be fallacious.

Datasets can be both large and complex. The complexity of data is influenced by the following: missing values, noise, outliers, high dimensionality, mixtures of data types, non-parametric distribution and non-homogeneity. Non-parametric distributed data refers to data which does not fit a known distribution. Non-homogeneous data is data where different relationships exist between variables within different parts of the decision space. *The curse of dimensionality* states that the more features present, the higher the variance of the data points is. Hence, the data is sparser and more spread out. The dimensionality of data can be reduced. However, the accompanying drawbacks are unfavourable. The drawbacks include the loss of data and a reduction in the ability to distinguish the influence of individual features. Most importantly, the interpretability of data is significantly reduced with dimension reduction.

There was a need for a method in which the most important features of the data are accentuated, the background noise disregarded and the conclusions interpretable to the analyst [5], which brought rise to the conceptualisation of the decision tree. The defining characteristic of decision trees is the ability to not only produce satisfactory results but to also give the user thoughtful information and insight into the data. For example, for classification problems, a decision tree can help to uncover the predictive nature of a problem. The decision tree helps the user to better understand how specific features contribute to the observed outcome [5].

2.1.2. The CART algorithm

One of the earliest documented works on the theory of decision tree induction was published in 1984 [5]. Breiman et al. were inspired by the deficiencies of existing tree-structured classifiers at the time to develop an improved algorithm which provided a more flexible and accurate solution. Breiman et al. named the proposed decision tree induction algorithm as classification and regression trees (CART). CART induces a tree through a series of binary questions, which when answered sequentially, produces a solution to the problem at hand.

A simplistic classification tree is portrayed in Figure 2.1, can classify an individual's gender based on two input features, namely weight and height. Nodes X_1 and X_3 each represent a test. Tests, also referred to as splits comprise two parts: a feature on which the test is performed and a value that indicates the decision boundary. Node X_1 is referred to as a *root* node because it is the starting node of the decision tree. Nodes X_2, X_4 and X_5 contain a label of the output class. A class label represents the majority class for the instances associated with the leaf node. These nodes are referred to as *leaf* nodes. Leaf nodes terminate all possible *paths* (sequences of nodes) that can be followed in the tree from the root node to any leaf node. *Neighbouring* leaf nodes refer to the adjacent leaf nodes at the end of every path, such as nodes X_2 and X_4 as well as nodes X_4 and X_5.

Each node of the decision tree encapsulates a subset of the dataset. The instances that make up the data subset of a node are characterised by the tests that lead up to the node. When combined, the data subsets that each leaf node of the decision tree denotes cover the entirety of the complete dataset. The *size* of a decision tree refers to the number of nodes it consists of. Finally, *depth* is the number of nodes in the longest path the tree contains. A decision tree is regarded as being *shallow* if its depth is small.

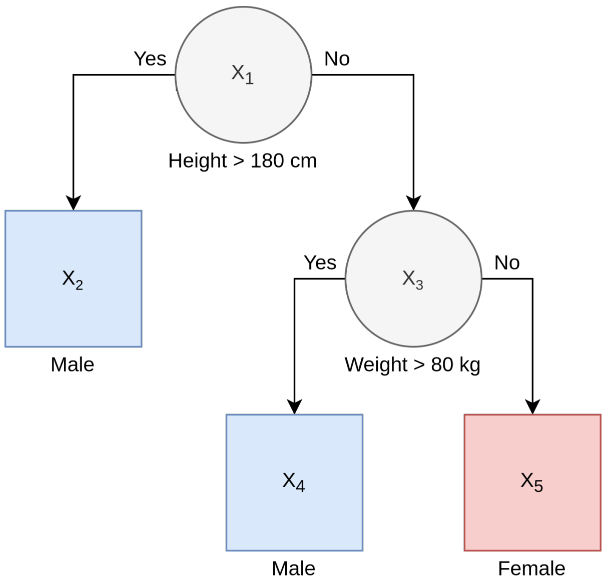


Figure 2.1: A hypothetical two-class tree-structured classifier.

2.1.3. Induction algorithm of CART

CART induces a decision tree for classification, such as shown in Figure 2.1, from a set of labelled instances through three fundamental steps:

1. Determine the tests.

2. Evaluate stopping rules.

3. Assign class labels to each leaf node.

At the root node, step 1 is repeated for each subsequent node until step 2 is satisfied for each path in the decision tree. To construct a decision tree, each split is chosen to minimise the mixture of classes in the descendant nodes' subsets. A splitting function determines the feature as well as the value of a split; multiple splits are compared for a node by the splitting function. A metric often used to express the result of the splitting function is impurity. The larger the impurity, the more heterogeneous the data subset is with respect to class distribution. CART produces binary splits from the value of a single input feature only, i.e. a single split cannot house multiple rules or features. For that reason, splits are always parallel to the decision space axes. CART allows input features to be both discrete and/or continuous.

At any node, the tree induction algorithm evaluates a total of $n_f \times n_i$ candidate splits, where n_f is the number of features and n_i is the number of instances in the dataset. A split is evaluated on every feature and the corresponding feature value for each instance in the dataset. The best-performing split is selected as the one split which minimises the impurity function. Therefore, the best-performing split also maximises the reduction in heterogeneity.

Once a node is reached where no significant decrease in impurity is produced, the node is declared a leaf node. This is one example of a stopping rule, another is to fix the maximum depth of a decision tree. The class that makes up the majority of each leaf is declared as the label of the leaf. The decision tree growing sequence, i.e. the induction algorithm, is thus completed.

In the application of the induction algorithm of CART to regression problems, little changes. The induction algorithm proceeds to partition the dataset into subsets via a series of tests on the input features. However, a different form of impurity metric is incorporated. The expected reduction in either variance or absolute deviation of the instances that make up the data subset for each candidate split is calculated. Thereafter, the splitting function of regression trees maximises the reduction in either variance or absolute deviation. A change is also present in the leaf nodes; instead of a class label, the decision tree now predicts a numerical value associated with a target feature. Recall that a classification tree predicts a categorical value to define a target feature. The constant output value of a leaf node is calculated as the mean squared error of the numerical target values for all instances in the subset of data denoted by the path to the leaf node.

Fundamentally, the numerical variable predicted by the regression tree is a dependent variable based on a multitude of independent variables. The independent variables represent the input features and the dependent variable is the target feature. The numerical values which represent the output in a regression tree are constant per leaf node. Subsequently, the regression tree models the relationship between the input and output variables through discrete piecewise approximation. Figure 2.2 illustrates the predicted constant values of a regression tree, induced on the values of $y = sin(x)$ for $x \in [0, 2\pi]$, with the target feature y being dependent on the input feature x. It is clear that the constant output values of a regression tree fail to properly fit a nonlinear function. Furthermore, constant values create discontinuities in the predicted values of the regression tree. The regression tree creates an unsatisfactory approximation of a sinusoidal wave.

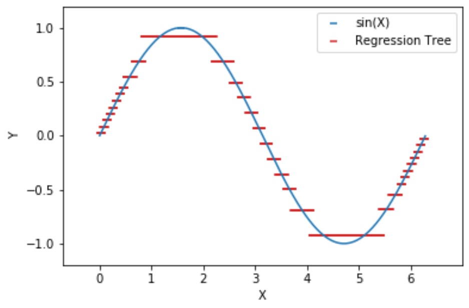


Figure 2.2: A regression tree's piecewise discrete approximation (red) of a continuous sinusoidal function (blue).

Despite the poor observed fit of nonlinear data, CART is a good solution to regression and especially classification problems. The advantages of the CART induction algorithm compared to other statistical models at the time of its conception include [5]:

1. The induction algorithm handles both categorical and/or numerical input as well as output data. A decision tree can thus be induced from a dataset that contains mixed feature types.

2. Once induced, the application of a decision tree on new incoming data is computationally efficient and interpretable.

3. Decision trees exploit the conditional relationships between variables present in non-homogeneous data to better model the instances.

4. The most beneficial information required to interpret the relationships between input and output features is incorporated at a split. Hence, automatic dimensionality reduction is effectively performed on the dataset.

5. CART showed to be quite robust when subjected to outliers and misclassified instances which are considered noise.

2.2. The bias-variance tradeoff

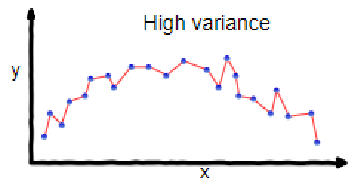
This section introduces the dilemma of the bias-variance tradeoff which is present in all predictive machine learning models. This section also introduces ensemble learning and aims to discuss how ensemble learning can be an effective way to mitigate the problems associated with the bias-variance tradeoff. Section 2.2.1 starts with a discussion of the bias-variance tradeoff and how it explains the tendency of a model to overfit or underfit. Next, Section 2.2.2 describes how the bias-variance tradeoff is specifically present in decision trees. Finally, Sections 2.2.3 and

2.2.4 introduce two popular ensemble methods and discuss how each addresses the bias-variance tradeoff.

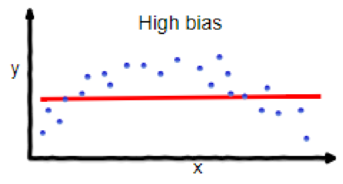
2.2.1. Overfitting and underfitting

The tendency to overfit or underfit given data stems from the degree of inaccuracy present in machine learning models and is explained through the bias-variance dilemma. The bias-variance dilemma is a result of the problem to balance model complexity with the desire to have accurate models. Both bias and variance are degrees of prediction error present in the model, that negatively affect prediction accuracy. Ideally, a machine learning model should exhibit perfect prediction accuracy, but perfect prediction accuracy is infeasible because models are exposed to unseen and often noisy data during testing. Therefore, during training, the focus is on the minimisation rather than the removal of the prediction error. To better explain how to mitigate the prediction error, the two components (bias and variance) must be examined.

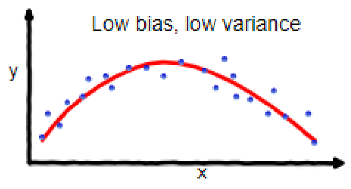
Bias error is defined as the average difference in the prediction that a model makes from each target value. Variance error is defined as the amount of variability or spread that a model is subject to when predicting the output of a given instance. Figure 2.3 illustrates the predicted output of three models as follows: the first model exhibits a high variance error, the second a high bias error and finally, the third has a balance of low bias as well as low variance error. Each model is trained on the same set of instances, shown in blue.



(a) An overly complex model's output.



(b) An overly simple model's output



(c) The output of a model with balanced complexity to the problem at hand.

Figure 2.3: Three separate models' predicted output (red) and target output (blue) of a simple regression problem with a one-dimensional input.

Models that are simple in their composition, such as the model in Figure 2.3b, cannot adequately capture any intricate patterns in the training data. These models are referred to as simple models because the complexity of the output functions used to make predictions are low. Simple models do not have the discriminative ability to fit nonlinear relationships within data. These models produce an oversimplified interpretation of the data. The testing of overly simple models on unseen data results in a high bias error [18]. A model with high bias error *underfits* the training data, i.e. it cannot capture the complex relationships between the input space and target space within the data. Hence, a model that underfits will produce poor prediction accuracy on both training and unseen data.

In contrast, models that incorporate complex algorithms for output functions are more vulnerable to noise in the training data. During training, the overly complex output function causes the model to learn relationships within data that are a product of noise and not true to the nature of the data. Subsequently, a large variance error is present and the model *overfits* the training data, shown through the jagged output in Figure 2.3a. Although the training of complex models may be low, such as the model in Figure 2.3a, the model performs poorly in terms of generalisation error [13]. An overly complex model fallaciously fits noisy patterns in the training data that are not associated with the desired output for all cases [18]. In training data where noise is associated with the target feature, may not properly represent any future unseen data. Hence, poor generalisation accuracy is obtained, similar to models with large bias errors. However, the training accuracy is low.

Thus, to produce favourable prediction accuracy requires that both bias and variance errors to be minimised. However, it is important to note that a model cannot simultaneously be less complex and more complex. Therefore, bias and variance, being respectively proportional to the complexity of a model, are error functions that increase in opposite directions. This tradeoff between bias and variance is portrayed in Figure 2.4. Models are designed to minimise both components of the tradeoff simultaneously. The point of optimal balance between the two errors is desired.

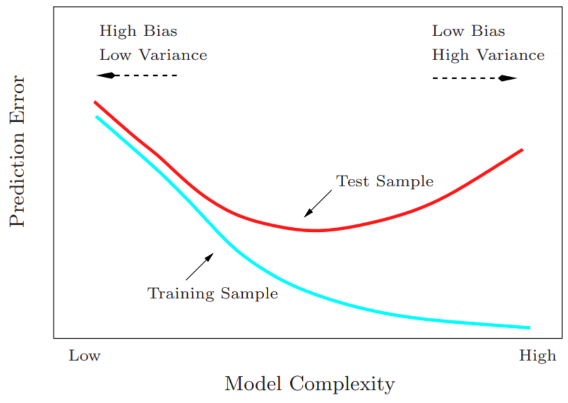


Figure 2.4: The change in a model's prediction error on the training set (red) and generalisation set (blue) as the complexity of the model is varied. Reproduced from [14].

2.2.2. How decision trees address the bias-variance tradeoff

Decision trees are classified as supervised learning models because they are trained to map multiple input variables to an observed output variable. Unsupervised learning models are instead tasked with the identification of patterns in unlabelled data to, for example, group the given data instances into homogeneous regions. In the case of a supervised learning model, the criteria used to assess prediction accuracy are both the training and test set errors. The ability of the model to generalise to unseen data is the most important of the two criteria [13].

When Breiman et al. started their initial work on decision trees, it was clear that arguably the biggest shortcoming was a product of the high variance present in decision trees [5]. Decision trees would often deliver satisfactory results on training sets, but generalised poorly on test sets. During induction, classification trees continuously optimise the classification boundaries by the addition of more nodes. Hence, decision trees are prone to capture noise in the training set the deeper the decision tree grows. This meant that trees were prone to overfitting and require something to remedy this.

The answer Breiman et al. put forth was a fundamental shift in focus. Instead of adjusting the stopping criterion to combat overfitting, the tree was induced with more lenient stopping rules and once fully grown, selectively *pruned*. Decision trees are not capped at a specific depth to ensure a thoroughly deep decision tree was induced. Classification trees are induced to comprise only homogeneous leaf nodes, i.e. all of the instances that a leaf node encapsulates are of the same class. Thereafter, pruning works as follows: starting at the leaf nodes and following the paths upwards into the tree, nodes are evaluated through cross-validation and the subtree below is removed if warranted. Each node along the path is temporarily pruned to a leaf node and the performance of the temporary pruned decision on unseen data is compared to that of the original unpruned decision tree. If the prediction error of the tree is found to have decreased, the node remains pruned. Otherwise, the tree reverts to the previous state before the node was pruned. Pruning showed to lower the variance in classification trees and significantly improve its performance with respect to generalisation [5]. Furthermore, less complex node sets are extracted from the data at hand.

As for regression trees, the tradeoff is more intricate. Because the induction step implements a stopping criterion based on the deviation over all target values within a node, regression trees are prone to overfitting. The stopping criterion is reached when the deviation of the target values is below a threshold. As a result, the tendency of a regression tree to overfit data is determined by the value of the threshold, i.e. the lower the value, the more likely it is that the regression tree is induced to overfit the data. However, the constant output value at a leaf node can be argued to underfit the subset of data denoted by the path to the leaf node. Figure 2.2 shows how poorly constant values fit the shape of a sinusoidal wave. This underfitting is only applicable to the subset of data though. If the regression tree is evaluated as a whole, a high variance error remains observed. Therefore, regression trees are pruned similarly to classification trees to reduce the variance error.

2.2.3. Ensemble learning strategies

An alternative method to pruning with the intent of minimising the variance of decision trees is the use of ensemble learning. The premise of ensemble learning is to develop multiple models in unison, which together produce a better result than the result an individual model is capable of producing.

One of the simplest decision tree ensemble methods is bootstrap aggregation, referred to as *bagging* [6]. Bagging is when several decision trees are induced in parallel, each on a subset of the training data. A subset is generated through random sampling of the dataset with replacement. Subsets of instances are always chosen from the entire dataset and the act of choosing an instance does not remove it from the dataset. For regression applied bagging, the final prediction of the ensemble is derived by averaging the predictions of each tree within the ensemble. In the case of classification, the majority vote is taken instead of the average predicted value. Note that this is simply the specific strategy bagging employs to predict the output.

The randomness in the data subsets on which the decision trees are induced contributes to the minimisation of the variance of the collective model. The induced splits within decision trees are highly sensitive to changes in data because of the greedy induction algorithm, especially in the case of noisy data. A greedy algorithm is a heuristic approach that makes choices that result in local optima. As a result, the induction of decision trees on random subsets produces dissimilar decision trees. The variance, $Var(\cdot)$, in the prediction a bagged ensemble produces for a given set of instances is decomposed into two terms and defined as [18]

$$\mathrm{Var}(X, \boldsymbol{y}) = \rho(X)\sigma^2(\boldsymbol{y}) + \frac{(1 - \rho(X))\sigma^2(\boldsymbol{y})}{n_l} \tag{2.1}$$

where ρ is the sampling correlation between the prediction of any two decision trees in the ensemble, σ^2 is the sampling variance over the target values of any single, randomly drawn, decision tree; n_l is the number of learners in the ensemble, in this case, decision trees. Finally, X represents a set of instances that a given decision tree uses to predict values and $\boldsymbol{y}$ a vector of target values. The set of instances is assumed to be a set of independent and identically distributed random variables. Due to the sensitivity of splits and the injection of randomness in the tree induction algorithm, decision trees within the ensemble are divergent from one another, i.e. $\rho(X) < 1$. The more random effects are introduced, the more ρ tends to 0. Thereby, Equation (2.1) is reduced to only $\frac{\sigma^2(\boldsymbol{y})}{n_l}$.

It is evident that as the number of decision trees in the ensemble increases, the total variance decreases further. The variance of the ensemble is thus strictly less than the variance of an individual tree. Furthermore, the combination of randomised models does not affect the bias error [14]. Therefore, decision trees are ideal for bagging, because they exhibit a very low bias individually. Bagged decision trees produce a model that is both low in bias and variance, which improves the prediction accuracy for both training and generalisation instances significantly over that of a single tree.

2.2.4. An alternative ensemble strategy for decision trees

In contrast to bagging is the *boosting* method. Boosting is fundamentally different from bagging; boosting employs a sequential learning strategy as opposed to the parallel induction of bagging. The models which make up a boosted ensemble are developed with a higher bias than variance error in mind and are referred to as weak learners. A weak learner is only slightly better at the prediction of a target value compared to a random guess [14]. In the context of decision trees, an example strategy of a decision tree induced to have a high bias error is to induce a shallow tree, such as a decision stump that comprises a single split. Weak learners are incapable of individually modelling complex observed relationships. However, weak learners collectively form a strong learner capable of adequately modelling more complicated patterns in data.

Sequential boosting induces a weak learner on repeatedly modified versions of the data. Data is modified through weights that are assigned to each training instance. Weights represent the probability of an instance being chosen as part of a subset on which a weak learner is induced [20]. After each additional weak learner is added to the ensemble, larger weights are assigned to the instances which have the highest contribution to the prediction error. Each successive weak learner thereby prioritises the improvement of the prediction error of the larger weighted instances. Through this sequential optimisation, the bias error of the collective model is reduced compared to that of a single weak learner. Once the induction of the entire ensemble is complete, weights are assigned to the outputs of each weak learner based on the prediction error.

Bagging and boosting have fundamentally different approaches to addressing the bias-variance tradeoff. To conclude, boosting reduces bias with an already low variance, whilst bagging reduces variance with an already low bias.

2.3. Model trees

This section discusses how model trees are an extension of regression trees conceptualised to produce an increased performance on regression problems. The section is outlined as follows. Section 2.3.1 elaborates on the need that brought rise to the conceptualisation of model trees. The induction algorithm of the M5 model tree is described in Section 2.3.2. Section 2.3.3 discusses the first published performance results of the M5 model tree. The linear models that the M5$'$ model tree incorporates are described in Section 2.3.4. Finally, Section 2.3.5 discusses the drawbacks associated with model trees that incorporate strictly linear models.

2.3.1. The limited capability of regression trees

The early work of Breiman et al. that improved the robustness of decision trees showed great success concerning the application of decision trees to classification problems [5]. However, the focus Breiman subsequently put on classification trees [6, 7] meant that regression applications of decision trees still left much to be desired. A big drawback of regression trees remained the discontinuous output that it produced through the discrete approximation of the mapping between inputs and outputs of a regression problem, as illustrated in Figure 2.2. This meant that regression trees, developed through the CART algorithm, had limited capability of adequately predicting the target value for continuous target features.

Before the conception of model trees, problems where the predicted value took on a continuous numeric value, favoured the use of learning techniques capable of producing continuous numerical predictions. Linear regression or neural networks are two example models capable of producing a continuous numerical output. However, no technique comes without its drawbacks and both linear regression and neural networks are no exception to this. Linear regression has limited capability because it imposes a linear relationship on the data Neural networks do not provide the user with an insight into the relationship of the data, due to its problem of opacity [30].

There was still a need for a learning model capable of performing nonlinear regression and providing its user with insights into the relationship among descriptive features that results in specific predictions. M5 model trees were first developed by Quinlan in 1992 as an extension of the regression trees of Breiman et al., with the fundamental difference that the M5 model tree incorporated leaf nodes that are not limited to having a constant prediction value [24]. Model tree leaves can incorporate any multivariate model to better capture the patterns present in the training data. These multivariate models are used to predict a continuous target feature.

Figure 2.5 illustrates how a model tree that comprises linear output models, approximates the continuous sinusoidal function $y = sin(x)$ for $x \in [0, 2\pi]$, as opposed to the regression tree in Figure 2.2. The use of linear models instead of constant values at the leaves allows a model tree to predict a numeric target value without discontinuities. Thus a better approximation of the nonlinear function is produced. The linear models also allow profiling of the regression problem, i.e. describing the conditions on the input features that will result in a specific linear trend.

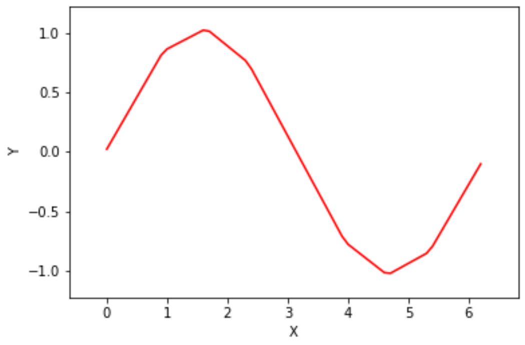


Figure 2.5: The continuous piecewise linear approximation of a sinusoidal wave using a model tree.

2.3.2. M5 model tree induction

The induction of an M5 model tree is quite similar to that of CART, with M5 model tree induction also employing a greedy approach. The first difference between the two methodologies is the splitting criterion used to evaluate a proposed split. Instead of selecting the split that maximises the reduction in variance or absolute deviation, the M5 induction algorithm selects the split which maximises an expected error reduction at a node, defined by [24]

$$\Delta_\epsilon = sd(D) - \sum_i \frac{|D_i|}{|D|} \times sd(D_i) \tag{2.2}$$

where D represents the set of training instances for the data in the node, and $sd(D)$ is the standard deviation of the target values in D. Every potential split is evaluated based on the subset of instances that the split produces. D_i denotes the subset of instances that belong to outcome i of a split and $sd(D_i)$ denotes the standard deviation of the target values in D_i. Therefore, the M5 induction algorithm chooses splits that are locally optimal for each node, similar to the induction of regression trees. After the model tree is induced, linear models are constructed for each node. The reason why linear models are constructed for non-leaf nodes in addition to leaf nodes is to accommodate for *pruning*. The linear model of the parent node is required to calculate the alternate performance of the tree if that leaf node were to be pruned away.

A multivariate linear regression model, at any given node, is constructed with only the instances characterised within the splits of that node and its subtree nodes. Once the linear model is constructed, each parameter of the linear model is evaluated for simplification of the linear model. If the exclusion of a variable minimises the error of the linear model, it is removed. This means that all of the variables of a linear model can be removed, with only a constant value remaining. The process of variable reduction within a multivariate linear regression model is referred to as the simplification step.

Once a simplified linear model is constructed for each node, the tree is pruned by examining all non-leaf nodes as if they were leaf nodes, starting at the bottom of the tree. If the error of the final model is reduced by regarding the specified node as a leaf node, the subtree, of that node, is withdrawn from the model and the specified node is pruned to a leaf node. The pruning step together with the simplification of the linear models at the nodes of the model tree helps to reduce the complexity of the final model, subsequently reducing its variance.

Smoothing is applied when the model is tasked with predicting a test instance. Discontinuities are prevented from forming between the linear models of adjacent leaf nodes. Smoothing adjusts the predicted values of the model tree as follows: starting at the leaf node, the predicted value is passed on along the path to the root node. At each node, the predicted value is adjusted such that it better resembles each predicted value produced by the linear model of the next node on the path. The formula used to recursively adjust the predicted value, y'_i, at the i^{th} node along the path from the leaf node to the root is

$$s(y'_i) = \frac{ny'_{i-1} + s_c y'_i}{n + s_c} \tag{2.3}$$

where n is the number of training instances at the previous node along the path, y'_{i-1} is the adjusted prediction from the previous node, y'_i is the predicted value of node i, and s_c is the specified numeric smoothing constant. The i^{th} value of $s(\cdot)$, produced by the root node at the end of the path, is the predicted value of the model tree given a test instance.

2.3.3. Early M5 model tree performance

Quinlan compared the M5 model tree to MARS (multivariate adaptive regression spline), a popular regression model that is effective at modelling nonlinear data, after the conception of the M5 model tree [24]. MARS is similar to M5 in that MARS is also non-parametric and utilises piecewise linear approximation. A non-parametric model has no preconceived notion regarding the number of parameters it is tasked with learning, nor the distribution that the data follows. M5 and MARS produced similar accuracy results, but what set M5 apart from MARS was the computational requirement. The computational requirements of MARS grew aggressively with an increase in dimensionality, severely limiting its applicability. M5 was able to handle tasks with up to hundreds of input features, whereas MARS struggled past no more than twenty [24]. The results Quinlan published provided evidence that model trees are a viable choice for solving regression problems. Model trees also compare better to alternative regression techniques as opposed to regression trees which struggle to compete against linear regression models or neural networks.

2.3.4. The M5$'$ model tree

Quinlan's work on the M5 model tree was not readily available and some design decisions, such as how missing values are handled, were not addressed. This prompted Wang and Witten to refine the induction steps to create the M5$'$ model tree [30]. One important aspect of the M5$'$ model tree is that it specifies the linear model implemented in a node. The linear model named the $(m+1)$-parameter model is denoted by [30]

$$w_0 + w_1x_1 + w_2x_2 + ... + w_mx_m \tag{2.4}$$

with x_m representing the m^{th} input feature, w_m the respective weight of that input feature in the linear model, and w_0 the bias term. Fundamentally, Equation (2.4) is a linear polynomial with weights that represents its coefficients. Through experimentation, Wang and Witten deemed that the simplification step of M5 compromises the size of the model tree. Wang and Witten found that occasionally much smaller trees were obtained when all features were left in the linear models. Therefore, the simplification step was omitted from the induction algorithm. The M5$'$ model tree was shown to perform better than the original M5 tree on the standard datasets for which results were available [30]. For the remainder of this thesis, the M5$'$ model tree is simply referred to as the M5 model tree.

2.3.5. Shortcomings of model trees comprising linear models

The M5 model has been shown to perform inadequately in comparison to other regression techniques on datasets that contain noisy patterns and highly nonlinear relationships between its input and target features [1]. The susceptibility of the M5 model tree to initially overfit noisy data can be attributed to the already high variance produced by decision tree models, together with the increased complexity of a model tree compared to a regression tree. To combat this, the model tree is pruned after induction. However, the excessive post-pruning of a decision tree, with the intent of increasing its resilience to overfitting noisy patterns, limits the complexity of the model and in turn its ability to capture highly nonlinear patterns. Post-pruning refers to the pruning of a decision tree after it has been grown to overfit the data, as opposed to stopping the growth of the tree prematurely to effectively prune it. Within the tree, multiple leaf nodes each with linear models are pruned away. Consequently, the collection of linear models in the subtree is simplified into one. The remaining linear model is likely to underfit the data. This is referred to as over-pruning and has been shown to have negative effects on the performance of a model tree [1].

A study published in 2016 hypothesised that the reason for the inferior results produced by the M5 model tree on highly nonlinear data is due to the limited capability piecewise linear functions have in fitting training data [15]. Six quantitative water quality parameters were used in predicting the monthly chemical oxygen demand of a river. The relationship between these parameters is highly nonlinear. A MARS model was able to achieve, on average, an accuracy of 19.1% better than the competing M5 model tree. It is important to note that Quinlan designed the M5 model tree with large datasets in mind, making it the preferred choice for problems with a large number of input-parameters [24].

Quinlan recommended that the application of nonlinear models at the leaf nodes of the M5 model tree be researched to improve the ability of the model tree to adequately capture highly nonlinear patterns present in complex datasets [24]. Nonlinear models for output functions would also allow for a model tree to be grown smaller, as a single nonlinear model can approximate the same function that requires multiple linear models to approximate. The number of times the training data requires splitting is reduced to produce a smaller tree. Therefore, a decreased variance error is observed making the model tree less sensitive to noise and less prone to overfitting.

Careful consideration should be given to the increase in computational cost that comes with the implementation of nonlinear models. A model has to justify its increased computation time with a clear and substantial improvement in its accuracy or even interpretability. Ockham's razor expresses the fallacy of expecting that the increase in complexity of a model will guarantee an improvement in its performance [4]. More often than not, a simpler approach is the favoured solution to a problem. These factors are hypothesised as contributing to the lack of abundant research on model trees that incorporate nonlinear models.

2.4. Nonlinear solutions

This section discusses two existing model tree approaches that incorporate nonlinear models within their leaf nodes. Section 2.4.1 discusses how the nonlinear model, present in partial linear trees (PLT), is broken down and the performance thereof examined. Next, Section 2.4.2 discusses a model tree which is induced via a genetic programming approach.

2.4.1. Kernel regression

Torgo developed the hybrid regression tree that, amongst other proposed models, incorporated kernel regression at its leaf nodes [29]. A kernel function empowers a linear model to interpret the relationships between the instances as nonlinear by mapping the instances to a higher-dimensional feature space. Feature space refers to the m dimensions through which the data is defined, excluding the target feature. The kernel regression that HTL incorporates is known as a lazy learner. Lazy learners only generalise the training data once a prediction must be made and never before. For this reason, lazy learners are also a popular choice for models where the training data is regularly updated.

The HTL structure is induced using the CART algorithm of selecting splits that minimise the mean square error (MSE), i.e.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2 \tag{2.5}$$

where n is the number of instances over which the MSE is computed, y_i is the actual target value of the i^{th} instance, and y_i' is the predicted value for the instance. The predicted value for each instance is chosen as the average target value in the subtree where the split is proposed. Effectively, the deviation is being calculated; Equation (2.5) is showcased this way to emphasise the potential of interchanging y_i with the kernel regression prediction. Torgo states that the use of the calculations of kernel regression to determine the MSE for each proposed split within the tree structure is computationally too expensive, Therefore, Torgo opts to use the CART algorithm to induce the tree structure instead [29]. After the splits have been determined, a kernel regression model within the leaf node is thereafter employed to calculate y_i' when making predictions. The use of kernel functions gives rise to the hybrid nature of HTL.

The kernel regression model comprises a nearest neighbours model and a Gaussian kernel function. The kernel regression model calculates predictions with only the training instances most similar to a given query. The similarity is based on a distance metric between two instances. The Gaussian kernel function increases the influence (based on the distance metric) neighbours have on the prediction the nearer they are to the instance in the feature space. Given the instance $\boldsymbol{q}$, the prediction is calculated using the following function [29]:

$$y'(\boldsymbol{q}) = \frac{1}{K_1(\boldsymbol{q})}\sum_{i=1}^{k_{nn}} y_i \times K_2\left(\frac{D(\boldsymbol{x_i}, \boldsymbol{q})}{h_b}\right) \tag{2.6}$$

with

$$K_1(\boldsymbol{q}) = \sum_{i=1}^{k_{nn}} K\left(\frac{D(\boldsymbol{x_i}, \boldsymbol{q})}{h_b}\right) \tag{2.7}$$

and

$$K_2(d) = e^{-d^2} \tag{2.8}$$

where $D(\cdot)$ is a distance function between two vectors, h_b is the bandwidth parameter, and $\boldsymbol{x}_i$ is a vector of input features with target value y_i for the i^{th} training instance. The number of nearest neighbours is indicated by k_{nn}. Only the training instances that fall into the same leaf node as $\boldsymbol{q}$ are evaluated as potential nearest neighbours.

HTL is computationally more expensive than other tree models because the nearest neighbours have to be re-evaluated for each query. Equation (2.6) portrays how the computational complexity grows as the value of k increases due to the added terms in both the kernel function and the encapsulated distance function. For experimentation, Torgo chose $k_{nn} = 3$ without specifying the reasoning behind that choice [29].

The performance of HTL was evaluated on eleven different benchmarking datasets. Torgo concluded that, compared to linear models, the use of kernel regression models at the HTL leaf nodes resulted in significantly better performance for most cases and never significantly worse. One dataset was considered an exception which Torgo claimed is biased towards linear models. Although Torgo mentioned that the M5 model tree incorporates similar linear models to the linear HTL variant, Torgo did not compare the performance of HTL directly to that of M5 [29].

HTL was further improved on by imposing the kernel regression model, described by Equation (2.6), on the linear model in Equation (2.4) [28]. The resulting model now incorporates the weight vector of Equation (2.4), $\boldsymbol{w}$, and is described by

$$y'(\boldsymbol{q}) = \boldsymbol{wq} - \frac{1}{K_2}\sum_{i}^{k_{nn}} \epsilon_i \times K_1\left(\frac{D(\boldsymbol{x_i}, \boldsymbol{q})}{h}\right) \tag{2.9}$$

where e_i is the error of the linear model calculated as $\epsilon_i = \boldsymbol{wx_i} - y_i$. The vector $\boldsymbol{w}$ is a set of weights that represent the coefficients of a linear polynomial, equivalently expressed in Equation (2.4). For a set of n training instances $\{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_n}, y_n)\}$, the weights of $\boldsymbol{w}$ are calculated by minimising a least squares error criterion, i.e.

$$\epsilon = \sum_{i=1}^{n} \left(y_i - \boldsymbol{w}^T \boldsymbol{x_i}\right)^2 \tag{2.10}$$

This adaption of HTL was referred to as the partial linear tree (PLT) [28]. PLT was designed with the goal of maintaining the accuracy of linear models while improving its comprehensibility of nonlinear patterns [28]. PLT was compared to MARS as well as a commercial version of M5 on 12 separate benchmarking datasets. The commercial version of M5, namely Cubist, also incorporates linear models. PLT produced highly competitive results, though Torgo described the use of PLT as being computationally heavy for the same reasons HTL is considered computationally expensive [28].

Torgo recommended further research be done on the use of an alternative splitting criterion when inducing the tree structure. PLT did not change the procedure that HTL follows, which is the same as the standard CART algorithm of selecting splits that minimise deviation over the target value of the training instances. A criterion which incorporates the models used at the leaf nodes, such as the one used in M5, can improve the fit of the models in leaf nodes to be more accurate. However, this also increases the computation time and therefore demands careful consideration.

2.4.2. Higher-order polynomial regression

Another approach to modelling nonlinear data is through the use of higher-order polynomial expressions, rather than piecewise linear polynomials. However, the estimation of the structure of a multivariate polynomial, as well as its parameters, is a time-consuming task. A multivariate polynomial function that incorporates all possible terms is expressed as [22]

$$f(\boldsymbol{x}) = \sum_{\tau=0,\Sigma_{j=1}^{m}\lambda_j=\tau}^{o} \left(w_{(\lambda_1,\lambda_2,\ldots,\lambda_m)} \prod_{i=1}^{m} x_i^{\lambda_i} \right) \tag{2.11}$$

where m is the number of input features, o is the degree of the polynomial order and $w_{(\lambda_1,\lambda_2,\ldots,\lambda_m)}$ the polynomial coefficients, i.e. the weights when used as a regression model. For a third-order polynomial describing a ten-dimensional feature space, there are 286 terms in total. The terms are referred to as monomials. Monomials are purposely excluded to change the characteristics of the polynomial function as desired. Subsequently, there are 2^{286} different possible combinations of monomials for a third-order, ten-dimensional polynomial function. Even with the coefficients already estimated, iteratively evaluating all 2^{286} possible monomials on a set of instances is already computationally demanding. Genetic algorithms, however, can be implemented to perform these tasks that are otherwise considered computationally too expensive [22].

A genetic algorithm is a heuristic search method that models aspects of natural selection mechanisms to evolve an optimal solution from multiple candidates [9]. "Survival of the fittest" is simulated across numerous generations, each generation having multiple candidate solutions, referred to as individuals [25]. The fitness function of a genetic algorithm dictates which individuals represent better solutions in the problem domain [23]. Only the individuals deemed fit are allowed to reproduce or to survive to the next generation. The genetic algorithm thereby optimises the fitness function to produce a globally optimal solution.

Potgieter and Engelbrecht developed a genetic algorithm capable of evolving polynomial expressions that are structurally optimal and retain favourable accuracy [22]. Potgieter and Engelbrecht defined optimality as the shortest polynomial with the best possible function approximation. GASOPE was evaluated by approximating several nonlinear functions. GASOPE was shown to be significantly faster than a neural network approach, whilst producing comparable results [22]. Note that GASOPE is regarded as a polynomial regression model in the context of this paper.

