UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

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/ Engineering Management) in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof. A.P. Engelbrecht

November 2021

Acknowledgements

TBD

Declaration

Date: 18 November 2021

Abstract

English

The English abstract.

In this paper, an ensemble approach that reduces the high variance error exhibited by model trees that comprise multi-variate nonlinear models as well as increase their overall robustness is conceptualised, tuned for, and evaluated against competing regression models on ten separate benchmarking datasets. The ensemble, labeled 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 randomised splitting feature spaces during tree induction, sufficiently decorrelates the base learners within the ensemble, thereby reducing the variance error of MTF compared to that of a single model tree whilst retaining the favourable low bias error that model trees exhibit. The multi-variate nonlinear models that dictate the output enables MTF to produce approximations of highly nonlinear data whilst the addition of ensembling methods passively combat overfitting brought forth from the increased model complexity, compared to a previous implementation of GASOPE within a tree structure which is shown to exhibit overfitting in specific cases.

Afrikaans

Die Afrikaanse uittreksel.

Contents

List of Figures

List of Tables

Nomenclature

Variables and functions

$p(x)$	Probability density function with respect to variable x.
$P(A)$	Probability of event A occurring.
TBD	

Acronyms and abbreviations

MTF	model tree forest
TBD	

Chapter 1

Introduction

1.1. Motivation

TBD

1.2. Objectives

TBD

1.3. Contribution

TBD

1.4. Thesis Outline

TBD

Chapter 2

Literature Review

This chapter presents the background of model trees and 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 an extension over decision trees. Hence, decision trees are first discussed in this chapter. Understanding 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 to decision trees and the induction thereof. The bias-variance tradeoff and how the two ensemble methods, *bagging* and *boosting*, address the tradeoff is discussed in Section 2.2. Section 2.3 presents the differences a model tree has over a decision tree as well as a general discussion on the performance of the M5 model tree. Two different model trees, HTL and GPMCC, which incorporate non-linear models at their leaf nodes are discussed in Section 2.4. Finally, two existing model tree ensembles, RMT and MTE, which incorporate M5 model trees are introduced in Section 2.5

2.1. Decision trees

In this section the reader is introduced to the concept of decision trees in the context of machine learning. Section 2.1.1 discusses decision trees in the application of classification and regression. Section 2.1.2 introduces CART and the portrayal of a decision tree. Finally, Section 2.1.3 enumerates the induction process employed by CART to grow both regression and/or classification trees.

2.1.1. Introduction to decision trees

Decision trees are tree-like predictive models used in machine learning on both classification or 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 which the decision tree is tasked with predicting. 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 values are qualitative as they take on labelled values and numerical values are quantitative as they take on real values.

A decision tree organises data recursively through a hierarchical representation of tests on input features, giving it the appearance of a tree data structure. Through a series of tests, present in the nodes of the tree structure, the data is divided with branches representing the different outcomes of a test. Alternatively, decision trees are expressed as a collection of if-else statements to model decision boundaries in data. Decision trees were inspired by the *divide-and-conquer* paradigm, which entails recursively breaking an intricate problem down into smaller sub-problems which each can be solved more simply.

Before the conception of decision trees, many of the statistical techniques used for classification problems were developed with small datasets in mind [4]. Furthermore, the variables were all of the same type 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 [4]. The application of these simple models to larger datasets, with the assumption that the data retains its homogeneous structure, is flawed.

Datasets are not only subject to being large, but complex as well. The complexity of data is influenced by several factors: missing values, noise, outliers, high dimensionality, mixtures of data types, non-parametric distribution[1] and non-homogeneity[2]. "The curse of dimensionality" states the more features present, the higher the variance of the data points are. Hence the data is sparser and more spread out. The dimensionality of data can be reduced, however the accompanying drawbacks are unfavourable. 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 [4], which brought rise to the conceptualisation of decision tree induction. The defining characteristic of decision trees is its ability to not only produce satisfactory results but also give the user thoughtful information and insight into the data. For example, in classification problems, a decision tree helps uncover the predictive nature of a problem and helps the user better understand how specific features contribute to the outcome that is being observed [4].

[1]Data which does not fit a known distribution.

[2]Different relationships hold between variables within different parts of the decision space

2.1.2. The CART methodology

One of the earliest documented work on the theory of decision tree induction was published in 1984 [4]; Breiman et al. was inspired by the deficiencies of existing tree-structured classifiers at the time to develop an improved methodology which provided a more flexible and accurate solution. Breiman labelled his proposed decision tree induction algorithm as CART (Classification and Regression Trees). CART grows 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, capable of classifying an individual's gender based on two input features: weight and height. Nodes X_1 and X_3 each represents a test. Tests, also referred to as splits, comprise two parts: a feature on which the test is performed and a value indicating 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 the label of the output class. These nodes are referred to as *leaf* nodes, as they terminate all possible *paths* (sequences of nodes) that can be followed in the tree from the root node to any leaf node.

Each node of the decision tree encapsulates a subset of the dataset; the data subset of a node is described by the tests that lead up to the node. When the data subsets each leaf node of the decision tree denotes is combined, the entirety of the instance space is accounted for [18]. 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. Therefore a tree is regarded as being *shallow* if its depth is small.

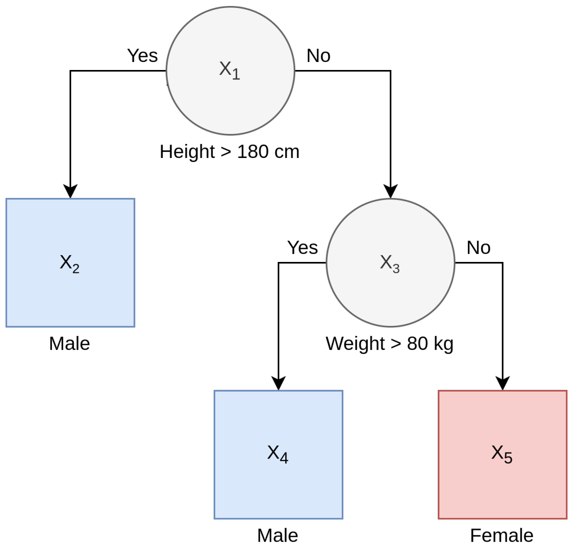


Figure 2.1: A hypothetical two-class tree-structured classifier.

2.1.3. CART's induction algorithm

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

1. Determining the tests.
2. Evaluating stopping rules.
3. Assigning class labels to each leaf.

Starting 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 the ideal 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 higher the number of classes present in the subset is. CART produces binary splits from the value of a single input feature only. 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 $F \times N$ splits, where F is the number of features and N the number of observations in the dataset. A split is therefore proposed on every feature and the corresponding feature value for each observation in the dataset. The best performing split is selected as the one split which minimises the impurity function.

Once a node is reached where no significant decrease in impurity is produced, the node is declared a leaf node. This is an example stopping rule; stopping rules can also be explicit, such as fixing the depth of a tree. The class making up the majority of each leaf is assigned as the leaf's label. The decision tree growing sequence, i.e. the induction algorithm, is thus completed.

In the application of CART's induction algorithm to regression problems, little changes. The induction algorithm proceeds to partition the dataset into subsets via a series of tests on the input features. A different metric in place of impurity is incorporated; the expected reduction in either variance or absolute deviation for a proposed split is calculated. The splitting function of regression trees maximises the reduction in either variance or absolute deviation.

Changes are also present in the leaf nodes; instead of predicting which class an observation belongs to, the decision tree now predicts a numerical value associated with a class[3]. The numerical value of a leaf node is calculated as the sum of squares over the target values for all observations in the subset of data denoted by the path to the leaf node.

[3]As apposed to a classification tree which predicts a categorical value to define a class.

Fundamentally, the numerical value predicted by the regression tree is a dependent variable based on a multitude of independent variables[4]. 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 $y = sin(x)$ for $x \in (0, 2\pi)$, with the target value y being dependent on the input feature x.

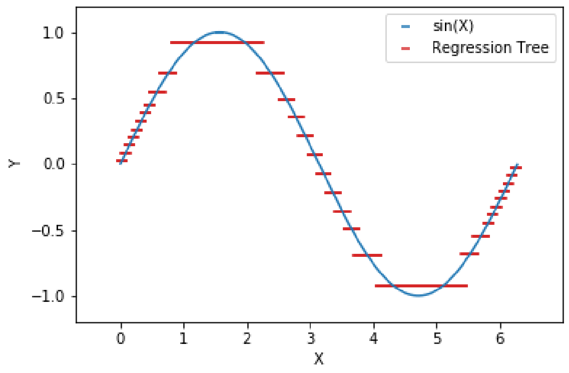


Figure 2.2: A regression tree's piecewise discrete approximation (red) of a continuous sinusoidal function (blue).

The advantages CART's induction algorithm had over statistical models at the time of its conception included [4]:

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

2. Once grown, 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 observations.

4. The most beneficial information for identification is incorporated at each node, forming a subset of the dataset. Hence, automatic dimensionality reduction is effectively performed on the dataset.

5. CART proved to be quite robust when subjected to outliers and missclassified observations.

[4]The independent variables represent the input features and the dependent variable the target value

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 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 by discussing the bias-variance tradeoff and how it causes 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 discuss two popular ensemble methods and how each addresses the bias-variance tradeoff.

2.2.1. Overfitting and underfitting

The bias-variance dilemma stems from the degree of inaccuracy present in machine learning models. Both bias and variance are degrees of prediction error present in the model, negatively affecting its 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 mitigating the prediction error, rather than trying to remove it altogether. To better mitigate the prediction error, its two components (bias and variance) must be examined.

Bias error is defined as the average difference in prediction a model makes from the target value; variance error is defined as the amount of variability or spread a model is subject to when predicting a certain data value. 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 observations, shown in blue.

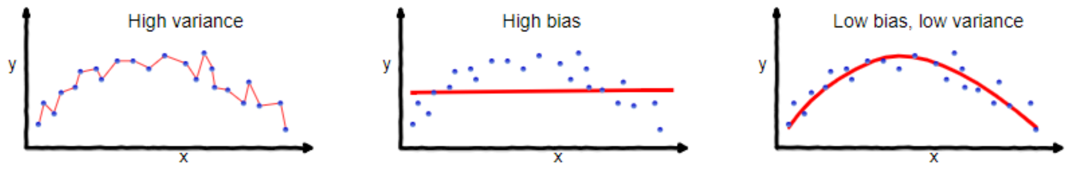


Figure 2.3: Three separate models' predicted output (red) and target output (blue) of a simple regression problem in a two-dimensional space. Reproduced from [21]

Models that are simple in their composition, such as the second model in Figure 2.3, are incapable of capturing any intricate patterns in the training data. These models produce an oversimplified interpretation of the data. Testing overly simple models on unseen data likely results in a high bias error [14]. A model with high bias error *underfits* the training data i.e. it cannot capture the complex relationships within the data. Hence, a model that underfits will produce poor prediction accuracy.