Building on the success of GASOPE, Potgieter and Engelbrecht employed GASOPE to evolve the models used at the leaf nodes of a model tree. The proposed model tree labelled GPMCC, made use of a genetic program to evolve a tree structure [23]. Hitherto discussed decision trees all employed greedy approaches to induce a decision tree structure. A greedy approach can be problematic because it is susceptible to becoming stuck in locally optimal solutions. The problem is attributed to the short-sighted nature of a greedy approach. Genetic approaches to inducing the structure of a decision tree outperform greedy approaches in the size of the induced tree but at the expense of computational cost [3, 23]. The advantage a genetic approach exhibits over greedy approaches is due to the search for a globally optimal solution as opposed to iteratively favouring solutions that yield local optima.

The genetic heuristic of GPMCC initialises multiple tree candidates by recursively adding nodes to each individual, until a maximum depth. At each node, the feature on which to split and the splitting value were randomly selected upon initialisation. The remaining operators ensured that only the optimal splits are maintained through the generations. Specialised mutation and crossover operators were introduced. Mutation ensures that new genetic material, utilising domain-specific knowledge, is injected into the population of candidate decision trees. The crossover operator was used to splice together two candidates to produce new candidate decision trees for the following generation. The fitness function incorporates the complexity of a candidate tree, in terms of the size and number of polynomial terms within all the leaf nodes, as well as the difference in the predicted and actual output of an instance. Thereby, the fitness functions can minimise both the prediction error as well as the structure of the candidate solution.

GPMCC was compared to Cubist on 13 benchmarking/artificial datasets producing significantly smaller tree structures, whilst being competitive with the accuracy of its predictions. For the majority of datasets, the order of the polynomials produced by GASOPE at the leaf nodes of GPMCC never exceeded three. This meant that cubic surfaces were sufficient in describing the nonlinear relationships within these datasets. A disadvantage of GPMCC is the speed at which the genetic algorithm induces a model tree. Potgieter and Engelbrecht attributed the slow computational speed to the recursive procedures that perform fitness evaluation, crossover, and mutation of genetic candidates [23].

2.5. Decision tree ensembles

Decision tree ensembles are discussed in this final section of the chapter. Firstly, a popular ensemble method for regression trees, named random forests (RFs), is discussed in Section 2.5.1. Model tree ensemble research is not widespread and the computational cost associated with model tree ensembles is hypothesised as the main factor contributing to the lack of research. Section 2.5.2 motivates the reason why further research into model tree ensembles can be beneficial and introduces two successful ensembles based on the M5 model tree. Section 2.5.3 discusses the performance of random model trees (RMT). Finally, model tree ensemble (MTE) is examined in Section 2.5.4.

2.5.1. Random forest

To build on the success achieved by bagging, Breiman conceptualised the *random forest* method as an extension over bagging [7]. In RF, similar to bagging, each base learner is once again trained on a randomly selected subset of the data. Each base learner in a bagged ensemble is induced on a subset of the training set equal to the original training set size. However, due to sampling with replacement, approximately two-thirds of the instances are unique [6]. The remaining third are duplicate instances. The base learners of RF are regression trees. Regression trees and model trees both employ the divide-and-conquer induction approach to recursively split the training data using a set of rules. Recursive splitting of a training set contributes to the ever-high variance error decision trees exhibit because a decision tree tends to overfit the training data through each additional split.

The difference that RF incorporates over bagging is a change to the induction algorithm of the base learners that make up the ensemble. A subset of input features is randomly selected at each node of the tree to determine the best split for that node. Hence, the splitting function only evaluates proposed splits on the randomly selected subset of input features. Breiman proposed the number of splitting features evaluated for each node equal $\log_2(m+1)$ where m is the dimensionality of the feature space [7]. The number of splitting features was approximated to $\sqrt{m}$ for classification problems and $m/3$ for regression problems.

The random subset of splitting features serves to strengthen the random effects present in the induction algorithm, decorrelating the base learners within the ensemble. Therefore, the decorrelation of base learners minimizes the first term in Equation (2.1). Consequently, the collective variance of the ensemble is reduced. Simultaneously, pruning is omitted in the base learners of an RF to promote *fully grown* regression trees. Pruning prohibits a decision tree from exhibiting the advantageous low bias error associated with decision trees. Thereby, RFs retain a low bias, whilst minimising variance through the induction of numerous, decorrelated decision trees. RFs have the benefit of performing well on higher dimensionality data, due to the dimension reduction-like induction algorithm and was shown to outperform many competing classifiers whilst being robust to overfitting [7].

2.5.2. M5 ensembles

The challenges of developing ensembles of model trees are the increased computational cost and the difficulty of interpreting the models [25]. The increased interpretation difficulty is due to the increased number of learners as well as the injection of randomness into the selection of splits. One of the advantages that decision trees have over competing supervised learning techniques is the interpretability that decision trees exhibit. Decision trees not only produce adequate performance but also gives insight into the relationships between the input and output features for a given problem. The performance increase of ensembling decision trees has to be adequate to justify the loss of interpretability. Due to the added complexity of a polynomial output function, model trees exhibit an even greater variance error and lower bias error than regression trees. Ensemble methods have proven effective in minimizing the variance error of model trees [25]. The right choice of ensemble method that comprises model trees is critical to producing an effective ensemble model. There are various approaches to ensemble modelling, those applicable in the context of this thesis were discussed in Section 2.2.

Aleksovski [1] and Pfahringer [21] are two of the few who have applied ensemble methods to model trees for regression problems, with each author incorporating a unique approach. The common denominator between the approaches that these authors employed was the use of the M5 model tree as the base learner. Each of the approaches to model tree ensembling discussed here also incorporates an aspect of randomness in the induction step. The authors referenced the success that Breiman achieved with random forests as justification for the injection of randomness into RMT and MTE. Incorporation of randomness into an ensemble increases the decorrelation present between the base learners that comprise the ensemble, allowing for greater prediction accuracy [7] as shown in Equation (2.1).

Both Aleksovski and Pfahringer opted to only incorporate model trees that comprise linear models. To the best knowledge of the author of this thesis, there is no research published yet on ensembles of model trees that comprise nonlinear models.

2.5.3. Pfahringer's random model trees

Pfahringer modified the M5 algorithm to grow balanced trees within an ensemble, largely based on Breiman's random forest with little deviation other than the use of model trees as base learners [21]. Balanced trees are defined as trees that incorporate the same number of instances in each leaf node. Pfahringer developed balanced trees by always selecting the median of a feature as the split value.

Pfahringer did not directly compare RMT to a single M5 tree. Instead, RMT was compared to a simple linear regression model, Gaussian process regression (GPR), and additive groves[1] (AG) [21]. At the time, GPR and AG were considered state-of-the-art regression methods. RMT outperformed both GPR and AG in terms of computational efficiency, whilst having competitive predictive accuracy [21].

2.5.4. Aleksovski's model tree ensembles

Aleksovski was tasked with developing models for dynamic systems, for which data was not evenly distributed in the feature space [1]. Aleksovski, therefore, opted not to incorporate balanced trees for base learners to avoid poor approximations of the data [1]. Aleksovski's approach to growing a model tree within an ensemble included three key features, namely randomised splitting features, *fuzzification*, and ensemble pruning.

The induction process of MTE is based on a bagged approach. A subset of features is randomly selected with replacement for each node within each tree in the ensemble. Subsets of features are always chosen from the entire pool of available features and the act of choosing a feature does not remove it from this pool. The standard induction of M5 follows on each subset, i.e. growth that favours a reduction in standard deviation and pruning nodes to reduce prediction error.

MTE omits the smoothing process that M5 incorporated and instead uses fuzzification to prevent the discontinuities that each split within the model tree introduced. Aleksovski stated that the smoothing procedure of M5 produces poor-performing models, and fuzzification is used instead to combat this [1]. Fuzzification removes discontinuities from the prediction of a model by creating a smooth transition between two local models. Splits are transformed into fuzzy splits via the implementation of a sigmoid function. For a tree that comprises a single split, the prediction $y'(\boldsymbol{x})$ of a given instance is accordingly calculated as

$$y'(\boldsymbol{x}) = \mu(x_j, s, \alpha) f_1(\boldsymbol{x}) + \mu(x_j, s, \alpha) f_2(\boldsymbol{x}) \tag{2.12}$$

[1]Additive groves is an ensemble of bagged additive regression trees which are iteratively trained.

with

$$\mu(x_j, s, \alpha) = \frac{1}{1 + e^{-\alpha(x_j - s)}} \tag{2.13}$$

where f_1 and f_2 are the linear models of two adjacent leaf nodes separated by a split on the j^{th} feature with split value s. Finally, α is a fuzzy parameter calculated using cross-validation. It is important to note that Aleksovski did not incorporate fuzzification into the induction of the tree structure because it showed to decrease the efficiency of the M5 algorithm. The intended omission is due to the added computational cost fuzzification demanded [1].

Once the ensemble is fully grown, i.e. a specified number of learners have been induced on the dataset, the ensemble is pruned as a whole via a greedy selection procedure. As is the case with individual tree pruning, the generalisation error of MTE is evaluated both with and without each tree in the ensemble. In the case of individual tree pruning, nodes are removed. If a particular tree contributes to an increase in the prediction error, it is removed from the ensemble. The prediction of MTE is calculated through the uniformly weighted average of the predictions each tree within the ensemble makes.

MTE performed competitively against other state-of-the-art methods, including neural networks, in terms of its prediction error, whilst having the advantage of being quite robust to noise. Aleksovski described the MTE as being resilient to data that exhibited up to 20% noise. This resilience to noise is contributed to the injection of randomness in the induction of MTE and helped MTE to produce a low variance error [1].

2.6. Conclusion

This chapter aimed to provide background information about how decision trees and ensemble methods thereof each address the bias-variance dilemma. Furthermore, this chapter investigated existing model tree methods and the application of model trees within an ensemble. The application of model trees that comprises higher-order polynomial output function as the base learners within an ensemble model is under-researched. There are studies conducted on model trees that comprises linear output functions within an ensemble which confirm the advantages ensemble techniques have. The advantages are a successful reduction in the variance errors to improve the robustness to overfitting and overall performance over the base learner. Model trees that comprise higher-order polynomial output functions have been shown to outperform their linear output function counterparts. Consequently, further research into the application of model trees with higher-order polynomial output functions as the base learner of an ensemble is required.

Chapter 3

Genetic algorithm to evolve structurally optimal polynomial expressions

This chapter discusses the genetic algorithm to evolve structurally optimal polynomial expressions. Genetic algorithms are a paradigm of evolutionary computing. Evolutionary computing allows computers to solve problems systematically through the iterative application of analogues of naturally occurring genetic operations [16, 17]. Based on a high-level problem statement, evolutionary computing breeds a population of candidate solutions and transforms the population with each iteration into a new generation. Each following generation is selected to incorporate more favourable characteristics to solve an optimisation problem. In this chapter, a genetic program is tasked explicitly with symbolic regression to find the model that best fits a given dataset.

Potgieter and Engelbrecht incorporated GASOPE in the leaves of the GPMCC model tree [23]. However, since the conception of GASOPE, computational limitations have lessened. Therefore, changes from the original implementation have been made. The model tree forest proposed in this thesis employs an adaption of GASOPE in the leaf nodes of the model trees that make up the ensemble. The adaption of GASOPE was developed from the ground up. The complete genetic algorithm and all the deviations from the original are discussed with pseudo-code in Section 3.1. The adaptation of GASOPE is evaluated and compared to the original results obtained by Potgieter and Engelbrecht in Section 3.2. In the final section, GASOPE is evaluated on higher dimensional problems against other regression models.

3.1. The genetic algorithm

This section discusses the genetic algorithm. The drawbacks of the original implementation of GASOPE to accommodate a small computational budget are discussed. The changes made to address the drawbacks of the original GASOPE and form an improved adaptation are motivated. The pseudo-code of the genetic algorithm of the adapted GASOPE is discussed. Section 3.1.1 discusses the drawbacks of the original implementation of GASOPE and the parts that are omitted for the adaption of GASOPE. The remaining sections each discuss the genetic algorithm which remains fundamentally unchanged. Section 3.1.2 discusses the representation of an individual in the genetic algorithm. Section 3.1.3 illustrates the initialisation of an individual. Sections 3.1.4 and 3.1.5 illustrate the mutation and crossover operators of the genetic algorithm respectively. Section 3.1.6 introduces discrete least-squares approximation, used to produce the coefficients of an individual. Section 3.1.7 discusses the fitness function employed by the genetic algorithm to rank individuals. Finally, Section 3.1.8 discusses the entire genetic algorithm used by GASOPE to evolve a population of individuals for a given dataset.

3.1.1. Drawbacks and adaptations from the original implementation of the genetic algorithm

The original implementation of GASOPE was heavily influenced by the computational limitations at the time and developed with techniques to minimise computational cost [22]. The algorithm consists of a three-stage process:

1. Clustering of the training instances,
2. the genetic algorithm to evolve a structurally optimal polynomial expression and
3. a hall-of-fame to combat the drawback clustering causes.

The drawback of clustering is that the genetic algorithm is only exposed to a select number of training instances and not the dataset as a whole. The clustering stage draws a clustered random sample of training instances for each generation. A cluster is drawn to represent a collection of homogeneous training instances. The clustering stage is implemented to minimise the number of training instances over which the genetic algorithm must iterate. Thereby, the computational cost of the genetic algorithm is reduced. Clustering significantly reduces the time taken for the genetic algorithm to evolve the final individual, without reducing the accuracy of the genetic algorithm [22]. However, Potgieter and Engelbrecht acknowledged that, due to the aspect of randomness injected into the clustering stage, the data subset of a given generation may not inherit a true representation of the entire dataset [22]. In turn, an individual may be evolved that does not present the true nature of the dataset, particularly for extremely noisy datasets [22].

A hall-of-fame was incorporated to ensure that the clustering stage does not hinder the accuracy of GASOPE. The hall-of-fame stores the set of individuals that performed the best for each generation of the genetic algorithm. In each generation, the training instances differ, due to the clustering stage. Subsequently, the individual that performs best on the last generation does not necessarily perform the best of all individuals on the entire dataset, but only on a sample of the dataset. Therefore, once the maximum number of generations is reached, the individual that performed best over all generations is chosen from the hall-of-fame, as opposed to the individual that performed best in the last generation.

Compared to when GASOPE was first conceptualised, computational limitations are now more lenient. Consequently, the new version of GASOPE developed in this chapter and used in MTF does not require the same three-stage process to minimise computational demand. Instead, the dataset in its entirety is utilised, ensuring the best individual is evolved through each of the generations of the genetic algorithm. The clustering stage and therefore the hall-of-fame stages are dropped in the adaption of GASOPE. These two stages are no longer necessary for the evolution of higher-order polynomial functions due to the increased computational budget available today versus when GASOPE was conceptualised. The removal of the clustering stage is the fundamental difference between the original version of GASOPE and the adaption of GASOPE. The adaptation of GASOPE follows the same representation, genetic operators and fitness function as the original implementation of GASOPE.

3.1.2. Representation

Each individual in the population is a representation of a multivariate higher-order polynomial function which is evolved to best fit the given data. Any given individual is made up of a set, I, of unique term-coefficient mappings that collectively represent Equation (2.11). For an individual that incorporates p terms, I is expressed as

$$I = \{(t_0 \to w_0), ..., (t_p \to w_p)\} \tag{3.1}$$

where $w_i, i \in \{0, \dots, p-1\}$, is the real-valued coefficient for the corresponding term t_i. Each term in the set that I comprises, consists of m unique variable-order pairs to form a set T, where m is strictly equal to the number of input variables in the dataset. Therefore, a set of variable-order pairs that represent any single term in the polynomial function is expressed as

$$T = \{(x_1 \to \lambda_1), ..., (x_m \to \lambda_m)\} \tag{3.2}$$

where λ_j is the whole-valued polynomial order of the respective input variable $x_j, j \in \{1, \dots, m\}$.

3.1.3. Initialisation

Algorithm 1 describes how each individual in the population is initialised. The initialisation algorithm is illustrated in Figure 3.1. Random variable-order pairs are repeatedly selected for each term-coefficient mapping. To help promote the growth of smaller-sized polynomials, the polynomial orders for a term are randomly selected without replacement from all available orders up to the maximum polynomial order. This results in a set of variable-order pairs that do not strictly include every available polynomial order (all remaining input variables within the set T are assigned a polynomial order of zero). The maximum allowed number of terms, p, as well as the maximum polynomial order, o, are both user-specified parameters.

An increase in either p or o increases the computational cost of the genetic algorithm. This is because an additional coefficient is estimated for each additional term. In the case of the maximum polynomial order, an additional arithmetic operation is computed for each increment of o. However, an increase in o increases the complexity of the output function through an increase in the order. The complexity of the overall model which employs GASOPE can thereby be tuned based on the problem at hand by tuning o. Note that a large value for o is not ideal for problems that comprise fairly linear relationships between input and target features because overfitting and increased computational costs are facilitated. Additionally, for highly nonlinear relationships, a very large value for o is unnecessary, because the use of ensemble learning mitigates the need for a single output function to produce a perfect fit of given training instances. For example, Boosting discussed in Section 2.2.4 motivates how weak base learners that cannot independently produce an adequate fit of given data, are able to produce a better approximation within an ensemble. Several output functions together can produce a satisfactory fit of highly nonlinear relationships even with a moderate value for o, such as three. An increase in the value of p allows for the evolution of polynomials with a larger number of allowed terms. Problems with high dimensionality may better suit larger p values as more terms are required to sufficiently express each feature within a polynomial equation.

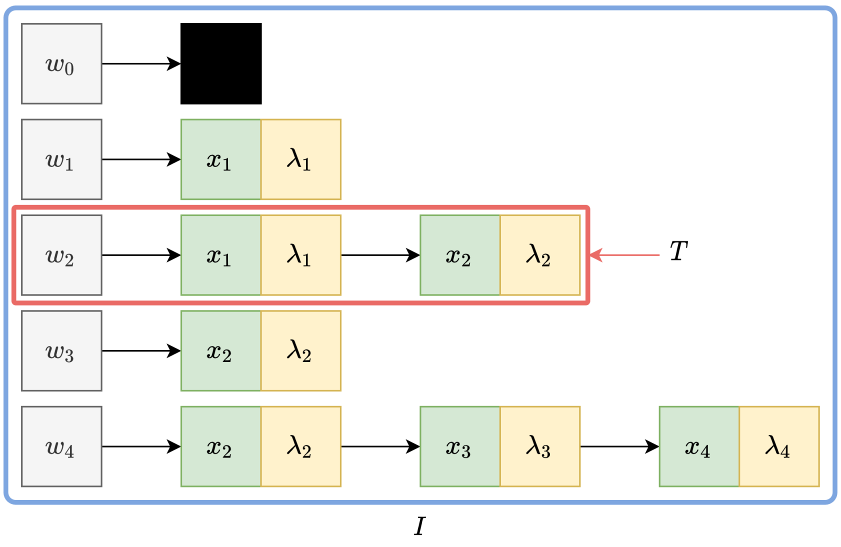


Figure 3.1: Illustration of the GASOPE initialisation algorithm adapted from [22].

Algorithm 1: Pseudo-code for the initialisation of an individual.

```
Set $I = \{\}$
while $|I| < p$ do
    Set $T = \{\}$
    Select $\boldsymbol{e} \in \{1, \ldots, o\}$ as random integers up to the maximum polynomial order $o$.
    while $|\boldsymbol{e}| > 0$ do
        Select $f$ to be a random order within $\boldsymbol{e}$
        Select $g$ randomly from $\{1, 2, ..., m\}$
        if $|T| < |T \cup \{(g \to f)\}|$ then
            $\boldsymbol{e} := \boldsymbol{e}\{f\}$, i.e. remove $f$ from the available orders
            Set $T = T \cup \{g \to f\}$
        end
    end
    Set $I = I \cup \{T \to 0\}$ as illustrated in Figure 3.1
end
```

It is important to note that the coefficients of an individual cannot be estimated until the polynomial structure is fixed. Therefore, during initialisation, each coefficient is assigned the preliminary value of one. Furthermore, when an empty set of variable-order pairs is initialised, all input variables are assigned a polynomial order of zero.

3.1.4. Mutation operators

The mutation operators randomly inject new genetic material into selected individuals of the population. The mutation operators ensure that the search space is better explored by the genetic algorithm. Thereby, the probability of the algorithm producing a local optimum solution is reduced. Each operator aims to optimise the structure of the polynomial by either making adjustments to variable-order pairs or an entire term-coefficient mapping. The four, equally probable, mutation operators are shrink, expand, perturb and reinitialise.

The first mutation operator, *shrink*, is simple in its execution and is responsible for decreasing the size of the polynomial structure of an individual. One term-coefficient pair is arbitrarily selected from the set I and removed. The shrink operator is described in Algorithm 2.

Algorithm 2: Pseudo-code for performing the shrink operator on an individual.

```
Select $T \in I$ randomly from the set of terms $I$
Set $I = I/\{T\}$ as illustrated in Figure 3.2
```

Figure 3.2 illustrates the shrink operator. A set of variable-order pairs T is randomly selected and removed from the set of term-coefficient mappings I, as indicated by the red arrow in Figure 3.2.

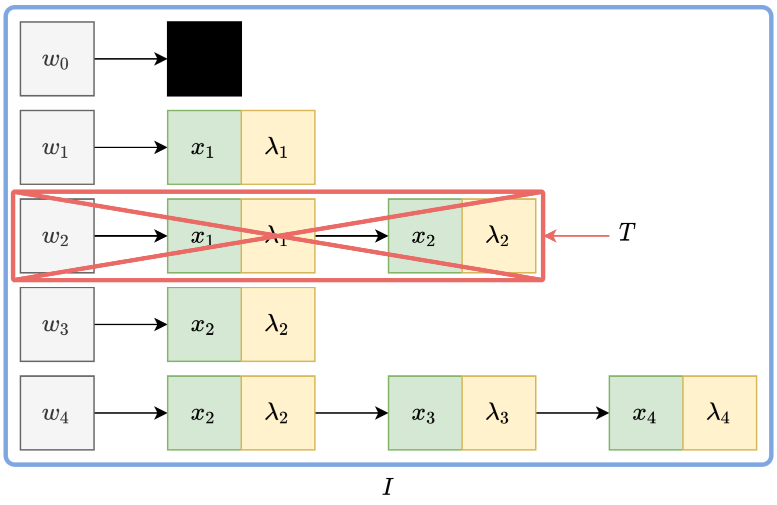


Figure 3.2: Illustration of the GASOPE shrink algorithm, adapted from [22].

The second operator, *expand*, is responsible for increasing the size of the polynomial structure of an individual. The expand operator adds a new term-coefficient mapping to the set I as described in Algorithm 3. Expand is illustrated in Figure 3.3 through the insertion of a new variable-order pair, indicated by the red arrow, into the term-coefficient mapping. In comparison to the original implementation, the expand operator is executed on an individual even if the individual already contains or exceeds the maximum number of allowed terms p. The removal of a maximum term threshold allows for larger polynomial structures to be explored if the increase in accuracy justifies it. For this reason, polynomials that incorporate a structure too large should be adequately penalised during fitness calculations to prevent individuals from evolving polynomial structures that may compromise the computational cost of the genetic algorithm.

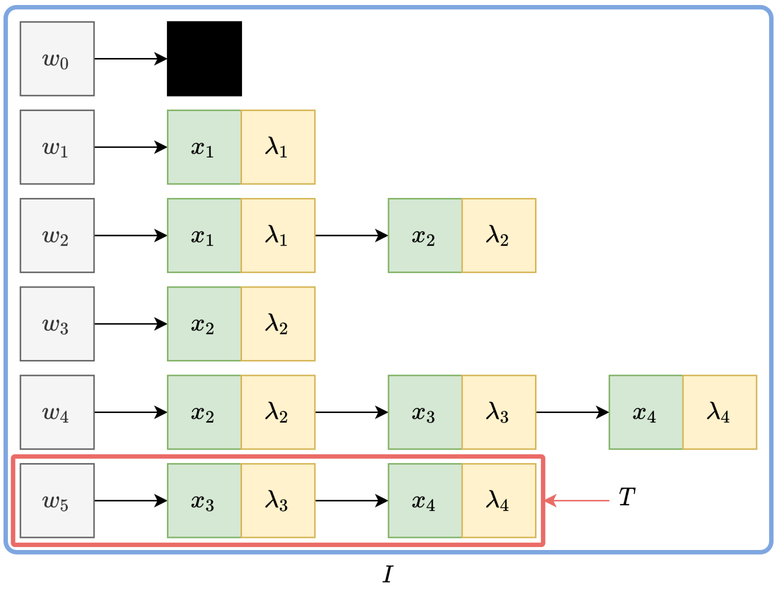


Figure 3.3: Illustration of the GASOPE expand algorithm, adapted from [22].

Algorithm 3: Pseudo-code for performing the expand operator on an individual.

```
Set $T = \{\}$
Select $\boldsymbol{e} \in \{0, \ldots, o\}$ as random integers up to the maximum polynomial order $o$
while $|\boldsymbol{e}| > 0$ do
    Select $f$ to be a random order within $\boldsymbol{e}$
    Select $g$ randomly from $\{1, 2, ..., m\}$
    if $|T| < |T \cup \{(g \to f)\}|$ then
        $\boldsymbol{e} := \boldsymbol{e}/\{f\}$, i.e. remove $f$ from the available orders
        Set $T = T \cup \{g \to f\}$
    end
end
Set $I = I \cup \{T \to 0\}$ as illustrated in Figure 3.3
```

The third mutation operator, *perturb*, is focused on the alteration of a randomly selected variable-order pair, $\{x_m \to \lambda_m\}$, within another randomly selected term T in the set of term-coefficient mappings I. When perturb is performed on an individual, one of three possible adjustments is made: a new variable-order pair is added to the term, an existing variable-order pair is removed from the term or an existing variable-order pair is adjusted. This process is described in Algorithm 4 and illustrated in Figure 3.4. A change in the perturb operator from the original implementation of GASOPE is introduced. When a variable-order pair is either adjusted or added, the available orders $\boldsymbol{e}$ remain unchanged. Previously, once an order was randomly selected, it was removed from $\boldsymbol{e}$. The change is implemented to increase the diversity of variable-order pairs that are introduced through mutation.

Algorithm 4: Pseudo-code for performing the perturb operator on an individual.

```
Select $T \in I$ randomly from the set of terms in $I$
Select $\boldsymbol{e} \in \{0, \dots, o\}$ as random integers up to a maximum polynomial order $o$
Select $g$ randomly from $\{1, 2, ..., m\}$
Select $h$ randomly from a uniform distribution over $[0, 1]$
if $h < 0.333$ then
    Set $T = T/\{g \to \lambda\}$ where $\lambda > 0$, i.e. remove the $g^{th}$ variable-order pair present in
      $T$, as portrayed by the crossed out variable-order pair in Figure 3.4
else if $h < 0.666$ then
    Select $f$ to be a random order within $\boldsymbol{e}$.
    Set $T = T \cup \{g \to f\}$ as portrayed by the entirely circled variable-order pair in
      Figure 3.4
else
    Set $T = T/\{g \to \lambda\}$ where $\lambda > 0$
    Select $f$ to be a random order within $\boldsymbol{e}$
    Set $T = T \cup \{g \to f\}$ as portrayed by the partially circled order of the variable-order
      pair in Figure 3.4
end
```

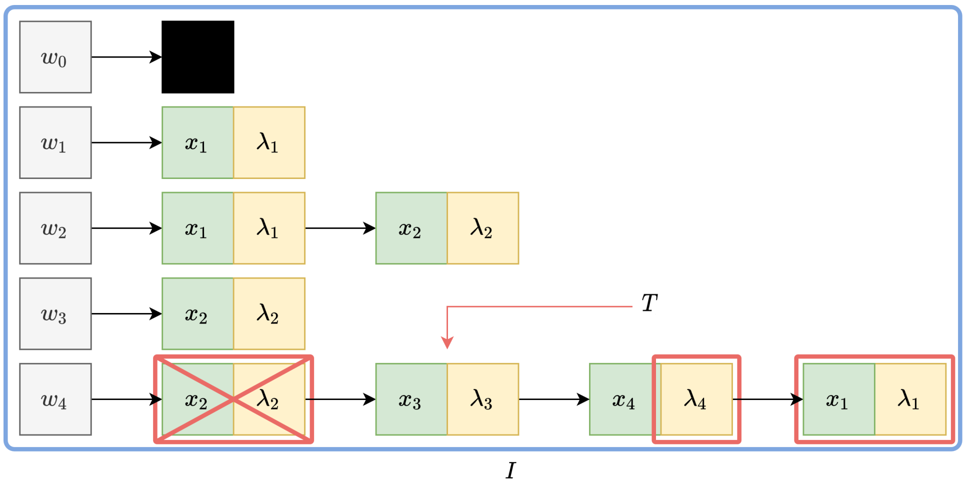


Figure 3.4: Illustration of the GASOPE perturb algorithm, adapted from [22]. Each of the three variable-order pairs marked by the red square describes a peturb operation.

The fourth and final mutation operator, *reinitialise*, is simply a re-invocation of Algorithm 1, as discussed in Section 3.1.3. The purpose of the reinitialise operator is to introduce new, random genetic material into the pool of candidate solutions.

3.1.5. Crossover operator

The crossover operator recombines favourable genetic material from one generation of individuals to produce the offspring candidates of the next generation. Consequently, the favourable genetic material is transferred over to the next generation. The crossover operator serves to aid the genetic algorithm. The crossover operator produces new individuals from the top-ranked genetic material to promote positive exploration of the search space. When the crossover operator is called two of the best performing individuals are randomly selected and their term-coefficient mappings are evaluated before a new individual is evolved which retains some of their genetic makeup. Any term-coefficient mappings simultaneously present in both individuals have a larger probability of being transferred over to the newly evolved individual. The probability of a mutual or non-mutual term-coefficient mapping being passed on from parent to offspring is determined by a user-specified parameter, namely the selection ratio. The selection ratio represents the probability of offspring retaining the inclusive and exclusive term-coefficient mappings. Potgieter and Engelbrecht found that a selection ratio of 80%:20% resulted in newly generated individuals roughly equalling the length of its longer parent [22]. Consequently, inclusive terms between parents have an 80% chance of being passed on and exclusive terms have a 20% chance.

Algorithm 5 describes how the selection ratio is implemented in the crossover operator to evolve a new individual and Figure 3.5 presents the crossover operator. Only one change exists from the original implementation of GASOPE. Newly generated individuals may now exceed the maximum number of terms. The increased computational budget allows for the exploration of candidate solutions with larger polynomial structures. Note that a larger polynomial structure requires a significant improvement in accuracy to justify any increased computational cost.

Algorithm 5: Pseudo-code for performing crossover between two individuals.

```
Set $I_\alpha = \{\}$, i.e. a new, empty term-coefficient set (individual)
Select $I_\beta, I_\gamma \in P$ as two unique, randomly chosen individuals from a given population
Set $A = I_\beta \cap I_\gamma$, i.e. the intersection of the term-coefficient mappings
Let $B = (I_\beta / I_\gamma) \cup (I_\gamma / I_\beta)$ be the union of the exclusion
for each term-coefficient mapping ($T_A$) in $A$ do
    Select $h$ randomly from a uniform distribution over $[0, 1]$
    if $h < 0.8$ then
        Set $I_\alpha = I_\alpha \cup \{T_A\}$.
    end
end
for each term-coefficient mapping ($T_B$) in $B$ do
    Select $h$ randomly from a uniform distribution over $[0, 1]$
    if $h < 0.2$ then
        Set $I_\alpha = I_\alpha \cup \{T_B\}$.
    end
end
```

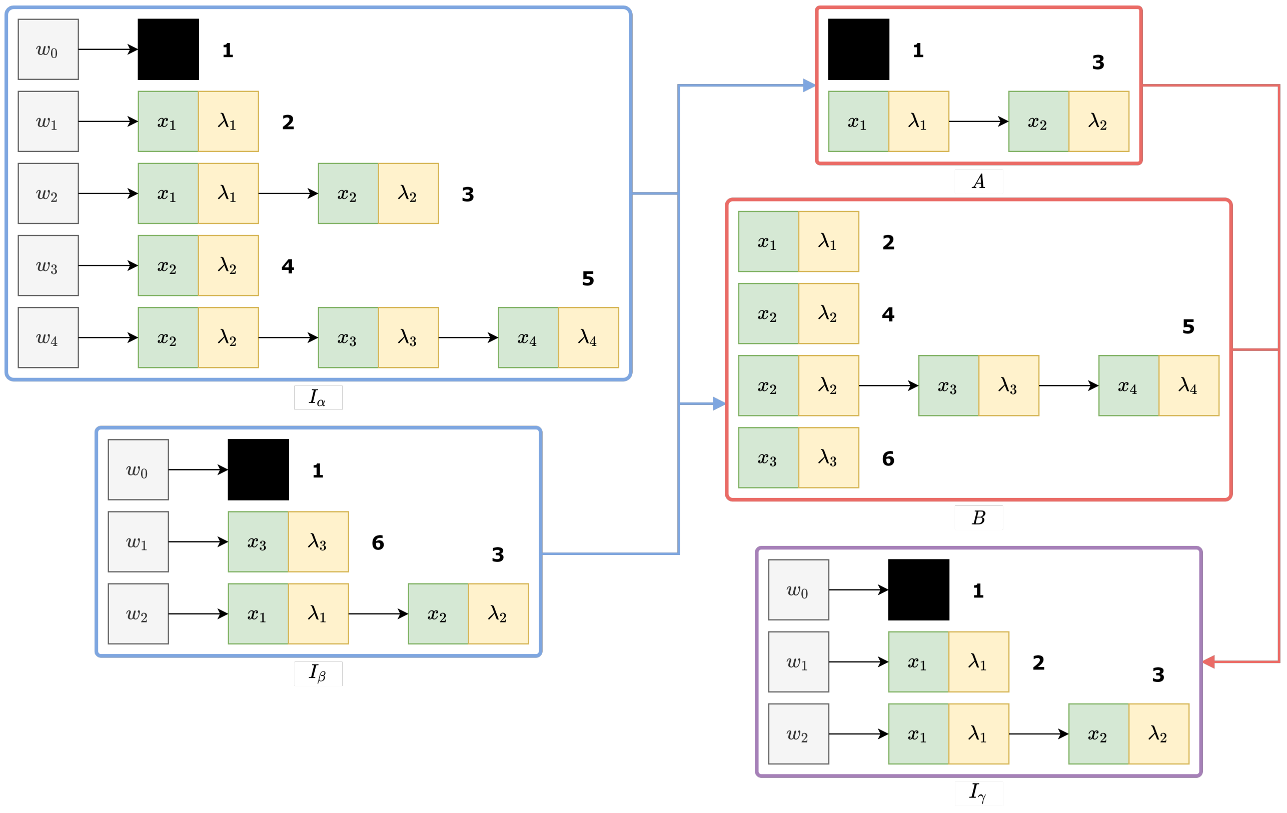


Figure 3.5: Illustration of the GASOPE crossover algorithm, adapted from [22].

3.1.6. Discrete least-squares approximation

With each generation, an individual is evaluated to be structurally optimal through two phases. In the first phase, the terms (which encapsulate variable-order pairs) of the polynomial function are evolved with the crossover operator and possibly mutated. In the second phase, the coefficients are approximated using a discrete least-squares approximation to complete a set of term-coefficient mappings of an individual. Discrete least-squares approximation for a arbitrary set of n data points, $\{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_n}, y_n)\}$, minimises the least squares error:

$$\epsilon = \sum_{i=1}^{n} (y_i - y_i')^2 \tag{3.3}$$

where y_i' is the predicted output of the polynomial function of an individual, for the i^{th} instance. The predicted output is described by the equation (similar to Equation (2.11)),

$$y_i' = \sum_{\tau=0, \Sigma_{j=1}^{m} \lambda_j = \tau}^{o} \left(w_{(\lambda_1, \lambda_2, \ldots, \lambda_m)} \prod_{k=1}^{m} x_{i,k}^{\lambda_k} \right) \tag{3.4}$$

where m represents the number of input variables and o is the maximum polynomial order. Equation (3.4) allows for the representation of an arbitrary individual as

$$y_i' = w_{(0,0)} + w_{(1,0)} x_{i,1} + w_{(0,1)} x_{i,2} + w_{(1,1)} x_{i,1} x_{i,2} + w_{(2,0)} x_{i,1}^2 + w_{(0,2)} x_{i,2}^2 \tag{3.5}$$

for $m = 2$ and $o = 2$. The vector of coefficients, $\boldsymbol{w}$, of the polynomial function is determined by solving the linear system;

$$\boldsymbol{y} \approx \boldsymbol{X}\boldsymbol{w} \tag{3.6}$$

where $\boldsymbol{w}^{\mathrm{T}} = \begin{bmatrix} w_0 & w_1 & \cdots & w_p \end{bmatrix}$ for a polynomial incorporating p terms, which is rewritten as

$$\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)\boldsymbol{w} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} \tag{3.7}$$

There is no exact solution to Equation (3.7). In the case of a univariate function, the matrix $\boldsymbol{X}$ is of the form

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^o \\ 1 & x_2 & x_2^2 & \cdots & x_2^o \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^o \end{bmatrix} \tag{3.8}$$

and vector $\boldsymbol{y}^{\mathrm{T}} = \begin{bmatrix} y_0 & y_1 & \cdots & y_n \end{bmatrix}$. Matrix X has no universally set form and is adapted to correspond with the set of variable-order pairs present in any given individual.

The coefficients of an individual are estimated by solving the above-mentioned linear system as follows. The matrix $\boldsymbol{X}$ in Equation (3.8) is first populated from left to right using the combination of terms that each term-coefficient mapping of an individual represents. With vector $\boldsymbol{y}$ containing the target output for each pattern, the linear system in Equation (3.7) is reduced, producing vector $\boldsymbol{w}$ which is populated with the coefficient of each term-coefficient mapping. Thus, the evolution of an individual for a single generation is concluded. After the term-coefficient mappings are estimated successfully, the fitness of the individual is calculated.