In contrast, models that incorporate complex algorithms are more vulnerable to noise in the training data. Subsequently a large variance error is present and the model referred to as *overfitting* the training data. Although the prediction error on the training data of complex models may be low, such as the first model in Figure 2.3, the model still generalises poorly to unseen data [11]. The overly complex model fallaciously captures noisy patterns in the training data that are not associated with the desired output [14]. Hence, poor prediction accuracy is obtained, similar to models with large bias error.

Thus producing favourable prediction accuracy requires 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 increasing in opposite directions. This tradeoff between bias and variance is portrayed in Figure 2.4. Models are designed with the strategy of minimising the tradeoff and finding the point of optimal balance between the two errors.

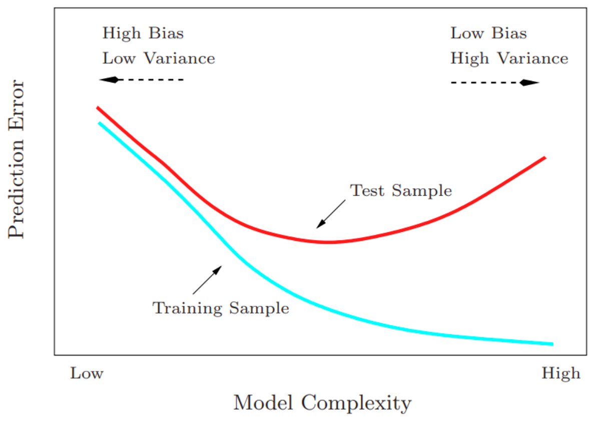


Figure 2.4: The change in a model's prediction error on the training set (red) and generalisation set (blue) as the complexity of the model is varied. Reproduced from [12].

2.2.2. In the context of decision trees

Decision trees are classified as supervised learning models, as they are trained with the task of mapping an observed output variable to multiple input variables. Unsupervised learning models are instead tasked with clustering unlabelled data into homogeneous regions. In the case of a supervised learning model, the criterion used in assessing its prediction accuracy is the ability of the model to generalise to unseen data [11].

When Breiman started his initial work on decision trees, it was clear that arguably the biggest shortcoming was a product of the high variance present in decision trees [4]. 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 adding additional decision nodes. Hence, decision trees are prone to capturing noise in the training set. This meant that trees were prone to overfitting and require something to remedy this.

The answer Breiman put forth was a fundamental shift in focus. Instead of adjusting the stopping criterion to combat overfitting, the tree was grown with lenient stopping rules and once fully grown, selectively *pruned*. 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 removed if warranted. Each node along the path is temporarily pruned to a leaf node and this temporary pruned tree's performance on unseen data compared to that of the original unpruned tree. If the tree's prediction error is found to have decreased, the node remains pruned Otherwise, the tree is reverted to its state before the concerning node was pruned. Pruning proved to lower the variance in classification trees and significantly improve its performance [4].

As for regression trees, the tradeoff is more intricate. Because of its tree structure, regression trees are prone to overfitting. 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. This is due to the splits being highly sensitive to changes in data, especially in the case of noisy data. Therefore regression trees are pruned similarly to classification trees to reduce the variance error.

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 produces a better result than the result an individual model is capable of producing.

2.2.3. Bagging

One of the simplest tree ensemble methods is bootstrap aggregation, referred to as *bagging* [5]. Several decision trees are induced in parallel, each on a subset of the training data. A subset is determined through random selection. For regression applied bagging, the final prediction of the ensemble is derived through 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.

The randomness in the data subsets on which the decision trees are induced contributes to minimising the variance of the collective model. The total variation in the prediction a bagged model produces for a given set of observations[5] is decomposed into two terms and

[5]The set of observations is assumed to be a set of independent and identically distributed random variables

expressed as [14]

$$Var(\boldsymbol{x}) = \rho(\boldsymbol{x})\sigma^2(\boldsymbol{x}) + \frac{(1-\rho(\boldsymbol{x}))\sigma^2(\boldsymbol{x})}{N} \tag{2.1}$$

where ρ is the sampling correlation between the prediction of any two decision trees in the ensemble, σ is the sampling variance of any single, randomly drawn, decision tree and N is the number of decision trees in the ensemble. Finally, $\boldsymbol{x}$ represents a set of observations. Due to the injection of randomness in the tree induction algorithm, it can be assumed that trees are divergent from one another, i.e. $\rho(\boldsymbol{x}) < 1$. The stronger random effects are the more ρ tends to 0, reducing Equation 2.1 to only $\frac{\sigma^2(\boldsymbol{x})}{N}$. It is evident that as the number of decision trees in the ensemble is increased, the total variance decreases further. The variance of the ensemble is thus strictly less than the variance of an individual tree. Furthermore, combining randomised models does not affect the bias error [12]. Therefore, decision trees are ideal for bagging as they exhibit a very low bias. Bagging decision trees produces a model that is both low in bias and variance, improving its accuracy significantly over that of a single tree.

Building on the success achieved by bagging, Breiman conceptualised the *random forest* method as an extension over bagging [6]. In a random forest, each tree is once again trained on a randomly selected subset of the data. The difference random forest incorporates is a change to the induction algorithm of the individual trees that make up the ensemble to strengthen random effects. 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 splits on the randomly selected features. The decision trees within the ensemble are further decorrelated as a result, decreasing the collective model's variance. Random forest has the benefit of performing well on higher dimensionality data, due to the dimension reduction-like induction algorithm [6]. Note that individual trees are left unpruned in random forest. Therefore, each decision tree within the ensemble is *fully grown*, retaining the low bias of decision trees. Random forests were shown to outperform many competing classifiers whilst being robust to overfitting [6].

2.2.4. Boosting

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 in mind and referred to as weak learners. A weak learner's error rate is only slightly better than a random guess [12]. In the context of a decision tree, an example strategy of inducing a high bias decision tree is a growing a shallow tree. Weak learners are incapable of individually modelling complex observed relationships. However, as a collective model, weak learners form a strong learner capable of adequately modelling more complicated patterns in data.

Boosting sequentially induces a weak learner on repeatedly modified versions of the data. Data is modified through weights that are assigned to each training observation. Weights represent the probability of an observation being chosen as part of a subset on which a weak learner is induced [15]. After each additional weak learner is added to the ensemble, larger weights are assigned to the observations which have the highest contribution to the prediction error. Each successive weak learner thereby prioritises improving the prediction error of the larger weighted observations. Through this sequential optimisation, the collective model's bias is reduced compared to that of a single weak learner. Once induction of the entire ensemble is complete, weights are assigned to the outputs of each weak learner based on their prediction error.

Bagging and boosting has now been shown to have similarities, such as subsampling of the dataset but these methods 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 over decision trees conceptualised to produce increased performance on regression problems. The section is outlined as follows. Section 2.3.2 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 which the M5' model tree incorporates is 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

Breiman's early work on improving the robustness of decision trees showed great success concerning the application of decision trees to classification problems [4]. However, Breiman's focus on classification trees [5, 6] meant that regression applications of decision trees still left much to be desired. A big drawback of regression trees remained the discontinuous output it produced, as shown in Figure 2.2. This meant that regression trees, developed through the CART methodology, 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 data, whilst neural networks do not provide the user with an insight into the relationship of the data, due to its problem of opacity [26].

There was still a need for a learning model capable of performing non-linear regression and providing its user with insights into the relationship among descriptive features that results in specific predictions. Model trees were first developed by Quinlan in 1992 as an extension over Breiman's regression trees with the fundamental difference that Quinlan's M5 model tree's leaf nodes are not limited to having a constant prediction value [19]. 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 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 producing discontinuities and thus better approximate the non-linear function $y = sin(x)$. The linear models also allows profiling, i.e. describing the conditions on the input features that will result in a specific linear trend.

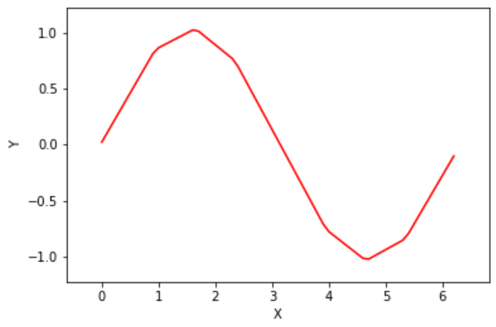


Figure 2.5: The continuous piecewise linear approximation of a sinusoidal wave using a model tree.

2.3.2. M5 model tree induction

The induction of an M5 model tree is quite similar to that of CART, as M5 model tree induction also employs a greedy approach[6]. 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

[6]A greedy approach is a heuristic approach that makes choices resulting in local optima, with the intentions of reaching an global optimum.

induction algorithm selects the split which maximises an expected error reduction at a node, defined by the following formula [19]

$$\Delta error = sd(T) - \sum_{i} \frac{|T_i|}{|T|} \times sd(T_i) \tag{2.2}$$

where T represents the collection of training samples for the data encapsulated by the node, and $sd(T)$ is the standard deviation of the target values in T. Every potential split is evaluated through the subset of samples it produces. T_i denotes the subset of samples that belong to outcome i of a split and $std(T_i)$ the standard deviation of the target values in T_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 grown, linear models are constructed for each node. The reason why linear models are constructed for non-leaf in addition to leaf nodes is to accommodate for *pruning*. The linear model of a leaf node's 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 features defined within the splits of that node and its subtree nodes. This ensures that the performance of a non-leaf node linear model is compared to the performance of the subtree on a level playing field during the pruning process [19]. 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, leaving only a constant behind.

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, encapsulated by the node, is withdrawn from the model and the specified node 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 sample. Thereby, 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 with at the leaf node, the predicted value is passed on along the path to the root node and adjusted at each node such that it better resembles each predicted value produced by the linear model of a given node on the path. The formula used to adjust the predicted value, y_i, at the i^{th} node along the path to the root is

$$y_i = \frac{ny_{i-1} + ky}{n + k} \tag{2.3}$$

where n is the number of training samples at the previous node along the path, y_{i-1} is the adjusted prediction from the previous node, y is the predicted value of node i, and k is the specified numeric smoothing constant. The value of y_i produced by the root node at the end of the path is the predicted value of the model tree given the test sample.

2.3.3. Early M5 model tree performance

Quinlan compared the M5 model tree to MARS (multivariate adaptive regression spline), which was a popular regression model that proved effective on non-linear data at the time of the M5 model tree's conception [19]. MARS is similar to M5 as it is also non-parametric[7] and utilises piecewise linear approximation. 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 [19]. The results Quinlan published proved 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. One important aspect of the M5' model tree is that it specifies the linear model implemented in a node. The linear model, named the $(k+1)$-parameter model, is denoted by the formula [26]

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

with x_k representing the k^{th} input feature, w_k 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 the simplification step of M5 (the simplification step of M5 removes terms from the linear model) compromising to the size of the model tree. Sometimes 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 where available [26].

[7]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.

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 non-linear relationships between its features [1]. The M5 model tree's susceptibility to initially overfitting noisy patterns can be attributed to the already high variance exhibited by tree based models, together with the model tree's increased complexity over the regression tree. To combat this, the model tree is pruned during induction. However, excessively post-pruning[8] 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 non-linear patterns. Within the tree multiple leaf nodes, each with linear models, are pruned away and consequently these linear models simplified into one. This is referred to as over-pruning and has been shown to have negative affects on a model tree's performance [1].

A study published in 2016 [13] hypothesised that the reason for the M5 model tree's inferior results on highly non-linear data is its piecewise linear functions' limited capability to fit the training data. In this study, only six quantitative water quality parameters were used in predicting the monthly chemical oxygen demand of a river. The relationship between these parameters is highly non-linear. 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 the M5 model trees were designed with large datasets in mind, making it the preferred choice for problems with a large number of input parameters [19].

Quinlan recommended researching the application of non-linear models at the leaf nodes [19] to improve the ability of a model tree to adequately capture highly non-linear patterns present in complex datasets. This would also allow for a model tree to be grown smaller, as a single non-linear model can approximate the same function which demands multiple linear models to approximate. The number of splits of the training data is reduced to produce a smaller tree, and should therefore decrease variance in the model making it 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 non-linear 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 in expecting that the increase in complexity of a model will guarantee an improvement in its performance. More often than not a simpler approach is the favoured solution to a problem [3]. These factors are hypothesised as contributing to the lack of abundant research on model trees that incorporate non-linear models.

[8]Post-pruning refers to pruning of the decision tree after it has been grown to overfit on the data, as opposed to stopping the growth of the tree prematurely to effectively prune it.

2.4. Non-linear solutions

This section discusses two existing model trees that incorporate non-linear models within their leaf nodes. For the model tree forest that will later be proposed in this paper it is important to first evaluate the performance of existing model trees that comprise non-linear models. Inducing model tree comprising non-linear models is already computationally expensive, and growing an entire ensemble may be inapproachable. In Section 2.4.1 the non-linear model present in PLT is broken down and the performance thereof examined. Next, Section 2.4.2 discusses GPMCC, a model tree which is induced via a genetic programming approach.

2.4.1. Kernel regression

Torgo developed the hybrid regression tree (HTL) that, amongst other proposed models, incorporated kernel regression at its leaf nodes [25]. A kernel function empowers a linear model to interpret the relationships between the observations as non-linear by mapping the observations to a higher-dimensional feature space. The kernel regression that HTL incorporates is known as a lazy learner, because the system only generalizes the training data once a prediction is made[9].

The HTL structure is induced using the CART methodology of selecting splits that minimizes the mean square error (MSE)

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

where N is the number of observations over which the MSE is computed, y_i is the actual target value of the i^{th} observation, and y_i' is the predicted value for the observation. The predicted value for each observation is chosen as the average target value in the subtree where the split is proposed[10]. A kernel regression model is thereafter used to calculate o_i when making predictions. This gives rise to the hybrid nature of HTL. Torgo states that using the kernel regression's calculations to determine the MSE for each proposed split within the tree structure is computationally too expensive [25], and therefore opts to use the CART methodology to induce the tree structure.

The kernel regression model comprises a k-nearest neighbours model and a Gaussian kernel function. The kernel regression model calculates predictions using only the training samples most similar to a given query. The similarity is based on a distance metric between two observations. The Gaussian kernel function increases the influence, based on the

[9]For this reason, lazy learners are also a popular choice for models where the training data is regularly updated.

[10]This means that effectively the deviation is being calculated; it is showcased this way to emphasise the potential of interchanging o_i with the kernel regression prediction.

distance metric, neighbours have on the prediction the nearer they are to the query in the feature space. Given the query point $\boldsymbol{q}$, the prediction is calculated using the following function [25]

$$o(\boldsymbol{q}) = \frac{1}{SK} \sum_{i}^{k} y_i \times K\left(\frac{D(\boldsymbol{x_i}, \boldsymbol{q})}{h}\right) \tag{2.6}$$

with

$$SK = \sum_{i}^{k} K\left(\frac{D(\boldsymbol{x_i}, \boldsymbol{q})}{h}\right) \tag{2.7}$$

and

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

where $D(\cdot)$ is a distance function between two vectors, h is the bandwidth parameter, and $\boldsymbol{x}_i$ is a vector of input variables with target value y_i for the i^{th} observation of k nearest neighbours. Only the training samples that fall into the same leaf node as $\boldsymbol{q}$ is evaluated to be one of its nearest neighbours. HTL is computationally more expensive than other tree models as 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 $k = 3$ without specifying the reasoning behind that choice [25].

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[11]. Although Torgo mentioned the M5 model tree incorporating similar linear models to the linear HTL variant, Torgo did not compare the performance of HTL directly to that of M5 [25].

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

$$o(\boldsymbol{q}) = \boldsymbol{wq} - \frac{1}{SK} \sum_{i}^{k} \epsilon_i \times K\left(\frac{D(\boldsymbol{x_i}, \boldsymbol{q})}{h}\right) \tag{2.9}$$

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 M data points $\{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_m}, y_m)\}$, the weights of $\boldsymbol{w}$ are

[11]With the exception of one dataset which Torgo claimed as being biased towards linear models.

calculated by minimising a least squares error criterion:

$$\epsilon = \sum_{i=1}^{m} [y_i - \boldsymbol{w}\boldsymbol{x_i}]^2 \tag{2.10}$$

This continuation of HTL was named the partial linear tree (PLT). PLT was designed with the goal of maintaining the accuracy of linear models while improving its comprehensibility of non-linear patterns [23]. PLT was compared to MARS as well as a commercial version of M5, Cubist which incorporates only linear models, on 12 separate benchmarking datasets. 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 [23].

Torgo recommended further research be done on the use of an alternative splitting criterion when inducing the tree structure as PLT did not change the procedure HTL follows, which is the standard CART methodology of selecting splits that minimise deviation. 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 [23].

2.4.2. Higher-order polynomial regression

Another approach to modelling non-linear data is through the use of higher-order polynomial expressions, rather than piecewise linear polynomials. However, estimating the structure of a multivariate polynomial as well as its parameters is a difficult task. The formula of a multivariate polynomial function incorporating all possible terms can be expressed as [17]:

$$f(\boldsymbol{x}) = \sum_{\tau=0, \Sigma_{j=1}^{m} \lambda_j = \tau}^{n} \left(w_{(\lambda_1, \lambda_2, \ldots, \lambda_m)} \prod_{q=1}^{m} x_q^{\lambda_q} \right) \tag{2.11}$$

where m is the dimensionality of the feature space, n 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 3^{rd} order polynomial describing a 10-dimensional feature space, there are 286 terms in total. Terms can be dropped to change the characteristics of the polynomial function, meaning there are 2^{286} different possible combinations of those terms. Even with the coefficients already estimated, iteratively testing all 2^{286} possible functions on a set of observations is already computationally demanding. Genetic algorithms, however, can be implemented to perform these tasks that are otherwise considered computationally too expensive [17].

A genetic algorithm is a heuristic search method that models aspects of natural selection mechanisms to evolve an optimal solution from multiple candidates [8]. "Survival of the fittest" is simulated across numerous generations, each generation having multiple candidate solutions [20]. The fitness function of a genetic algorithm dictates which candidate solutions correspond to better solutions in the problem domain [18]. Only the candidates deemed fit are allowed to reproduce or to survive to a next generation. The genetic algorithm thereby optimises the fitness function to produce a globally optimum solution.

Potgieter and Engelbrecht developed a hybrid genetic algorithm (GASOPE) capable of evolving polynomial expressions that are structurally optimal. The authors defined optimality as the shortest polynomial with the best possible function approximation. GASOPE was tested by approximating several non-linear functions and evaluating its prediction error. GASOPE was shown to be significantly faster than a neural network approach, whilst producing comparable results [17]. 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 as the model used at the leaf nodes of a model tree. The proposed model tree (GPMCC) made use of a genetic program to evolve its tree structure [18]. Hitherto discussed decision trees all employed greedy approaches to induce its structure. The use of a greedy approach can be problematic as they are susceptible to becoming stuck in locally optimal solutions. This 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 [2]. This is due to genetic approaches seeking a globally optimal solution as opposed to iteratively favouring solutions that yield local optima.

The genetic heuristic of GPMCC generates multiple tree candidates by randomly choosing a node to expand on. The feature on which to split, as well as the splitting value was randomly selected. Specialized mutation and crossover operators were introduced. Mutation ensured that new genetic material, utilising domain-specific knowledge, would be injected into the population. This was done to improve the quality of a candidate solution. The crossover operator was used to splice together two candidates. The fitness function combined the complexity of a tree, in terms of its size and number of polynomial terms it encapsulated, as well as the difference in the predicted and actual output of an observation.

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 non-linear relationships within these datasets. The disadvantage of GPMCC proved to be 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 [18].

2.5. Model tree ensembles

In the final section of this chapter, model tree ensembles are discussed. 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 abundant research. Section 2.5.1 motivates the reason why further research into model tree ensembles can be beneficial and introduces two successful ensembles which comprise the M5 model tree. Section 2.5.2 discusses the performance of RMT. Finally, MTE is examined in Section 2.5.3.

2.5.1. M5 ensembles

The challenges of developing ensembles of model trees are the increased computational cost and the difficulty of interpreting the models [20]. One of the advantages decision trees have over competing supervised learning techniques is the interpretability it exhibits. Decision trees not only produce adequate performance, but also gives insight into the relationships between the input and output variables for a given problem. When ensembled, decision trees lose its interpretability and mores so when random effects are introduced. The performance increase of ensembling decision trees has to be adequate to justify the loss of interpretability. However, model trees have an even greater variance than decision trees. Ensemble methods have proven effective in minimizing the variance of model trees [20]. There are various approaches to ensemble modelling, those applicable in the context of this paper were discussed in Section 2.2.