3.1.7. Fitness calculation

The fitness function determines the rank of each individual in the population. The fitness function serves to guide the genetic algorithm into a favourable search space with each generation. This rank is determined by how well the individual solves the problem at hand. In the case of GASOPE, the fitness function is maximised. Therefore, the individual with the highest scoring fitness is ranked first. In each generation, individuals with the lowest fitness scores are replaced preventing the genetic algorithm from exploring poor solutions. The fitness function of GASOPE penalises individuals with polynomials that are not structurally optimal and well-fitted function approximations. The fitness function of a given individual is defined as [22]

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(y_i - y_i'\right)^2}{\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2} \cdot \frac{n-1}{n-c} \tag{3.9}$$

where n is the number of instances in the dataset, y_i is the target output, and y'_i is the output predicted by individual I for instance i. The model complexity c is,

$$c = \sum_{i=1}^{|I|} \sum_{j=1}^{|T_i|} \lambda_{i,j} \tag{3.10}$$

where $\lambda_{i,j}$ is the order of a variable-order pair within a term-coefficient mapping T_i of individual I. Thereby, c penalises an individual based on the number of multiplications needed to calculate the predicted output. The c variable in the fitness function is crucial in ensuring the genetic algorithms develop solutions that are structurally optimal both in the number of terms and their orders.

3.1.8. The complete genetic algorithm

Algorithm 6 presents the complete procedure executed by the genetic algorithm to evolve a structurally optimal polynomial function from a given training dataset. The genetic algorithm starts with a randomly initialised population of individuals P. For each generation, each individual within the population is subjected to a least-squares approximation that produces the set of coefficients needed to perform fitness calculations. Once the fitness of each individual is calculated, the population is ranked accordingly.

An elite few of the top-ranked individuals survive to partially form the population of the following generation. The top-ranked individuals survive to the next generation without any alterations to their polynomial structure. The number of individuals selected for elitism is specified by the user-controlled parameter, i.e. the elitism rate. From the top-ranked individuals, two unique individuals are randomly selected for crossover. Parent selection and crossover are repeated to populate the remainder of the population for the next generation. The number of individuals selected for crossover is specified by the user-controlled parameter, i.e. crossover rate.

Once the population of the following generation is fully populated, mutation is performed. Specified by the user-controlled parameter, i.e. the mutation rate, several randomly selected individuals are mutated. Because adapted GASOPE does not incorporate a hall-of-fame, the best-performing individual from any given generation is no longer saved into a separate set. Only the best performing individual from the final generation is of importance. Consequently, the mutation operator may select the best-performing individual of a generation to be mutated and possibly cause the best candidate solution to be lost during evolution. To prevent the ideal candidate from being, lost a duplicate of the individual selected for mutation is stored in the population prior to mutation. After the maximum number of generations has passed, the highest ranked individual is presented as the optimal polynomial.

Algorithm 6: Complete genetic algorithm for evolving polynomials. Adapted from [22].

```
Let $i = 0$ be the generation counter
Initialise a population $P_i$ of $n_j$ individuals:
  $P_i = \{I_j | j = 1, ..., n_j\}$
Set $n_m$ = mutation rate$\times n_j$
Set $n_c$ = crossover rate$\times n_j$
while  $i < G$, where $G$ is the maximum number of generations do
    for each individual $I_j$ in $P_i$ do
        Perform least square optimisation to determine the coefficients of $I_j$
        Calculate the fitness of $I_j$ from Equation (3.9)
    end
    Let $P'_i \subset P_i$ be the top $x\%$ of the individuals selected for elitism
    $P_{i+1} := P'_i$
    Let $P''_i \subset P_i$ be the top $y\%$ of the individuals selected for crossover
    while  $|P_{i+1}| < (n_j - n_c - n_m)$ do
        Select two unique individuals $I_\alpha$ and $I_\beta$ randomly from $P''_i$
        Perform crossover between $I_\alpha$ and $I_\beta$ to produce $I_\gamma$
        $P_{i+1} := P_{i+1} \cup \{I_\gamma\}$
    end
    Let $k = 0$ be the mutation counter.
    while  $k < n_m$ do
        Duplicate a randomly selected individual as $I_\delta$ from $P_{i+1}$
        Perform a randomly chosen mutation operator on $I_\delta$
        $P_{i+1} := P_{i+1} \cup \{I_\delta\}$
        $k := k + 1$
    end
    $P_i := P_{i+1}$
end
```

3.2. Evaluation of the genetic algorithm

This section focuses on the tuning of the adapted GASOPE and the evaluation of the predictive accuracy of the adapted GASOPE. Section 3.2.1 presents the experimental procedure and the accuracy of the adapted GASOPE compared to originally recorded results of Potgieter and Engelbrecht [22]. The hyperparameters of the adapted GASOPE are tuned in section 3.2.2 specifically for a multi-dimensional dataset. Finally, adapted GASOPE is compared in section 3.2.3 with other regression models and the results obtained by GPMCC with the original GASOPE on a dataset incorporating nine input features.

3.2.1. Experimental procedure and results

Four functions were used to evaluate the predictive accuracy of adapted GASOPE on unseen data. Table 3.1 lists these four functions with their function definitions. All functions were injected with uniformly generated noise over the interval $[-1, 1]$. A total of 12 000 instances were created for each function to make up a dataset. Of these 12 000, 10 000 were allocated for the training set, 1 000 for the validation set and 1 000 for the test set. All tests were repeated 100 times with the dataset being shuffled for each.

Table 3.1: Function definitions on which GASOPE was evaluated.

Name	Function
h_1	$h(x_0) = \sin(x_0); x_0 \in [0, 2\pi]$
h_2	$h(x_0, x_1) = \sin(x_0) + \sin(x_1); \{x_0, x_1\} \in [0, 2\pi]$
h_3	$h(x_0) = x_0^5 - 5x_0^3 + 4x_0; x_0 \in [-2, 2]$
h_4	$h(x_0, x_1) = x_0^5 - 5x_0^3 + 4x_0 + x_1^5 - 5x_1^3 + 4x_1; \{x_0, x_1\} \in [-2, 2]$

The hyperparameters of the adapted GASOPE were chosen as that specified by Potgieter and Engelbrecht [22]. The hyperparameters values that were used are shown in Table 3.2.

Table 3.2: Chosen values for the hyperparameters of the adapted GASOPE.

Hyperparameter	Value
Elite	0.1
Mutation rate	0.1
Crossover rate	0.2
Maximum terms	20
Maximum order	5
Population size	100
Generations	30

Table 3.3 lists the results achieved by GASOPE in [22] on the same functions, together with the results obtained from the adaptation of GASOPE.

Predictive performance was measured using the average mean square error on the test set over the 100 test runs, with σ_{MSE} indicating the standard deviation. The statistical significance of the results could not be evaluated. However, from observation there are no notably large discrepancies. It is clear from the results shown in Table 3.3 that the alterations implemented in the adapted version of GASOPE did not compromise its predictive accuracy. However, further testing on multi-dimensional data is required to verify this.

It is unclear if the hyperparameter values listed in Table 3.2 are indeed the optimal values for adapted GASOPE. The hyperparameter values of the original GASOPE were initially chosen based on numerous experimental runs conducted in [22]. Therefore, before further evaluation of the adapted GASOPE, the hyperparameters require tuning.

Table 3.3: The accuracy of GASOPE compared to the original from [22].

Function	Model	$\overline{MSE}$	σ_{MSE}
h_1	Original GASOPE	0.343135	0.001644
	Adapted GASOPE	0.332955	0.009163
h_2	Original GASOPE	0.345282	0.001698
	Adapted GASOPE	0.330054	0.009625
h_3	Original GASOPE	0.339414	0.001765
	Adapted GASOPE	0.334083	0.009059
h_4	Original Gasope	0.332791	0.001935
	Adapted GASOPE	0.342808	0.007728

3.2.2. Tuning process of the genetic algorithm

Shortcomings of the original evaluation of GASOPE include the lack of multi-dimensional datasets and motivation for the choice of hyperparameters. The choice of hyperparameter values (listed in Table 3.2) for original GASOPE were applicable on only one- and two-dimensional functions as listed in Table 3.1.

To ensure that the adapted GASOPE is the optimal model to employ at the leaves of an MTF, it was first tuned and evaluated on a multi-dimensional problem. There are many openly available datasets from the UCI Machine Learning Repository[1]. The Abalone dataset was chosen because the results GPMCC previously produced on this dataset were not as favourable [23]. It is hypothesised that Appropriate tuning of the hyperparameters of GASOPE, as well as the newly implemented changes, could improve the performance of the adapted GASOPE on the Abalone dataset.

Evaluation of the predictive accuracy of GASOPE on the dataset was done for 30 independent runs. For each run, the dataset was randomly split into a training set, validation set, and test set in an 80%:10%:10% split. Tuning makes use of the training and validation set. The first three hyperparameters that require optimisation are

1. population size,
2. the number of generations evolved for, and
3. the maximum number of terms of an individual.

The computational expense for the above-mentioned hyperparameters required critical evaluation during tuning. Furthermore, each of the three hyperparameters negatively influenced the computational cost of the adapted GASOPE as the value of the hyperparameter was increased.

[1]https://archive.ics.uci.edu/ml/datasets.php

The reason for this is that each hyperparameter increases the number of computations required to evolve an optimal polynomial.

To find the optimal value for the population size, the adapted GASOPE with population size varying from 10 to 60 in increments of five was used to evolve polynomial functions to fit the training set. Figure 3.6 illustrates the validation set RMSE for each instance of the adapted GASOPE with varying population sizes. From the box plots in Figure 3.6, which portray the RMSE distribution over the 30 independent runs, it is clear that for very small population sizes the RMSE value tended to fluctuate more. The smallest population size that produced consistently low RMSE values was that of 30 individuals. Figure 3.7 shows how the average training set RMSE decreased with each generation. Population sizes smaller than 30 resulted in an increased average training set RMSE. Once the population size was increased above 30, both the training set RMSE as well as the validation set RMSE converged.

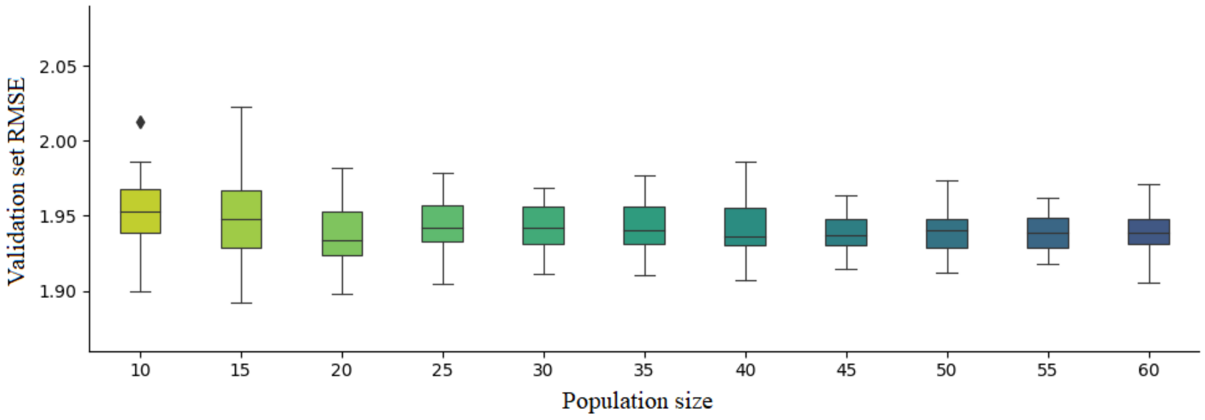


Figure 3.6: RMSE on the Abalone validation set for the adapted GASOPE models with a varying number of individuals within the population.

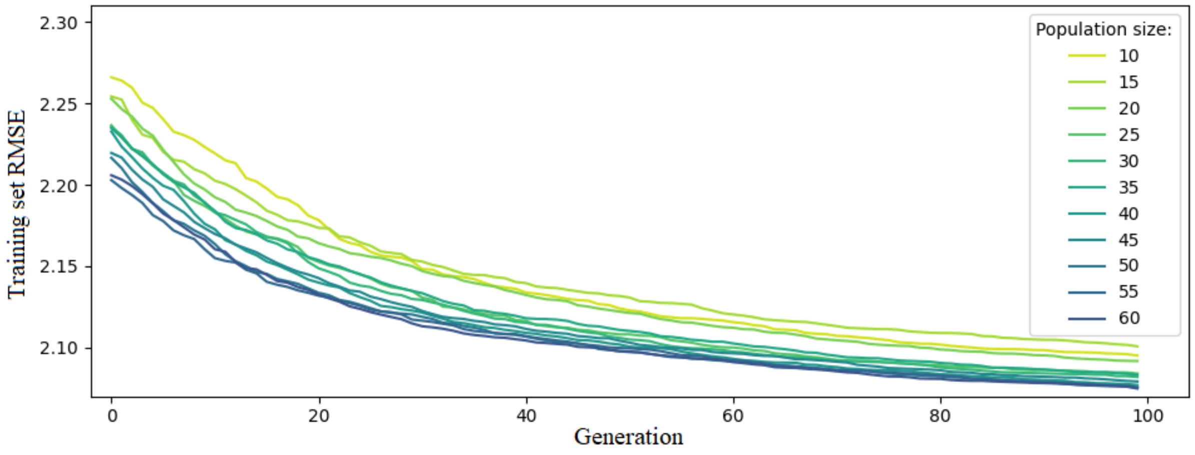


Figure 3.7: RMSE on the Abalone training set for each generation of the adapted GASOPE models with a varying number of individuals.

The next hyperparameter which influences computational costs is the number of generations the adapted GASOPE takes to evolve a population of individuals. Figure 3.8 provides the box plots of 30 independent RMSE scores of the best polynomial evolved using the adapted GASOPE for generations varying from 10 to 200 in increments of 10. As the number of generations increased, the RMSE values tended to decrease as well as fluctuate less. However, this tendency started to drop off after 100 generations and the RMSE converged.

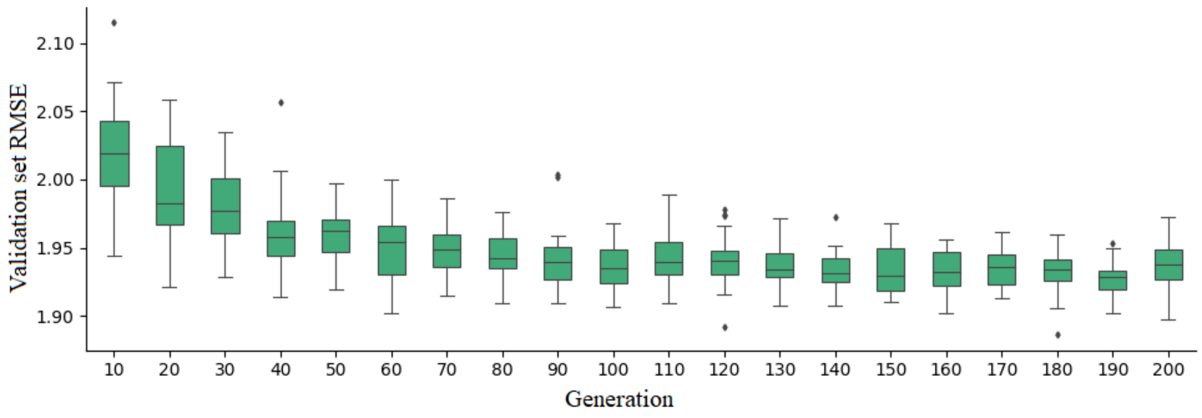


Figure 3.8: RMSE GASOPE with a population size of 30 produced on the Abalone validation set over each generation for 30 repeated runs.

Next, the maximum number of terms is discussed. The maximum number of terms had the largest contribution to the computational cost associated with evolving an optimal polynomial. Therefore, the maximum number of terms was chosen as small as possible without compromising the performance of the adapted GASOPE, whilst also large enough to ensure that the initial search space was adequately covered. From the implementation of GASOPE in GPMCC, a value of 10 for the maximum number of terms was found to produce sufficient results in multi-dimensional problems and the best performing individual was found to often comprise less than 10 terms [23]. Note that the maximum number of terms hyperparameter is only applicable to the initialisation of an individual.

The next hyperparameter for discussion, i.e. maximum polynomial order, is considered unique to each case that the adapted GASOPE is applied. Polynomials should not be allowed to incorporate large polynomial orders when the dataset itself does not exhibit highly nonlinear tendencies. For polynomial orders that are too large, the adapted GASOPE may impose nonlinear relationships that are not necessarily prevalent in the data, which leads to overfitting. Despite the fitness function penalising larger polynomial orders more severely, there is still a need to tune the maximum polynomial order per problem. As discussed in Section 2.2, if a model is too complex in its composition for the data at hand, it will increase the variance error and subsequently overfit the data. The Abalone dataset is quite linear, but the original GASOPE within GPMCC evolved polynomials orders up to a value of four [22]. Limitation of the maximum polynomial order of the adapted GASOPE to a value smaller than four may produce improved results. However, the adapted GASOPE cannot adequately capture any nonlinear relationships in the dataset when a small maximum polynomial order is chosen.

To analyze the effect that the maximum polynomial order had on the performance of the adapted GASOPE in the case of the Abalone dataset, three different instances of the adapted GASOPE were further tuned and compared. Each instance was evolved with a respective maximum polynomial order, o, set to 1, 2 and 3. For each of the three adapted GASOPE instances, the remaining hyperparameters that require optimisation were the

- crossover rate,
- mutation rate, and
- elitism rate.

Bayesian optimisation was applied to tune the remaining hyperparameters of the adapted GASOPE on the validation set. Bayesian optimisation is a tuning method used to determine the hyperparameter values that minimise a given objective function [10]. Bayesian optimisation was used to build probabilistic models of the function mappings between hyperparameter values and the objective function. Bayesian optimisation used the RMSE on the validation set as the error metric that formed the objective function. Although Bayesian optimisation is not as exhaustive as a gridsearch, it is less time-consuming for large parameter spaces [27]. Table 3.4 lists the final hyperparameter values obtained from Bayesian optimisation for each GASOPE instance.

Table 3.4: GASOPE hyperparameters obtained through Bayesian optimisation.

Hyperparameter	GASOPE with $o = 1$	GASOPE with $o = 2$	GASOPE with $o = 3$
Elitism rate	0.15	0.3	0.25
Mutation rate	0.3	0.2	0.3
Crossover rate	0.3	0.2	0.2
Maximum terms	10	10	10
Population size	30	30	30
Generations	100	100	100

3.2.3. Evaluation against other regression models

For the comparison of the prediction accuracy of adapted GASOPE on a higher-dimensional problem, competing regression models were also trained and evaluated on a test set of the Abalone dataset. The regression models were a support vector regression (SVR) model, incorporating a Gaussian kernel function, as well as a simple linear regression model used by M5 at its leaf nodes. GPMCC, which utilises the original GASOPE, was evaluated on the same higher-dimensional problem and therefore also serves as an adequate comparison for the adapted GASOPE [23]. The experimental procedure of the evaluation follows.

For the linear regression model, there were no hyperparameters to tune. For the SVR model, two hyperparameters were tuned with Bayesian optimisation. The SVR hyperparameters were the values for γ and C. The γ value represented the coefficient of the Gaussian kernel function and the C value was a regularisation parameter.

To evaluate the accuracy of a model, the results of 30 independent runs were recorded. Recall that the Abalone dataset was split into a training, validation, and test set. Here the training and test sets are used. For all models, excluding the original GASOPE within GPMCC, both RMSE and MAE values with standard deviation, σ, were used for evaluation. Note that RMSE is considered a more robust estimation of the performance of a model than MAE, because it penalises larger errors more severely [8]. For each run, the five models that were compared were ranked according to the RMSE and MAE errors respectively. The average ranking was thereafter used to test the statistical significance of the results.

The Friedman test [26] is a non-parametric statistical test to detect significant differences between the performance of the models. The null-hypothesis was that all models are equal in their performance based on the results. In the event that the null-hypothesis was rejected, a post-hoc test was performed. A post-hoc test is a statistical test performed to further distinguish the results. In this case, the post-hoc test showed which models were statistically dissimilar in accuracy. The post-hoc pairwise test used was the Bonferroni-Dunn test because it is best suited for comparisons of several methods with a control method (in this case, the linear regression model serves as the control method) [11]. In the statistical tests, a significance level of 5% was used. The significance level is the probability of rejecting the null-hypothesis, given that it is true.

Table 3.5 presents the results obtained for each model together with the results that the original GASOPE within GPMCC obtained on the Abalone dataset [23]. SVR scored the lowest MAE and average rank amongst all models. Adapted GASOPE with a maximum polynomial order of two achieved the lowest RMSE and second-best average ranking. The null-hypothesis was rejected and the post-hoc test was performed to show which algorithms were statistically dissimilar.

Table 3.5: Comparison of different adapted GASOPE instances on Abalone.

Model	$\overline{RMSE}$	σ_{RMSE}	$\overline{MAE}$	σ_{MAE}	Average Rank
GPMCC	n/a	n/a	1.6051	0.0156	n/a
GASOPE with $o = 1$	2.159098	0.082020	1.531867	0.052391	3.875
GASOPE with $o = 2$	2.125721	0.084797	1.480717	0.055245	2.033
GASOPE with $o = 3$	2.142215	0.080056	1.497536	0.055468	2.783
SVR with Gaussian Kernel	2.136233	0.086790	1.434549	0.053255	1.733
Linear Regression	2.162230	0.080725	1.544813	0.053264	4.575

Figure 3.9 presents the critical difference diagram that summarises the results of the post-hoc test. A critical difference diagram indicates the average rank of each method and connects those that were not statistically different. It is evident from Figure 3.9 that the results of SVR and the adapted GASOPE with $o = 2$ were statistically equivalent. The linear regression and the adapted GASOPE with $o = 1$ were also considered statistically similar. This is expected because the adapted GASOPE with $o = 1$ produced a polynomial with an equal order to linear regression. Thereby, the two models had a similar discriminative ability. GASOPE with $o = 3$ is shown to have produced statistically worse results than both SVR and GASOPE with $o = 2$.

This confirms that the imposition of higher-order functions on data that did not comprise nonlinear relationships impacted the accuracy of the adapted GASOPE negatively. However, GASOPE with $o = 3$ was ranked better than GASOPE with $o = 1$. This means that a too-small maximum polynomial order was also detrimental to the accuracy of GASOPE.

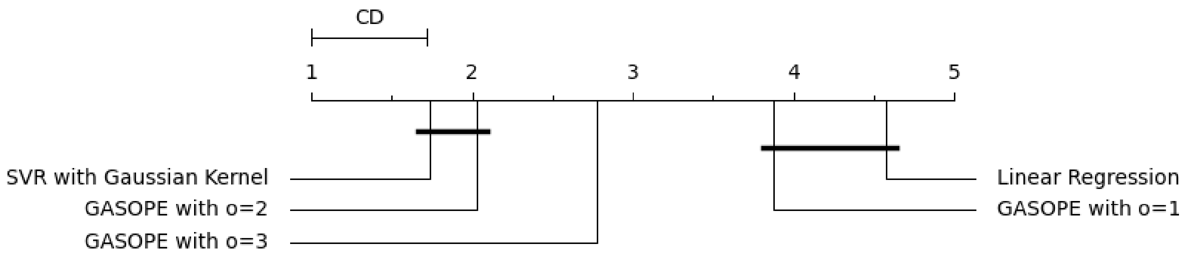


Figure 3.9: Critical difference diagram for the results obtained in Table 3.5.

3.3. Conclusion

This chapter investigated the original implementation of GASOPE and defined the proposed changes that are required for the use of GASOPE with increased computational power. After testing, it was evident that the changes brought into the adapted GASOPE produced improved results when compared to the original implementation of GASOPE within GPMCC. The results GASOPE produced were adequate to justify implementation in the leaf nodes of MTF.

To prevent overfitting in any future instances of the adapted GASOPE, careful consideration should be given to the maximum polynomial order value. There is no universal maximum polynomial order value that will result in the evolution of the optimal polynomial for all cases. Although GASOPE already has systems in place to penalise overly complex polynomials, in some cases the fitness function alone cannot prevent the evolution of overly complex polynomials. Through testing on a dataset which comprises linear relationships, it was found that a too-large maximum polynomial order flooded the population of individuals with overly complex polynomials during initialisation. The fitness function as well as crossover and mutation operators cannot escape these local optima which resulted in overly complex evolved solutions. Therefore, the maximum polynomial order hyperparameter should be carefully chosen for each unique dataset. Doing so ensures that the initial search space is close enough to the globally optimal solution.

Chapter 4

Model tree forest

The model tree forest induction algorithm proposed in this thesis is presented and discussed in this chapter[1]. MTF consists of an RF-inspired tree induction algorithm that applies GASOPE to evolve optimal multivariate polynomial functions for each leaf node within a given tree of the ensemble. Section 4.1 starts with a discussion of the successful techniques other existing decision tree ensembles utilise to mitigate high variance errors. Thereafter, Section 4.1 continues with a discussion on how the robustness and computational efficiency of MTF is maximised. Finally, Section 4.1 concludes with the pseudo-code of the MTF induction algorithm. Section 4.2 concludes the chapter with a discussion on the conceptualisation of MTF.

4.1. The model tree forest method

This section presents the methodology of the model tree forest. The required steps to ensure that the base learner model trees within MTF are sufficiently decorrelated, similarly to the trees within RF, are motivated and listed in Section 4.1.1. Thereafter, Section 4.1.2 discusses the steps implemented in the induction algorithm of MTF to address robustness and computational costs. Finally, Section 4.1.3 presents the complete pseudo code of the MTF induction algorithm.

4.1.1. Inspirations from existing ensemble methods

As discussed in Chapter 2, model trees have a greater intrinsic ability to approximate continuous nonlinear data compared to the simpler regression trees found in RFs. This is due to the polynomials that are employed to produce the output of a model tree for a given instance as opposed to the singular output value in the leaf nodes of a regression tree. The substitution of constant values for polynomials, specifically higher-order polynomials, brings an increase in the observed variance error. However, it has been shown by Equation (2.1) in Section 2.2.3 that the variance error can be effectively reduced through the use of an ensemble of decorrelated trees. Recall that RFs extend the idea of the decorrelation of decision trees through the introduction of randomly selected subsets of input features. The subsets of input features are used for the evaluation of a proposed split during the induction of a single tree. Therefore, RF-inspired induction steps are implemented in MTF to help reduce the expected high variance error that MTF inherits from both the adapted GASOPE and the decision tree architecture of MTF.

[1]The entire model is openly available at https://github.com/WernerVdM97/Masters-Thesis.

The base learners of MTF, which together form an ensemble, are referred to as model trees (MTs). An MT employs the same splitting function as the M5 model tree (refer to Equation (2.2)). What differentiates the base learners of MTF from all other model trees discussed in this thesis, is the use of the adapted GASOPE to evolve higher-order polynomials to describe the mapping between the input and output features for all of the instances within a leaf node (refer to Equation (3.4)). For the M5 model tree, a simpler linear model is employed, previously defined by Equation (2.4). For a regression tree employed by an RF, the output functions are simply replaced by a constant value in the leaf nodes. The increased complexity of linear models and higher-order polynomials used to produce the output of a model tree leads to an increase in variance error. Therefore, the high variance error of MTF is attributed to two factors:

1. The decision tree structure of the base learners that comprise MTF.
2. The higher-order polynomials evolved via GASOPE in the leaf nodes of MT.

In turn, these two factors together also result in a low bias error.

Similar to RFs, the decorrelation of base learners may help an MTF to retain a more favourable bias error, whilst the variance error is kept low enough to not degrade the performance of the MTF. A previous approach employed to combat the high variance error of an ensemble was the introduction of base learner pruning. Base learner pruning is omitted in RFs due to the introduction of decorrelated base learners to sufficiently reduce the observed variance error of an ensemble. The advantageous low bias error associated with decision trees is increased when pruning is applied to a decision tree. Equation (2.1) in Section 2.2.3 showed that the high variance error of bagged decision tree ensembles can be reduced through decorrelation of the base learners within the ensemble. An increase in the number of decision trees in the ensemble effectively reduces the variance error. However, the reduction in variance starts to diminish the larger the number of trees becomes.

In conclusion, random splitting feature space selection together with an increased ensemble size is implemented in MTs and MTFs to reduce the high variance error of MTFs. The only factor that limits the ensemble size of an MTF is the computational cost of the induction of large numbers of MTs in serial.

4.1.2. Increased robustness and computational efficiency

An MTF attempts to not only perform well on complex data with highly nonlinear mappings but on simpler data as well. As discussed in the previous chapter, GPMCC performed well on nonlinear cases, but lacked the flexibility to also perform sufficiently on linear cases. GPMCC is not robust enough and is hypothesised to overfit simpler data. Furthermore, it was shown that the adapted GASOPE, if correctly tuned, can generalise adequately to data that comprises linear mappings between the input and output features. To ensure that an MTF produces favourable results for both linear and nonlinear cases, the resilience of MTFs to overfitting is addressed in this section.

As discussed in Section 2.5.4, Aleksovski includes a greedy ensemble pruning step to optimise MTEs. Decision trees that do not contribute to the accuracy of the ensemble are removed. The removal of inaccurate decision trees aims to produce a more accurate model of the training data. The contribution of an individual tree is evaluated through the performance of the ensemble both with and without the tree at hand. The same greedy optimisation step implemented in MTEs is employed in MTFs.

Previously discussed approaches used to increase the robustness of a decision tree model were pruning and smoothing or fuzzification. Pruning was addressed in the previous section and omitted because the ensemble structure of MTF replaces the need for pruning. Smoothing is applied to decision trees, such as the M5 model tree, to minimise the discontinuities that are formed between the output functions of adjacent leaf nodes. Aleksovski concluded that the modifications introduced to the original M5 model tree, including fuzzification, did not yield conclusive results for MTEs [1]. Furthermore, smoothing and fuzzification were both applied to model trees that incorporated linear models which are more likely to produce discontinuities as opposed to nonlinear models.

For an MTF, the computational cost is a major dictator in the freedom to which the model is allowed to increase in its complexity. All parameters and aspects of the induction algorithm of an MTF should be designed to minimise computational costs. Therefore, the added computational cost of the implementation of smoothing, fuzzification, or a similar process into an MTF is detrimental to the performance. This fact holds especially in the case of smoothing for which an output function is induced for each node within the model tree, not only the leaf nodes. Through the omission of the smoothing step, MTFs are given an increased computational budget for the induction of the base learner model trees. Additionally, the nonlinear output function at the leaf nodes of an MT already minimises any discontinuities between neighbouring leaf nodes.

In the event that a polynomial evolved by the adapted GASOPE produces a poor approximation of the given data, the induction algorithm of an MTF employs a baseline linear model comparison (BLMC) step. The BLMC step evaluates the performance of the higher-order polynomial for a given leaf node against a simpler linear model and if the linear model is found to outperform the higher-order polynomial, the linear model is instead used to model the output for the given leaf node. This step acts as a fallback mechanism in the case that the adapted GASOPE is unable to properly approximate or overfits a given data subset. The BLMC step is aimed at the improvement of the approximation of leaf nodes with a low number of training instances. A decrease in the number of training instances increases the sample variance and thereby increases the difficulty for the adapted GASOPE to produce a fair approximation.

Given that each base learner is trained on a randomly drawn subset of the original dataset, outliers may be more prolific by chance. Decision trees are typically good at the isolation of outlier instances in leaf nodes. However, MTs incorporate the adapted GASOPE which is susceptible to the evolution of poor approximations for data with outliers. Therefore, MTFs require additional measures in place to combat the influence of base learners that exhibit poor performance. In the case where a model tree is induced that significantly decreases the accuracy of the ensemble, MTFs include an ensemble pruning step to ensure that the accuracy of the final ensemble remains favourable.

Ensemble pruning is implemented as follows: The size of the ensemble is reduced through the singular omission of each base learner model tree from the output calculation. Only one base learner is omitted from the ensemble at a time. If the validation set error metric of the ensemble model is improved with the omission of the base learner which is subject to evaluation, the base learner is permanently removed from the final ensemble.

Finally, to further ensure a robust estimation is made for a set of input data, MTFs employ a trimmed mean output function to calculate the predicted value. Trimmed mean refers to the removal of a certain number of minimum and maximum values from a set before the mean value is computed [19]. In the context of an MTF, the output values of the base learners which produce the minimum and maximum numerical values for the same set of input values, are excluded when the value predicted by the ensemble as a whole is calculated. Trimmed mean removes the negative impact of outlier predictions. Note that the omission of minimum and maximum values are applied per data instance.

The addition of the BLMC step, ensemble pruning and trimmed mean prediction increase the robustness of an MTF, ensuring that the MTF performs adequately on unseen data.

4.1.3. Pseudo code

The pseudo-code that describes the induction of each MT, as well as MTF, is presented in this section. The induction algorithm of MTFs is described in Algorithm 7 and the induction algorithm of MTs is described in Algorithm 8.

Algorithm 7: Pseudo-code for MTF induction.

```
Initialise empty ensemble array, $E = []$
while stopping criterion is not met do
    Randomly sample with replacement from $D_t$ to produce $D_\gamma$
    Call MT induction algorithm and parse $D_\gamma$
    Insert the returned MT into $E$
end

Given the training set $D_t$
and the validation set $D_v$
Compute a baseline $MSE$ on $D_v$
for $i = 1$ to $|E|$ do
    Prune $MT_i$ from $E$
    Compute $MSE'$ on validation set
    if $MSE \times c_s > MSE'$ then reinsert $MT_i$ into $E$
end
```

Algorithm 8: Pseudo-code for MT induction.

```
Initialise root node, $N_0$.
Given the training set $D_t$ with $n$ instances, initialise an empty set of nodes $Z = \{\}$
Let $Z = Z \cup N_0$
Call Grow($N_0, D_t$)

Grow ($N_i, D_i$):
if stopping criteria is met then
    Label $N_i$ a leaf node
    Parse $D_i$ to GASOPE
    Save the returned output function in $N_i$
else
    Select $\{g_1, g_2, .., g_n\}$ as a random subset of feature indexes from $D_i$
    Initialise $S = \{g_1, s_1\}$
    for $j = 1, .., n$ do
        for $k = 1, .., l$ do
            Select sample value $s_k$ from $D_i$
            Let $S' = \{g_j, s_k\}$
            Compute $SDR(S')$
            if $SDR(S') < SDR(S)$ then $S = S'$
        end
    end
    Initialise left child node, $N_L$, and right child node, $N_R$
    Split $D_i$ into $D_L$ and $D_R$ according to $S$
    $Z = Z \cup \{N_L, N_R\}$
    Grow( $N_L, TS_L$ )
    Grow( $N_R, TS_R$ )
end
```

A single MTF model comprises multiple MT base learners and a single MT further comprises multiple higher-order polynomial output functions evolved through GASOPE. Therefore, the MTF induction algorithm consists of three nested induction stages:

1. evolving polynomials with GASOPE,
2. MT induction, and
3. MTF induction.

The MT induction algorithm incorporates a recursive node splitting function. Starting at the root node, multiple candidate splits are evaluated and unless stopping criteria are met, the best split is chosen. The best split maximises the reduction in standard deviation over the target values of the instances characterised by the split. For MTFs, the stopping criteria are either when a maximum tree depth is met, or when the number of instances on which to split is smaller than twice the minimum number of leaf instances. The latter ensures that both children nodes house more instances than the minimum number of leaf instances. Thereafter, each child node is evaluated for further splitting. Once a stopping criterion for a node is met, the node is labelled a leaf node and the recursive splitting function continued on the remaining nodes. All splits that together define the decision tree structure of each MT is stored in a recursive set.

The i^{th} node, N_i, within a decision tree is described by the following set:

$$N_i = \{S, N_L, N_R, f_i(\boldsymbol{x})\} \tag{4.1}$$

where S is a set that describes the splitting conditions for node N_i; N_L and N_R are the left and right children nodes of N_i, and $f_i(\boldsymbol{x})$ is the function evolved by adapted GASOPE that defines the output of N_i for a given input vector $\boldsymbol{x}$. In the case of a leaf node S, N_L and N_R are left void, whilst $f_i(\boldsymbol{x})$ is defined. For any other node, the opposite is true, i.e. only the $f_i(\boldsymbol{x})$ parameter is void, whilst S, N_L and N_R are defined. A split is defined through the set of two parameters:

$$S = \{g, s\} \tag{4.2}$$

where g is the index of the input feature on which to split and s is the value of the split. for a dataset with 10 features and 100 instances, there are $10 \times 100 = 1000$ possible splits for the root node. However, splits are proposed from a randomly selected subset of the feature space at each node. Furthermore, proposed splits are only evaluated on the set of instances housed by the node. Evaluation of a proposed split is performed with the reduction in expected error function (refer to Equation (2.2)) returns a value that represents the standard deviation reduction (SDR) of the proposed split. The optimal split for a given dataset is chosen after iterating over each value of each feature and calculating the corresponding SDR of the proposed splits. The split with the highest SDR is selected as the optimal split.

The induction of an MT for a given path is halted when a stopping criterion is met. Once a stopping criterion is met, the applicable node is labelled a leaf node. The data subset associated with a leaf node is determined through the splits on the path from the root node to the leaf node. Thereafter, the adapted GASOPE evolves the output function that is stored in the leaf node. The polynomial evolved by the adapted GASOPE is compared to the baseline linear function. The accuracy of both functions is compared through the MSE scores on the training data within the leaf node. Finally, the function which minimises the MSE is chosen as the output function of the leaf node.

When an MTF induces an entire forest of model trees, Algorithm 8 is executed on several bootstrapped subsamples of the training set. Each bootstrap subset is selected with replacement. The size of the sampled dataset is equal to the original dataset. Given sampling with replacement, the bootstrap subset contains approximately 63% unique instances of the original dataset [6]. Thereby, model trees are decorrelated in the ensemble. Once the stopping criterion is met, the induction phase concludes. For MTF the only stopping criterion is the maximum ensemble size.

MTF calculates a vector of predicted outputs, $\boldsymbol{y'}$, for a given set of instances, X, as the uniformly weighted average over the predictions for a set of model trees within the ensemble:

$$\boldsymbol{y}'(X) = \frac{1}{n_t} \sum_{i=1}^{n_t} g_i(X) \tag{4.3}$$

where $\boldsymbol{y}'_i(X)$ denotes the predicted value of the i^{th} model tree within the ensemble, given the set X. The number of base learners within the ensemble is denoted by n_t. The set of model trees on which Equation (4.3) is computed is determined by the omission of a number of base learners equal to 10% of the ensemble size. This 10% comprises the minimum 5% and maximum 5% from the set of total values predicted by the base learners of the ensemble. For example, in an ensemble of 100 MTs, the five lowest and five highest predicted values are omitted. Equation (4.3) is applied to the remaining 90 values to compute the value predicted by MTF for a given instance.

After the first phase follows the ensemble pruning phase. At the start of the ensemble pruning phase, a baseline MSE is calculated on the validation set. MTF employs a greedy pruning step where each MT in the ensemble is individually omitted from the ensemble and the MSE on a validation set is thereafter calculated. If the MSE with the respective tree omitted is significantly smaller than the baseline MSE, the tree is permanently pruned from the ensemble. For an increase in MSE to be regarded as significant, the MSE must have increased with at least a factor of c_s, where $c_s = 1 + \frac{1}{|E|}$ ($|E|$ refers to the ensemble size). The value of c_s was chosen such that MTs must increase the MSE of the ensemble by a factor greater than the uniform weight of a base learner on the predicted value within the ensemble. For example, an ensemble that comprises 20 MTs requires that a single MT increases the MSE score by more than 5% to justify pruning it away.

Figure 4.1 illustrates the ensemble method employed by MTF to induce the base learners and to produce an output from the ensemble. Figure 4.2 illustrates the complete MTF induction algorithm.

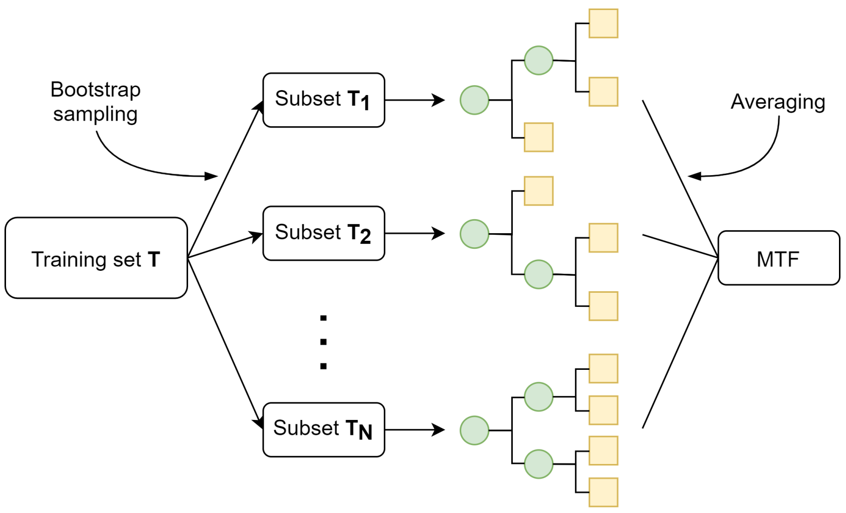


Figure 4.1: A diagram of the MTF bagging and output calculation.

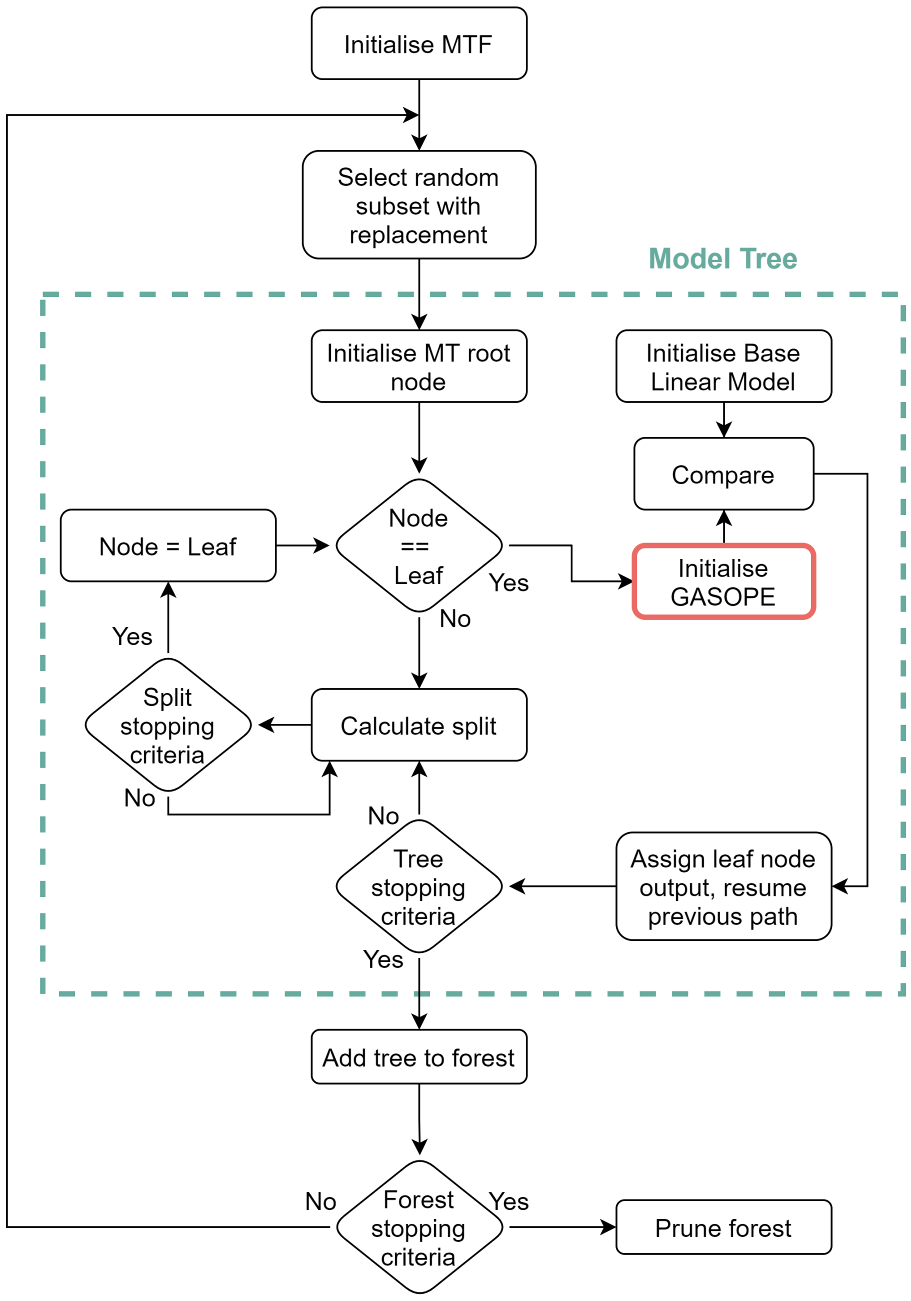


Figure 4.2: A diagram of the MTF induction algorithm.

4.2. Conclusion

This chapter introduced the complete model tree forest method. The decorrelation step of RF that minimises the high variance error of regression trees is also implemented in MTF. The decorrelation step includes the induction of base learners on bagged subsets of the dataset and the random selection of splitting feature subsets.

To avoid overfitting data, the induction algorithm and output function of MTF incorporate steps to improve the robustness of the ensemble. The induction algorithm prunes the ensemble as a whole after a user-specified number of MTs were induced. After the induction of a higher-order polynomial function for a leaf node, a baseline linear comparison step is included to overwrite a polynomial that generalises poorly to unseen data with a simple linear model. Finally, the ensemble output function trims away the minimum and maximum predicted values of the base learners for an unseen data instance before the uniformly weighted average is computed. Unnecessary pruning and smoothing techniques of singular model trees are omitted from MTF to reduce computational costs.

Chapter 5

Experimental procedure

This chapter discusses the experimental procedure for the tuning and comparison of MTF against four state-of-the-art regression ensemble models on ten benchmarking datasets. The chapter is outlined as follows. Section 5.1 discusses the datasets, models selected for comparison and metrics used to define the performance of a given model. The tuning process of all the hyperparameters of each model is discussed in Section 5.2. Section 5.3 discusses the series of experiments designed to evaluate the performance of MTF and the statistical tests that were applied to the experimental results. Finally, this chapter is concluded in Section 6.3 with a discussion of the experimental procedure.

5.1. Empirical analysis

Section 5.1.1 discusses the datasets on which MTF and all other models selected for the comparison were evaluated. Thereafter, Section 5.1.2 motivates the selection of models used for comparison against MTF. Finally, the metrics used to empirically analyse the performance of models are discussed in Section 5.1.3.

5.1.1. Datasets

To ensure that the evaluation of an MTF is thorough, ten datasets each with a unique composition were obtained from the UCI machine learning repository and the L. Torgo Repository [1]. Six datasets from the evaluation of GPMCC were selected and an additional four datasets were chosen to expose MTF to larger datasets [23]. All datasets contained a single numerical target feature that a given regression model was assigned with predicting. Predictions were made on a set of input features that were nominal and/or continuous. The ten datasets are listed in Table 5.1 with their attributes.

Large datasets are defined as datasets comprising more than 10000 instances. In the evaluation of GPMCC, only one large dataset was present from the UCI Machine Learning Repository, namely the Elevators dataset [23]. The other two large datasets were artificially generated. At the time, there was a lack of sufficient large databases covered by the UCI Machine Learning Repository [23]. For the evaluation of MTFs, there are additional large benchmarking datasets available, namely Physicochemical Properties of Protein Tertiary Structure (CASP)

[1]Openly available at https://www.dcc.fc.up.pt/ ltorgo/Regression/DataSets.html

Table 5.1: The benchmark datasets used for the evaluation of the performance of an MTF.

Dataset	Number of features (N = Nominal, C = Continuous)	Number of instances
Abalone	1N 7C	4,177
Auto-MPG	3N 4C	398
Boston housing	0N 13C	506
California housing	0N 8C	20,460
CASP	0N 9C	45,730
Elevators	0N 18C	16,559
Friedman artificial domain	0N 10C	40,768
Machine CPU	0N 6C	209
MV artificial domain	3N 7C	40,768
Servo	4N 0C	167

and California housing. Two large artificial datasets are also used in the evaluation of MTFs, which are the MV artificial domain and Friedman artificial domain datasets. The first was used by Torgo in comparing the performance of PLT and MARS, two favourable regression techniques at the time [28]. The second artificial dataset was used by Friedman in the conceptualisation of MARS and the evaluation of bagging predictors [6, 12].

The ten datasets were chosen to ensure that all degrees of dimensionality are covered during testing. The number of features ranges from a minimum of 4 to a maximum of 18. Decision tree-based methods are expected to have the advantage of embedded feature selection on higher dimensional datasets.

The datasets are first processed before any evaluation takes place. Missing values are all dropped, only California housing and Abalone incorporate missing values. Feature values are scaled into a range of $[0, 1]$.

5.1.2. Selected models for comparison

The focus is on a comparison of decision tree-based methods and how the introduction of ensembling and decorrelation of base learners can produce improved performance. It is also necessary to first confirm that the ensembling of MT is indeed beneficial. This is the only comparison of a singular model with an ensemble of models. To ensure a fair comparison, MTF is compared only to other ensembled methods. There are several questions that motivate the choice of models against which the performance of MTF is evaluated:

- Does ensembling a model tree help to reduce the problematic high variance error that model trees exhibit?
- Apart from bagging, can additional decorrelation steps within an ensemble of model trees further reduce the high variance error?

- Is the complexity of MTF unfavourable for datasets that have linear relationships between their input and target features?
- Is MTF able to outperform the state-of-the-art ensembled regression techniques which are specifically catered towards nonlinear problems?

The first question is addressed through a comparison of an MTF to an MT. M5 is also included in this comparison. The accuracy of MTF is directly compared to singular model trees. This comparison illustrates the efficiency of ensembled model trees in reducing variance. The reduction of variance improves the generalisation accuracy.

The second question is answered through a comparison of an MTF to a bagged M5 ensemble (M5E). M5E incorporates only a simple bagging ensemble technique with no other decorrelation strategies implemented. Both M5s and M5Es are openly available through the Cubist package[2].

The third question is addressed by the addition of an RF and an artificial feed-forward neural network ensemble (NNE). Both RFs and NNEs have the capability to produce favourable results for data with varying degrees of complexity in the relationships between the input and output variables.

For the final question, a bagged support vector regression model ensemble (SVRE) is added to the list of comparative models. SVREs incorporate a Gaussian kernel function in the SVR base learners to model nonlinear problems.

5.1.3. Performance metrics

Multiple performance metrics are chosen to provide insight into which model performs best given a set of unseen data. Performance is determined by both the prediction accuracy of unseen data and the computational cost of training the model.

The error metric used to express the accuracy of a given model is either MSE defined in Equation (2.5) or the square root thereof (RMSE). For the induction algorithm and the tuning process of an MTF, MSE is chosen to minimise computational costs, because MSE has fewer arithmetic calculations than RMSE. In turn, RMSE is chosen for the evaluation of the predictive accuracy of a tuned model. RMSE is more interpretable than MSE because it is measured in the same unit as the target variable. Therefore, RMSE is the given metric for tabulated results. MAE is omitted because MSE penalises larger errors more severely than MAE.

For each independent test run, the RMSE is computed on both the training and test sets. The comparison of both the training and test set error metrics allows for the identification of overfitting. Given that the test set error is worse than the training set error, if the difference between the errors increases a model is likely to have overfitted the data at hand. The variation of the RMSE over the 30 independent test runs is included to indicate how stable a model is, i.e. how consistent is a model in making predictions for each shuffled set of the data.

The last applicable metric is the absolute computational time to execute the induction algorithm for a model, expressed in seconds. It is important to put the performance of a

[2]Cubist is an R port of the Cubist GPL C code released at http://rulequest.com/cubist-info.html.

model into perspective, given how computationally expensive it is to induce. A model which only marginally outperforms another at a significantly higher computational cost is not always favourable. For cases which require regular training of new instances of a model, it is advantageous to evaluate and minimise the time to train a model.

5.2. Tuning process

This section discusses the hyperparameters of each model and the experimental process used to tune them. The hyperparameters of all models selected for comparison are first discussed in Section 5.2.1. Thereafter, the hyperparameters of MTF are discussed in Section 5.2.2.

5.2.1. Hyperparameters of comparative models

Before the evaluation of performance, all models are first tuned for each problem. Tuning also minimises the likelihood that a model underfits or overfits a dataset. The predictive problem of each dataset is uniquely different. Each dataset has a different number of continuous or nominal features and data instances. Therefore, the hyperparameters of each model are individually tuned for each dataset prior to performance evaluation. Table 5.2 lists each model and their applicable hyperparameters.

Table 5.2: The hyperparameters of a base learner for each ensemble model.

Model	Hyperparameters
M5	None
M5 Ensemble	Neighbours Committees
Neural network ensemble	Ensemble size Learning rate Batch size Hidden layer size
Random forest	Ensemble size Minimum number of leaf instances Maximum tree depth Splitting features
Support vector regression ensemble	Ensemble size C Gamma

Models are tuned on each dataset as a whole. The computational cost of tuning a model with continuous hyperparameters and on large datasets with high dimensionality is larger than the training of a model. This is because multiple instances of a model are induced on the training set and compared on the testing set during the tuning process. Although strategies such as Bayesian optimisation are implemented to minimise the time taken to properly tune a model, it remains too computationally demanding to individually tune a model for each of the 30 independent test runs.

Each comparative model is trained and evaluated through a k-fold cross-validation approach per dataset, with a value of five chosen for k. Consequently, each model is induced on 80% of the instances in the dataset and the remaining 20% is reserved for the evaluation of the performance of a given set of hyperparameters through the error metric.

Of the five models, M5 is the only model without any hyperparameters applicable for tuning. This is a restriction of the Cubist package. The size of the M5E ensemble is referred to by Cubist as the number of committees. The number of committees may only range from 1 to 100. Cubist improves the predictive accuracy of M5E through the combination of a nearest-neighbour technique with the output calculation. When Cubist performs a prediction, the predicted values of the nearest instances in the feature spaces are also determined and the average of these predictions is used for the final prediction. The possible values of neighbours are only 1, 3, 5, 7 and 9. Since Cubist limits the range of possible values, the two hyperparameters of M5E can be optimised with grid-search optimisation within the computational limitations.

Grid-search optimisation can ensure that the best possible set of discrete integer hyperparameter values is obtained if every possible combination is evaluated. However, a continuous real hyperparameter search space is too large to cover through grid-search optimisation. The remaining ensemble models also have more hyperparameters compared to M5E. Therefore, the remaining ensemble models are all tuned with Bayesian optimisation.

The next model for discussion is the NNE. A simple feed-forward neural network with a single hidden layer is used as the base learner. The size of the hidden layer depends on the complexity of each problem and is tuned for each dataset separately. Neural networks can be tuned by the addition or removal of single nodes or layers of nodes to increase or decrease their complexity to suit the predictive task at hand. For the activation function, a leaky rectified linear unit is implemented. A leaky ReLU activation function improves the likelihood that the NNE converges with each training sequence. The NNE is trained through stochastic gradient descent (SGD) [2], which requires the specification of a learning rate. Finally, the batch size used for training the NNE also influences the quality of the model. To train the NNE as best as possible, the learning rate, batch size and hidden layer size were tuned using Bayesian optimisation. With each training epoch, the error metric on both the training set and the validation set is calculated. The NNE employs the validation set to detect overfitting during training Once the error metric converges or overfitting is detected, training is halted. The NNE is deemed to overfit once the error on the validation set stops decreasing, whilst the error on the training set continues to decrease.

The base learner hyperparameters of the SVRE are the values for C and gamma, previously discussed in Chapter 3. The two hyperparameters of the base learners SVRE are both tuned using Bayesian optimisation.

SVREs and NNEs both have complex base learners. Consequently, the base learners of SVREs and NNEs exhibit a low bias and high variance error. Therefore, an increase in the ensemble size only increases the predictive accuracy of the ensemble as a whole. However, the increase in accuracy converges beyond a given ensemble size. Any increase in ensemble size beyond this value only provides diminishing returns in increased accuracy. The ensemble size of each comparative model is maximally chosen such that it exceeds the point of convergence, whilst the computational cost of training the ensemble remains viable. The training time of an MTF was used to determine the computational budget for the comparative models.

The final tree-based model used for comparison, i.e. an RF, has four hyperparameters to tune for. The hyperparameters are the ensemble size, the minimum number of leaf instances, the maximum tree depth, and the number of splitting features. All of the aforementioned hyperparameters are also present in MTF and were discussed in Chapter 4. The hyperparameters of RF are all tuned using Bayesian optimisation

5.2.2. Hyperparameters of MTF

The tuning of MTF is done with the intent of minimising the computational cost of the induction algorithm without compromising the predictive accuracy of the ensemble given a set of unseen instances. Due to the genetic algorithm employed in MTF, the computational cost of induction for an MT is far greater than all other base learners of the competing ensemble models. Furthermore, certain genetic operators in GASOPE are randomly applied during induction of MTF as discussed in Chapter 3. Therefore, MTF demands greater repetition in tuning than a greedy induction algorithm to ensure that the outcome is reliable. Table 5.3 lists all applicable hyperparameters and which part of MTF it originates from.

Table 5.3: Hyperparameters from each component in MTF.

Component	Hyperparameter
Adapted GASOPE	Generations
	Crossover rate
	Mutation rate
	Maximum number of terms
	Maximum polynomial order
	Population size
	Generations
MT	Number of splitting features
	Maximum tree depth
	Minimum number of leaf instances
MTF	Ensemble size

MTF has the largest number of hyperparameters of all the evaluated ensemble models. The hyperparameters of MTF originate from three different components of MTF. The tuning of an

MTF also requires a greater number of repetitions, due to the use of a genetic algorithm. Altogether the tuning of MTF is nigh impossible with a gridsearch approach. Bayesian optimisation also falls short of being computationally practical.

Therefore, the tuning of each hyperparameter of an MTF is performed separately through a process of analysing and choosing the most sensible value. This allows the most computationally costly hyperparameters to be tuned individually and eliminates the need for a gridsearch or the Bayesian optimisation tuning approach.

To reduce the complexity of tuning MTF, two hyperparameter values are simply chosen with motivation. Firstly, the minimum number of leaf instances is chosen because it depends on the least-squares algorithm. An explanation follows. The Python package (SciPy) used to implement least-squares approximation in the adapted GASOPE incorporates the Leven-Marquardt (LM) algorithm to solve the nonlinear least-squares problem (refer to Equation (3.7)). The prerequisite for the implementation of the LM algorithm is that the number of training instances is greater than the number of coefficients to solve for. Therefore, the minimum number of leaf instances is chosen as equal to the maximum number of terms in the adapted GASOPE. Minimisation of the value for the number of leaf instances allows for an increased number of splits to be evaluated during training. This is because a split is only evaluated for a node if the node comprises at least twice the number of minimum leaf instances as explained in Section 4.1.3.

Secondly, the number of splitting features is chosen with motivation as follows. MTFs aim to grow decorrelated decision tree structures similar to RFs, be it at a much shallower size. The only difference between the two models is present in the leaf nodes of the base learners. Therefore, the function to calculate the number of splitting features for RF is also implemented per dataset in MTF. A value of $m/3$ is chosen for the number of splitting features for MTF, where m is the number of input features in the dataset.

The hyperparameters of adapted GASOPE were all previously tuned to minimise computational cost without compromising predictive accuracy in Section 3.2. These values are retained with the exception of the maximum polynomial order. Careful consideration should be given in choosing the maximum polynomial order for a given problem to prevent MTF from overfitting or underfitting. Therefore, the remaining hyperparameters that require thorough tuning are the maximum tree depth, maximum polynomial order, and ensemble size. For the maximum tree depth and maximum polynomial order, larger values increase the complexity of a given base learner, which leads to an increased variance error. Although an increased ensemble size helps reduce the variance error, the computational cost increases for each additional base learner that is induced. The same holds true for the maximum tree depth and the maximum polynomial order. Furthermore, an increase in the ensemble size can only decrease the variance error up to a given amount.

For each unique dataset on which an MTF is applied, an analysis of the best-performing values for the three hyperparameters is performed individually. It is possible to tune these three hyperparameters individually by only inducing the required component. The maximum polynomial order applies to the adapted GASOPE, the maximum tree depth to an MT, and the ensemble size to the MTF as a whole. Evaluation of the maximum tree depth is done with both BLMC enabled and disabled in the induction of MTs. The performance metrics used to evaluate a given component of MTF for varying hyperparameter values and datasets are training execution time and MSE as discussed in Section 5.1.3.

For maximum polynomial order, values of one to four are compared. A maximum value of four was chosen for the following reasons. The complexity of higher-order polynomials is unmanageably large beyond four for the fitness function of GASOPE to properly penalise overfitting, as shown by the evaluation of GPMCC on the Abalone dataset (refer to Chapter 3). Furthermore, for the remaining eleven datasets on which Potgieter and Engelbrecht provided a visualisation of the best performing GPMCC model, no evolved polynomial ever exceeded an order of four [23]. Note that a maximum polynomial order of ten was used in the tests. Self-evidently, an increase in the polynomial order also increases the computational costs of fitness evaluations within the adapted GASOPE exponentially. If a maximum polynomial order is too large the computational cost of the adapted GASOPE quickly becomes infeasible. A lower maximum polynomial order also allows for GASOPE to converge faster because the genetic algorithm has a smaller number of possible terms to evaluate.

For the maximum tree depth, values up to ten are evaluated. The total number of nodes in a binary tree structure can maximally be equal to $2^d - 1$, where d is the maximum tree depth. Therefore, similar to the maximum polynomial order, computational costs grow exponentially with an increase in maximum tree depth. The induction algorithm of MTF evolves a polynomial via GASOPE for each additional leaf node present in an MT. At a depth of ten, a total of $2^{10} - 1 = 1023$ instances of the adapted GASOPE are induced if an MT is fully grown. A value greater than 10 is computationally infeasible given that multiple MTs are induced to form an ensemble.

For each given hyperparameter and dataset, the corresponding component of MTF is induced on a training set and the MSE is calculated on a generalisation set for 30 independent runs. The absolute computational time taken to execute the induction algorithm for each hyperparameter value is also recorded for each independent run. The training set and generalisation set are randomly split in the ratio 80:20. In the case of tuning the ensemble size, a validation set is included for ensemble pruning and the training set, validation set, and test set are randomly split in the ratio 80:10:10.

5.3. Comparison process

The series of experiments to evaluate both the predictive and computational performance of MTF are discussed in this section. Section 5.3.1 first discusses the process used to evaluate the performance of MTF against both singular model trees and the ensemble methods selected for comparison. Thereafter, Section 5.3.2 discusses the statistical tests that were applied to the experimental results.

5.3.1. Performance evaluation

After the tuning process of MTF and each comparing ensemble model was completed, the performance of MTF was evaluated against singular model trees and the selected ensemble models. It was first necessary to confirm the advantage ensemble methods offered model trees. This was done by the induction of singular MT and M5 models for comparison against MTF. Thereafter, the ensemble models selected for comparison were included in the performance evaluation.

For both the performance comparison of MT against singular model trees and ensemble models, each dataset was shuffled randomly and divided into three sets, the training set, validation set, and test set in an 80:10:10 ratio. Models were induced on the training set. The validation set provided unseen data during induction. MTF utilised the validation set for ensemble pruning, whilst NNE utilised the validation set for the detection of overfitting. Finally, the training set was used to evaluate the generalisation capability of each model on unseen data. This was repeated for a total of 30 independent runs per dataset. The performance metrics discussed in Section 5.1.3 were compared over the 30 independent runs.

5.3.2. Statistical Tests

Once error metrics were calculated, statistical tests were performed to determine which differences in performance were considered significant. The null-hypothesis was that all models were equal in their performance based on the results for a given dataset. The Friedman test was used to test the null-hypothesis. To perform the Friedman test, each independent test run out of the 30 was ranked according to the RMSE scores. A significance level of 5% was used. If the null-hypothesis was rejected, pairwise tests were performed to determine which models produced results that are statically dissimilar. The post-hoc test was the Bonferroni-Dunn test.

5.4. Conclusion

This chapter discussed the series of experiments that were used to tune and evaluate the performance of all compared models. Performance refers to both the predictive accuracy of a model given unseen data and the computational costs of the induction of a model on a set of instances. For diverse evaluation, datasets with a number of instances ranging from 167 to $45,730$ and a number of features ranging from 4 to 18 were selected.

The models used to compare the performance of an MTF were selected to evaluate a number of questions, listed in Section 5.1.2. It was necessary to compare an MTF to a singular model tree to show that the ensemble method discussed in the previous chapter did indeed improve the accuracy compared to a single model tree. Other decision tree ensemble methods were included in the selection to compare the effectiveness of decorrelation in ensemble methods to reduce a high variance error. State-of-the-art ensemble models were also chosen to compare the performance of MTF on data with nonlinear relationships between the input and output features.

It was important for all models to be finely tuned for each problem at hand to ensure a fair comparison was made. Once each model was tuned, the performance of the models would be compared and statistical tests performed to determine whether any difference in performance could be considered significant.

Chapter 6

Experimental results

This chapter discusses the experimental results for the tuning and performance of MTF against the four ensemble models selected for comparison on ten benchmarking datasets. The results provide insight into the potential and flexibility of MTFs to produce good accuracy on various different problems. Section 6.1 presents and discusses the experimental results of the tuning process of MTF and lists the final hyperparameter values. Section 6.2 firstly motivates the use of ensemble methods to reduce variance error based on the results of MTF, MT and M5. Thereafter, the comparison results of all ensemble models are discussed. This section is then concluded with the outcomes of the statistical tests used to determine the significance of the findings. Finally, Section 6.3 concludes this chapter with a discussion on the performance of MTF.

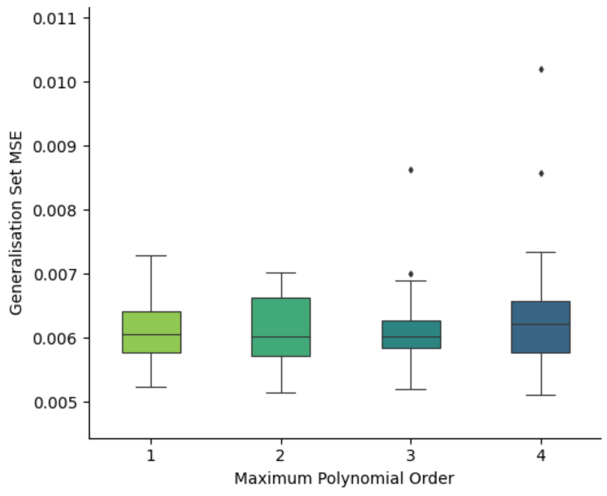
6.1. Tuning results of MTF

This section presents and discusses the experimental results of the tuning process of MTF. The ideal hyperparameter values minimise the computational cost of inducing an MTF without compromising the predictive accuracy of the MTF. Section 6.1.1 presents and discusses the maximum polynomial order. Next, Section 6.1.2 presents and discusses the maximum model tree depth. Thereafter, Section 6.1.3 presents and discusses the final hyperparameter, i.e. ensemble size. This section is concluded with a discussion of the chosen hyperparameter values in Section 6.1.4.

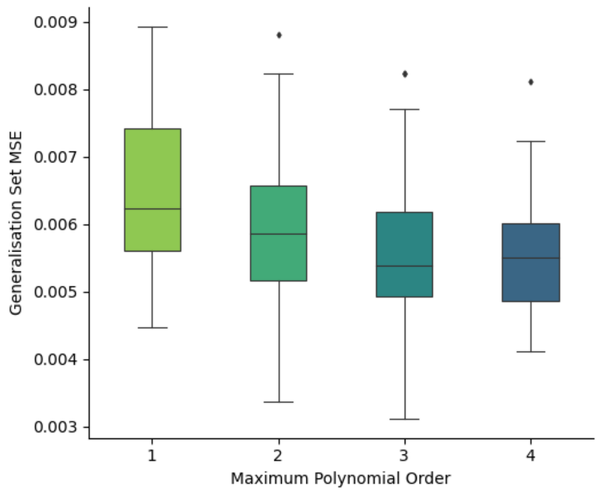
6.1.1. Maximum polynomial order

Figure 6.1 visualises the generalisation MSE scored by the adapted GASOPE for each dataset for a varying maximum polynomial order. For the majority of datasets, a maximum polynomial order value of two resulted in the lowest generalisation MSE. Only the Machine CPU dataset favoured a maximum polynomial order of one. This showed that the datasets chosen for the evaluation of MTFs were all fairly nonlinear except for the Machine CPU dataset. The minority of datasets was highly nonlinear because only three of the ten datasets favoured a maximum polynomial order of three. Appendix A lists the generalisation MSE scores visualised in Figure 6.1 in full.

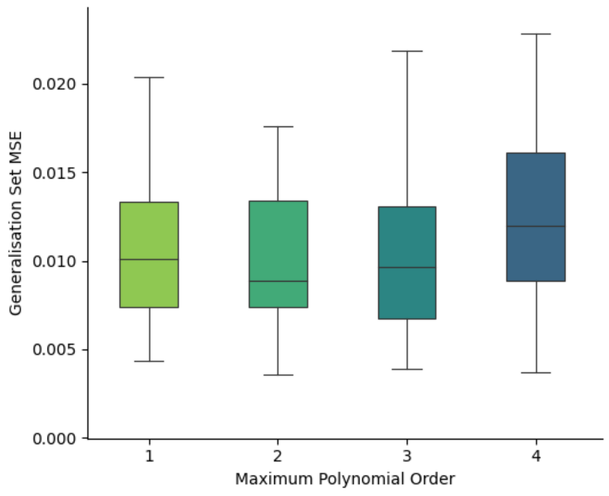
It was worth noting that the use of a maximum polynomial order higher than the favourable value was not fatal to the performance of the algorithm, because the fitness function of GASOPE penalised higher-order functions that did not improve the error metric when compared to a lower-order function. For the cases where there was little discrepancy between the generalisation MSE for maximum polynomial orders of one and two, two was chosen as the preferred value. Additionally, a value of two was chosen because the BLMC step in MTFs replaced any higher-order output functions evolved through GASOPE that performed poorly on the given set of data instances.



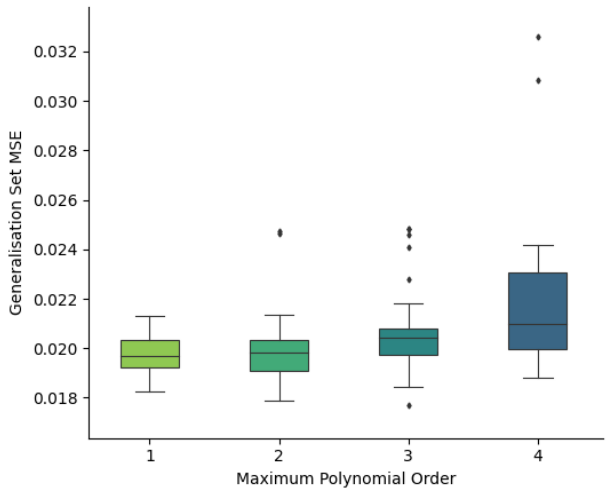
(a) Abalone



(b) Auto-MPG

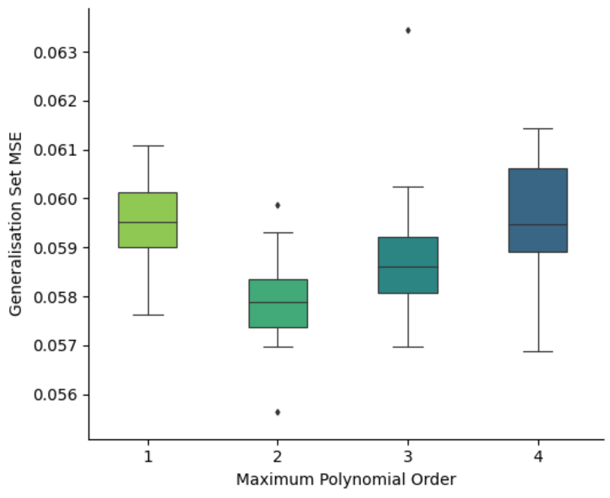


(c) Boston housing

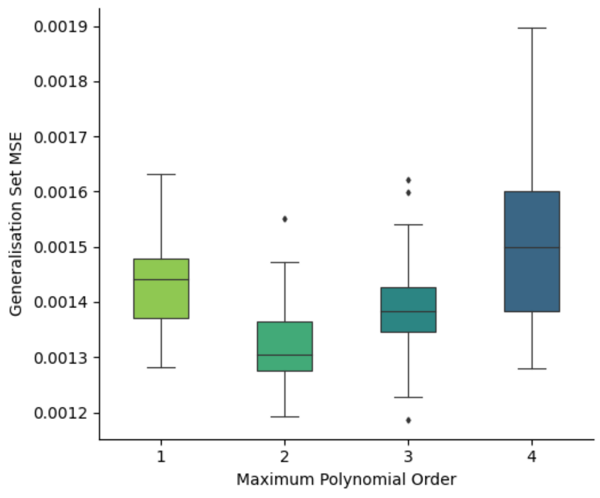


(d) California housing

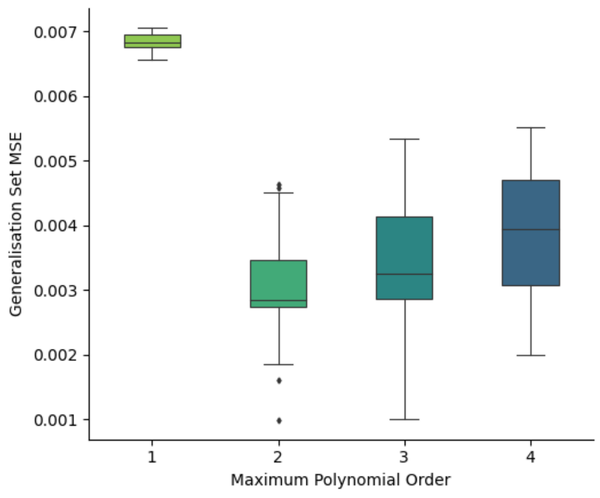
Figure 6.1: Generalisation MSE of the adapted GASOPE for varying degrees of maximum polynomial order on the benchmarking datasets.