Aleksovski and Pfahringer are two of the few who have applied ensemble methods to model trees for regression problems, with each author incorporating a unique approach [1, 16]. The common denominator between the approaches these authors employed was the use of the M5 model tree as the base learner. Each of the model tree ensemble approaches discussed here also incorporated an aspect of randomness in the induction step. The authors reference the success Breiman achieved with random forests as justification. Incorporation of randomness into an ensemble increases the diversity present in the learners that comprise the ensemble, allowing for greater prediction accuracy [6].

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

2.5.2. Pfahringer's random model trees

Pfahringer modified the M5 algorithm to grow balanced trees within an ensemble (RMT), largely based on Breiman's random forest with little deviation other than the use of model trees as base learners [16]. Balanced trees are defined as trees incorporating the same number of observations in each leaf node. Pfahringer developed balanced trees by always selecting for the split value the median of a feature.

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 (AG) [16]. At the time, GPR and AG were considered competing state-of-the-art regression methods. RMT outperformed both GPR and AG in terms of computational efficiency, whilst having competitive performance regarding predictive accuracy [16].

2.5.3. Aleksovski's model tree ensembles

Aleksovski was tasked with developing models for dynamic systems, whose 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 (MTE) included three key features: randomised splitting features, *fuzzification*, and ensemble pruning.

MTE's induction is based on bagging, meaning that subsets of randomly selected features with replacement[12] are created for each tree in the ensemble. 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 implemented fuzzification to prevent the discontinuities each split within the model tree introduced. Aleksovski stated that the smoothing procedure of M5 produced low performing models, and fuzzification is used instead to combat this [1]. Fuzzification removes discontinuities from the model's prediction 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 comprising a single split, the prediction of a given observation, $o(\boldsymbol{x})$, is accordingly calculated as:

$$o(\boldsymbol{x}) = \mu(x_j, c, \alpha) f_1(\boldsymbol{x}) + \mu(x_j, c, \alpha) f_2(\boldsymbol{x}) \tag{2.12}$$

with

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

where, f_1 and f_2 are the linear models of two adjacent leaf nodes divided by a split on the j^{th} feature with value c. Finally, α is a fuzzy parameter calculated using cross-validation.

[12]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.

It is important to note that Aleksovski did not incorporate fuzzification into the induction of the tree structure as it proved to decrease the efficiency of the M5 algorithm due to the added computational cost it 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 output error of MTE is evaluated both with and without each tree[13] in the ensemble. 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 injection of randomness in the induction of MTE and helped MTE to produce a low variance error [1].

[13]In the case of individual tree pruning, nodes are removed.

Chapter 3

GASOPE

The following chapter presents the genetic algorithm capable of evolving structurally optimal polynomial expressions (GASOPE). Potgieter and Engelbrecht incorporated GASOPE in the leaves of the GPMCC model tree. Since the conception of GASOPE computational limitations has lessened. Therefore, changes from the original implementation have been made. The Model Tree Forest (MTF) proposed in this paper employs a revision of GASOPE in the leaf nodes of the model trees that make up the ensemble. The revision of GASOPE was developed from the ground up in Python using libraries Numpy and Scikit-learn. The complete genetic algorithm and all the deviations from the original are discussed with pseudo code in Section 3.1. GASOPE is tested and compared to the original results obtained by Potgieter and Engelbrecht in Section 3.2.

3.1. Replicating GASOPE

The original implementation of GASOPE was heavily influenced by the computational limitations at the time and developed with techniques that minimise computational cost [17]. The algorithm consists of a three-stage process:

1. Clustering of the training patterns.
2. The genetic algorithm, and
3. a hall-of-fame.

The clustering stage is implemented to minimise the number of training patterns to iterate over for evolving each individual in every generation. Clustering significantly reduced the time taken for the genetic algorithm to evolve the final individual, without reducing the performance of the genetic algorithm [17]. Due to the clustering stage, a given generation's data subset may not inherit a true representation of the entire dataset. In turn, an individual may be evolved that does not present the true nature of the dataset, this holds particularly for extremely noisy datasets [17]. A hall-of-fame was subsequently incorporated to ensure that the clustering stage does not hinder GASOPE's performance. Once the maximum number of generations is reached, the individual that performed best over all generations is chosen, 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, 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.

All other aspects of the original genetic algorithm were retained, some with slight deviations, and are presented in the following sections: Section 3.1.1 discusses the representation of an individual in the genetic algorithm. Section 3.1.2 illustrates the initialisation of an individual. Sections 3.1.3 and 3.1.4 illustrate the mutation and crossover operators of the genetic algorithm respectively. Section 3.1.5 introduces discrete least-squares approximation, used to produce the coefficients of an individual. Section 3.1.6 discusses the fitness function employed by the genetic algorithm to rank individuals. Finally, Section 3.1.7 presents the entire process followed by the GASOPE method to evolve a population of individuals for a given dataset.

3.1.1. 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:

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

where $w_\xi, \xi \in \{0, \dots, p-1\}$ is the real-valued coefficient for the corresponding term. Each term w_ξ the set 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:

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

where λ_τ is the whole-valued order of the respective input variable $x_\tau, \tau \in \{1, \dots, m\}$.

3.1.2. Initialisation

For each individual in the population, the initialisation process is described by Algorithm 1. Random variable-order pairs are repeatedly selected for each term-coefficient mapping. To help promote the growth of smaller sized polynomials, the available polynomial orders for a term are randomly selected without replacement. 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 are assigned a polynomial order of 0). The maximum number of terms, p, as well as the maximum polynomial order, n, are both user-specified parameters. Increasing either p or n increases the computational cost of the genetic algorithm. This is because for each additional term an additional coefficient is estimated for fitness calculations. In the case of the maximum polynomial order, an additional operation is computed for each increment of n.

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

```
Set $I = \{\}$.
while $|I| < p$ do
    Set $T_\xi = \{\}$.
    Select $e \in \{1, \dots, n\}$ as a random subset of integers up to the maximum
      polynomial order $n$.
    while $|e| > 0$ do
        Select $f$ to be a random order within $e$.
        Select $1 \leq g \leq m$ randomly from the set of $m$ input variables.
        if $|T_\xi| < |T_\xi \cup \{(g \to f)\}|$ then
            $e := e/\{f\}$, i.e. remove $f$ from the available orders.
            Set $T_\xi = T_\xi \cup \{g \to f\}$.
        end
    end
    Set $I = I \cup \{T_\xi \to 0\}$ 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 0. Furthermore, when an empty set of variable-order pairs is initialised, all input variables are assigned a polynomial order of 0[1].

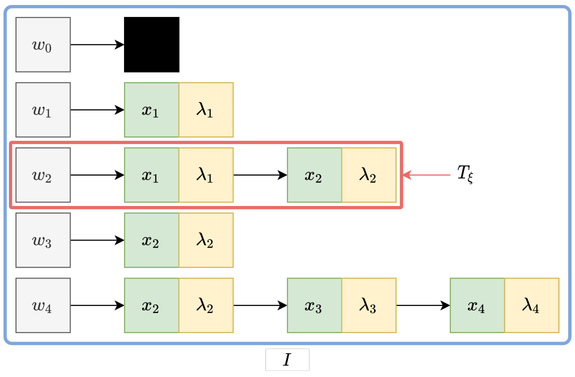


Figure 3.1: Illustration of the GASOPE initialisation algorithm, adapted from [17].

[1]Assigning a polynomial order of 0 to an input variable effectively excludes it from the term.

3.1.3. Mutation operators

The mutation operators randomly inject new genetic material into selected individuals of the population. The mutation operators ensure that larger search space is covered 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. Arbitrarily, one term-coefficient pair is 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 $T_\xi \in I$ randomly from the set of terms $I$.
Set $I = I/\{T_\xi\}$ as illustrated in Figure 3.2.
```

The second operator, *expand*, adds a new term-coefficient mapping to the set I as described in Algorithm 3. 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[2].

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

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

The third mutation operator, *perturb*, is focused on altering a randomly selected variable-order pair within one of the terms in I. When perturb is performed on an individual, one of three possible adjustments are made: a new variable-order pair is added to the term; an

[2]Polynomials that incorporate a large structure are adequately penalised during fitness calculations, preventing individuals from evolving too large polynomial structures that may compromise the genetic algorithm's computational cost

existing variable-order pair is removed from the term or a variable-order pair is adjusted. This process is described in Algorithm 4. As opposed to the original implementation of GASOPE, when a variable-order pair is either adjusted or added, the selection of λ is not restricted.

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

```
Select $T_\xi \in I$ randomly from the set of terms in $I$.
Select $e \in \{0, \ldots, n\}$ as a random subset of integers up to the maximum
  polynomial order $n$.
Select $1 \leq g \leq m$ uniformly from the set of $m$ inputs.
Select $h \in U(0, 1)$ as a uniformly distributed random number.
if $h < 0.333$ then
    Set $T_\xi = T_\xi / \{g \to \lambda\}$ where $\lambda > 0$, i.e. remove the $g^{th}$ variable-order pair
      present in $T_\xi$, 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 $e$.
    Set $T_\xi = T_\xi \cup \{g \to f\}$ as portrayed by the entirely circled variable-order pair
      in Figure 3.4.
else
    Set $T_\xi = T_\xi / \{g \to \lambda\}$ where$\lambda > 0$.
    Select $f$ to be a random order within $e$.
    Set $T_\xi = T_\xi \cup \{g \to f\}$ as portrayed by the partially circled order of the
      variable-order pair in Figure 3.4.
end
```

The fourth and final mutation operator, *reinitialise*, is simply a re-invocation of Algorithm 1, already discussed in Section 3.1.2.

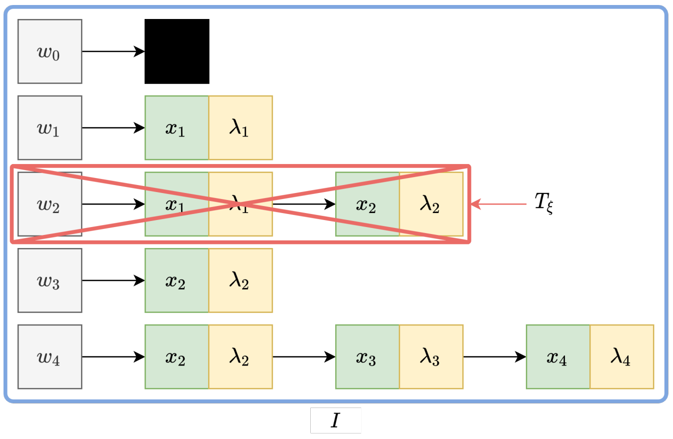


Figure 3.2: Illustration of the GASOPE shrink algorithm, adapted from [17].

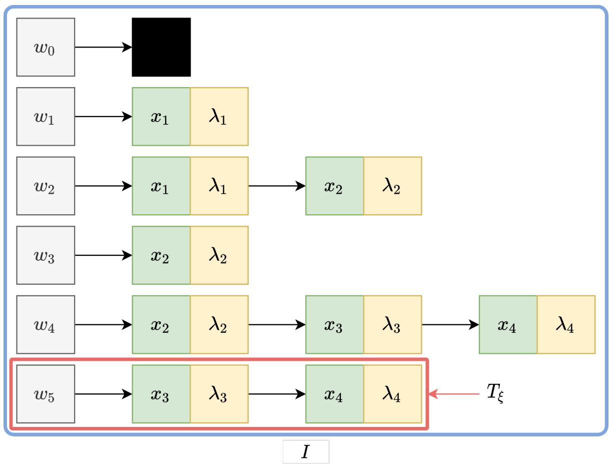


Figure 3.3: Illustration of the GASOPE expand algorithm, adapted from [17].

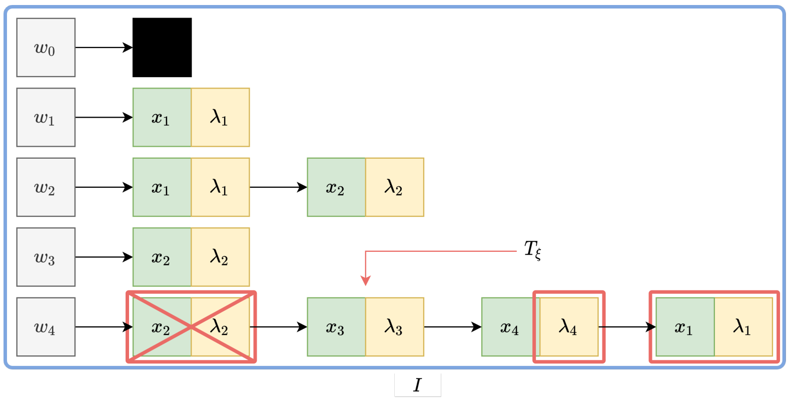


Figure 3.4: Illustration of the GASOPE perturb algorithm, adapted from [17].

3.1.4. Crossover operator

The crossover operator transfers over favourable genetic material from one generation of individuals to the next. The crossover operator serves to narrow the search space with each generation around a particular solution. When the crossover operator is called two unique individuals are randomly selected and their term-coefficient mappings 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. A selection ratio of 80:20 [3] is chosen as this was found to result in newly generated individuals to roughly equal the length of its longer parent [17]. Algorithm 5 describes how the selection ratio is implemented in the crossover operator to evolve a new individual. Contradictory to the original implementation of GASOPE, newly generated individuals may exceed the maximum number of terms.

[3]The 80:20 selection ratio dictates the probability of inclusive and exclusive term-coefficient mappings being retained respectively.