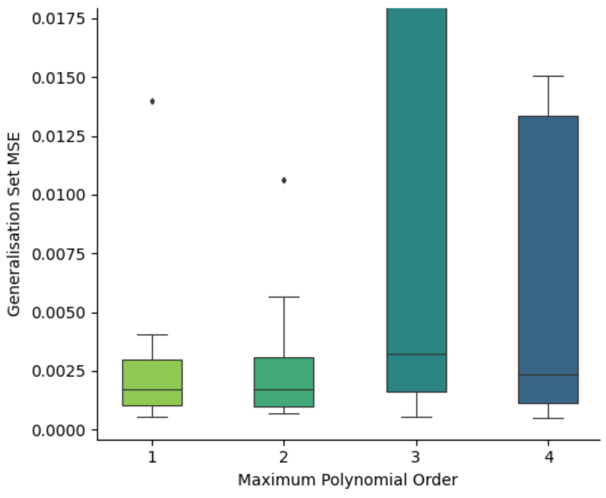
(e) CASP



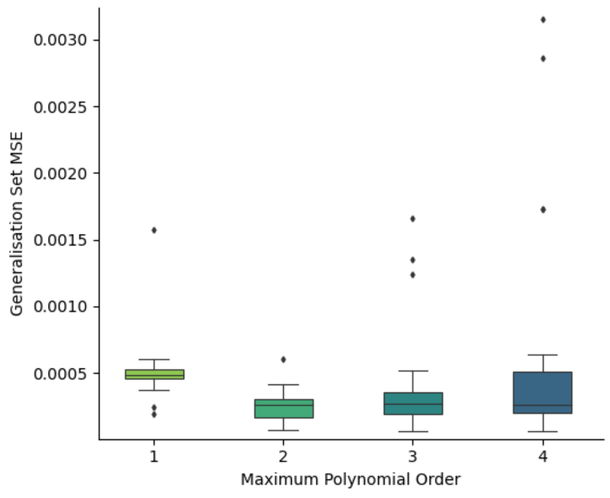
(f) Elevators



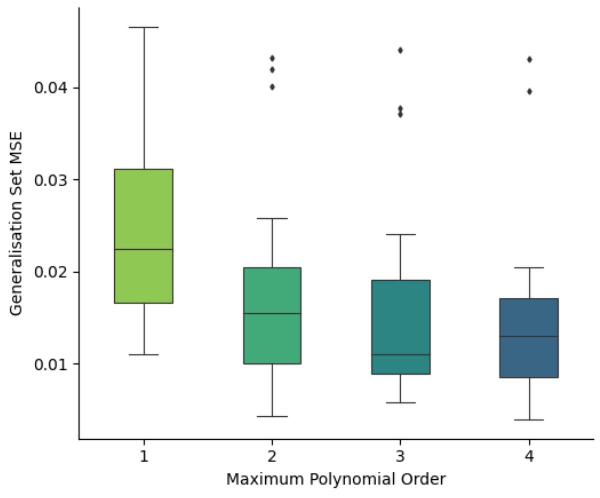
(g) Friedman artificial domain



(h) Machine CPU



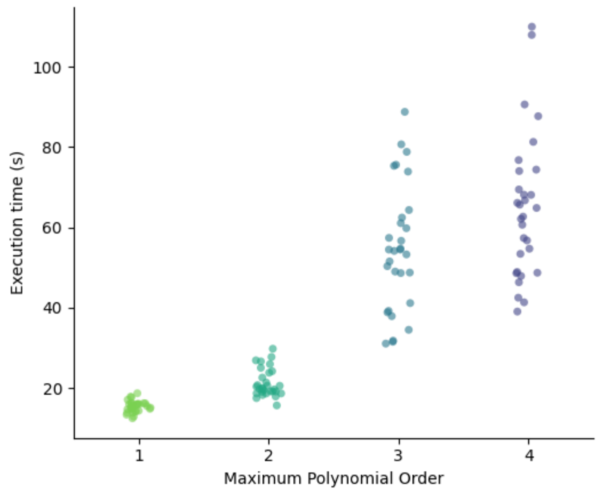
(i) MV artificial domain



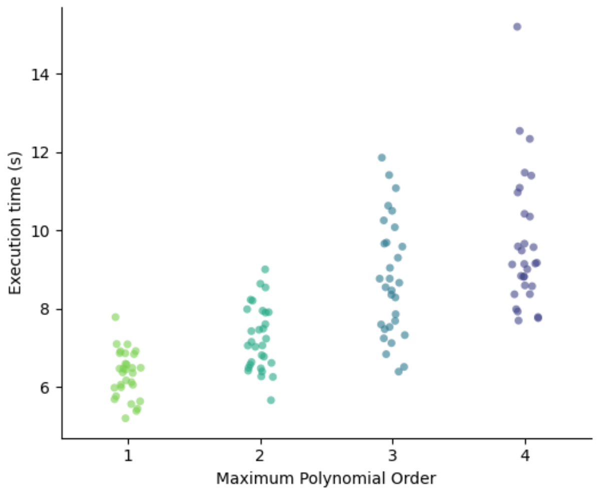
(j) Servo

Figure 6.1: (continued)

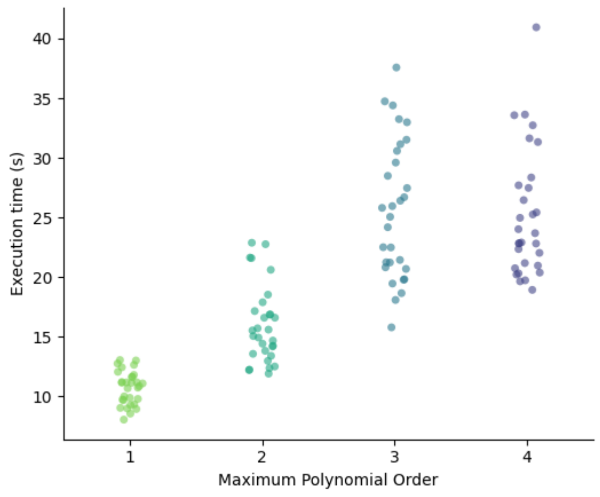
The execution time to evolve a polynomial using the adapted GASOPE for each maximum polynomial order and each dataset is shown in Figure 6.2. For large datasets, a maximum polynomial order greater than three significantly increased the time taken to evolve a polynomial function. This reiterates the need to minimise the polynomial order without compromising predictive accuracy and substantiates the choice of a maximum polynomial order of two for large datasets. For small datasets, i.e. sets comprising less than 1000 instances, the computational cost was not a limiting factor because the absolute execution time of the MTF induction algorithm was up to two orders of magnitude lower than for large datasets.



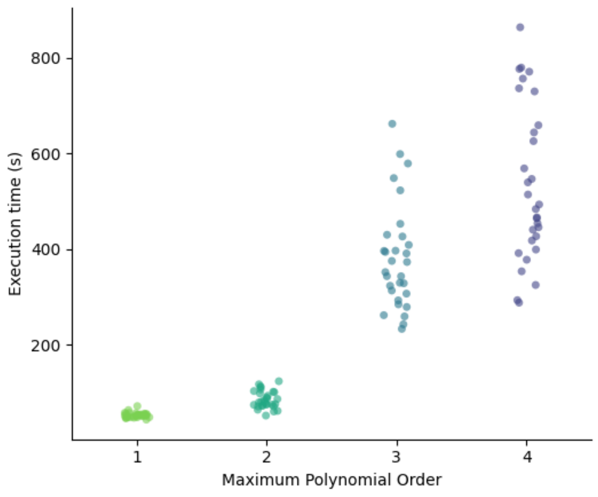
(a) Abalone



(b) Auto-MPG

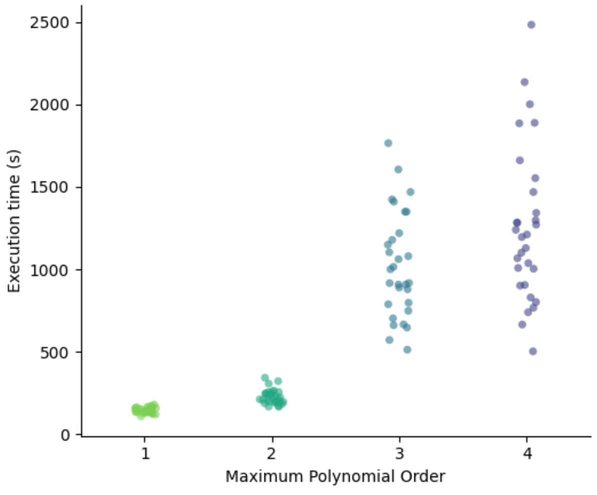


(c) Boston housing

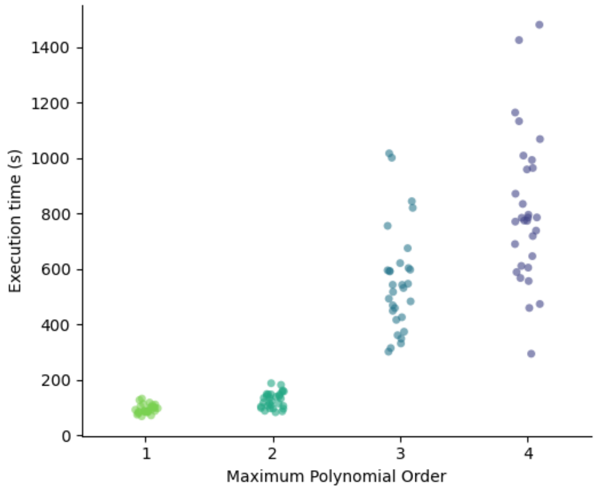


(d) California housing

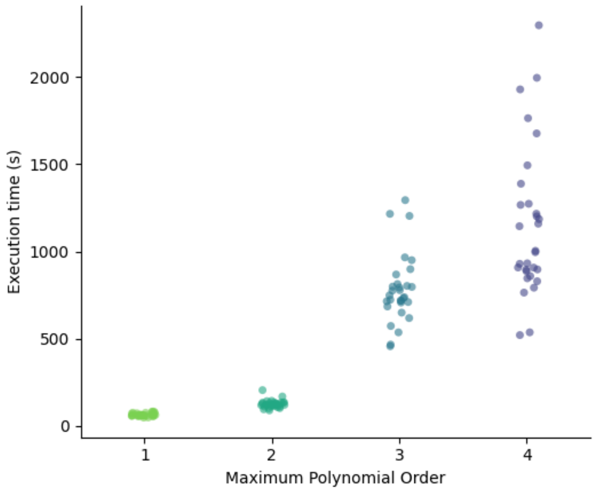
Figure 6.2: The training time of the adatped GASOPE for varying degrees of maximum polynomial order on the benchmarking datasets.



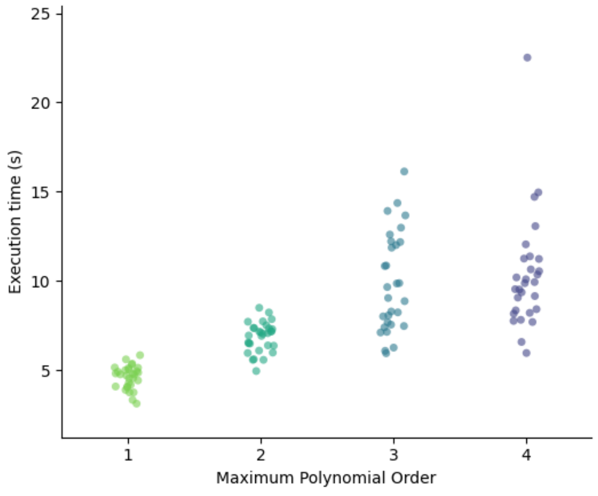
(e) CASP



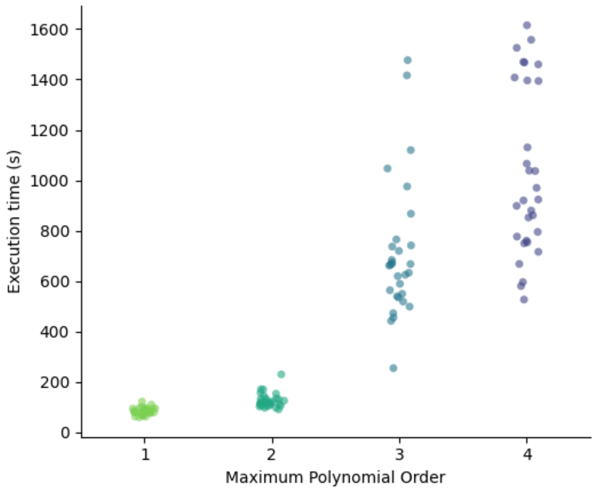
(f) Elevators



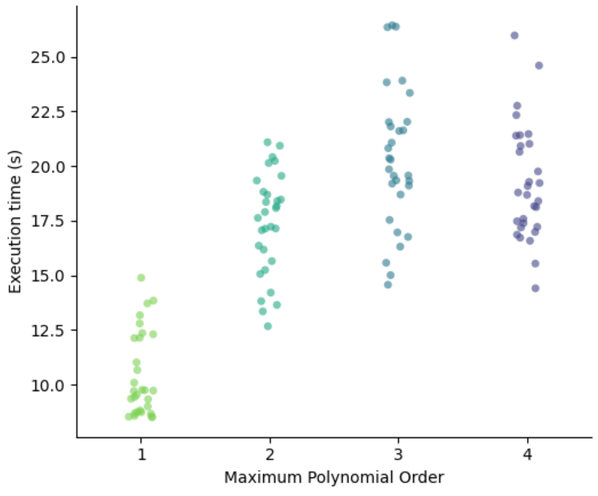
(g) Friedman artificial domain



(h) Machine CPU



(i) MV artificial domain



(j) Servo

Figure 6.2: (continued)

The maximum polynomial order served as an estimation of the linearity of the relationships in each dataset. Four adapted GASOPE models with differing maximum polynomial orders on the entire training set were induced and compared on the generalisation set. The linearity of the relationships between the features in the dataset was represented through the maximum polynomial order of the most accurate model. As the polynomial order was increased from one to four, the polynomial functions first underfit, then adequately fit, and then overfit a given dataset.

For each dataset, the maximum polynomial order that matched the linearity of the relationships between the input and output features was the value that produces the lowest generalisation MSE. The polynomial order value of the best-performing adapted GASOPE instance was also chosen as the maximum polynomial order for MTF to match the linearity of a given dataset.

The linearity characteristic for each dataset was derived from the results of tuning the maximum polynomial order of the adapted GASOPE and listed below in Table 6.1.

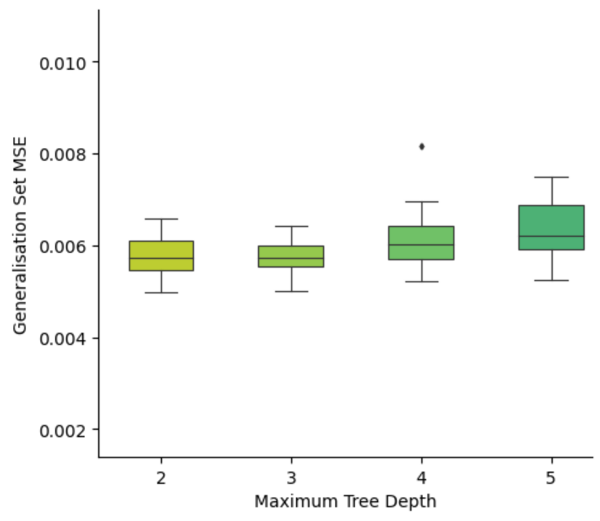
Table 6.1: Linearity estimation of datasets used for the comparison of models.

Dataset	Number of features (N = Nominal, C = Continuous)	Number of instances	Estimated linearity through GASOPE
Abalone	1N 7C	4,177	linear to quadratic
Auto-MPG	3N 4C	398	quadratic to cubic
Boston housing	0N 13C	506	quadratic to cubic
California housing	0N 8C	20,460	linear to quadratic
CASP	0N 9C	45,730	purely quadratic
Elevators	0N 18C	16,559	purely quadratic
Friedman artificial domain	0N 10C	40,768	quadratic to cubic
Machine CPU	0N 6C	209	purely linear
MV artificial domain	3N 7C	40,768	purely quadratic
Servo	4N 0C	167	purely cubic

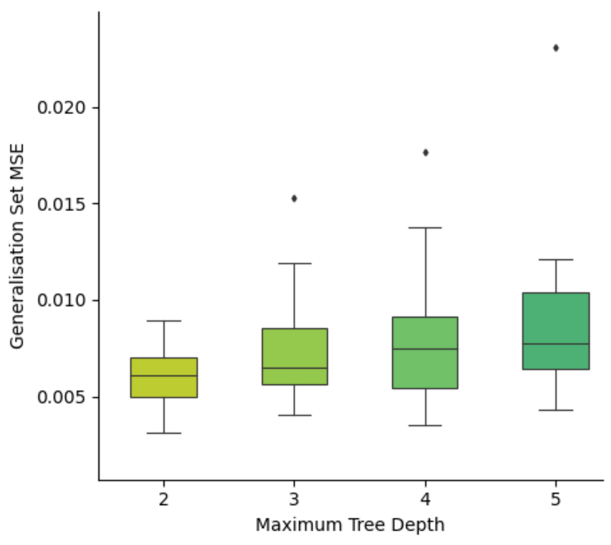
6.1.2. Maximum model tree depth

The next hyperparameter that was evaluated, was the maximum tree depth of the MT base learners that collectively form an MTF. Recall that the larger a decision tree was grown, the more likely it was to overfit the training set. In conjunction, the deeper a decision tree was grown, the higher the number of leaf nodes was. It was therefore beneficial for the MTF induction algorithm to cap the depth of each MT at a value tuned for computational cost and to avoid overfitting.

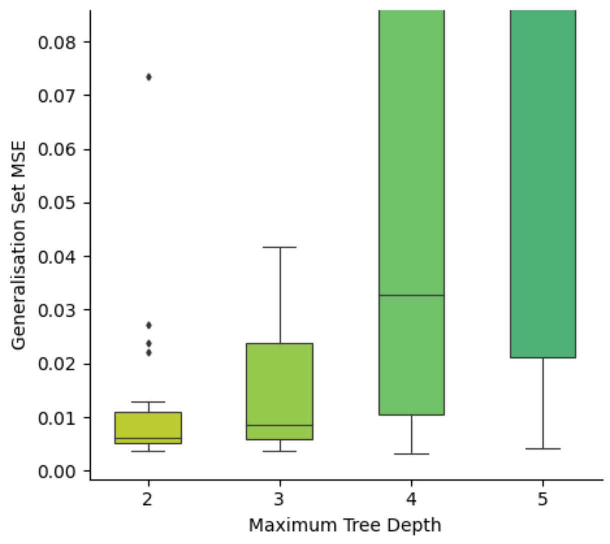
The MSE on the generalisation set is shown for each dataset in Figure 6.3. Figure 6.4 shows how the execution time of training an MT increases as the maximum tree depth is increased. The maximum tree depth of an MT is shown to have a large influence on both the total computational cost and collective variance error of MTF.



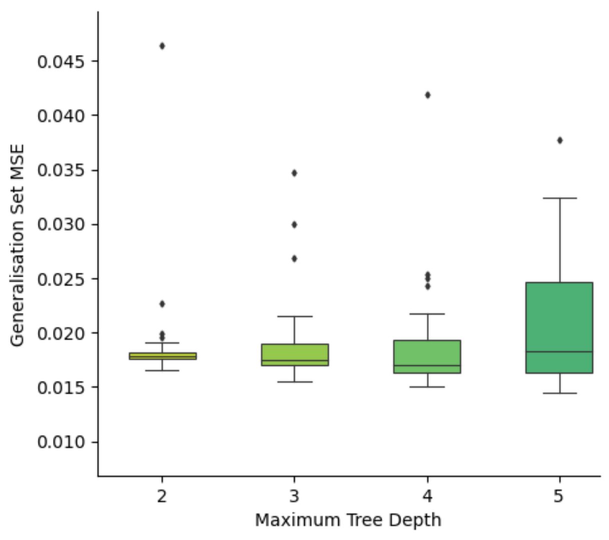
(a) Abalone



(b) Auto-MPG

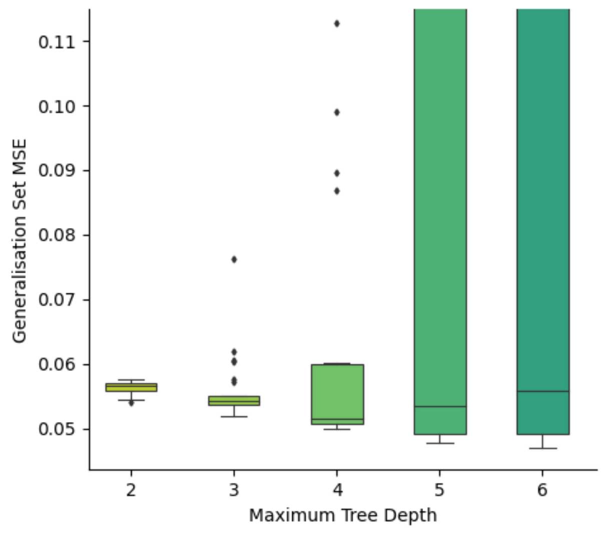


(c) Boston housing

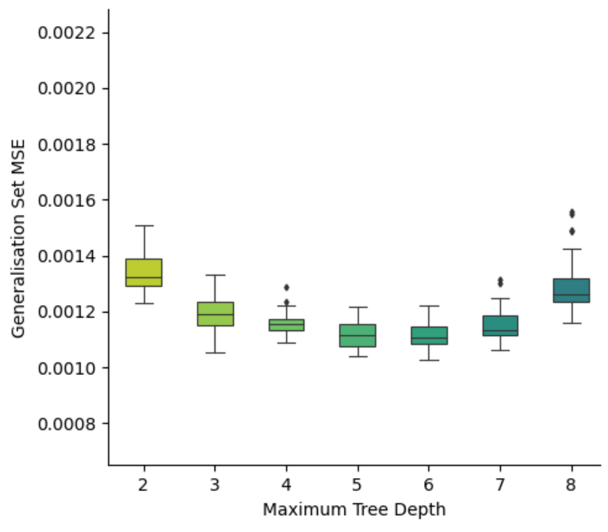


(d) California housing

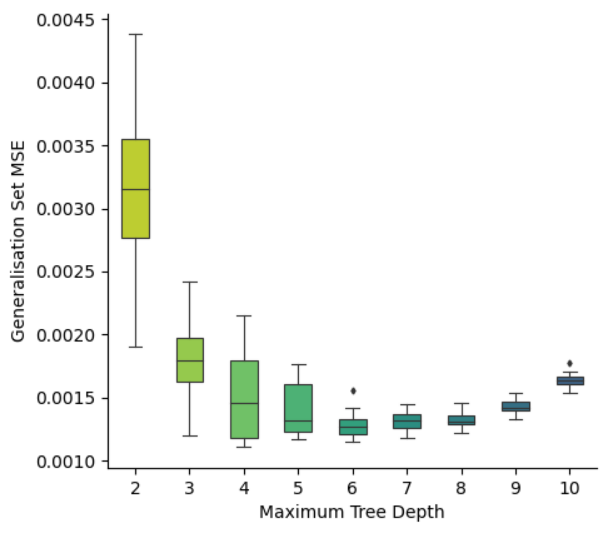
Figure 6.3: Generalisation MSE of MT for varying degrees of maximum tree depth on the benchmarking datasets.



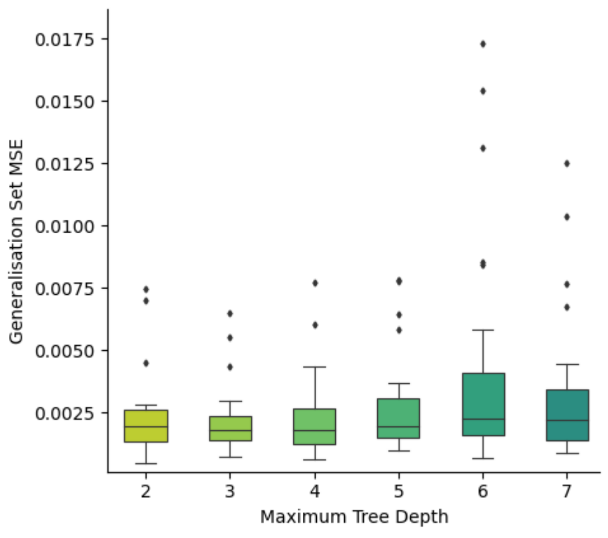
(e) CASP



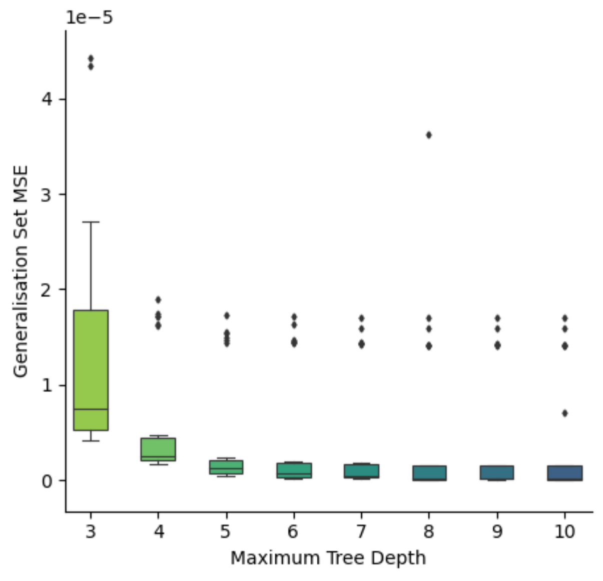
(f) Elevators



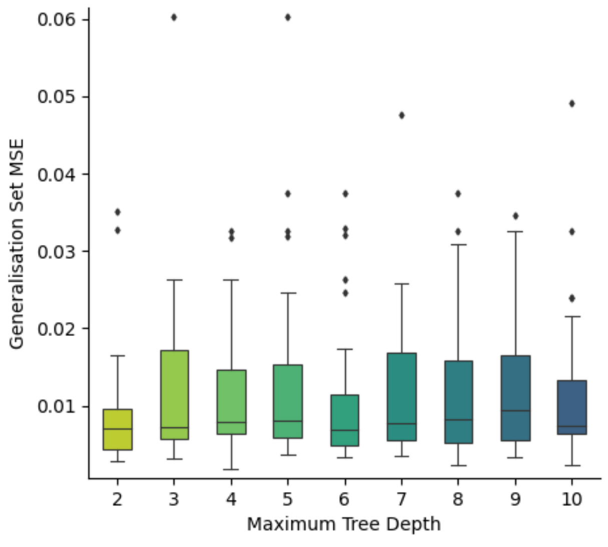
(g) Friedman artificial domain



(h) Machine CPU



(i) MV artificial domain

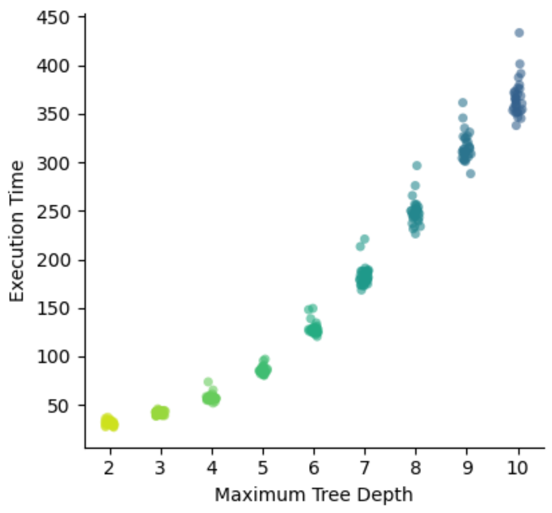


(j) Servo

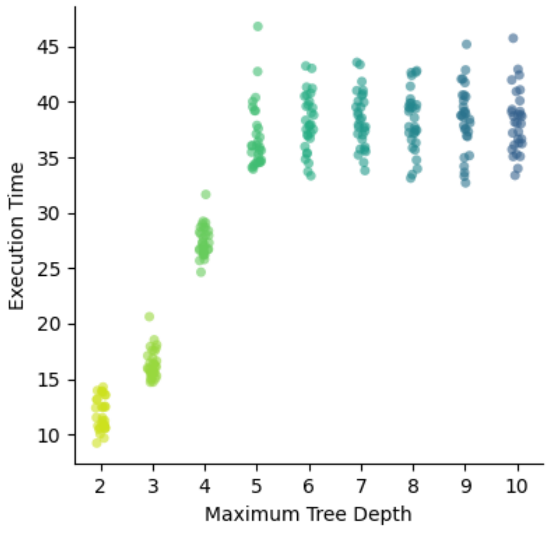
Figure 6.3: (continued)

An unintended consequence of the limitation of the maximum tree depth was a reduction in the variance error of MTF. In the case of traditional regression trees, a too-strict limitation on the depth of the tree resulted in an increased bias error. However, with MTF this increase was minimal because the complex composition of higher-order polynomial output functions ensured a low bias with high variance error was retained in MTF.

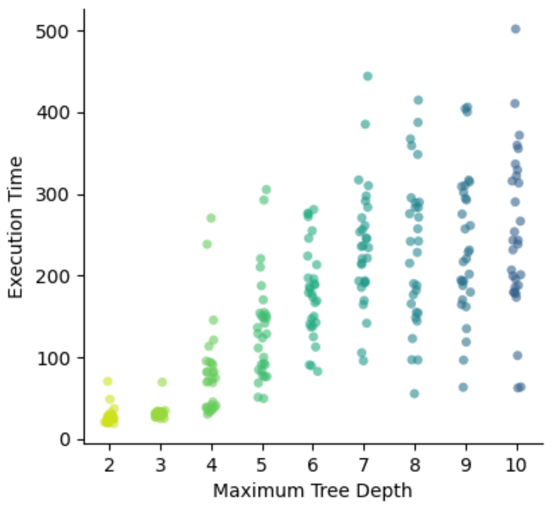
The maximum tree depth was not tuned only according to the best-performing generalisation MSE. An MT with a large generalisation MSE due to a high variance error was reduced through bagging and it was possible to outperform an ensemble with a smaller generalisation MSE per base learner. However, the bagging of base learners that produced unstable MSEs was not guaranteed to reduce the collective variance error effectively. For some datasets, such as Boston housing and CASP, the generalisation MSE became unstable at greater depths, i.e. it fluctuated more. Therefore, shallow trees were preferred for datasets where MTs produced unstable generalisation MSE scores at greater depths.



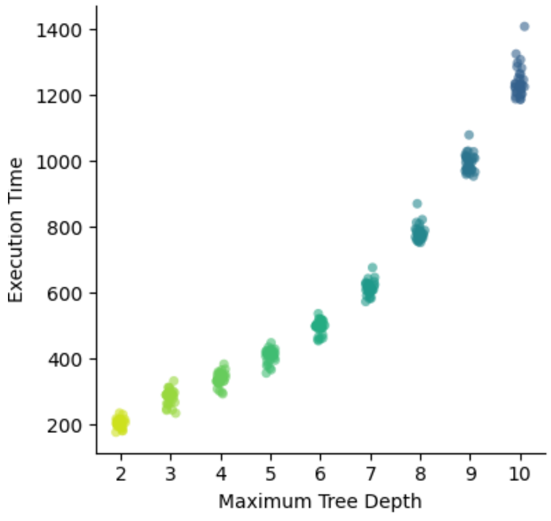
(a) Abalone



(b) Auto-MPG

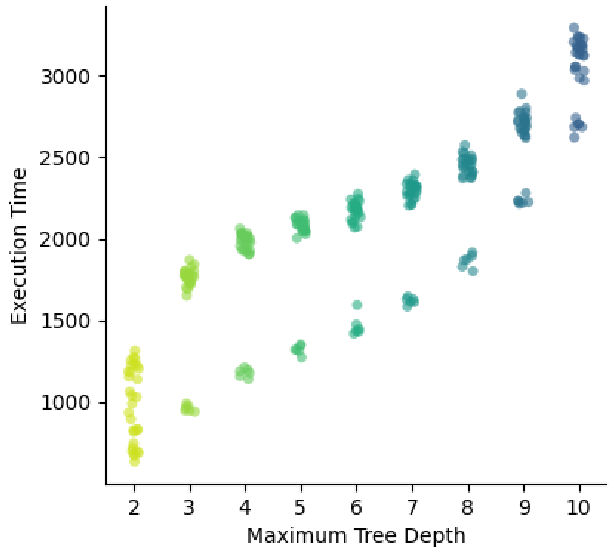


(c) Boston housing

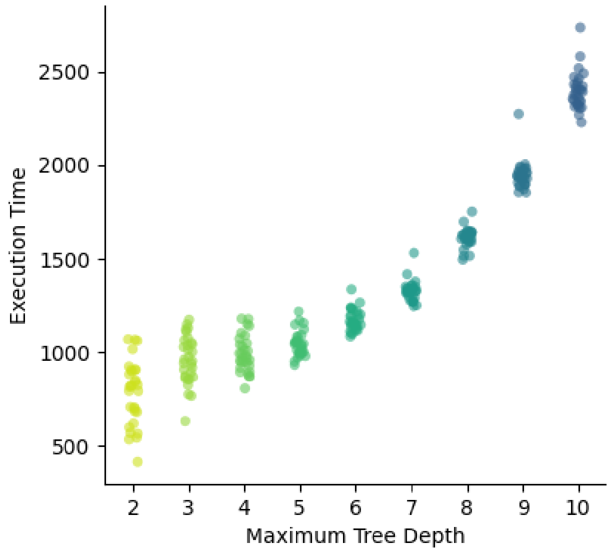


(d) California housing

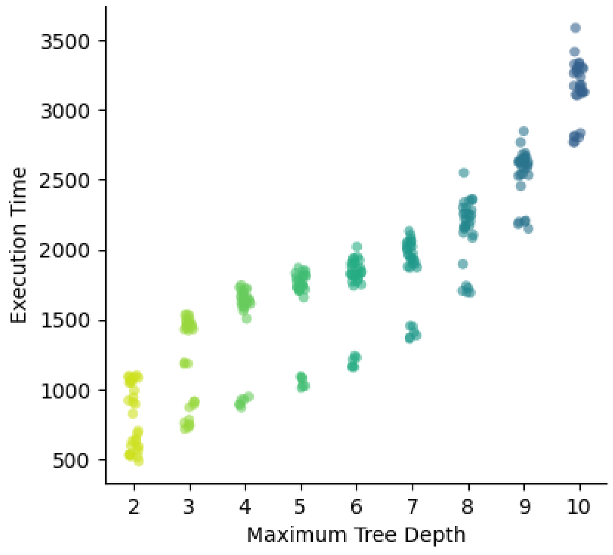
Figure 6.4: The computational cost of MTs for varying degrees of tree depth on the benchmarking datasets.



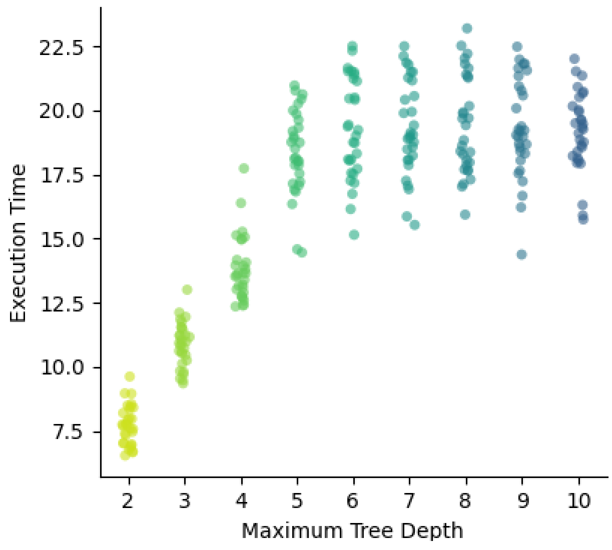
(e) CASP



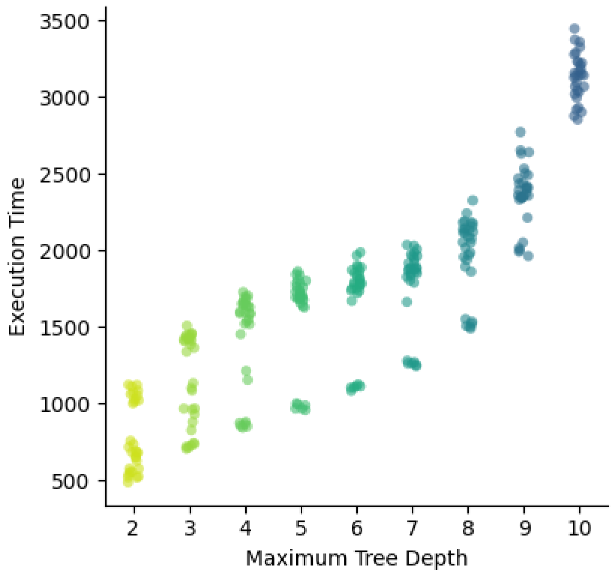
(f) Elevators



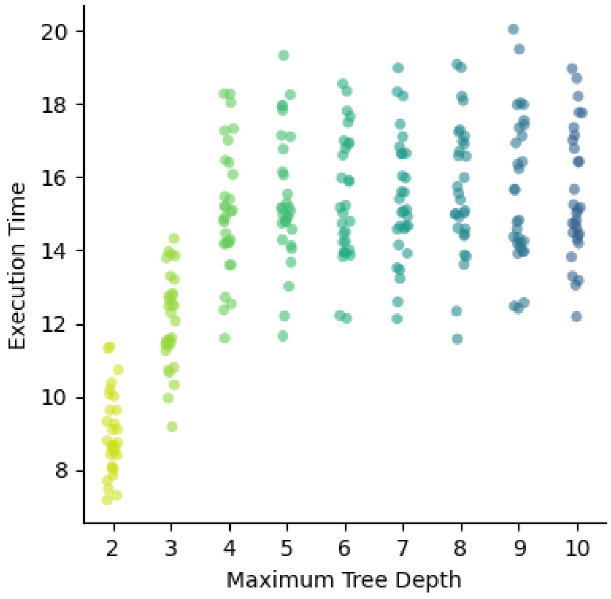
(g) Friedman artificial domain



(h) Machine CPU



(i) MV artificial domain



(j) Servo

Figure 6.4: (continued)

The value at which the best-performing MSE on the generalisation set was produced without large fluctuations was selected as the optimal maximum tree depth. If multiple maximum tree depths produced a favourable generalisation MSE, the larger-valued maximum tree depth was preferred up until the execution time became too high. A deeper tree was likely to produce a larger variance error, but the ensemble as a whole combated this through bagging and decorrelation of the base learners. For the CASP, Elevators, MV artificial domain and Friedman artificial domain datasets, the computational costs started to grow impractically high for each additional tree depth value past six.

For illustrative purposes, the severe outlier results were omitted from Figure 6.3. Generalisation MSE scores with different orders of magnitude were obtained over the varying maximum tree depths. Tables B.1 through B.10 in Appendix B shows the complete results of generalisation MSE scores and variation with both BLMC enabled and disabled. As expected, BLMC always improved the predictive accuracy of the generalisation set at optimal tree depths. With BLMC enabled, the generalisation MSE and the variation of the generalisation MSE over the 30 independent runs were reduced.

For smaller datasets, the execution time was shown to plateau after a given maximum tree depth. The minimum number of leaf instances was the stopping criterion which prohibited the remainder of the dataset from being split further during the induction process. For example, in Figure 6.4b beyond a maximum tree depth of six, the execution time no longer increased. When an MT was induced on the Auto-MPG dataset, nodes at a depth of six could no longer comprise enough instances, prohibiting the node from being split further by the induction algorithm. For larger datasets, such as Abalone and California Housing, the execution time grew exponentially, confirming the need to keep the tree depth and consequently computational cost at a minimum. Larger datasets had a greater number of instances associated with the root node and demanded more splits before all possible paths in the decision tree led to a leaf node that met the stopping criteria.

6.1.3. Ensemble size

The test set MSE scored by MTF on the test set for each consecutively added model tree in the ensemble is shown in Figures 6.5 through 6.14. The results are tabulated in Appendix C.

A high bias error was observed in the test set MSE for a minority of the 30 runs, shown through the convergence of outlier values. The offset was clearly observed in Figures 6.7 and 6.10. The offset was due to one or two model trees in the ensemble performing exceptionally poor and biasing the average prediction enough to compromise the ensemble as a whole. As the ensemble size was increased, the uniform weight of each MT on the predicted value of the ensemble decreased. However, with each additionally induced model tree, there was the risk of adding another poorly performing tree. Furthermore, the computational cost associated with growing an ensemble large enough to sufficiently reduce the weight of a base learner was problematically high for MTF. The ensemble pruning and trimmed mean steps allowed for the selection of a small ensemble size without poor-performing MTs biasing the prediction of MTF.

The chosen ensemble size was the point at which the test set MSE started to converge.

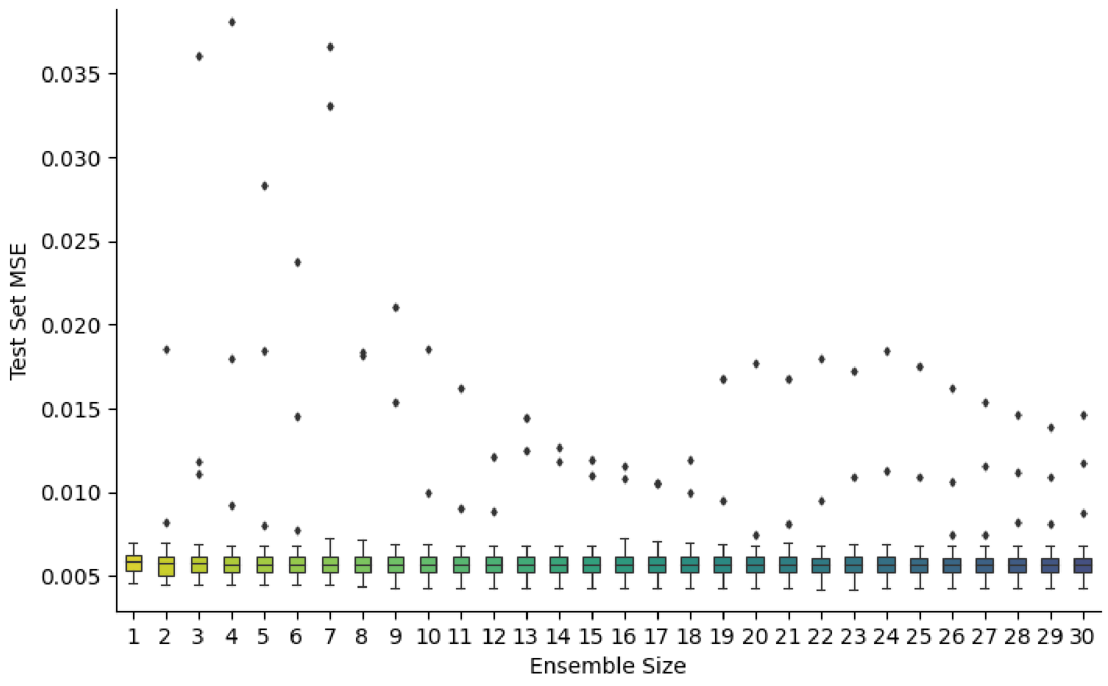


Figure 6.5: Abalone: Test set MSE vs ensemble size of MTF.

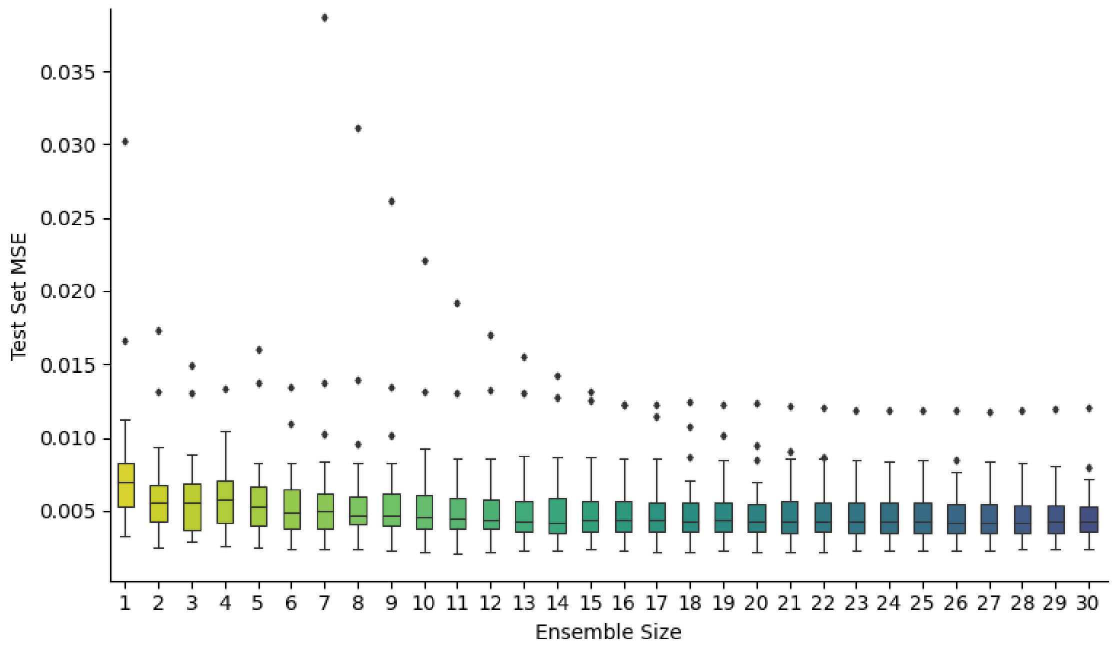


Figure 6.6: Auto-MPG: Test set MSE vs ensemble size of MTF.

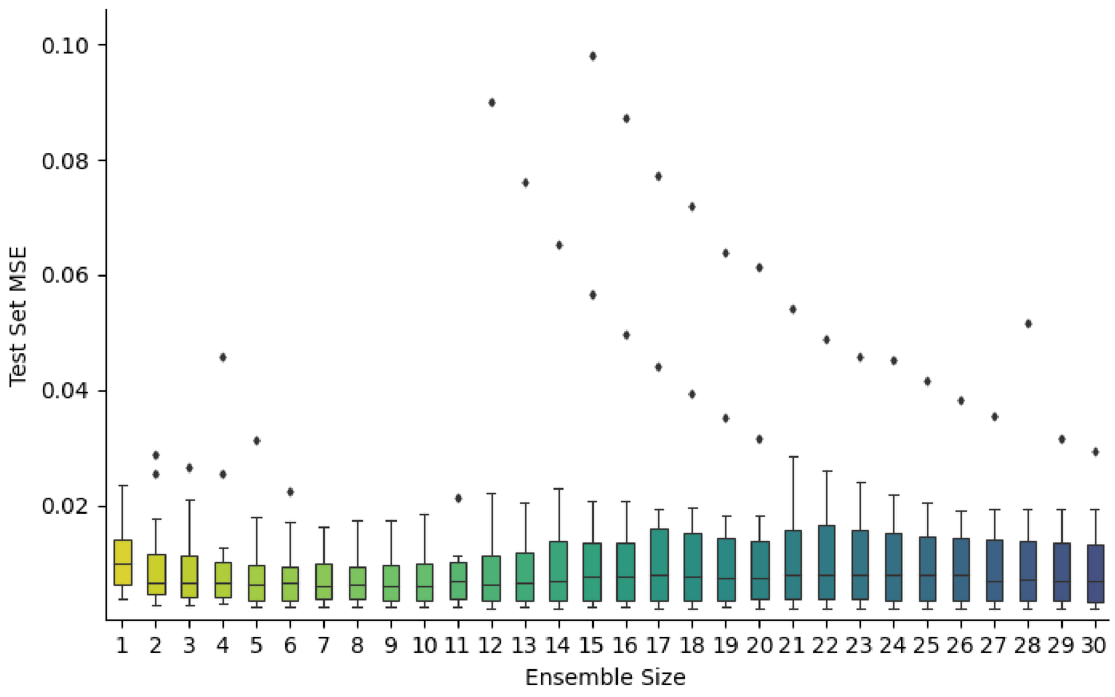


Figure 6.7: Boston Housing: Test set MSE vs ensemble size of MTF.

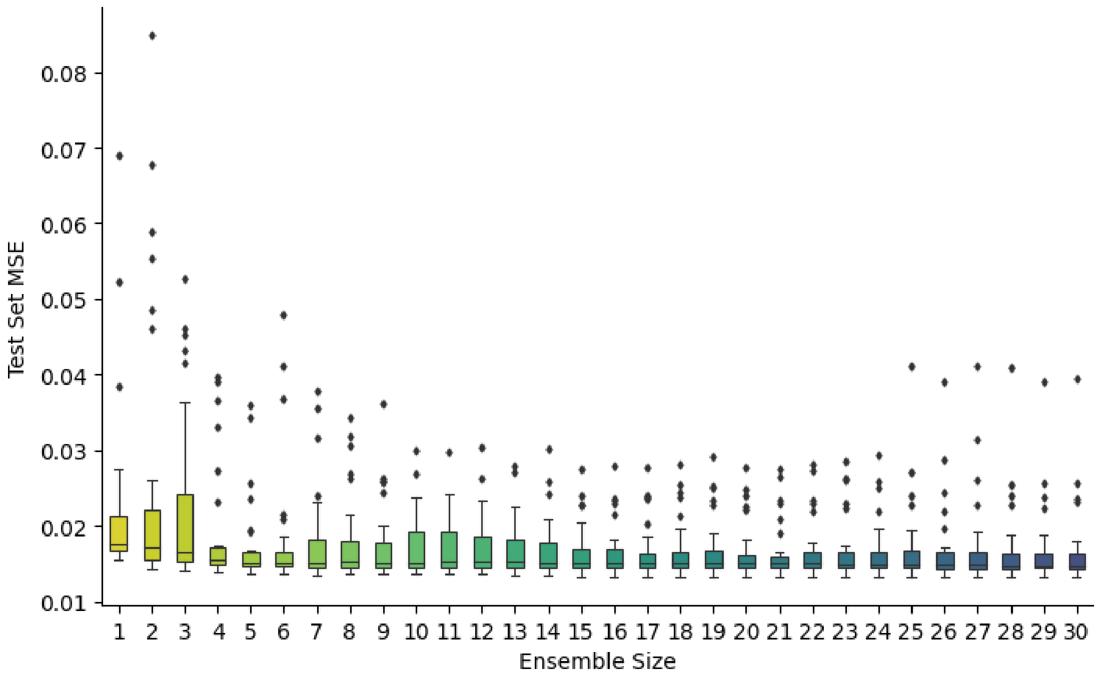


Figure 6.8: California Housing: Test set MSE vs ensemble size of MTF.

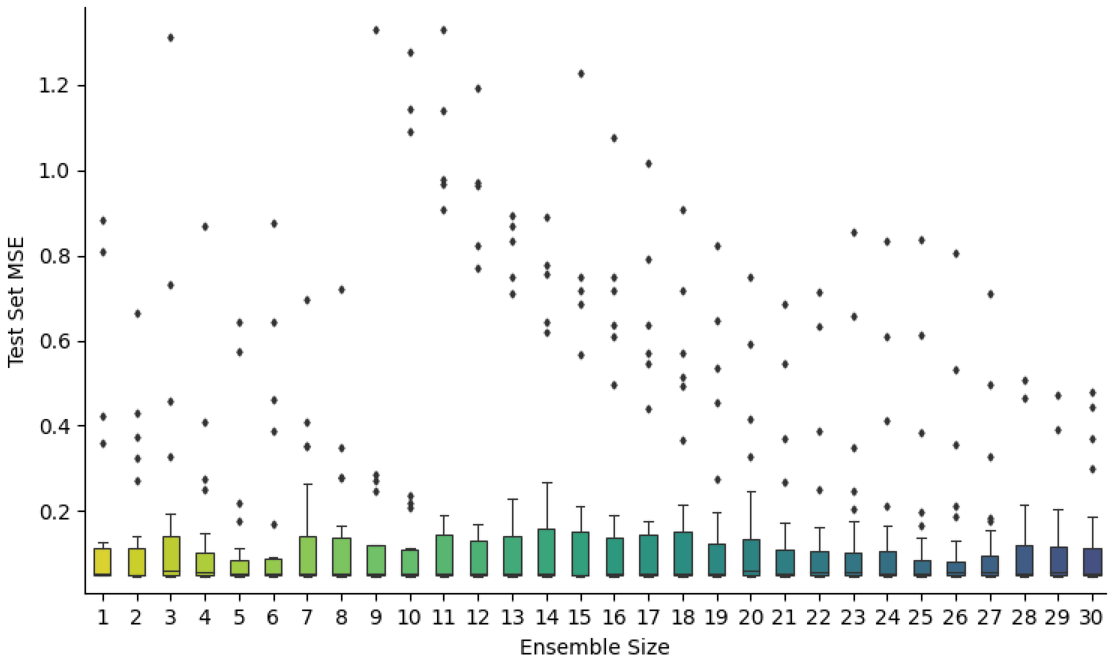


Figure 6.9: CASP: Test set MSE vs ensemble size of MTF.

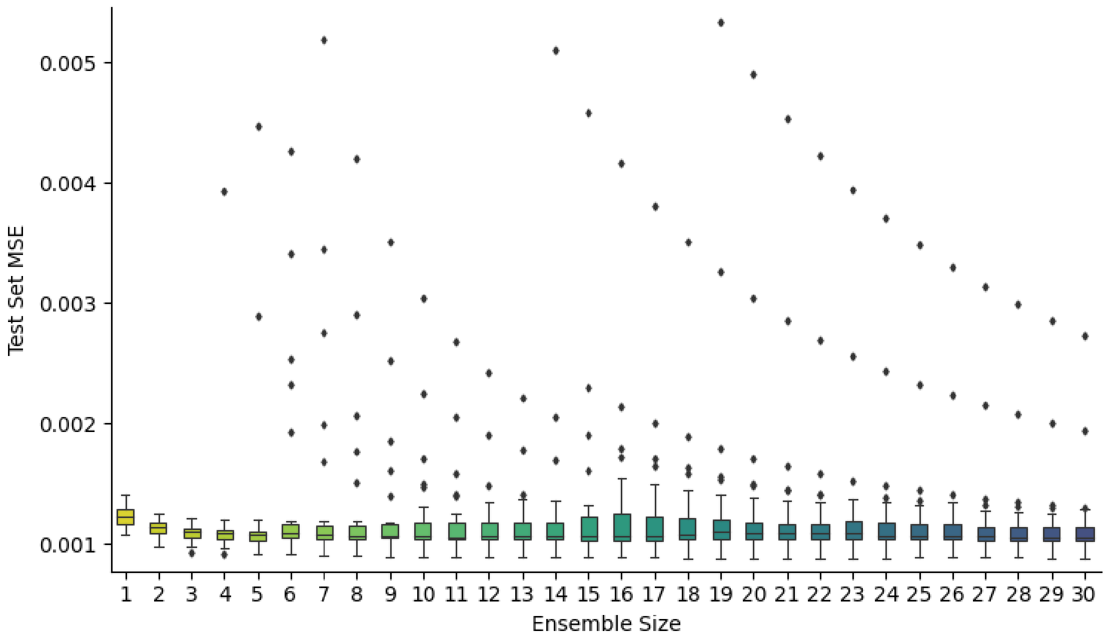


Figure 6.10: Elevators: Test set MSE vs ensemble size of MTF.

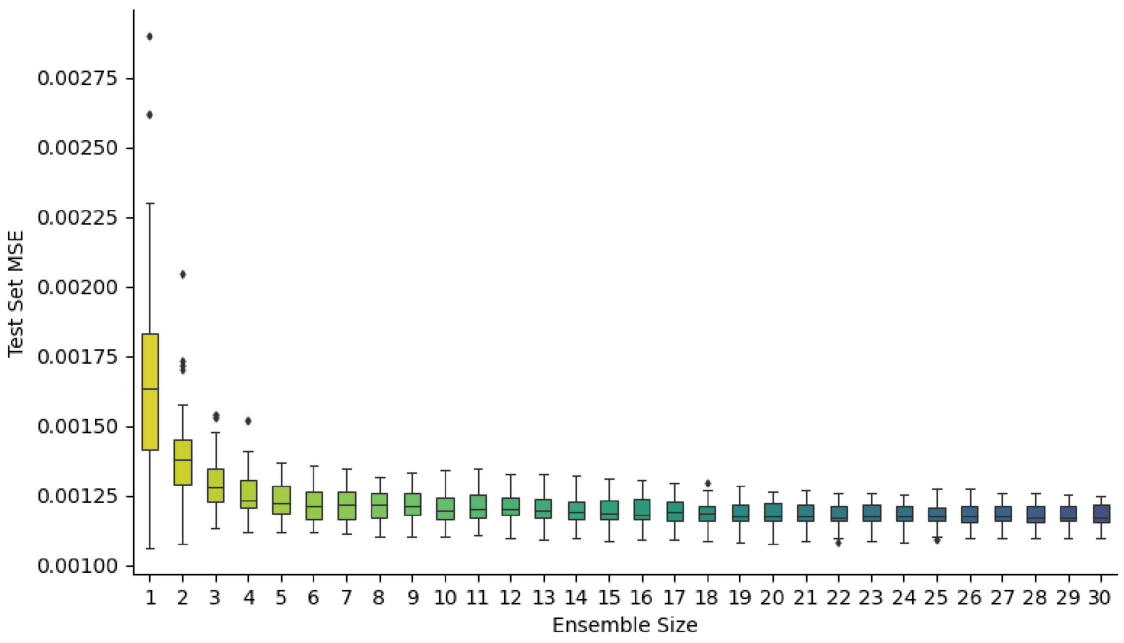


Figure 6.11: Friedman Artificial Domain: Test set MSE vs ensemble size of MTF.

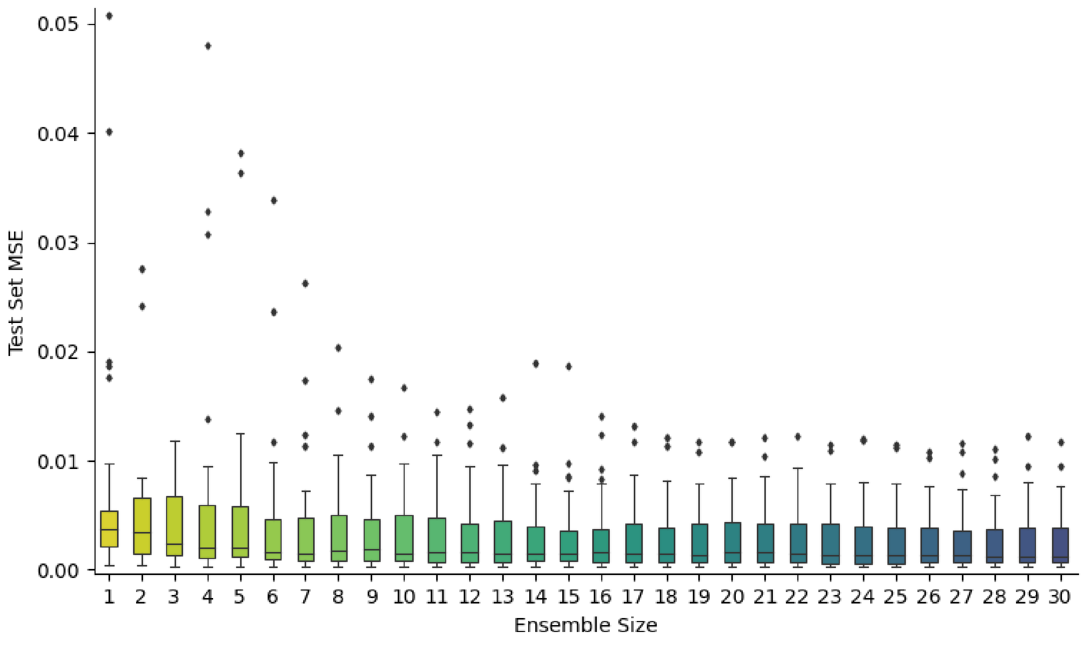


Figure 6.12: Machine CPU: Test set MSE vs ensemble size of MTF.

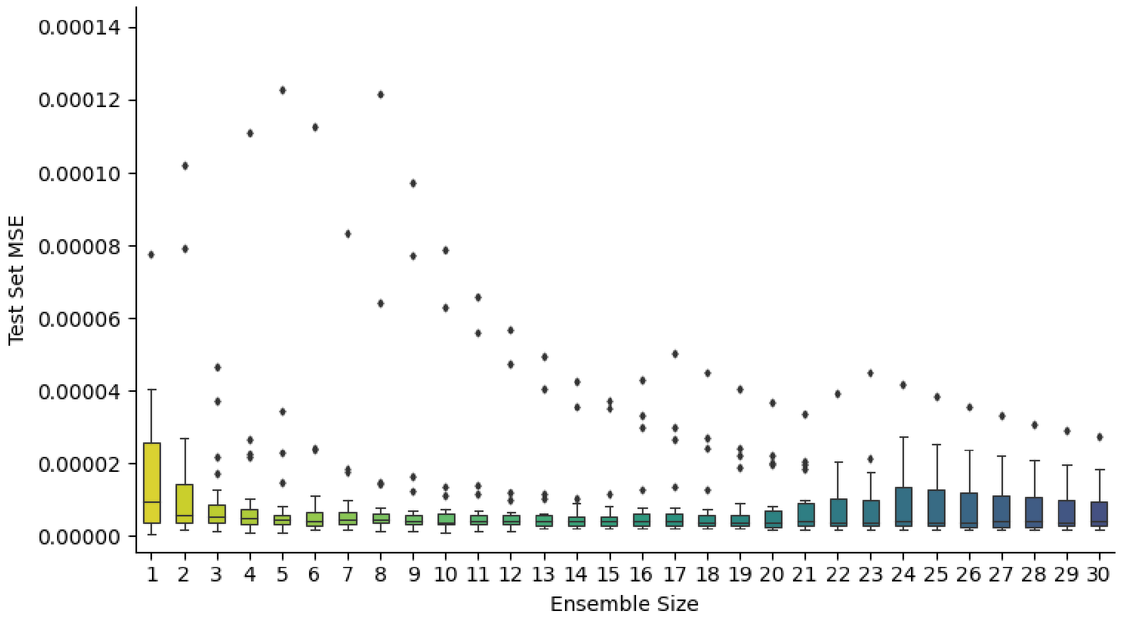


Figure 6.13: MV Artificial Domain: Test set MSE vs ensemble size of MTF.

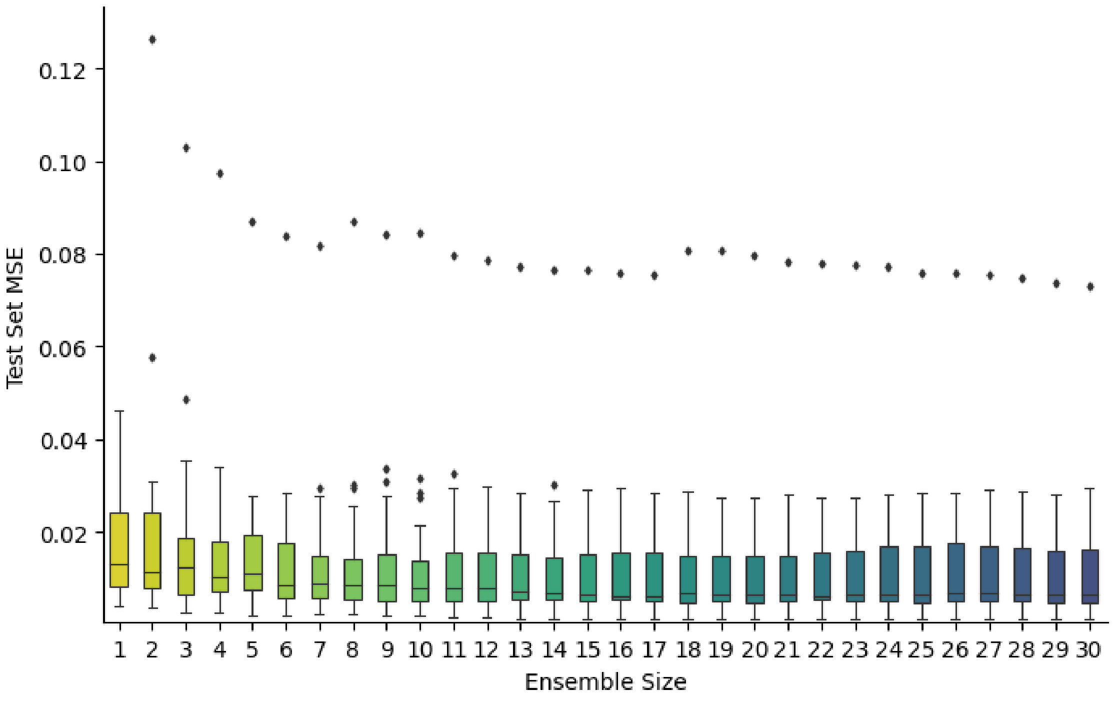


Figure 6.14: Servo: Test set MSE vs ensemble size of MTF.

The ensemble size at which the test set MSE started to converge was lower than initially anticipated, compared to RF models which typically started with a minimum ensemble size of 100. The error in prediction for the majority of datasets plateaus at an ensemble size of 25, excluding the outlier results.

The test set MSE on datasets that favoured shallow MTs for base learners, discussed in the previous section, converged at low ensemble sizes. These datasets were Auto-MPG, Boston housing and Servo. Note that these datasets were all small datasets. The difference between the test set MSE after the first base learner and the MSE after convergence was observed to be small, shown clearly in Figure 6.7. Although it seems that a single shallow MT is capable of adequately fitting the small datasets with minimal overfitting, multiple MTs reduce the variance error and provide stabilised results over repeated test runs.

For datasets that favoured deep MTs, namely Elevators, Friedman artificial domain, and MV artificial domain, the MSE converged at a larger ensemble size. The difference between the initial MSE and the converged MSE was also large as shown clearly in Figure 6.11. Furthermore, the latter datasets were all large datasets. This observation was explained through the bias-variance tradeoff. For deeper MTs with lower bias and higher variance errors compared to shallow MTs, bagging demanded a higher number of base learners to effectively reduce the higher variance error. Therefore, the MSE difference between a single MT and the collective ensemble was also greater the deeper the MTs are.

The reasons hypothesised for the smaller required ensemble size in both shallow and deep MTs compared to other decision tree-based ensembles such as RFs are as follows. The base learners of RF in this thesis were grown up to depths of 250 and exhibited higher variance errors compared to the MTs. Therefore, a larger number of base learners are required to effectively reduce the higher variance error. MTs that are capped at a depth of two to six, have a lower variance error and therefore converge quicker with each additional base learner in the ensemble. The number of model trees required to significantly reduce the variance error is minimal in comparison to the number of regression trees. Furthermore, due to the randomness injected by the adapted GASOPE, MTs in MTF are decorrelated to a greater extent than regression trees in RF.

6.1.4. Tuned MTF hyperparameters

The best performing hyperparameter values for MTFs on each of the ten benchmark datasets are listed in Table 6.2 together with the attributes of the dataset. All three of the hyperparameters were minimally chosen to optimise the computational costs of the induction algorithm of MTF. In the case of datasets that were relatively small in size (less than a thousand instances), MTF favoured stumps for its base learners. The only exception to this was for Machine CPU where the maximum tree depth that produced the best result was only one greater.

Table 6.2: Chosen hyperparameter values of MTF for each benchmark dataset.

Dataset	Maximum Polynomial Order	Maximum Tree Depth	Ensemble Size
Abalone	2	3	20
Auto-MPG	3	2	30
Boston housing	3	2	30
California housing	2	4	25
CASP	2	2	25
Elevators	2	6	25
Friedman artificial domain	2	6	20
Machine CPU	1	3	25
MV artificial domain	2	6	20
Servo	3	2	30

For datasets with few training instances, it seemed that partitioning the feature space through too many splits resulted in degraded performance. It was hypothesised that the reason for this was the likeliness of GASOPE to overfit smaller-sized subsets of data. For larger datasets, there were more training instances during induction and subsequently, leaf nodes comprised larger subsets of data. Therefore, small values were chosen for the maximum tree depth for smaller datasets. An exception to this was the choice of inducing tree stumps for CASP, because the predictive accuracy was unstable in deeper MTs, as shown in Figure 6.3e.

An increase in the ensemble size beyond the value where the test set MSE converged, had no benefits beyond stabilising the predictions of MTF For large datasets, the smallest ensemble sizes that did not compromise the ability of the ensemble to reduce the high variance of a base leaner were chosen as the final hyperparameter value. The computational cost of the induction of an MTF on small datasets was significantly smaller and therefore the minimisation of the ensemble size was less strict.

6.2. Comparison results

The experimental results and significance of the comparison process are listed and discussed in this section. Section 6.2.1 first discusses the comparison of the predictive accuracy of MTF against MT and M5 to motivate the ensemble method. Section 6.2.2 discusses the results of the comparison between the selected ensemble models in both predictive accuracy and computational cost of induction. Section 6.2.3 discusses the statistical significance of the results through the non-parametric Friedman test together with the post-hoc pairwise Bonferroni-Dunn test.

6.2.1. The necessity of ensemble methods

Table 6.3 lists the experimental results of MTF, MT and M5 on all ten benchmarking datasets. MTF produced a lower average test set RMSE than MT for nine out of the ten datasets. For all ten datasets, MTF produced a more stable test set RMSE than MT. For Boston housing, a dataset which was characterised as highly nonlinear, high dimensional and comprised few training instances, MT produced very poor predictive accuracy. For Boston housing, the average test set RMSE score was large and highly variable. The model tree forest method provided valuable stability for MTs to produce a much more competitive test set RMSE. For the more linear characterised datasets, i.e. Abalone and Machine CPU, the difference between the average training and test set RMSE for an MT was much larger than for an MTF. This indicates that MTs were prone to overfitting, which MTF combated. Evidently, the necessity of the model tree forest method was justified.

The predictive performance of M5 was poor as it produced average test set RMSEs up to two orders of magnitude larger than both MT and MTF. The larger training set RMSE scores for M5s showed that M5s were unable to capture the relationships between the features of the dataset as efficiently compared to MTs.

Table 6.3: Performance comparison of M5s, MTs and MTFs on all 10 benchmarking datasets.

Dataset	Model	Training set		Test set	
		$\overline{RMSE}$	σ_{RMSE}	$\overline{RMSE}$	σ_{RMSE}
Abalone	M5	0.074866	0.000744	0.076545	0.004053
	MT	0.005389	0.000102	0.009037	0.018419
	MTF	0.005322	0.000145	0.005570	0.000611
Auto-MPG	M5	0.069695	0.004315	0.082646	0.015379
	MT	0.004162	0.000363	0.006142	0.002438
	MTF	0.003616	0.000303	0.005026	0.002090
Boston housing	M5	0.061264	0.007321	0.085531	0.025073
	MT	0.003934	0.000634	18.663880	81.130081
	MTF	0.003507	0.000216	0.006526	0.004654
California housing	M5	0.089944	0.000878	0.102740	0.003937
	MT	0.015467	0.000243	0.028865	0.049465
	MTF	0.014109	0.000218	0.015677	0.003643
CASP	M5	0.196434	0.000746	0.208270	0.002600
	MT	0.056032	0.000547	0.056498	0.001424
	MTF	0.054470	0.000202	0.054913	0.001172
Elevators	M5	0.032023	0.000412	0.033555	0.001241
	MT	0.000952	0.000015	0.001108	0.000706
	MTF	0.000866	0.000012	0.001073	0.000285
Friedman artificial domain	M5	0.032640	0.000164	0.033518	0.000289
	MT	0.001241	0.000098	0.001268	0.000087
	MTF	0.001055	0.000019	0.001092	0.000029
Machine CPU	M5	0.028839	0.005599	0.055650	0.032674
	MT	0.000620	0.000116	0.003962	0.007888
	MTF	0.000441	0.000094	0.001561	0.001675
MV artificial domain	M5	3.036e-04	1.534e-05	3.425e-04	6.922e-05
	MT	3.738e-07	2.158e-07	1.533e-06	5.140e-06
	MTF	1.254e-06	3.887e-07	1.861e-06	1.790e-06
Servo	M5	0.046675	0.014230	0.062162	0.022631
	MT	0.003746	0.001451	0.007405	0.007675
	MTF	0.003206	0.000576	0.006686	0.007606

6.2.2. Evaluation of the accuracy of each model

MTF was compared to the selected ensemble models in Tables 6.4 and 6.5. MTF outperformed all other models in terms of the average test set RMSE in five datasets. Two of the datasets, i.e. Abalone and Machine CPU, were characterised as fairly linear. The remaining three, i.e. Auto-MPG, Elevators, and Friedman artificial domain were considered quadratic. The five datasets had a number of instances ranging from 209 to 40778 and a number of features ranging from seven to eighteen. This showed the ability of MTF to perform competitively in various cases.

Table 6.4: Performance comparison of all ensemble models on the first five datasets.

Dataset	Model	Training set		Test set	
		$\overline{RMSE}$	σ_{RMSE}	$\overline{RMSE}$	σ_{RMSE}
Abalone	M5E	0.074970	0.000641	0.076379	0.003951
	MTF	0.005322	0.000145	0.005570	0.000611
	NNE	0.002511	0.000654	0.006733	0.001070
	RF	0.004111	0.000071	0.005754	0.000627
	SVRE	0.005256	0.000089	0.006143	0.000528
Auto-MPG	M5E	0.067171	0.003143	0.079311	0.013131
	MTF	0.003616	0.000303	0.005026	0.002090
	NNE	0.000471	0.000286	0.008663	0.003216
	RF	0.000723	0.000054	0.005127	0.002334
	SVRE	0.004830	0.000306	0.005588	0.001997
Boston housing	M5E	0.052854	0.002909	0.072946	0.019754
	MTF	0.003507	0.000216	0.006526	0.004654
	NNE	0.000130	0.000088	0.012144	0.007102
	RF	0.000697	0.000044	0.004818	0.002253
	SVRE	0.028804	0.001155	0.028275	0.007454
California housing	M5E	0.082564	0.000552	0.093782	0.003744
	MTF	0.014109	0.000218	0.015677	0.003643
	NNE	0.011292	0.001629	0.015273	0.000998
	RF	0.001411	0.000014	0.010421	0.000777
	SVRE	0.012838	0.000135	0.013745	0.000901
CASP	M5E	0.175107	0.000511	0.187256	0.002092
	MTF	0.054470	0.000202	0.054913	0.001172
	NNE	0.039984	0.002481	0.048893	0.001177
	RF	0.003781	0.000025	0.027099	0.000904
	SVRE	0.017463	0.000086	0.031247	0.000954

The poorest predictive accuracy of MTF was observed for California housing and CASP. For both large datasets, MTF scored the fourth-best average training and test set RMSE. The average training and test set values were almost equal, meaning MTF did not overfit the data but was simply unable to produce an adequate fit. In comparison, RF produced a training set RSME one order of magnitude smaller and still produced a lower test set RSME than MTF on both datasets. However, the predictive accuracy of MTF is within the same order of magnitude compared to the remaining ensemble models. RF was the only ensemble model robust to outlier values.

Table 6.5: Performance comparison of all ensemble models on the second five datasets.

Dataset	Model	Training set		Test set	
		$\overline{RMSE}$	σ_{RMSE}	$\overline{RMSE}$	σ_{RMSE}
Elevators	M5E	0.031444	0.000253	0.032502	0.000924
	MTF	0.000866	0.000012	0.001023	0.000285
	NNE	0.000532	0.000077	0.001075	0.000066
	RF	0.000229	0.000003	0.001598	0.000118
	SVRE	0.009327	0.000060	0.012461	0.000563
Friedman artificial domain	M5E	0.031481	0.000127	0.032136	0.000334
	MTF	0.001055	0.000019	0.001092	0.000029
	NNE	0.000759	0.000039	0.001131	0.000027
	RF	0.000306	0.000002	0.001622	0.000047
	SVRE	0.013553	0.000037	0.024714	0.000441
Machine CPU	M5E	0.025148	0.002403	0.046558	0.021910
	MTF	0.000441	0.000094	0.001561	0.001675
	NNE	0.000102	0.000145	0.006425	0.008666
	RF	0.000404	0.000068	0.001832	0.002874
	SVRE	0.013404	0.001762	0.012200	0.013372
MV artificial domain	M5E	2.270e-04	1.576e-05	2.504e-04	3.419e-05
	MTF	1.254e-06	3.887e-07	1.861e-06	1.790e-06
	NNE	8.122e-08	3.701e-08	4.335e-07	4.343e-07
	RF	4.007e-07	7.107e-08	1.948e-06	1.270e-06
	SVRE	1.980e-03	2.098e-05	1.992e-03	3.596e-05
Servo	M5E	0.046675	0.014230	0.062162	0.022631
	MTF	0.003206	0.000576	0.006686	0.007676
	NNE	0.000678	0.000903	0.023160	0.016373
	RF	0.000761	0.000123	0.005843	0.006780
	SVRE	0.009718	0.001250	0.011579	0.010459

MTF produced the most stable and lowest average test set RMSE on Machine CPU compared to all other models. Machine CPU is a piecewise linear dataset on which GPMCC struggled to produce competitive results [23]. This portrays the flexibility of MTF to produce adequate results on both linear as well as nonlinear data, given that GASOPE is properly tuned to the problem at hand.