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 $h \in U(0,1)$ as a uniformly distributed random number.
    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 $h \in U(0,1)$ as a uniformly distributed random number.
    if $h < 0.2$ then
        Set $I_\alpha = I_\alpha \cup \{T_B\}$.
    end
end
```

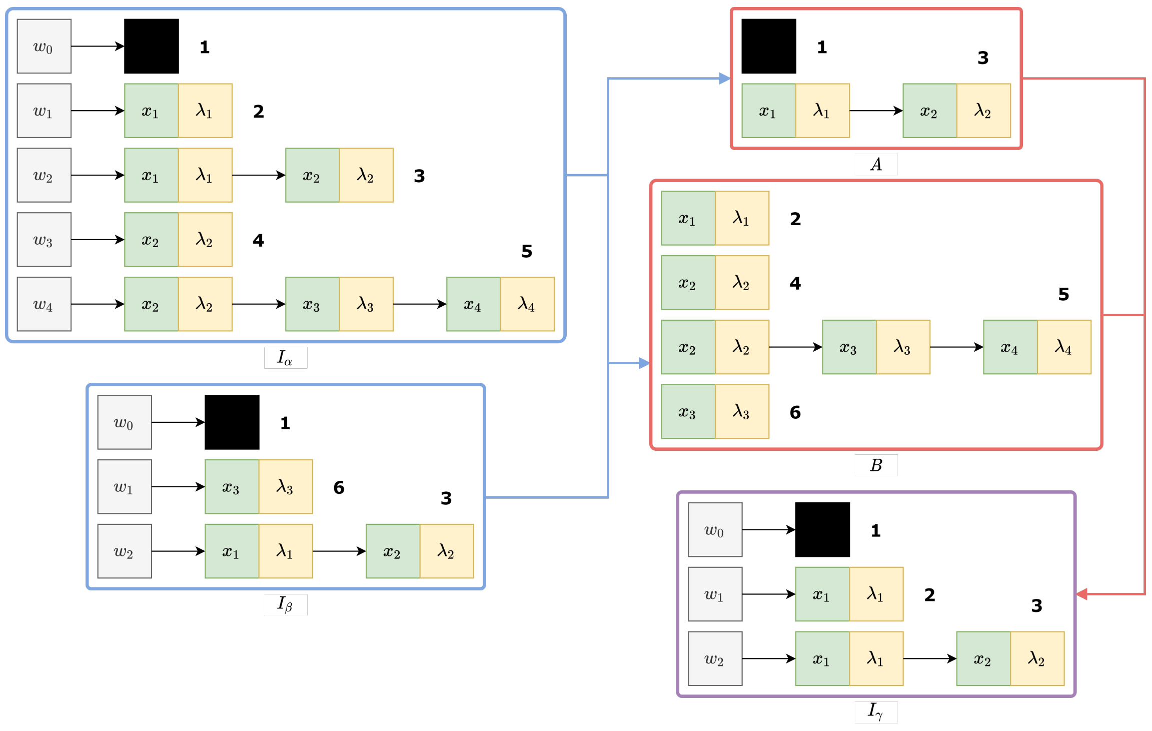

w_0
1
w_1 x_1 λ_1
2
w_2 x_1 λ_1 x_2 λ_2
3
w_3 x_2 λ_2
4
5
w_4 x_2 λ_2 x_3 λ_3 x_4 λ_4
I_α
I_β
6
A
B
I_γ

Figure 3.5: Illustration of the GASOPE crossover algorithm, adapted from [17].

3.1.5. Discrete least-squares approximation

In every generation a structurally optimal polynomial is constructed in two phases. In the first phase, the terms (which encapsulate variable-order pairs) of the polynomial function are evolved with the aforementioned operators. In the second phase, the coefficients of these terms are approximated using discrete least-squares approximation to complete an individual's set of term-coefficient mappings. Discrete least-squares approximation for a arbitrary set of L data points $\{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_l}, y_l)\}$, minimises the least squares error:

$$\epsilon = \sum_{i=1}^{l} [y_i - y_i']^2 \tag{3.3}$$

where y_i' is the predicted output of an individual's polynomial function of the form (similar to Equation (2.11),

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

where m represents the number of input variables and n the maximum polynomial order. An extrapolation of Equation 3.4 for an arbitrary individual is, for example,

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

The set of coefficients, $\boldsymbol{w}$, of the polynomial function are determined by solving the linear system:

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

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

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

and has no exact solution. In the case of a univariate function, Matrix X may be of the form[4]:

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_l & x_l^2 & \cdots & x_l^n \end{bmatrix} \tag{3.8}$$

and vector $\boldsymbol{y}^{\mathrm{T}} = \begin{bmatrix} y_0 & y_1 & \cdots & y_l \end{bmatrix}$.

Once the structure of an individual is constructed in the first phase, the coefficients are estimated by solving the above-mentioned linear system. The X Matrix from Equation (3.8) is first populated from left to right using the combination of terms each term-coefficient

[4]Matrix X has no universally set form and is adapted to comprise the set of variable-order pairs present in any given individual.

mapping 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. This concludes the evolution of an individual for a single generation. After the term-coefficient mappings are estimated successfully, the individual's fitness is calculated.

3.1.6. Fitness calculation

The fitness function dictates an individuals rank amongst all other individuals in the population. This rank is determined by how well the individual solves the problem at hand. Each generation, individuals with the lowest fitness scores are replaced preventing the genetic algorithm from exploring poor solutions. In the case of GASOPE, the fitness function favours structurally optimal and well-fitted function approximations. The fitness function is defined as [17]:

$$R_a^2 = 1 - \frac{\sum_{i=1}^{L}\left(y_i - y'_{i,I}\right)^2}{\sum_{i=1}^{L}\left(y_i - \bar{y}\right)^2} \cdot \frac{s-1}{s-k} \tag{3.9}$$

where L is the dataset size, y_i is the target output and $y_{i,I}$ the output predicted by individual I for pattern i. The model complexity k is,

$$k = \sum_{\xi=1}^{|I|} \sum_{\tau=1}^{|T_\xi|} \lambda_{\xi,\tau} \tag{3.10}$$

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

3.1.7. The complete GASOPE process

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

An elite few of the top-ranked population are installed into the following generation's population i.e. these 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, elitism rate.

From the top-ranked individuals, two unique individuals are randomly selected for crossover. Crossover is 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, crossover rate.

Algorithm 6: Process [17]

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

Once the following generation's population is fully populated, mutation is performed Dictated by the user-controlled parameter, mutation rate, several randomly selected individuals are mutated. Because GASOPE no longer incorporates a hall of fame, the best-performing individual from any given generation is not saved, only the best performing individual from the final generation is. This means that 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 combat this, when an individual is selected for mutation, a duplicate of the individual prior to mutation is stored in the population.

After the maximum number of generations have passed, the individual with the highest-ranking fitness score is presented as the ultimate solution. This concludes the revised version of GASOPE.

3.2. Testing

The following section focuses on the tuning as well as testing of GASOPE. In Section 3.2.1, the original experimental procedure of GASOPE is partly replicated to compare the performance of revised GASOPE against the originally recorded results of Potgieter and Engelbrecht. As MTF is aimed at solving multi-dimensional problems, GASOPE must produce satisfactory results on -multi-dimensional problems before it is incorporated in the leaves of MTF. GASOPE was originally only applied to one- and two-dimensional problems, but GPMCC was tested on higher dimensional problems and therefore servers as an adequate comparison for revised GASOPE [17,18]. In Section 3.2.2 GASOPE's hyper-parameters are tuned specifically for multi-dimensional data using Bayesian optimisation. Finally, in Section 3.2.3 GASOPE is tested against the results obtained by GPMCC on a dataset incorporating 9 input variables under the same experimental procedure.

3.2.1. Original experimental procedure and results

A subset of functions from the original testing procedure was used to initially test GASOPE. Table 3.1 lists these four functions with their function definitions.

Table 3.1: Function definitions on which GASOPE is tested.

Name	Function
f_1	$f(x_0) = \sin(x_0); x_0 \in [0, 2\pi]$
f_2	$f(x_0, x_1) = \sin(x_0) + \sin(x_1); \{x_0, x_1\} \in [0, 2\pi]$
f_3	$f(x_0) = x_0^5 - 5x_0^3 + 4x_0; x_0 \in [-2, 2]$
f_4	$f(x_0, x_1) = x_0^5 - 5x_0^3 + 4x_0 + x_1^5 - 5x_1^3 + 4x_1; \{x_0, x_1\} \in [-2, 2]$

The following experimental aspects were replicated from original testing. All functions are injected with uniformly generated noise over the interval $[-1, 1]$. A total of 12 000 patterns are created for each function to make up a dataset. From these 12 000, 10 000 are 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. The validation set originally used to select the best individual from the hall-of-fame, no longer servers a purpose in the implementation GASOPE other than hyperparameter tuning. The hyperparameters of GASOPE are chosen to equal that of the original testing; these values are shown in Table 3.2.

Table 3.2: Original values for the hyperparameters of 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 original results of GASOPE, reproduced from [17], together with the results obtained from the new version of GASOPE. Predictive performance was measured using mean squared error (MSE) on the test set, with σ_{MSE} indicating the standard deviation for multiple test runs These results' statistical significance 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 new version of GASOPE did not compromise its performance. However, further testing on multi-dimensional data is needed to verify this.

Table 3.3: GASOPE's predictive performance of functions 1 through 4 with noise compared against the original results from [17].

Function	Model	$\overline{MSE}$	σ_{MSE}
f_1	Original Gasope	0.343135	0.001644
	Gasope	0.332955	0.009163
f_2	Original Gasope	0.345282	0.001698
	Gasope	0.330054	0.009625
f_3	Original Gasope	0.339414	0.001765
	Gasope	0.334083	0.009059
f_4	Original Gasope	0.332791	0.001935
	Gasope	0.342808	0.007728

It is unclear if the hyperparameter values listed in Table 3.2 are indeed the optimal values. The hyperparameter values were initially only chosen "based on numerous experimental runs" [17]. Therefore, before further testing of GASOPE, tuning of the hyperparameters is performed and discussed in the section that follows.

3.2.2. Tuning GASOPE for multi-dimensional problems

Shortcomings of the original testing of GASOPE include the lack of multi-dimensional datasets and sounded motivation to the choice of hyperparameters. The choice of hyperparameters values, listed in Table 3.2, for GASOPE was evaluated on only one- and two-dimensional functions, listed in Table 3.1. Therefore, the chosen hyperparameter values are not necessarily adequate for problems with a dimensionality greater than 2. Only in the testing of GPMCC was an alteration of GASOPE applied to multi-dimensional problems. Due to the added computational cost introduced in GPMCC when growing its tree-like structure, GASOPE was implemented with reduced capability. However, the same hyperparameter values as those listed in Table 3.2 were used for the multi-dimensional problems [18].

To ensure GASOPE is the optimal model to employ at the leaves of MTF, it is first tuned and tested on a multi-dimensional problem. There are many openly available UCI (University of California at Irvine) datasets. The Abalone dataset is chosen as the results GPMCC previously produced on this dataset were not as favourable when compared to Cubist (the commercial version of M5) [18]. The appropriate tuning of GASOPE's hyperparameters as well as the newly implemented changes can improve the performance of GASOPE on the Abalone dataset.

Evaluation of the dataset is done for 30 repeated runs. For each run the dataset is split into a training set, validation set and test set (80%:10%:10%). The validation set is used to perform Bayesian optimisation for the tuning of GASPOPE's hyperparameters. Although Bayesian optimisation is not as exhaustive as a gridsearch, it is less time-consuming for large parameter spaces [22]. However, Before Bayesian optimisation is applied, the computational expense for the following hyperparameters is first evaluated:

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

The above-mentioned hyperparameters only improve GASOPE's performance as their values are enlarged, but beyond a certain value the improvement in the error metric converges. Each hyperparameter also negatively influences the computational cost of GASOPE as its value is increased. Maximum number of terms is the single hyperparameter that has the highest contribution to computational cost. Therefore, the maximum number of terms is chosen as small as possible without compromising the performance of GASOPE whilst also ensuring that the initial search space is adequately covered. Note that the maximum number of terms parameter is only applicable to the initialisation of an individual. Furthermore, GASOPE rewards the minimisation of the number of terms an individual comprises with each generation[5]. From GASOPE's implementation 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 [18].

To find the point where increase in population size causes the error metric to converge, GASOPE with population size varying from 10 to 60 in increments of 5 was trained for 30 repeated runs. For each run and varying population size, the root mean square error (RMSE) on the validation set is recorded. From the box plots that portray the RMSE distribution in Figure 3.6, is is clear that for very small population sizes the RMSE value tends 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 value over the 30 repeated runs decreases with each generation that is evolved for. For population sizes smaller than 30, the average training set RMSE was significantly

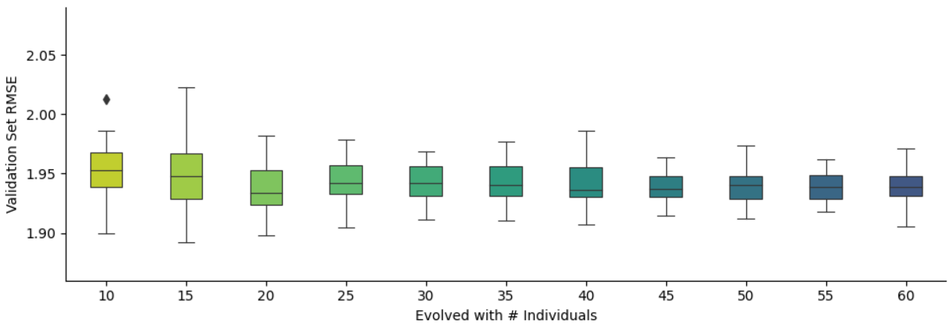


Figure 3.6: Pop size vs RMSE

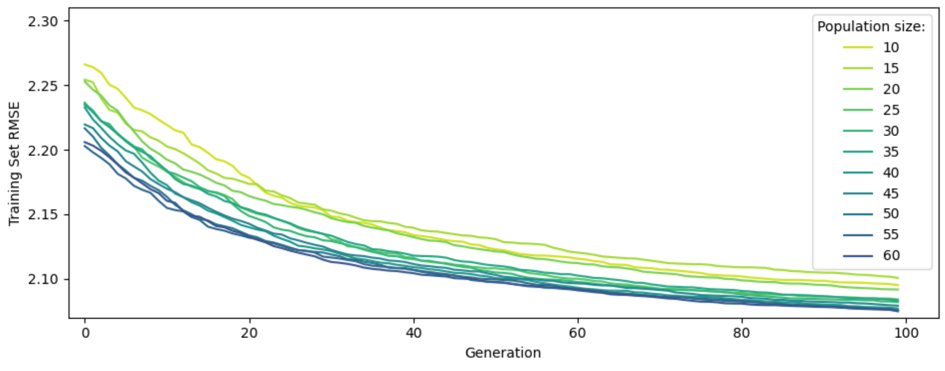


Figure 3.7: Pop size vs RMSE

higher. Once the populations size is increased above 30, both the training set RMSE as

[5]Equation (3.10) illustrates how the number of terms negatively impacts the fitness ranking of an individual.

well as validation set RMSE converges.

The final hyperparameter which heavily influences computational costs is the number of generations GASOPE evolves a population of individuals for. Figure 3.8 shows the box plots of 30 repeated RMSE values for the best performing individual evolved through GASOPE for generations varying from 10 to 200 in increments of 10. As the number of generations increased, the RMSE values tend to decrease as well as fluctuate less. However this tendency starts dropping off after 100 generations and the RMSE converges.

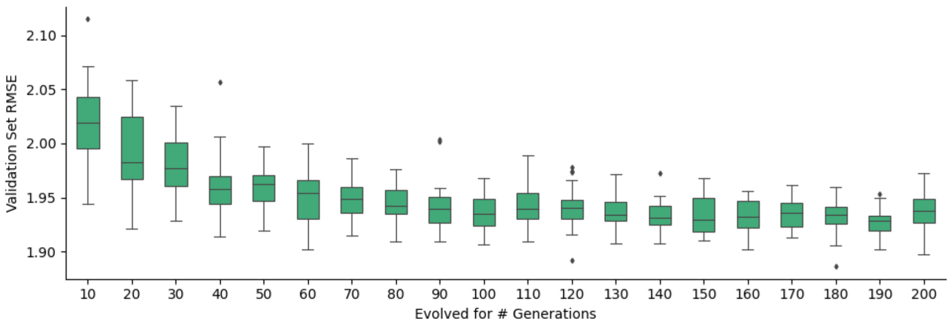


Figure 3.8: Generations vs RMSE.

The next hyperparameter for discussion, maximum polynomial order, is considered unique to each case GASOPE is applied to. Allowing polynomials to incorporate large polynomial orders when the dataset itself does not exhibit highly non-linear tendencies results in GASOPE imposing non-linear relationships that are not necessarily prevalent in the data. The Abalone dataset is quite linear[6] but GPMCC incorporated polynomials orders up to a value of four. As discussed in the 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. This is hypothesised as the cause of degradation in performance of GASOPE in the case of the Abalone dataset during the testing of GPMCC. Furthermore, the majority of the remaining datasets, of which were often more complex than the Abalone dataset, very rarely produced polynomials orders exceeding three [18].

Limiting GASOPE's maximum polynomial order to a smaller value than four may produce improved results. However, specifying an order too small is also faulty as this would prevent GASOPE from adequately capturing any non-linear relationships in the dataset. To portray the effect the maximum polynomial order has on the performance of GASOPE in the case of the Abalone dataset, three different instances of GASOPE are compared. Each instance GASOPE is trained with a respective maximum polynomial order, o, set to 1, 2 and 3.

For each of the three GASOPE instances, the remaining hyperparameters are tuned using Bayesian optimisation:

[6]The Abalone dataset is not strictly linear and still incorporate non-linear relationships to a certain extent.

1. Crossover rate
2. Mutation rate
3. Elitism rate

The error metric of the objective function used by Bayesian optimisation to evaluate a set of hyperparameters is chosen as the RMSE on the validation set. Table 3.4 lists the final hyperparameter values obtained from Bayesian optimisation for each GASOPE instance.

Table 3.4: GASOPE hyperparameters.

Hyperparameter	GASOPE with $o = 1$	GASOPE with $o = 1$	GASOPE with $o = 1$
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. Results

For comparison, 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, are also trained and tested on the Abalone dataset. For the linear regression model there are no hyperparameters to tune. For the SVR model, two hyperparameters are tuned with Bayesian optimisation. The SVR hyperparameters are the *gamma* and C values. Gamma represents the coefficient of the Gaussian kernel function and C a regularisation parameter.

Table 3.5 presents the results obtained from 30 repeated tests for each model together with the results GPMCC obtained on the Abalone dataset [18]. The only available metric for GPMCC's performance is mean absolute error (MAE). For all other models both RMSE and MAE values with their standard deviations are listed. RMSE is considered a more robust estimation of a model's performance as it penalises larger errors more severely [7]. For each test run the five comparing models are ranked according to their RMSE and MAE errors respectively. The average ranking is thereafter used for further statistical tests.

SVR is shown scoring the lowest MAE and average rank amongst all models. GASOPE with a maximum polynomial order of 2 achieved the lowest RMSE and second best average ranking. Applying a Friedman test is necessary to determine whether any results can be considered statistically significant. The null-hypothesis (All models are equal in their performance based on the results) was rejected and a post-hoc test performed to show which algorithms are statistically dissimilar.

Table 3.5: Comparison of different 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

The post-hoc pairwise test used is the Bonferroni-Dunn's test as it is best suited for comparisons of several methods with a control method (in this case, the M5's linear regression model serves as control) [9]. Figure 3.9 shows a critical diagram summarising the results of the post-hoc test. A critical diagram plots the average rank of each method and connects those which are not statistically different.

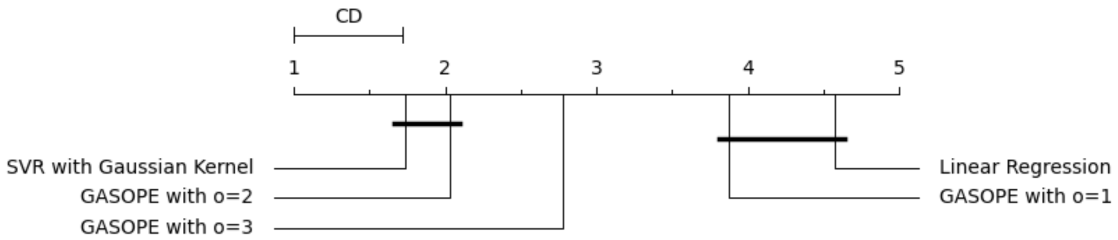


Figure 3.9: Criticial difference diagram for the results obtained in Table 3.5.

It is therefore evident from Figure 3.9 that the results of the SVR and GASOPE with $o = 2$ are statistically equivalent. The same goes for linear regression and GASOPE with $o = 1$. This is expected as GASOPE with $o = 1$ simply produces a linear polynomial. GASOPE with $o = 3$ is shown producing statistically worse results than both SVR and GASOPE with $o = 2$, confirming that imposing higher order relationships on data that is not necessarily of such sort will impact GASOPE's performance negatively.