The computational cost of the induction of a model on the training set of each dataset is listed in Tables 6.6 and 6.7. For all small datasets, i.e. Auto-MPG, Boston housing, Machine CPU and Servo, MTF had a mean training time up to four orders of magnitude greater than M5E, SVRE and RF. For large datasets, the mean training time for SVRE increased, but M5E and RF remained up to three orders of magnitude smaller than MTF. For California housing, CASP, and Servo, no ensemble model was close to the performance of RF in both mean training time and average test set RMSE simultaneously. For Servo, MTF produces an average test set RMSE approximately 7% larger than RF, but at a mean training time approximately 800 times greater.

Table 6.6: Computational cost comparison of all ensemble models on the first five datasets.

Dataset	Model	Mean Training Time (s)	Standard Deviation
Abalone	M5E	0.861	0.076
	MTF	807.823	53.069
	NNE	6577.322	283.357
	RF	0.661	0.048
	SVRE	12.447	0.236
Auto-MPG	M5E	0.099	0.032
	MTF	391.760	20.691
	NNE	123.384	47.866
	RF	1.074	0.075
	SVRE	0.186	0.015
Boston housing	M5E	0.236	0.040
	MTF	789.494	52.867
	NNE	5814.841	233.535
	RF	1.643	0.063
	SVRE	0.348	0.023
California housing	M5E	13.221	0.878
	MTF	5761.600	283.561
	NNE	7593.792	400.557
	RF	19.328	0.290
	SVRE	405.501	10.067
CASP	M5E	2.251	0.304
	MTF	11638.337	854.065
	NNE	11336.146	558.241
	RF	44.905	0.672
	SVRE	17209.047	378.836

The mean training time for MTF is the closest to the mean training time of NNE over the ten datasets. MTF outperforms NNE in the test set RSME for seven out of the ten datasets. For Boston housing, MTF produced a value with one order of magnitude less for both the average test set RMSE and the mean training time than NNE. SVRE is the model with the greatest difference between the fastest and longest mean training time over the ten datasets.

Table 6.7: Computational cost comparison of all ensemble models on the second five datasets.

Dataset	Model	Mean Training Time (s)	Standard Deviation
Elevators	M5E	8.544	0.542
	MTF	15470.549	285.525
	NNE	6921.400	399.231
	RF	29.984	0.493
	SVRE	414.859	11.172
Friedman artificial domain	M5E	23.006	1.863
	MTF	13692.882	780.484
	NNE	9159.162	483.096
	RF	51.611	0.835
	SVRE	3429.283	92.476
Machine CPU	M5E	0.065	0.018
	MTF	396.721	13.284
	NNE	642.023	186.528
	RF	0.967	0.073
	SVRE	0.085	0.005
MV artificial domain	M5E	6.195	0.436
	MTF	12030.477	882.228
	NNE	10473.581	534.142
	RF	31.578	0.521
	SVRE	70.161	2.218
Servo	M5E	0.107	0.031
	MTF	779.965	17.676
	NNE	42.441	13.787
	RF	0.915	0.071
	SVRE	0.120	0.008

6.2.3. Statistical significance of the results

For each dataset, statistical tests were performed on the individually ranked test set RSME scores for which the average values were listed in Table 6.4 and Table 6.5. Additionally, a statistical test was performed for the ranked test set RMSE values over all 10 datasets together. Each of the eleven Friedman tests resulted in a rejection of the null-hypothesis and post-hoc tests were performed accordingly. The results of the post-hoc tests were presented through the critical difference diagrams in Figures 6.15 to 6.24.

For example, Figure 6.15 shows that on average, MTF was ranked the most accurate model whilst M5E was the worst on the Abalone dataset. Furthermore, NNE and SVRE were shown to produce results that are not statistically distinguishable. The same applied to MTF and RF. However, MTF and RF were considered to be statistically more accurate models than SVRE, NNE and M5E on the Abalone dataset.

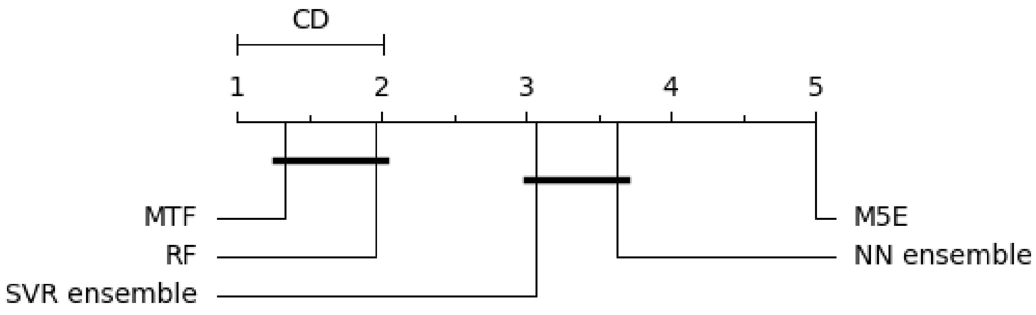


Figure 6.15: Critical difference diagram on Abalone.

For the remaining critical diagrams, notice the difference in the performance of MTF compared to the more simple model tree ensemble of M5E. MTF was shown to produce a similar or better rank in all ten datasets, nine of which are statistically significant.

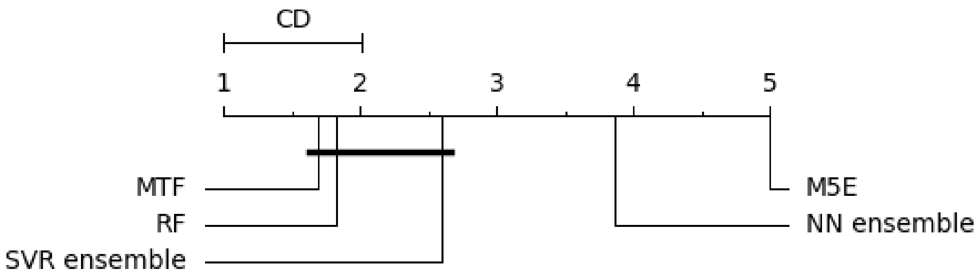


Figure 6.16: Critical difference diagram of Auto-MPG.

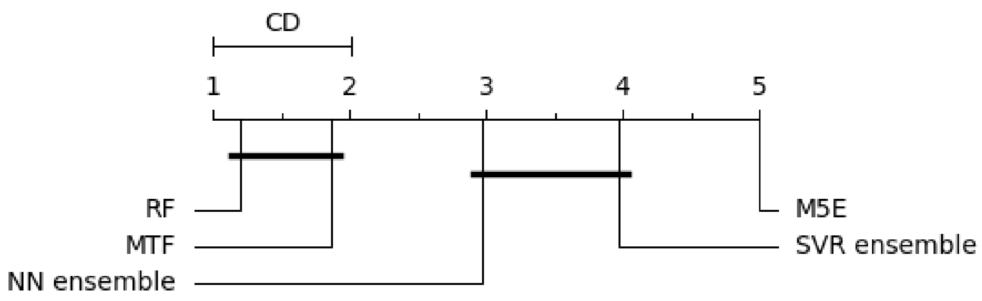


Figure 6.17: Critical difference diagram of Boston housing.

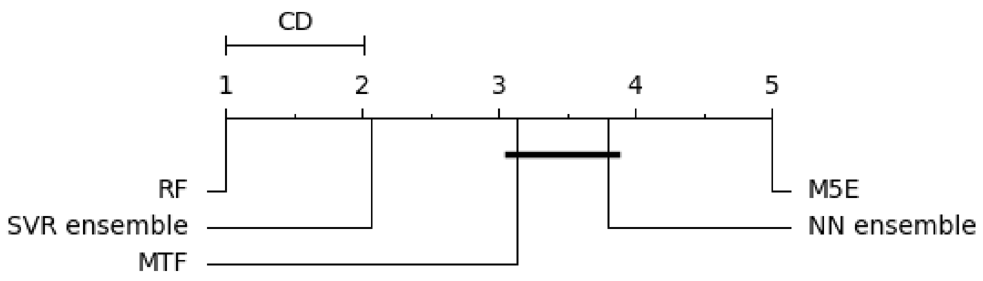


Figure 6.18: Critical difference diagram of California housing.

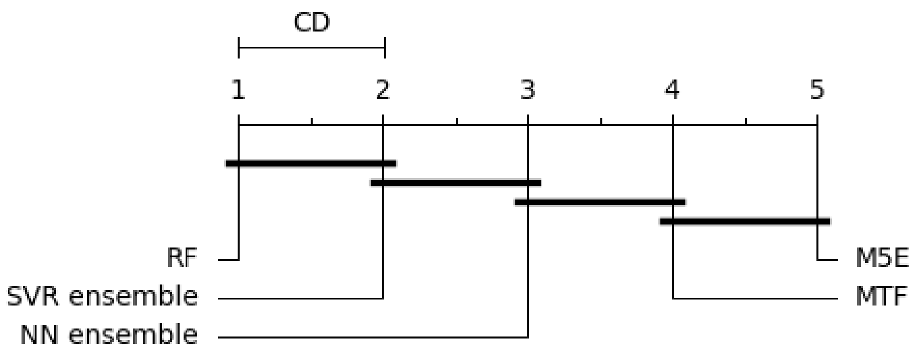


Figure 6.19: Critical difference diagram of CASP.

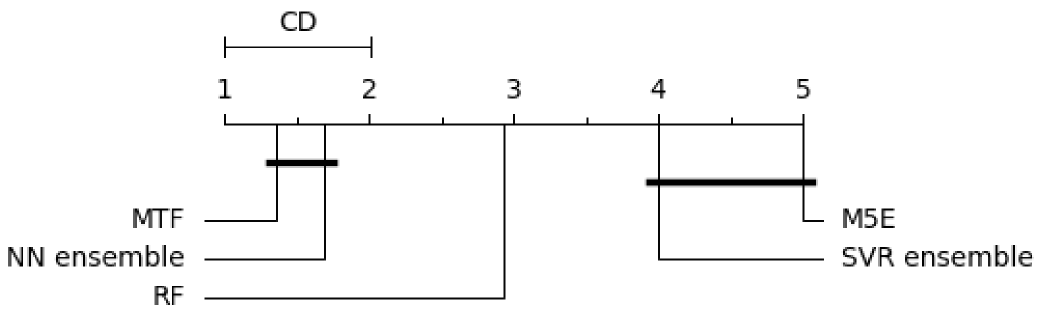


Figure 6.20: Critical difference diagram of Elevators.

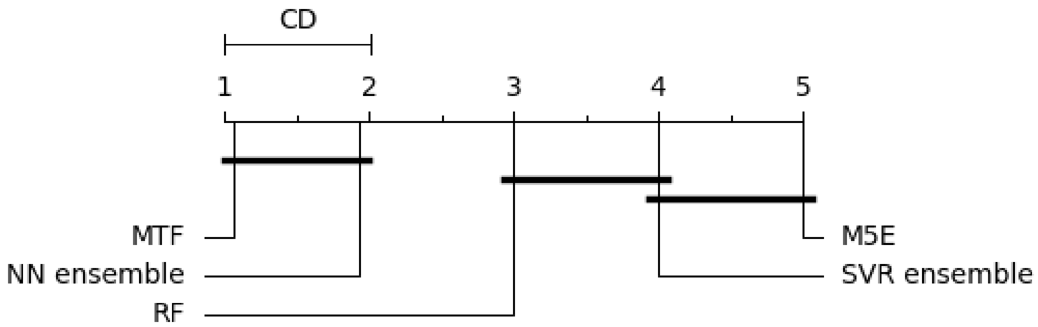


Figure 6.21: Critical difference diagram of Friedman artificial domain.

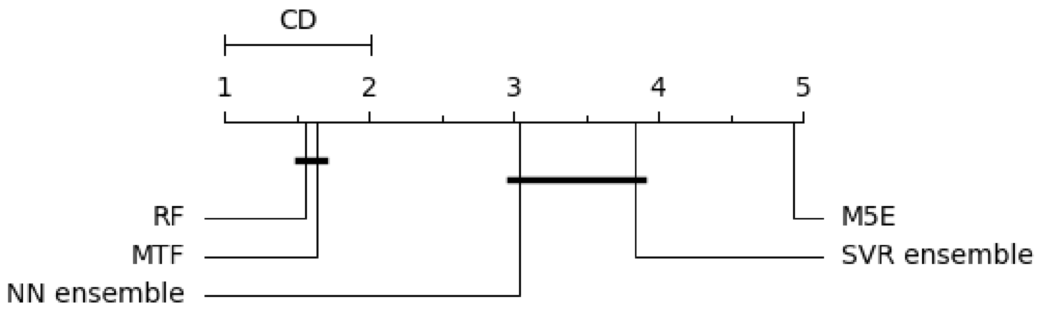


Figure 6.22: Critical difference diagram of Machine CPU.

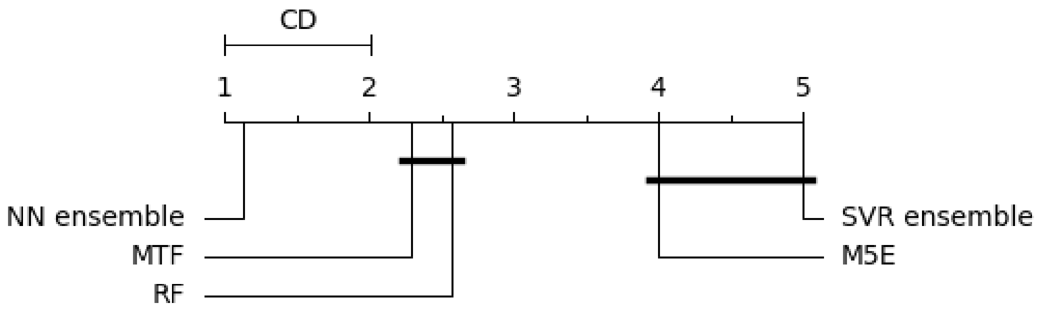


Figure 6.23: Critical difference diagram of MV artificial domain.

Statistically, MTF produced the joint best results for seven of the ten datasets. Of these seven, the performance of the MTF was matched five times by an RF, but only twice by an NNE and once by an SVRE.

The RF statistically matched the predictive accuracy of an MTF in six of the ten datasets. Out of the remaining four datasets, the prediction accuracy of the MTF was significantly better than an RF on the Friedman artificial domain and Elevators dataset. The prediction accuracy of the RF was statistically better than an MTF for CASP and California housing.

The four datasets characterised as the most nonlinear of the ten datasets are Auto-MPG, Boston housing, Friedman artificial domain, and Servo. Compared to an NNE and an SVRE, which are state-of-the-art solutions to nonlinear regression problems, MTF significantly outperformed both on two out of the four highly nonlinear datasets. For the remaining two datasets, the MTF matched an NNE while significantly outperforming an SVRE on the Friedman artificial domain and vice-versa on the Auto-MPG.

Furthermore, the predictive accuracy of the MTF on the purely linear characterised Machine CPU dataset was statistically similar to an RF and better than the remaining ensemble models. This showed the high flexibility potential of an MTF on both linear and nonlinear characterised datasets.

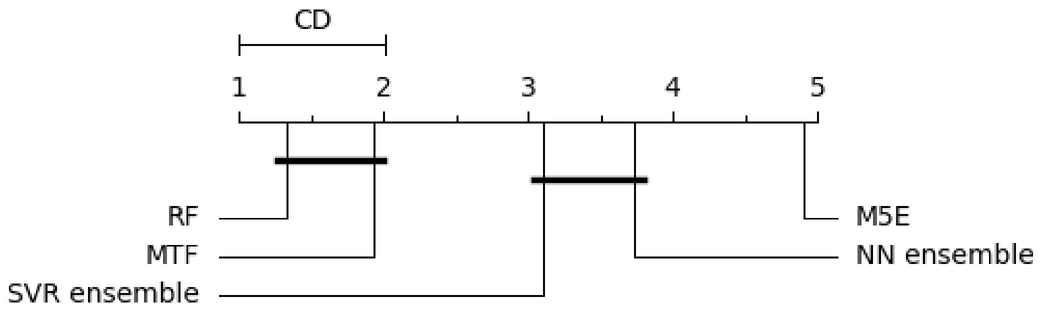


Figure 6.24: Critical difference diagram of Servo.

Figure 6.25 shows the critical difference diagram over the combined testing runs of all ten datasets. The RF-inspired modifications brought into the MTF to improve its performance over a single model tree and bagged ensemble of model trees were shown to have produced a statistically better predictive accuracy compared to all models except an RF.

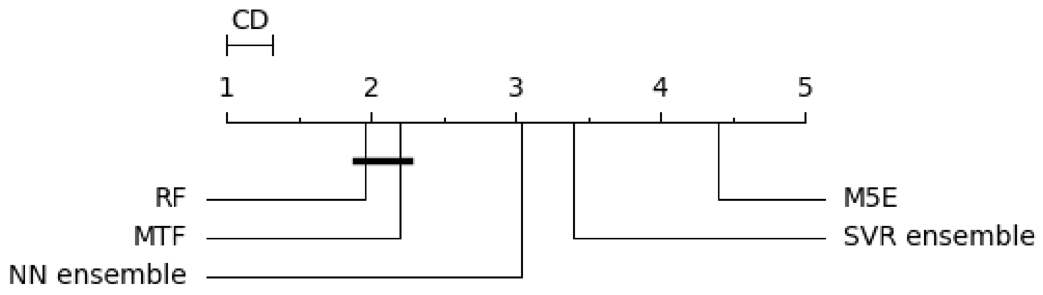


Figure 6.25: Critical difference diagram on the combined testing runs of all ten datasets.

6.3. Conclusion

This chapter presented the experimental results of the tuning and comparison process.

The tuning results of an MTF motivated the use of a maximum polynomial no greater than three for the datasets. An order exceeding three overfitted each dataset and raised the computational cost of the induction algorithm. A maximum tree depth of six produced the best tradeoff between predictive accuracy and computational cost for large datasets. In comparison, small datasets favoured the use of tree stumps to stabilise the predicted values of an MT. Finally, a small ensemble size between 20 and 30 was adequate in reducing the variance error of the base learners.

Next, it was shown that the ensembling of MTs through the model tree forest method, guaranteed an improvement in predictive accuracy and stability. Furthermore, the model tree forest method significantly increased the robustness and combated the overfitting of MT on more linearly characterised datasets, i.e. Abalone and Machine CPU.

Compared to the selected ensemble methods, an MTF was shown to produce competitive predictive accuracy on data that had linear relationships between the input and target features. This showed that the advantages of model trees are not limited to applications on nonlinear data. However, the model must be thoroughly tuned to the problem at hand. Statistically, MTF was ranked joint best for seven out of the ten datasets.

The performance of the MTF was unable to compete with an RF in both predictive accuracy and computational cost simultaneously. Although the predictive accuracy matched an RF for the majority of datasets, the computational cost of the induction of the MTF was up to four orders of magnitude greater. However, compared to the ensemble methods other than RFs, an MTF was significantly more accurate in its predictions of unseen data for the majority of datasets.

Chapter 7

Conclusion

This chapter concludes the thesis through a summary in Section 7.1, followed by a discussion of the possible avenues for future research regarding the topic of model tree ensembles in Section 7.2.

7.1. Summary

The main goal of this thesis was to thoroughly research, design and implement a model tree that comprises higher-order polynomial output functions within an ensemble. To achieve this, Chapter 2 started with the introduction of decision trees and the existing ensemble methods that can be applied to reduce the high variance error that decision trees tend to exhibit. It was also shown that model trees that comprise linear output functions have been used within an ensemble to successfully combat the high variance error. However, the only two model trees with nonlinear output functions, i.e. PLT and GPMCC, were never tested within an ensemble [23, 28]. Model trees that comprise higher-order polynomial output functions were shown to outperform their linear output function counterparts. A disadvantage of PLT and GPMCC is the computational demand associated with the induction of nonlinear output functions. Therefore, to successfully implement MTF, the tree structure in which GASOPE was employed to produce higher-order output functions, required a less computationally expensive induction algorithm.

The adaptation of GASOPE was discussed in Chapter 3. As opposed to the original GASOPE within GPMCC which evolved polynomials with orders up to five, the adapted GASOPE favoured a maximum polynomial order of two in the approximation of the Abalone dataset. A value greater than two led to overfitting and poor predictive accuracy for Adapted GASOPE. This accentuates the need for the optimisation of such a complex regression model as GASOPE, as well as any model tree that employs it. With proper optimisation of the applicable hyperparameters, GASOPE produced adequate results to justify the implementation thereof within MTF.

Next, Chapter 4 presented the decision tree structure as well as the ensemble method that was employed to induce MTF. The induction algorithm was designed to decorrelate the MTs within an ensemble and to provide the benefit of the reduction of the high variance error. Decorrelation was achieved through the use of bagged sets of training data and a randomly selected subset of features for split evaluation.

Chapter 5 discussed the experimental process to tune and compare MTF against other models.

An MTF was first compared to an MT and an M5 to determine whether the decorrelation steps were successful in improving predictive accuracy. The ensemble models selected for the performance comparison of MTF were RF, NNE, SVRE and M5E. Performance was analysed in both predictive accuracy and the computational cost of induction.

The penultimate Chapter 6 presented and discussed the experimental results of the tuning and comparison processes. MTF never exceeded a maximum polynomial order of three, as doing so resulted in too costly computational costs and the tendency to overfit the data at hand. It was found that the ensembling of model trees on small datasets favoured the use of stumps for base learners. For large datasets, base learners which were allowed to be grown up to a depth of six produced the best results. The greedy induction and decorrelation of the MTs within an ensemble provided the benefits of a reduced variance error. The MTF was shown robust to overfitting, which a single model tree is susceptible to.

Furthermore, the accuracy of MTFs against the selected four ensemble models on a set of ten benchmarking datasets was discussed and the statistical significance was determined. An MTF was shown to produce competitive performance on both nonlinear as well as linear problems. An MTF produced statistically similar predictive accuracy to an RF whilst outperforming SVRE, NNE and M5E for the majority of datasets. However, the induction of the RF had a computational cost of up to four orders of magnitude less than that of an MTF.

7.2. Future Research

Throughout the conceptualisation of MTF, several design aspects were considered but not yet implemented. These possible ideas are reserved for future research and are discussed below.

Boosting. Boosting is an alternative ensemble technique to bootstrap aggregation which is employed in MTF. Boosting induces base learners sequentially with the aim of reducing the net error at every step. Boosting favours the use of weak learners as base learners. Section 2.2.4 discussed boosting as a popular ensemble method for decision tree stumps. Although it was hypothesised that bagging would better suit the application of model trees within an ensemble, the implementation of boosting may also produce good results. Bagging was chosen as the preferred ensemble method due to the notion that MTF would incorporate base learners that are deep in nature. Furthermore, boosting demands an ensemble of weak learners. Simply limiting the depth of MTF does not produce a weak learner, due to the complex nature of the higher-order polynomial output functions evolved with GASOPE. The complexity of GASOPE should also be scaled back to properly investigate the benefits of boosting.

Hybrid model trees. The next idea, hybrid model trees, refers to the base learners of MTF. One big disadvantage of model trees is the inability to isolate outliers into a separate leaf node. Due to the multivariate output functions of model trees, leaf nodes have a minimum number of instances hyperparameter which has to be satisfied during induction. Therefore, outliers are grouped with other observations within a leaf node. As a result, the output function produces a weak fit of the data subset associated with the leaf node. A proposed hybrid model tree comprises leaf nodes of either constant values or polynomials of differing orders to dictate the output function of a decision tree. Thereby, the need for a minimum number of instances within a leaf node is dropped and outliers are allowed to be isolated within their own leaf nodes. Note that such a decision tree would require quite a deep tree structure to allow for the use of leaf nodes with constant values.

Data extraction. The implementation of decorrelated model trees within an ensemble results in the initial interpretability of decision trees being lost. Interpretability is one of the biggest advantages that decision trees have over other models. However, there are steps that can be implemented to extract information from MTF. One such step is the process of variable importance extraction. By measuring the frequency of the unique feature splits within the base learners of the ensemble, underlying relationships that determine which variable has the most discriminative power within the dataset are uncovered. The results of the evolutionary algorithm of GASOPE can be mined to offer additional information regarding the most prolific input features. Note that this does not directly increase the performance of the model, but restores the advantage that model trees offer through interpretability. Thereby, allowing for the use of interpretable decorrelated model trees within an ensemble.

Bibliography

[1] D. Aleksovski, J. Kocijan, and S. Džeroski, "Model-tree ensembles for noise-tolerant system identification," *Advanced Engineering Informatics*, vol. 29, no. 1, pp. 1–15, 2015.

[2] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

[3] R. C. Barros, D. D. Ruiz, and M. P. Basgalupp, "Evolutionary model trees for handling continuous classes in machine learning," *Information Sciences*, vol. 181, no. 5, pp. 954 – 971, 2011.

[4] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's razor," *Information Processing Letters*, vol. 24, no. 6, pp. 377 – 380, 1987.

[5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees.* Monterey, CA: Wadsworth and Brooks, 1984.

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

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

[8] T. Chai and R. R. Draxler, "Root mean square error (rmse) or mean absolute error (mae)?– arguments against avoiding rmse in the literature," *Geoscientific Model Development*, vol. 7, no. 3, pp. 1247–1250, 2014.

[9] H. Chen, J. Periaux, and A. Ecer, "Domain decomposition methods using gas and game theory for the parallel solution of cfd problems," in *Parallel Computational Fluid Dynamics 2000*, C. Jenssen, H. Andersson, A. Ecer, N. Satofuka, T. Kvamsdal, B. Pettersen, J. Periaux, and P. Fox, Eds. Amsterdam: North-Holland, 2001, pp. 341 – 348.

[10] A. I. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R. R. Griffiths, H. Jianye, J. Wang, and H. B. Ammar, "An empirical study of assumptions in bayesian optimisation," *arXiv preprint arXiv:2012.03826*, vol. 445, 2020.

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

[12] J. H. Friedman, "Multivariate Adaptive Regression Splines," *The Annals of Statistics*, vol. 19, no. 1, pp. 1 – 67, 1991.

[13] P. Geurts, "Contributions to decision tree induction: bias/variance tradeoff and time series classification," Ph.D. dissertation, University of Liège Belgium, 2002.

[14] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer New York, 2013.

[15] O. Kisi and K. S. Parmar, "Application of least square support vector machine and multivariate adaptive regression spline models in long term prediction of river water pollution," *Journal of Hydrology*, vol. 534, pp. 104–112, 2016.

[16] J. R. Koza and R. Poli, *Genetic Programming.* Springer US, 2005, pp. 127–164.

[17] W. B. Langdon and R. Poli, *Foundations of genetic programming.* Springer Science & Business Media, 2013.

[18] G. Louppe, "Understanding random forests," *Cornell University Library*, vol. 10, 2014.

[19] P. Mair and R. Wilcox, "Robust statistical methods in r using the wrs2 package," *Behavior research methods*, vol. 52, no. 2, pp. 464–488, 2020.

[20] G. Moisen, "Classification and regression trees," *Encyclopedia of Ecology*, vol. 1, pp. 582–588, 2008.

[21] B. Pfahringer, "Semi-random model tree ensembles: An effective and scalable regression method," in *Australasian Joint Conference on Artificial Intelligence.* Springer, 2011, pp. 231–240.

[22] G. Potgieter and A. Engelbrecht, "Genetic algorithms for the structural optimisation of learned polynomial expressions," *Applied Mathematics and Computation*, vol. 186, no. 2, pp. 1441 – 1466, 2007.

[23] G. Potgieter and A. P. Engelbrecht, "Evolving model trees for mining data sets with continuous-valued classes," *Expert Systems with Applications*, vol. 35, no. 4, pp. 1513 – 1532, 2008.

[24] J. R. Quinlan *et al.*, "Learning with continuous classes," in *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, vol. 92. World Scientific Press, 1992, pp. 343–348.

[25] M. Sattari, M. Pal, R. Mirabbasi, and J. Abraham, "Ensemble of m5 model tree based modelling of sodium adsorption ratio," *Journal of AI and Data Mining*, vol. 6, no. 1, pp. 69–78, 2018.

[26] D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures.* Chapman and Hall/CRC, 2003.

[27] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems.* Curran Associates, Inc., 2012, vol. 25, pp. 2951–2959.

[28] L. Torgo, “Partial linear trees.” in *Proceedings of the International Conference on Machine Learning*, 2000, pp. 1007–1014.

[29] ——, “Functional models for regression tree leaves,” in *Proceedings of the International Conference on Machine Learning*, vol. 97, 1997, pp. 385–393.

[30] Y. Wang and I. Witten, “Induction of model trees for predicting continuous classes,” in *Proceedings of the European Conference on Machine Learning*, 1998.

Appendix A

Dataset linearity estimation via GASOPE

Tables A.1 and A.2 list the tuning results of the maximum polynomial order hyperparameter in full. $\overline{TMSE}$ refers to the mean squared error on the training set, $\overline{GMSE}$ refers to the mean squared error on the generalisation set, σ indicates the standard deviation and $\bar{t}$ is the average simulation completion time in seconds.

Table A.1: Tuning results of the adapted GASOPE for varying maximum polynomial orders on the first five datasets.

Dataset	Order	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
	1	5.937e-03	1.091e-04	6.162e-03	5.371e-04	15.38
Abalone	2	5.741e-03	1.364e-04	8.266e-03	6.799e-03	21.23
	3	5.815e-03	1.207e-04	1.310e-02	3.833e-02	54.66
	4	5.890e-03	1.490e-04	2.971e-01	1.588e+00	64.74
	1	5.407e-03	3.680e-04	6.412e-03	1.201e-03	6.328
Auto MPG	2	4.763e-03	2.655e-04	5.934e-03	1.158e-03	7.246
	3	4.860e-03	3.566e-04	5.664e-03	1.195e-03	8.756
	4	4.855e-03	3.376e-04	5.583e-03	9.479e-04	9.646
	1	7.638e-03	1.007e-03	1.079e-02	4.433e-03	10.73
Boston Housing	2	5.794e-03	7.578e-04	9.984e-03	3.912e-03	15.99
	3	6.155e-03	5.732e-04	1.078e-02	5.225e-03	25.60
	4	6.713e-03	7.766e-04	1.895e-02	3.733e-02	25.18
	1	1.979e-02	2.982e-04	1.972e-02	8.077e-04	52.60
California Housing	2	1.934e-02	4.362e-04	1.996e-02	1.600e-03	85.06
	3	2.000e-02	9.267e-04	2.348e-02	1.517e-02	381.3
	4	2.045e-02	9.323e-04	2.595e-02	2.309e-02	534.0
	1	5.932e-02	4.584e-04	5.955e-02	9.046e-04	145.4
CASP	2	5.759e-02	5.679e-04	5.790e-02	8.208e-04	226.9
	3	5.819e-02	6.763e-04	5.873e-02	1.214e-03	1023
	4	5.915e-02	8.167e-04	6.839e-02	4.853e-02	1255

Table A.2: Tuning results of the adapted GASOPE for varying maximum polynomial orders on the last five dataset.

Dataset	Order	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
	1	1.457e-03	4.007e-05	1.436e-03	8.418e-05	95.63
Elevators	2	1.342e-03	5.493e-05	1.328e-03	8.691e-05	127.8
	3	1.389e-03	5.332e-05	1.385e-03	9.578e-05	555.0
	4	1.516e-03	1.311e-04	1.513e-03	1.618e-04	811.6
	1	6.870e-03	3.401e-05	6.840e-03	1.364e-04	63.63
Friedman Artificial Domain	2	3.045e-03	8.219e-04	3.050e-03	8.374e-04	124.9
	3	3.434e-03	8.894e-04	3.436e-03	8.780e-04	781.6
	4	3.914e-03	9.047e-04	3.891e-03	8.994e-04	1150
	1	1.033e-03	1.952e-04	3.764e-03	8.445e-03	4.605e
Machine CPU	2	6.853e-04	1.528e-04	1.953e-02	4.898e-02	6.851
	3	6.550e-04	1.198e-04	9.943e-02	3.234e-01	9.872e
	4	7.416e-04	1.560e-04	3.093e+00	1.535e+01	11.31
	1	8.496e-04	1.316e-03	8.554e-04	1.332e-03	84.29
MV Artificial Domain	2	2.514e-04	1.149e-04	2.512e-04	1.167e-04	126.2
	3	4.755e-04	7.047e-04	4.771e-04	7.134e-04	706.1
	4	5.578e-04	7.588e-04	5.614e-04	7.717e-04	1026
	1	1.862e-02	3.113e-03	2.481e-02	9.850e-03	10.42
Servo	2	8.867e-03	2.171e-03	1.691e-02	1.027e-02	17.37
	3	8.137e-03	2.152e-03	1.485e-02	9.958e-03	20.31
	4	8.972e-03	2.050e-03	1.435e-02	8.698e-03	19.20

Appendix B

Complete tuning results for maximum tree depth

Tables B.1 through B.10 list the mean tuning results of the maximum tree depth hyperparameter in full, σ indicates the standard deviation.

Table B.1: Tuning results of MT for varying tree depth on Abalone.

Baseline comparison	Depth	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
	2	5.506e-03	1.060e-04	9.780e-03	1.713e-02	30.90
	3	5.413e-03	1.006e-04	1.292e-02	2.363e-02	41.16
	4	5.246e-03	1.038e-04	1.734e-02	3.813e-02	57.30
	5	5.007e-03	9.336e-05	2.801e-01	1.460e+00	85.74
Disabled	6	4.606e-03	1.084e-04	2.620e-02	1.042e-01	125.8
	7	4.173e-03	1.129e-04	9.535e-02	4.691e-01	180.1
	8	3.690e-03	1.223e-04	2.786e-01	1.460e+00	246.1
	9	3.255e-03	1.301e-04	3.237e-01	1.663e+00	314.3
	10	2.985e-03	1.387e-04	1.241e+00	4.020e+00	366.3
	2	5.505e-03	1.044e-04	1.255e-02	3.436e-02	31.33
	3	5.392e-03	9.197e-05	1.044e-02	2.224e-02	41.75
	4	5.223e-03	9.516e-05	1.675e-02	5.850e-02	57.58
	5	5.009e-03	1.030e-04	9.728e-02	4.108e-01	85.57
Enabled	6	4.640e-03	9.666e-05	4.151e-02	1.650e-01	128.7
	7	4.239e-03	1.338e-04	9.124e-03	7.163e-03	183.2
	8	3.759e-03	1.422e-04	1.346e-02	2.198e-02	248.9
	9	3.359e-03	1.591e-04	1.635e-02	2.225e-02	315.9
	10	3.144e-03	1.228e-04	1.936e-02	2.361e-02	366.6

Table B.2: Tuning results of MT for varying tree depth on Auto-MPG.

Baseline comparison	Depth	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
	2	4.123e-03	3.176e-04	6.480e-03	1.711e-03	11.77
	3	3.615e-03	3.054e-04	8.555e-03	4.047e-03	16.24
	4	2.684e-03	2.612e-04	9.941e+00	3.321e+01	27.25
	5	2.240e-03	2.300e-04	1.048e+02	3.292e+02	36.73
Disabled	6	2.220e-03	2.488e-04	9.508e+01	3.299e+02	37.49
	7	2.243e-03	2.636e-04	5.313e+01	2.696e+02	37.85
	8	2.169e-03	2.386e-04	9.978e+01	3.293e+02	37.94
	9	2.179e-03	2.959e-04	6.564e+01	2.698e+02	37.89
	10	2.202e-03	2.547e-04	9.381e+01	3.302e+02	37.55
	2	4.187e-03	3.071e-04	6.062e-03	1.441e-03	11.89
	3	3.642e-03	3.732e-04	7.256e-03	2.384e-03	16.37
	4	2.739e-03	3.173e-04	7.903e-03	3.251e-03	27.47
	5	2.308e-03	3.040e-04	9.544e-03	7.038e-03	36.81
Enabled	6	2.219e-03	3.046e-04	1.082e-02	7.357e-03	38.18
	7	2.228e-03	2.804e-04	2.316e-02	6.684e-02	38.39
	8	2.239e-03	2.950e-04	1.162e-02	9.239e-03	38.30
	9	2.214e-03	3.579e-04	1.863e-02	3.455e-02	38.44
	10	2.234e-03	3.070e-04	2.664e-02	9.646e-02	38.24

Table B.3: Tuning results of MT for varying tree depth on Boston housing.

Baseline comparison	Depth	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
	2	3.729e-03	4.277e-04	2.052e+01	1.123e+02	25.94
	3	2.851e-03	2.197e-04	4.656e+04	2.550e+05	36.51
	4	2.020e-03	2.475e-04	1.525e+01	5.375e+01	71.06
	5	1.420e-03	2.102e-04	1.173e+03	6.175e+03	117.6
Disabled	6	1.129e-03	2.018e-04	7.848e+02	3.913e+03	195.6
	7	1.034e-03	2.030e-04	3.957e+04	2.163e+05	223.3
	8	1.061e-03	2.307e-04	2.313e+02	6.416e+02	232.7
	9	1.044e-03	2.205e-04	1.193e+03	6.295e+03	212.2
	10	1.051e-03	2.037e-04	1.074e+06	5.881e+06	220.5
	2	3.677e-03	5.035e-04	2.252e-02	6.581e-02	27.60
	3	2.873e-03	2.888e-04	3.615e+02	1.569e+03	31.77
	4	2.001e-03	2.832e-04	2.507e+04	1.365e+05	80.01
	5	1.504e-03	2.639e-04	2.782e+04	1.523e+05	133.6
Enabled	6	1.325e-03	2.551e-04	1.229e+06	6.729e+06	180.4
	7	1.202e-03	2.388e-04	3.669e+02	1.711e+03	235.9
	8	1.240e-03	2.265e-04	1.191e+02	5.671e+02	231.0
	9	1.190e-03	2.150e-04	1.712e+04	7.744e+04	236.9
	10	1.234e-03	2.486e-04	2.049e+01	6.088e+01	250.1

Table B.4: Tuning results of MT for varying tree depth on California housing.