3.3. Conclusion

After testing, it is evident that the changes brought into GASOPE produced improved results when compared to its original implementation within GPMCC. In the case of the Abalone dataset, GPMCC clearly overfitted the data. GASOPE's results were adequate to justify implementing it in the leaf nodes of MTF, which will be further discussed in the following chapter. MTF can prove to produce more robust results by being able to not only adapt to highly non-linear data, but linear data as well.

To prevent future instances of MTF from overfitting data, careful consideration should be given to the maximum polynomial order value. There is no universal maximum polynomial order value that will result in the optimal polynomial being evolved for all cases. Although GASOPE already has systems in place to penalise overly complex polynomials, in some cases the fitness function alone cannot prevent overly complex polynomials from being evolved. Through testing it was found that leaving the maximum polynomial order too large, would result in the population of individuals being flooded by overly complex polynomials during initialisation. The fitness function as well as crossover and mutation operators cannot escape these local optimas resulted by overly complex 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 (MTF) algorithm is presented and discussed in this chapter in its entirety. MTF consists of a random forest (RF) inspired tree induction algorithm with GASOPE applied to evolve ideal multivariate polynomial functions for each leaf node within a given tree of the ensemble. Furthermore, the hyperparameters of MTF are listed and tuned for on a collection of datasets obtained from the UCI machine learning repository as well as artificially induced data. Firstly, Section 4.1 discusses the RF inspirations implemented to help combat the high variance associated with MTF and presents pseudo code to describe the MTF induction algorithm. Next, Section 4.2 showcases the hyperparameter tuning of MTF as well as the steps that were taken to combat initially unfavourable results. Finally, Section 4.2.5 concludes the results of the tuning of MTF.

4.1. Methodology

As discussed in Chapter 2, model trees have a greater intrinsic ability to approximate continuous nonlinear data over the simpler regression trees found in RF. This is due to the polynomials that are employed to produce the model tree's output for a given sample as opposed to the singular output value in the leaf nodes of a regression tree. The cost of employing polynomials, specifically higher-order polynomials, is a significant increase in the observed variance error. However, it has been shown in Equation (2.1) that the variance error can be effectively reduced through the use of an ensemble of decorrelated trees. RF extends the idea of decorrelating trees by introducing randomly selected subsets of splits 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 GASOPE and its decision tree architecture.

The steps taken to ensure the trees within MTF are sufficiently decorrelated, similarly to the trees within RF, are motivated and listed in Section 4.1.1. Section 4.1.2 thereafter discusses the considerations of successfully applying these steps as well as the reasoning behind the omission of solutions that have been presented in previous research. Finally, Section 4.1.3 presents the complete pseudo code of the MTF induction algorithm.

4.1.1. Random forest inspirations employed in model tree forest

The base learners of MTF, which together form an ensemble, are referred to as model trees. In the case of RF, the base learners are regression trees. Regression trees and model trees both employ a divide-and-conquer induction approach to recursively split the training data using a set of rules. The rules are referred to as the splitting functions and are obtained through a greedy search approach. During the induction of a decision tree, several splits are proposed and evaluated through a chosen criteria, for the M5 model tree this criterion is standard deviation reduction. Standard deviation reduction is calculated through the following formula, as previously discussed in Section 2.3.2:

$$\Delta error = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i) \tag{4.1}$$

where T is the collection of training samples for the data within a node, and $sd(T)$ is the standard deviation of the target values in T. T_i denotes the subset of samples that belong to outcome i of a split and $std(T_i)$ the standard deviation of the target values in T_i. MTF employs the same splitting function as in Equation (2.2). The greedy search approach to tree induction is what RF, MTF and similar M5 ensembles have in common. Recursively splitting the training set contributes to the high variance error as a decision tree tends to overfit the training data through each additional split.

What differentiates the base learners of MTF from more traditional model trees, such as the M5 model tree, is the use of GASOPE to evolve higher-order polynomial equations to describe the output of a leaf node. For the M5 model tree, as simpler linear model is employed, described by the following equation:

$$w_0 + w_1x_1 + w_2x_2 + ... + w_kx_k \tag{4.2}$$

where x_k represents the k^{th} input feature, w_k its respective weight, and w_0 the bias term. In comparison to Equation (4.2), stands the higher-order polynomial equation of the model trees that comprise MTF:

$$f(\boldsymbol{x}) = \sum_{\tau=0, \Sigma_{j=1}^{m}\lambda_j=\tau}^{n} \left(w_{(\lambda_1,\lambda_2,\ldots,\lambda_m)} \prod_{q=1}^{m} x_q^{\lambda_q} \right) \tag{4.3}$$

where m is the dimensionality of the feature space, n the degree of the polynomial order and $w_{(\lambda_1,\lambda_2,\ldots,\lambda_m)}$ the weight of each term. For a regression tree, employed by RF, Equations (4.2) and (4.3) are simply replaced by a constant value in the leaf nodes of base learners in RF. The increased complexity of linear models and higher-order polynomials used to produce the output of the decision tree leads to a significant increase in variance error of decision trees in which these output models are present. 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 base learners.

In turn, these two factors together bring the bias error down further than what RF is capable of.

Employing techniques similar to RF in MTF, may produce a more favourable bias error, whilst keeping the variance error low enough to not degrade the performance of MTF. A previous tactic employed to combat high variance error was the introduction of pruning. However, for RF pruning is omitted as RF introduced new methods to reduce variance. Pruning prohibits a decision tree from exhibiting the advantageous low bias error it has the potential to. RF and MTF both employ the bagging technique, discussed in Section 2.2.3 to ensembling. Each tree in a bagged ensemble is induced on a subset of the training set equal to two-thirds of the original training set size. A high variance error of bagged decision tree ensembles can be reduced through decorrelation of the base learners within the ensemble, previously proven through the following equation:

$$Var(\boldsymbol{x}) = \rho(\boldsymbol{x})\sigma^2(\boldsymbol{x}) + \frac{(1-\rho(\boldsymbol{x}))\sigma^2(\boldsymbol{x})}{N} \tag{4.4}$$

where ρ is the sampling correlation between the prediction of any two decision trees in the ensemble and σ is the sampling variance of any single, randomly drawn, decision tree. N is the number of decision trees in the ensemble and $\boldsymbol{x}$ represents a set of observations. It is clear from Equation (4.4) that increasing the number of decision trees in the ensemble will effectively reduce the variance error.

To further decorrelate trees within the ensemble, RF induces the splits of base learner decision trees on subsets of the input feature space. The splitting function, shown Equation (4.1), randomly draws a third of the available input features and computes the proposed splits only on this subset of input features. Thereby, RF can both minimise bias and variance errors through the induction of numerous, fully-grown, decorrelated decision trees as base learners. This random selection process is repeated for each node and is implemented in MTF to further reduce the high variance error of MTF. The only limiting factor for MTF's ensemble size is the computational cost of inducing large amounts of model trees in serial.

The output, $\boldsymbol{y}$, of MTF for a given set of observations, $\boldsymbol{x}$, is calculated as the uniformly weighted average over the predictions of each model tree within the ensemble:

$$y(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}) \tag{4.5}$$

where $f_i(\mathbf{x})$ denotes the predicted value for the i^{th} model tree within the ensemble given $\boldsymbol{x}$.

4.1.2. Steps taken to ensure model tree forest is robust

MTF attempts to not only perform well on complex, highly nonlinear data but simpler data as well. The previous chapter showed GASOPE, if correctly tuned, can generalise adequately to linear data. MTF, which incorporates GASOPE, also requires fine-tuning of its hyperparameters before being applied to a regression problem. To ensure MTF produces favourable results for both cases, its robustness must be addressed.

Previously discussed approaches to increasing the robustness of a decision tree model were pruning and smoothing. Pruning was addressed in the previous section and omitted as the ensemble structure of MTF replaces the need for pruning. Smoothing has been applied in various ways to decision trees to minimise the discontinuities that are formed between the output of adjacent leaf nodes. The M5 model tree incorporated a simple function, which is applied from the leaf node up the path to the root node, each time altering the final output based on the output of a given node on the path. This method of smoothing is previously described through Equation (2.3). Aleksovski's MTE, which comprised an ensemble of M5 model trees, improved on the smoothing process of the original M5 model tree through fuzzification. Fuzzification, as previously discussed in Section 2.5.3, employed Equation (2.12) to smooth the output of neighbouring leaf nodes through the use of a sigmoid function. As opposed to smoothing, which required each node in the tree to encapsulate an output function, fuzzification only required the leaf nodes of a decision tree to comprise an output function. This would allow for a lower computational cost, however, the computational cost is still prevalent.

Aleksovski concluded that the modifications introduced to the original M5 model tree, including fuzzification, did not yield conclusive results. For MTF, computational cost is a major dictator in the freedom to which the model is allowed to increase in its complexity. All parameters should be tuned to minimise computational cost. Furthermore, smoothing and fuzzification were both applied to model trees which incorporated linear models. These linear models are more likely to produce discontinuities as opposed to nonlinear models. Therefore, the added computational cost of implementing smoothing, fuzzification, or a similar process into MTF, which utilises nonlinear polynomials, is detrimental to its 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. By omitting this step, MTF is given an increased computational budget for the induction of the base learner model trees.

Replacing the output functions in the leaf nodes of a model tree with higher-order polynomials evolved through GASOPE brought forth a problem. During initial testing, which will be discussed in more depth in the coming Section 4.2, the model tree, induced on the training set, irregularly produced a poor fit for the given samples in a leaf node. Although this occurred rarely, it was often enough to demand further investigation. The following three factors are hypothesised as contributing to the problem:

1. A limited initial search space for the genetic algorithm,
2. insufficient training generations and/or
3. insufficient training samples.

In these cases, the higher-order polynomial evolved by GASOPE was incapable of outperforming a simple linear model. Simply increasing the search space as well as training generations did not remedy the problem, as computational costs proved too heavy. Furthermore, the problem was too rare to justify increasing the computational costs for all cases, including those where GASOPE performed adequately. For this reason, MTF employs a baseline linear model comparison step in its induction. The baseline linear model comparison 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. More often than not, this error from GASOPE was the product of trying to fit an overly complex polynomial on a set of observations that did not demand a nonlinear model approximation. Alternatively, the fitness function of GASOPE could be changed to punish higher-order polynomials more strictly. However, due to the rarity of the problem increasing the strictness of the fitness function to accommodate for the minority of training, would negatively impact the majority of training cases. The added baseline linear model comparison was found to successfully address the irregularities, although happening rarely, that GASOPE introduced in MTF whilst retaining the freedom of GASOPE to sufficiently evolve higher-order polynomials.

In the rare case where a model tree is induced which significantly decreases the accuracy of the ensemble, MTF has another step included ensuring the performance of the final ensemble is not compromised. This final step is ensemble pruning. As discussed in Section 2.5.3 Aleksovski includes a greedy ensemble pruning step to optimise the M5 ensemble. Trees that do not contribute to the accuracy of the ensemble are removed, thereby aiming to produce a better performing 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. One by one, the ensemble's size is reduced by one, omitting the respective tree from the output calculation described through Equation (4.5). If the model's error metric is improved with the omission of the tree being evaluated, it is permanently pruned from the final ensemble. The same greedy optimisation step implemented in the M5 ensemble is employed in MTF. The addition of the baseline linear model comparison step together with ensemble pruning increased the robustness of MTF, ensuring MTF performs adequately on unseen data.

4.1.3. Pseudo code

MTF induction algorithm consists of three nested stages:

1. GASOPE
2. Model tree (MT) induction
3. Model tree ensemble induction

A single MTF model comprises multiple MT base learners and a single MT further encapsulates multiple higher-order polynomial output functions evolved through GASOPE. The pseudo code describing GASOPE was previously discussed in Section 3.1. In this section, the pseudo code that describes the induction of MT as well as MTF is presented.

The induction algorithm of MT incorporates a recursive node splitting function. Starting at the root node, multiple splits are proposed and if the stopping criteria are not met, the split producing the best reduction in standard deviation is chosen. Thereafter, each child node is evaluated for further splitting. Once the stopping criterion for a node is met, the node is labelled a leaf node and the recursive splitting function continued on the remaining nodes. The final set of rules housing the splits of the MT are stored within a dictionary.

Each node within a decision tree is described through the following set:

$$N = \{s, N_L, N_R, f\} \tag{4.6}$$

where s is the split housed by the node, N_L and N_R the left and right children nodes of N and f the function defining the output of N. In the case of a leaf node s, N_L and N_R are left void. For any other node, the opposite is true i.e. only f is empty. A split is further defined through the set of two parameters:

$$s = \{F, S\} \tag{4.7}$$

where F is the feature on which to split and S is the value of the split. In a dataset with 10 features and 100 samples there are $10 \times 100 = 1,000$ possible splits. Parsing a proposed split to Equation (4.1) returns a value that represents the standard deviation reduction (SDR) of the proposed split. The optimal split for a given dataset is computed by iterating through each feature as well as the corresponding sample value and calculating the SDR of each proposed split. The split with the highest SDR is selected as the optimal split. For MTF, splits are computed on a random subset of the feature space. The number of features chosen to form the subset is equal to a third of the original feature space.

The induction of a given path within a decision tree is halted when the stopping criterion is met. For MTF the stopping criteria are either when a maximum tree depth is met, or when the number of samples on which to split is smaller than twice the minimum leaf samples. Once a stopping criterion is met, the applicable node is labelled a leaf node and the data within the leaf node is parsed to GASOPE. The dataset a node encapsulates is calculated through the splits on the path from the root node to the leaf node. GASOPE returns an output function that is stored in the leaf node. Algorithm 7 describes the induction of a single MT.

Algorithm 7: Pseudo-code for MT induction.