Baseline comparison	Depth	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
	2	1.750e-02	3.490e-04	1.797e-02	1.202e-03	192.0
	3	1.675e-02	3.012e-04	2.357e-02	2.990e-02	260.9
	4	1.552e-02	2.946e-04	1.793e-02	4.749e-03	322.0
	5	1.461e-02	1.596e-04	3.462e-01	1.732e+00	388.3
Disabled	6	1.317e-02	2.327e-04	1.927e-01	5.807e-01	475.4
	7	1.155e-02	1.818e-04	5.193e-02	1.062e-01	593.7
	8	9.990e-03	1.476e-04	1.377e-01	3.264e-01	770.1
	9	8.395e-03	1.656e-04	4.740e-01	1.105e+00	985.5
	10	7.050e-03	1.921e-04	1.382e-01	5.039e-01	1235
	2	1.769e-02	4.936e-04	1.899e-02	5.300e-03	205.0
	3	1.667e-02	2.925e-04	1.903e-02	4.239e-03	284.1
	4	1.551e-02	1.837e-04	1.888e-02	5.160e-03	338.8
	5	1.462e-02	2.061e-04	1.908e-01	9.237e-01	409.4
Enabled	6	1.323e-02	2.353e-04	8.328e-02	2.368e-01	496.6
	7	1.158e-02	1.982e-04	7.381e-02	1.645e-01	615.6
	8	9.992e-03	1.946e-04	1.346e-01	2.438e-01	782.7
	9	8.463e-03	1.671e-04	3.064e-01	8.081e-01	995.1
	10	7.220e-03	1.806e-04	9.248e-02	1.647e-01	1241

Table B.5: Tuning results of MT for varying tree depth on CASP.

Baseline comparison	Depth	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
	2	5.634e-02	4.317e-04	5.665e-02	1.367e-03	729.3
	3	5.350e-02	6.796e-04	5.432e-02	1.814e-03	1058
	4	5.068e-02	5.662e-04	5.846e-02	2.968e-02	1280
	5	4.803e-02	4.786e-04	1.478e-01	2.784e-01	1415
Disabled	6	4.541e-02	4.044e-04	1.478e+01	3.532e+01	1554
	7	4.206e-02	3.906e-04	3.521e+00	6.675e+00	1748
	8	3.897e-02	3.720e-04	1.024e+01	2.375e+01	1988
	9	3.528e-02	4.532e-04	7.447e+00	1.711e+01	2339
	10	3.146e-02	5.529e-04	1.367e+02	5.207e+02	2778
	2	5.631e-02	5.115e-04	5.638e-02	8.936e-04	982.8
	3	5.358e-02	5.591e-04	5.873e-02	1.773e-02	1606
	4	5.057e-02	4.347e-04	2.274e-01	7.066e-01	1822
	5	4.782e-02	5.241e-04	1.343e+00	5.481e+00	1930
Enabled	6	4.521e-02	4.826e-04	1.550e+00	4.711e+00	2032
	7	4.219e-02	5.274e-04	8.545e+00	2.967e+01	2160
	8	3.900e-02	5.348e-04	1.017e+04	5.564e+04	2334
	9	3.553e-02	5.290e-04	6.643e+01	2.149e+02	2617
	10	3.199e-02	7.173e-04	6.812e+04	2.754e+05	3050

Table B.6: Tuning results of MT for varying tree depth on Elevators.

Baseline comparison	Depth	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
	2	1.301e-03	3.115e-05	1.182e+02	6.477e+02	773.9
	3	1.156e-03	3.586e-05	2.529e-03	7.415e-03	898.3
	4	1.103e-03	2.578e-05	1.145e-03	5.774e-05	932.7
	5	1.021e-03	1.863e-05	8.159e-03	2.480e-02	998.2
Disabled	6	9.606e-04	1.594e-05	1.077e-01	3.710e-01	1108
	7	8.615e-04	1.301e-05	5.085e-01	1.187e+00	1296
	8	7.508e-04	1.451e-05	7.457e-01	1.081e+00	1563
	9	6.479e-04	1.406e-05	1.952e+00	1.767e+00	1923
	10	5.515e-04	1.412e-05	3.487e+00	2.006e+00	2360
	2	1.313e-03	3.851e-05	1.339e-03	7.128e-05	784.1
	3	1.158e-03	4.272e-05	1.192e-03	6.441e-05	960.0
	4	1.102e-03	2.722e-05	1.157e-03	4.344e-05	993.0
	5	1.024e-03	1.846e-05	1.117e-03	5.167e-05	1045
Enabled	6	9.608e-04	1.366e-05	1.115e-03	4.691e-05	1172
	7	8.757e-04	1.174e-05	1.588e-03	2.387e-03	1327
	8	7.839e-04	1.252e-05	2.088e-03	3.051e-03	1614
	9	7.012e-04	1.603e-05	2.524e-03	3.279e-03	1946
	10	6.245e-04	1.549e-05	3.229e-03	4.649e-03	2397

Table B.7: Tuning results of MT for varying tree depth on Friedman artificial domain.

Baseline comparison	Depth	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
	2	3.078e-03	4.716e-04	3.079e-03	4.816e-04	577.8
	3	1.965e-03	3.284e-04	1.976e-03	3.324e-04	848.5
	4	1.400e-03	2.623e-04	1.408e-03	2.695e-04	1028
	5	1.336e-03	1.492e-04	1.352e-03	1.527e-04	1158
Disabled	6	1.277e-03	1.183e-04	1.309e-03	1.245e-04	1298
	7	1.204e-03	7.137e-05	1.272e-03	7.356e-05	1502
	8	1.132e-03	4.402e-05	1.296e-03	4.749e-05	1823
	9	1.050e-03	4.036e-05	1.410e-03	5.247e-05	2286
	10	9.777e-04	2.149e-05	1.628e-03	4.409e-05	2892
	2	3.175e-03	5.443e-04	3.181e-03	5.556e-04	799.3
	3	1.807e-03	3.202e-04	1.804e-03	3.162e-04	1224
	4	1.496e-03	3.405e-04	1.506e-03	3.450e-04	1499
	5	1.386e-03	2.073e-04	1.403e-03	2.016e-04	1626
Enabled	6	1.246e-03	8.520e-05	1.281e-03	9.509e-05	1724
	7	1.233e-03	7.704e-05	1.315e-03	7.173e-05	1871
	8	1.150e-03	5.687e-05	1.321e-03	5.749e-05	2145
	9	1.067e-03	4.219e-05	1.431e-03	5.755e-05	2552
	10	9.793e-04	2.179e-05	1.634e-03	5.213e-05	3153

Table B.8: Tuning results of MT for varying tree depth on Machine CPU.

Baseline comparison	Depth	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
	2	6.026e-04	9.361e-05	9.390e-03	2.212e-02	7.683
	3	5.810e-04	1.355e-04	7.829e-03	1.986e-02	10.64
	4	5.716e-04	1.407e-04	7.551e-03	1.956e-02	13.66
	5	5.554e-04	1.362e-04	4.273e-01	2.291e+00	18.23
Disabled	6	5.571e-04	1.174e-04	4.223e-01	2.292e+00	18.68
	7	5.484e-04	1.213e-04	4.221e-01	2.292e+00	18.69
	8	5.562e-04	1.162e-04	4.226e-01	2.292e+00	18.41
	9	5.481e-04	1.378e-04	4.227e-01	2.292e+00	18.57
	10	5.361e-04	1.238e-04	4.219e-01	2.292e+00	18.73
	2	6.512e-04	1.253e-04	3.506e-03	5.197e-03	7.731
	3	5.780e-04	1.273e-04	4.126e-03	8.305e-03	10.86
	4	5.849e-04	1.195e-04	4.260e-03	8.707e-03	13.81
	5	5.669e-04	1.707e-04	4.607e-03	8.222e-03	18.37
Enabled	6	5.519e-04	1.203e-04	5.220e-03	8.165e-03	19.28
	7	5.625e-04	1.137e-04	4.994e-03	8.366e-03	19.32
	8	5.671e-04	1.637e-04	5.989e-03	1.180e-02	19.43
	9	5.562e-04	1.312e-04	4.480e-03	8.186e-03	19.31
	10	5.655e-04	1.368e-04	4.728e-03	7.767e-03	19.17

Table B.9: Tuning results of MT for varying tree depth on MV artificial domain.

Baseline comparison	Depth	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
	2	5.506e-03	1.060e-04	9.780e-03	1.713e-02	30.90
	3	5.413e-03	1.006e-04	1.292e-02	2.363e-02	41.16
	4	5.246e-03	1.038e-04	1.734e-02	3.813e-02	57.30
	5	5.007e-03	9.336e-05	2.801e-01	1.460e+00	85.74
Disabled	6	4.606e-03	1.084e-04	2.620e-02	1.042e-01	125.8
	7	4.173e-03	1.129e-04	9.535e-02	4.691e-01	180.1
	8	3.690e-03	1.223e-04	2.786e-01	1.460e+00	246.1
	9	3.255e-03	1.301e-04	3.237e-01	1.663e+00	314.3
	10	2.985e-03	1.387e-04	1.241e+00	4.020e+00	366.3
	2	5.505e-03	1.044e-04	1.255e-02	3.436e-02	31.33
	3	5.392e-03	9.197e-05	1.044e-02	2.224e-02	41.75
	4	5.223e-03	9.516e-05	1.675e-02	5.850e-02	57.58
	5	5.009e-03	1.030e-04	9.728e-02	4.108e-01	85.57
Enabled	6	4.640e-03	9.666e-05	4.151e-02	1.650e-01	128.7
	7	4.239e-03	1.338e-04	9.124e-03	7.163e-03	183.2
	8	3.759e-03	1.422e-04	1.346e-02	2.198e-02	248.9
	9	3.359e-03	1.591e-04	1.635e-02	2.225e-02	315.9
	10	3.144e-03	1.228e-04	1.936e-02	2.361e-02	366.6

Table B.10: Tuning results of MT for varying tree depth on Servo.

Baseline comparison	Depth	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
	2	4.078e-03	1.095e-03	8.603e-03	5.128e-03	8.685
	3	4.159e-03	1.215e-03	1.254e-02	9.328e-03	12.12
	4	4.078e-03	1.199e-03	1.115e-02	5.915e-03	15.26
	5	4.077e-03	1.121e-03	1.151e-02	6.450e-03	14.99
Disabled	6	4.173e-03	1.297e-03	1.227e-02	9.013e-03	14.86
	7	4.092e-03	1.296e-03	1.163e-02	6.847e-03	14.83
	8	4.280e-03	1.070e-03	1.119e-02	6.258e-03	15.04
	9	4.055e-03	1.253e-03	1.091e-02	6.089e-03	15.12
	10	4.099e-03	1.220e-03	1.123e-02	6.389e-03	14.86
	2	4.234e-03	1.362e-03	8.808e-03	7.529e-03	8.977
	3	4.235e-03	1.307e-03	1.235e-02	1.156e-02	12.09
	4	4.448e-03	1.171e-03	1.112e-02	8.168e-03	15.10
	5	4.162e-03	1.165e-03	1.353e-02	1.273e-02	15.42
Enabled	6	4.273e-03	1.258e-03	1.095e-02	9.716e-03	15.40
	7	4.424e-03	1.275e-03	1.181e-02	9.668e-03	15.39
	8	4.096e-03	1.180e-03	1.162e-02	9.248e-03	15.69
	9	4.232e-03	1.175e-03	1.246e-02	9.169e-03	15.64
	10	4.293e-03	1.037e-03	1.158e-02	1.006e-02	15.50

Appendix C

Complete results for ensemble size tuning of MTF

Tables C.1 through C.11 list the tuning results of the ensemble size hyperparameter in full. $\overline{TMSE}$ refers to the mean squared error on the training set, $\overline{GMSE}$ refers to the mean squared error on the generalisation set and σ indicates the standard deviation.

Table C.1: Tuning results of varying ensemble size for MTF on Abalone for the first 15 values of Ensemble size.

Ensemble size	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
1	6.636e-03	2.660e-03	7.542e-03	9.630e-03
2	6.106e-03	8.084e-04	6.169e-03	2.463e-03
3	6.052e-03	8.801e-04	7.022e-03	5.713e-03
4	7.151e-03	5.392e-03	7.198e-03	6.311e-03
5	6.625e-03	3.447e-03	6.836e-03	4.740e-03
6	6.353e-03	2.407e-03	6.539e-03	3.699e-03
7	6.171e-03	1.885e-03	7.569e-03	7.471e-03
8	6.105e-03	1.491e-03	6.453e-03	3.281e-03
9	6.236e-03	1.995e-03	6.439e-03	3.355e-03
10	6.158e-03	1.619e-03	6.168e-03	2.564e-03
11	6.066e-03	1.291e-03	6.050e-03	2.124e-03
12	5.980e-03	1.073e-03	5.879e-03	1.465e-03
13	5.957e-03	9.348e-04	6.098e-03	2.122e-03
14	5.947e-03	8.078e-04	6.016e-03	1.814e-03
15	5.905e-03	7.591e-04	5.961e-03	1.627e-03

Table C.2: Tuning results of varying ensemble size for MTF on Abalone for the values 15 to 30 of Ensemble size.

Ensemble size	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
16	5.933e-03	8.061e-04	5.956e-03	1.572e-03
17	5.876e-03	7.114e-04	5.911e-03	1.420e-03
18	5.840e-03	6.412e-04	5.935e-03	1.537e-03
19	5.845e-03	6.413e-04	6.076e-03	2.245e-03
20	5.797e-03	5.809e-04	6.035e-03	2.320e-03
21	5.735e-03	5.208e-04	6.027e-03	2.174e-03
22	5.727e-03	5.106e-04	6.101e-03	2.439e-03
23	5.707e-03	4.645e-04	6.137e-03	2.404e-03
24	5.707e-03	4.733e-04	6.191e-03	2.624e-03
25	5.704e-03	4.518e-04	6.145e-03	2.449e-03
26	5.683e-03	4.229e-04	6.111e-03	2.224e-03
27	5.682e-03	3.972e-04	6.115e-03	2.178e-03
28	5.665e-03	3.728e-04	6.103e-03	2.062e-03
29	5.663e-03	3.695e-04	6.064e-03	1.921e-03
30	5.666e-03	3.951e-04	6.139e-03	2.132e-03

Table C.3: Tuning results of varying ensemble size for MTF on Auto-MPG.

Ensemble size	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
1	2.098e-02	7.727e-02	7.792e-03	4.987e-03
2	8.826e-03	1.859e-02	1.836e-02	6.745e-02
3	6.338e-03	8.167e-03	1.146e-02	3.115e-02
4	5.883e-03	5.328e-03	9.019e-03	1.783e-02
5	5.316e-03	3.401e-03	7.865e-03	1.191e-02
6	4.970e-03	2.412e-03	6.793e-03	8.405e-03
7	4.738e-03	1.751e-03	6.401e-03	6.514e-03
8	4.588e-03	1.393e-03	6.128e-03	5.236e-03
9	4.505e-03	1.144e-03	5.946e-03	4.442e-03
10	4.390e-03	9.037e-04	5.701e-03	3.777e-03
11	4.328e-03	7.212e-04	5.513e-03	3.344e-03
12	4.274e-03	6.619e-04	5.351e-03	3.056e-03
13	4.237e-03	5.923e-04	5.231e-03	2.855e-03
14	4.205e-03	5.356e-04	5.160e-03	2.668e-03
15	4.168e-03	4.810e-04	5.109e-03	2.513e-03
16	4.145e-03	4.376e-04	5.059e-03	2.379e-03
17	4.123e-03	4.056e-04	5.024e-03	2.302e-03
18	4.113e-03	3.773e-04	5.015e-03	2.266e-03
19	4.086e-03	3.572e-04	4.934e-03	2.184e-03
20	4.063e-03	3.348e-04	4.884e-03	2.158e-03
21	4.047e-03	3.207e-04	4.862e-03	2.100e-03
22	4.034e-03	3.065e-04	4.862e-03	2.063e-03
23	4.035e-03	3.007e-04	4.834e-03	2.020e-03
24	4.018e-03	2.836e-04	4.813e-03	1.986e-03
25	4.014e-03	2.824e-04	4.813e-03	1.966e-03
26	4.009e-03	2.733e-04	4.770e-03	1.981e-03
27	4.002e-03	2.710e-04	4.760e-03	1.949e-03
28	3.996e-03	2.676e-04	4.748e-03	1.950e-03
29	3.993e-03	2.592e-04	4.736e-03	1.960e-03
30	4.005e-03	2.705e-04	4.727e-03	1.946e-03
31	4.003e-03	2.652e-04	4.713e-03	1.908e-03
32	4.000e-03	2.636e-04	4.699e-03	1.902e-03
33	3.993e-03	2.606e-04	4.687e-03	1.874e-03
34	3.985e-03	2.581e-04	4.671e-03	1.854e-03
35	3.984e-03	2.566e-04	4.669e-03	1.846e-03
36	3.979e-03	2.573e-04	4.663e-03	1.847e-03
37	4.023e-03	3.305e-04	4.777e-03	1.961e-03
38	4.014e-03	3.229e-04	4.763e-03	1.942e-03
39	4.011e-03	3.146e-04	4.898e-03	2.069e-03
40	4.007e-03	3.053e-04	4.882e-03	2.044e-03

Table C.4: Tuning results of varying ensemble size for MTF on Boston Housing.

Ensemble size	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
1	5.625e-01	3.025e+00	1.059e-02	5.311e-03
2	7.917e-01	2.390e+00	1.720e+00	6.587e+00
3	3.592e-01	1.058e+00	7.615e-01	2.897e+00
4	2.320e-01	6.180e-01	4.302e-01	1.621e+00
5	1.518e-01	3.951e-01	2.771e-01	1.036e+00
6	1.193e-01	2.773e-01	1.948e-01	7.213e-01
7	1.460e-01	3.495e-01	1.586e-01	5.314e-01
8	1.931e-01	5.402e-01	1.228e-01	4.068e-01
9	2.788e-01	7.831e-01	9.820e-02	3.205e-01
10	2.260e-01	6.322e-01	8.029e-02	2.577e-01
11	1.919e-01	5.215e-01	7.062e-02	2.111e-01
12	1.637e-01	4.378e-01	6.080e-02	1.777e-01
13	1.412e-01	3.731e-01	5.282e-02	1.501e-01
14	1.610e+00	8.116e+00	4.666e-02	1.286e-01
15	1.036e+01	4.918e+01	1.367e-01	5.284e-01
16	9.105e+00	4.324e+01	1.209e-01	4.646e-01
17	8.062e+00	3.831e+01	1.080e-01	4.117e-01
18	7.201e+00	3.417e+01	9.714e-02	3.671e-01
19	6.476e+00	3.067e+01	8.762e-02	3.295e-01
20	5.845e+00	2.768e+01	7.955e-02	2.972e-01
21	1.079e+01	3.840e+01	8.090e-02	2.717e-01
22	9.967e+00	3.496e+01	7.393e-02	2.471e-01
23	9.120e+00	3.198e+01	6.816e-02	2.263e-01
24	8.385e+00	2.940e+01	6.332e-02	2.079e-01
25	7.550e+00	2.651e+01	5.878e-02	1.916e-01
26	9.510e+00	2.750e+01	5.484e-02	1.772e-01
27	8.818e+00	2.549e+01	5.121e-02	1.645e-01
28	8.206e+00	2.370e+01	4.879e-02	1.529e-01
29	1.562e+01	4.759e+01	4.546e-02	1.425e-01
30	1.460e+01	4.447e+01	4.283e-02	1.332e-01
31	1.367e+01	4.164e+01	4.056e-02	1.247e-01
32	1.284e+01	3.908e+01	6.197e-02	1.700e-01
33	1.208e+01	3.675e+01	6.284e-02	1.605e-01
34	1.138e+01	3.462e+01	5.958e-02	1.512e-01
35	1.525e+01	3.972e+01	5.536e-02	1.421e-01
36	1.442e+01	3.754e+01	5.255e-02	1.341e-01
37	1.357e+01	3.548e+01	5.010e-02	1.269e-01
38	1.282e+01	3.364e+01	5.013e-02	1.216e-01
39	1.235e+01	3.186e+01	4.773e-02	1.153e-01
40	1.174e+01	3.028e+01	4.618e-02	1.095e-01

Table C.5: Tuning results of varying ensemble size for MTF on California Housing.

Ensemble size	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
1	1.759e-02	3.331e-03	2.212e-02	1.165e-02
2	2.039e-02	2.039e-02	2.594e-02	1.864e-02
3	1.739e-02	9.006e-03	2.244e-02	1.179e-02
4	1.636e-02	5.009e-03	1.889e-02	7.794e-03
5	1.599e-02	3.485e-03	1.726e-02	5.565e-03
6	1.581e-02	2.559e-03	1.827e-02	8.359e-03
7	1.544e-02	1.808e-03	1.789e-02	6.445e-03
8	1.522e-02	1.363e-03	1.783e-02	5.902e-03
9	1.503e-02	1.058e-03	1.738e-02	5.255e-03
10	1.489e-02	8.582e-04	1.709e-02	4.301e-03
11	1.482e-02	8.349e-04	1.683e-02	3.825e-03
12	1.472e-02	6.919e-04	1.700e-02	4.074e-03
13	1.471e-02	6.047e-04	1.678e-02	3.792e-03
14	1.464e-02	5.331e-04	1.678e-02	3.998e-03
15	1.461e-02	5.900e-04	1.649e-02	3.515e-03
16	1.458e-02	5.426e-04	1.654e-02	3.655e-03
17	1.457e-02	4.952e-04	1.645e-02	3.673e-03
18	1.447e-02	4.120e-04	1.667e-02	3.944e-03
19	1.448e-02	4.014e-04	1.670e-02	4.133e-03
20	1.446e-02	3.896e-04	1.643e-02	3.768e-03
21	1.444e-02	3.884e-04	1.642e-02	3.855e-03
22	1.446e-02	3.459e-04	1.651e-02	3.965e-03
23	1.457e-02	7.220e-04	1.660e-02	4.140e-03
24	1.456e-02	6.810e-04	1.648e-02	3.963e-03
25	1.456e-02	6.708e-04	1.691e-02	5.606e-03
26	1.454e-02	5.547e-04	1.688e-02	5.409e-03
27	1.451e-02	5.251e-04	1.710e-02	5.983e-03
28	1.512e-02	3.549e-03	1.685e-02	5.468e-03
29	1.505e-02	3.301e-03	1.678e-02	5.155e-03
30	1.498e-02	3.085e-03	1.678e-02	5.257e-03

Table C.6: Tuning results of varying ensemble size for MTF on CASP.

Ensemble size	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
1	3.664e-01	5.559e-01	2.627e+00	1.285e+01
2	2.583e-01	3.228e-01	6.190e-01	2.738e+00
3	3.631e-01	4.483e-01	5.605e-01	2.206e+00
4	4.197e-01	6.132e-01	3.445e-01	1.247e+00
5	3.118e-01	4.071e-01	3.066e-01	8.451e-01
6	2.565e-01	2.964e-01	2.342e-01	5.738e-01
7	2.191e-01	2.339e-01	1.982e-01	4.206e-01
8	2.048e-01	2.026e-01	2.528e-01	5.468e-01
9	1.686e-01	1.574e-01	2.859e-01	5.314e-01
10	1.574e-01	1.577e-01	2.409e-01	4.307e-01
11	1.415e-01	1.299e-01	2.383e-01	3.832e-01
12	1.269e-01	1.075e-01	2.159e-01	3.387e-01
13	1.217e-01	9.179e-02	1.961e-01	2.844e-01
14	1.152e-01	7.773e-02	1.861e-01	2.592e-01
15	1.074e-01	6.801e-02	1.933e-01	2.908e-01
16	1.032e-01	6.254e-02	1.965e-01	2.777e-01
17	9.743e-02	5.618e-02	1.891e-01	2.599e-01
18	9.461e-02	5.073e-02	1.773e-01	2.299e-01
19	8.960e-02	4.622e-02	1.503e-01	1.991e-01
20	9.031e-02	4.714e-02	1.400e-01	1.701e-01
21	8.428e-02	3.803e-02	1.242e-01	1.529e-01
22	8.317e-02	3.578e-02	1.271e-01	1.654e-01
23	7.965e-02	3.276e-02	1.351e-01	1.849e-01
24	8.004e-02	3.633e-02	1.293e-01	1.799e-01
25	8.103e-02	3.551e-02	1.241e-01	1.793e-01
26	7.876e-02	3.245e-02	1.214e-01	1.661e-01
27	7.814e-02	2.862e-02	1.178e-01	1.478e-01
28	7.763e-02	2.729e-02	1.083e-01	1.145e-01
29	7.530e-02	2.510e-02	1.007e-01	1.010e-01
30	7.381e-02	2.328e-02	1.142e-01	1.220e-01

Table C.7: Tuning results of varying ensemble size for MTF on Elevators.

Ensemble size	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
1	2.006e-03	2.727e-03	1.230e-03	8.217e-05
2	5.563e-03	2.093e-02	1.132e-03	6.643e-05
3	5.802e-03	1.703e-02	1.092e-03	6.779e-05
4	3.897e-03	9.556e-03	1.168e-03	5.258e-04
5	3.013e-03	6.076e-03	1.393e-03	1.068e-03
6	2.440e-03	4.223e-03	1.555e-03	1.245e-03
7	2.395e-03	3.432e-03	1.918e-03	3.025e-03
8	2.501e-03	3.011e-03	1.742e-03	2.531e-03
9	2.487e-03	2.515e-03	1.593e-03	2.003e-03
10	2.420e-03	2.249e-03	1.500e-03	1.619e-03
11	2.261e-03	1.917e-03	1.417e-03	1.336e-03
12	2.112e-03	1.635e-03	1.363e-03	1.123e-03
13	1.924e-03	1.371e-03	1.324e-03	9.568e-04
14	1.790e-03	1.181e-03	1.272e-03	7.613e-04
15	1.743e-03	1.023e-03	1.281e-03	6.910e-04
16	1.653e-03	8.947e-04	1.282e-03	6.160e-04
17	1.643e-03	8.459e-04	1.253e-03	5.470e-04
18	1.591e-03	7.547e-04	1.239e-03	4.856e-04
19	1.695e-03	8.764e-04	1.361e-03	8.680e-04
20	1.741e-03	1.240e-03	1.328e-03	7.819e-04
21	1.696e-03	1.116e-03	1.300e-03	7.090e-04
22	1.700e-03	1.010e-03	1.275e-03	6.468e-04
23	1.640e-03	9.218e-04	1.255e-03	5.921e-04
24	1.590e-03	8.441e-04	1.237e-03	5.451e-04
25	1.527e-03	7.645e-04	1.220e-03	5.024e-04
26	1.484e-03	7.057e-04	1.201e-03	4.647e-04
27	1.461e-03	6.618e-04	1.186e-03	4.320e-04
28	1.423e-03	6.150e-04	1.174e-03	4.022e-04
29	1.391e-03	5.728e-04	1.163e-03	3.759e-04
30	1.363e-03	5.340e-04	1.154e-03	3.525e-04

Table C.8: Tuning results of varying ensemble size for MTF on Friedman artificial domain.

Ensemble size	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
1	1.615e-03	3.828e-04	1.664e-03	4.212e-04
2	1.367e-03	1.830e-04	1.401e-03	2.047e-04
3	1.268e-03	9.960e-05	1.298e-03	1.074e-04
4	1.230e-03	8.186e-05	1.260e-03	8.765e-05
5	1.209e-03	6.644e-05	1.236e-03	7.432e-05
6	1.196e-03	6.262e-05	1.223e-03	6.789e-05
7	1.195e-03	6.114e-05	1.221e-03	6.659e-05
8	1.190e-03	5.435e-05	1.216e-03	6.106e-05
9	1.191e-03	5.347e-05	1.217e-03	5.972e-05
10	1.184e-03	5.204e-05	1.210e-03	6.185e-05
11	1.186e-03	5.005e-05	1.212e-03	6.093e-05
12	1.181e-03	4.578e-05	1.207e-03	5.687e-05
13	1.176e-03	4.540e-05	1.202e-03	5.770e-05
14	1.172e-03	4.223e-05	1.198e-03	5.398e-05
15	1.169e-03	4.077e-05	1.194e-03	5.307e-05
16	1.169e-03	4.031e-05	1.195e-03	5.216e-05
17	1.167e-03	3.863e-05	1.192e-03	5.016e-05
18	1.162e-03	3.739e-05	1.187e-03	4.893e-05
19	1.160e-03	3.595e-05	1.185e-03	4.694e-05
20	1.159e-03	3.405e-05	1.184e-03	4.499e-05
21	1.159e-03	3.197e-05	1.184e-03	4.322e-05
22	1.156e-03	3.298e-05	1.180e-03	4.266e-05
23	1.157e-03	3.179e-05	1.181e-03	4.016e-05
24	1.156e-03	3.024e-05	1.180e-03	3.791e-05
25	1.157e-03	3.148e-05	1.181e-03	3.911e-05
26	1.157e-03	3.152e-05	1.180e-03	3.928e-05
27	1.158e-03	3.145e-05	1.181e-03	4.031e-05
28	1.158e-03	3.017e-05	1.181e-03	3.921e-05
29	1.158e-03	2.911e-05	1.181e-03	3.800e-05
30	1.157e-03	2.945e-05	1.180e-03	3.892e-05

Table C.9: Tuning results of varying ensemble size for MTF on Machine CPU.

Ensemble size	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
1	9.193e-02	4.622e-01	7.587e-03	1.155e-02
2	2.557e-02	1.155e-01	1.242e-02	2.916e-02
3	1.215e-02	5.137e-02	7.721e-03	1.545e-02
4	7.641e-03	2.886e-02	6.539e-03	1.113e-02
5	5.677e-03	1.846e-02	5.746e-03	9.243e-03
6	4.360e-03	1.283e-02	4.777e-03	7.272e-03
7	3.565e-03	9.413e-03	4.252e-03	5.802e-03
8	2.878e-03	7.247e-03	3.992e-03	4.754e-03
9	2.479e-03	5.732e-03	3.746e-03	4.371e-03
10	2.261e-03	4.698e-03	3.658e-03	4.198e-03
11	2.053e-03	3.886e-03	3.520e-03	3.974e-03
12	1.968e-03	3.320e-03	3.635e-03	4.250e-03
13	1.807e-03	2.870e-03	3.445e-03	3.977e-03
14	1.712e-03	2.477e-03	3.217e-03	4.065e-03
15	1.641e-03	2.152e-03	3.168e-03	3.996e-03
16	1.533e-03	1.884e-03	3.115e-03	3.685e-03
17	1.473e-03	1.675e-03	3.057e-03	3.486e-03
18	1.405e-03	1.495e-03	2.931e-03	3.257e-03
19	1.333e-03	1.353e-03	2.915e-03	3.170e-03
20	1.291e-03	1.223e-03	2.969e-03	3.283e-03
21	1.370e-03	1.234e-03	2.897e-03	3.167e-03
22	1.302e-03	1.126e-03	2.783e-03	3.069e-03
23	1.257e-03	1.025e-03	2.740e-03	3.104e-03
24	1.237e-03	9.364e-04	2.759e-03	3.243e-03
25	7.040e-03	3.187e-02	2.641e-03	3.099e-03
26	6.556e-03	2.946e-02	2.566e-03	2.920e-03
27	6.163e-03	2.734e-02	2.639e-03	3.128e-03
28	5.792e-03	2.542e-02	2.566e-03	2.984e-03
29	5.477e-03	2.367e-02	2.582e-03	3.022e-03
30	5.171e-03	2.214e-02	2.557e-03	2.954e-03
31	4.904e-03	2.074e-02	2.521e-03	2.911e-03
32	4.660e-03	1.947e-02	2.477e-03	2.793e-03
33	4.441e-03	1.832e-02	2.490e-03	2.779e-03
34	4.228e-03	1.725e-02	2.500e-03	2.756e-03
35	4.038e-03	1.625e-02	2.481e-03	2.686e-03
36	3.883e-03	1.536e-02	2.511e-03	2.749e-03
37	3.717e-03	1.456e-02	2.500e-03	2.708e-03
38	3.587e-03	1.388e-02	2.546e-03	2.795e-03
39	3.556e-03	1.317e-02	2.566e-03	2.765e-03
40	3.439e-03	1.251e-02	2.553e-03	2.675e-03

Table C.10: Tuning results of varying ensemble size for MTF on MV artificial domain.

Ensemble size	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
1	9.820e-05	4.318e-04	2.615e-05	4.432e-05
2	1.083e-04	3.657e-04	2.010e-05	4.160e-05
3	6.084e-05	1.693e-04	1.507e-05	3.540e-05
4	3.562e-05	9.526e-05	2.388e-03	1.302e-02
5	3.036e-05	6.751e-05	1.532e-03	8.333e-03
6	2.353e-05	4.686e-05	1.073e-03	5.785e-03
7	1.853e-05	3.449e-05	7.893e-04	4.250e-03
8	1.511e-05	2.659e-05	6.026e-04	3.239e-03
9	1.262e-05	2.114e-05	4.777e-04	2.559e-03
10	5.069e-04	2.716e-03	2.734e-03	1.492e-02
11	4.201e-04	2.244e-03	2.260e-03	1.233e-02
12	4.932e-04	2.014e-03	1.900e-03	1.036e-02
13	4.208e-04	1.716e-03	1.620e-03	8.831e-03
14	1.203e-02	5.998e-02	1.397e-03	7.614e-03
15	1.048e-02	5.225e-02	1.217e-03	6.633e-03
16	9.208e-03	4.592e-02	1.072e-03	5.829e-03
17	8.142e-03	4.068e-02	9.502e-04	5.163e-03
18	7.268e-03	3.628e-02	8.479e-04	4.605e-03
19	7.299e-03	3.262e-02	3.540e-03	1.563e-02
20	6.588e-03	2.944e-02	3.195e-03	1.411e-02
21	6.181e-03	2.668e-02	2.910e-03	1.279e-02
22	5.632e-03	2.431e-02	2.652e-03	1.166e-02
23	5.175e-03	2.224e-02	2.427e-03	1.067e-02
24	4.753e-03	2.043e-02	2.230e-03	9.796e-03
25	4.381e-03	1.883e-02	2.056e-03	9.028e-03
26	4.053e-03	1.740e-02	1.901e-03	8.347e-03
27	3.759e-03	1.614e-02	1.763e-03	7.740e-03
28	3.495e-03	1.501e-02	1.640e-03	7.197e-03
29	3.259e-03	1.399e-02	1.529e-03	6.709e-03
30	3.328e-03	1.310e-02	1.429e-03	6.269e-03

Table C.11: Tuning results of varying ensemble size for MTF on Servo.

Ensemble size	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}
1	1.224e-02	7.994e-03	6.367e-02	2.255e-01
2	8.217e-03	2.808e-03	2.913e-02	5.965e-02
3	7.993e-03	5.595e-03	1.788e-02	1.950e-02
4	6.943e-03	3.314e-03	1.632e-02	1.757e-02
5	1.088e-02	2.572e-02	1.489e-02	1.560e-02
6	9.059e-03	1.788e-02	1.356e-02	1.517e-02
7	7.865e-03	1.330e-02	1.348e-02	1.491e-02
8	7.152e-03	1.013e-02	1.335e-02	1.587e-02
9	6.590e-03	8.284e-03	1.321e-02	1.580e-02
10	6.112e-03	6.671e-03	1.294e-02	1.569e-02
11	5.905e-03	5.564e-03	1.272e-02	1.498e-02
12	5.655e-03	4.734e-03	1.235e-02	1.472e-02
13	5.536e-03	4.046e-03	1.223e-02	1.442e-02
14	5.365e-03	3.522e-03	1.217e-02	1.446e-02
15	5.231e-03	3.101e-03	1.220e-02	1.451e-02
16	5.097e-03	2.726e-03	1.214e-02	1.441e-02
17	4.972e-03	2.471e-03	1.202e-02	1.435e-02
18	4.919e-03	2.266e-03	1.199e-02	1.510e-02
19	4.820e-03	2.081e-03	1.169e-02	1.497e-02
20	4.764e-03	1.926e-03	1.159e-02	1.475e-02
21	4.830e-03	1.853e-03	1.173e-02	1.456e-02
22	4.749e-03	1.692e-03	1.166e-02	1.444e-02
23	4.706e-03	1.594e-03	1.171e-02	1.440e-02
24	3.389e-02	1.598e-01	1.186e-02	1.442e-02
25	3.163e-02	1.473e-01	1.176e-02	1.422e-02
26	2.958e-02	1.364e-01	1.194e-02	1.429e-02
27	2.779e-02	1.267e-01	1.176e-02	1.416e-02
28	2.615e-02	1.178e-01	1.166e-02	1.399e-02
29	2.473e-02	1.102e-01	1.157e-02	1.381e-02
30	2.336e-02	1.029e-01	1.162e-02	1.376e-02
31	2.213e-02	9.644e-02	1.160e-02	1.366e-02
32	2.111e-02	9.093e-02	1.156e-02	1.363e-02
33	2.010e-02	8.552e-02	1.160e-02	1.362e-02
34	1.916e-02	8.060e-02	1.162e-02	1.392e-02
35	1.832e-02	7.608e-02	1.156e-02	1.384e-02
36	1.756e-02	7.189e-02	1.150e-02	1.377e-02
37	1.697e-02	6.863e-02	1.154e-02	1.382e-02
38	1.629e-02	6.508e-02	1.140e-02	1.370e-02
39	1.571e-02	6.187e-02	1.139e-02	1.376e-02
40	1.512e-02	5.878e-02	1.142e-02	1.408e-02