```
Given training set $T$ with $l$ samples.
Initialise an empty set of nodes $N = \{\}$.
Initialise root node, $N_0$.
Let $N = N \cup N_0$.
Call Grow($N_0, T$).

**Grow** ($N_i, T_i$):
**if** *stopping criteria is met* **then**
    Label $N_i$ a leaf node.
    Parse $T_i$ to GASOPE.
    Save the returned output function in $N_i$.
**else**
    Select $\{F_1, F_2, .., F_n\}$ as a random subset of features from $T_i$.
    Initialise $s = \{F_1, S_1\}$.
    **for** $j = 1, .., n$ **do**
        **for** $k = 1, .., l$ **do**
            Select sample $S_k$ from $T_i$.
            Let $s' = \{F_j, S_k\}$.
            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 $T_i$ into $T_L$ and $T_R$ according to $s$ .
    $N = N \cup \{N_L, N_R\}$.
    Grow( $N_L, T_L$ )
    Grow( $N_R, T_R$ )
**end**
```

When MTF induces an entire forest of model trees, Algorithm 7 is called on several bootstrapped subsamples of the training set. Each bootstrap subset is selected with replacement. Once the stopping criterion is met, the first phase of the induction process halts. For MTF the only stopping criterion is the maximum ensemble size.

After the first phase follows the second and final phase, namely the ensemble pruning phase. At the start of the ensemble pruning procedure, 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 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, where $c = 1 + \frac{1}{|E|}$. Here $|E|$ refers to the ensemble size. The value of c was chosen such that MTs that are on the border of increasing the MSE of the ensemble are not unnecessarily pruned. This concludes the induction of MTF.

Algorithm 8: Pseudo-code for MTF induction.

```
Given training set $T$.
Given validation set $V$.
Initialise empty ensemble array, $E = []$.
while stopping criteria is not met do
    Randomly sample with replacement from $T$ to produce $T_i$.
    Call MT induction algorithm and parse $T_i$.
    Insert the returned MT into $E$.
end

Compute a baseline $MSE$ on $V$.
for $i = 1$ to $|E|$ do
    Prune $MT_i$ from $E$.
    Compute $MSE'$ on validation set.
    if $MSE \times c > MSE'$ then reinsert $MT_i$ into $E$.
end
```

Figure 4.1 illustrates the complete induction algorithm of MTF. Figure 4.2 portrays the bagging process employed by MTF to train its base learners and produce an output from the ensemble.

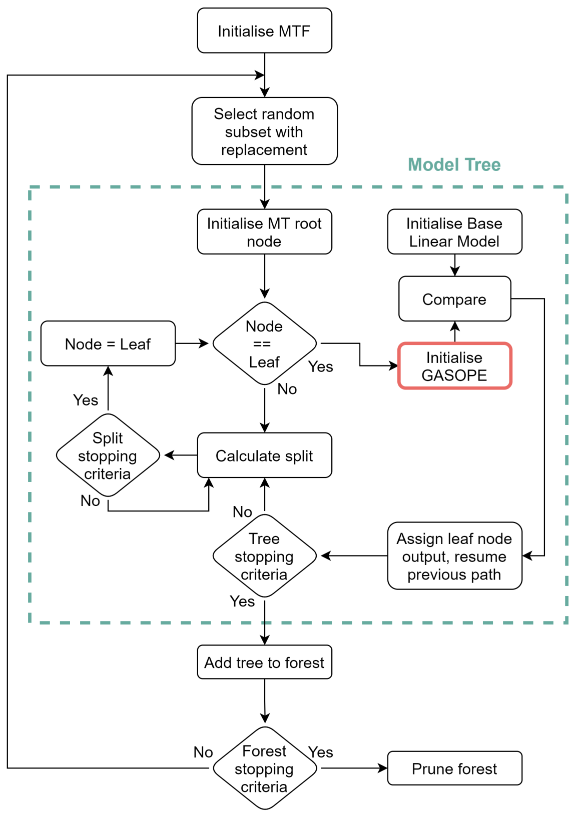


Figure 4.1: Illustration of the MTF induction algorithm.

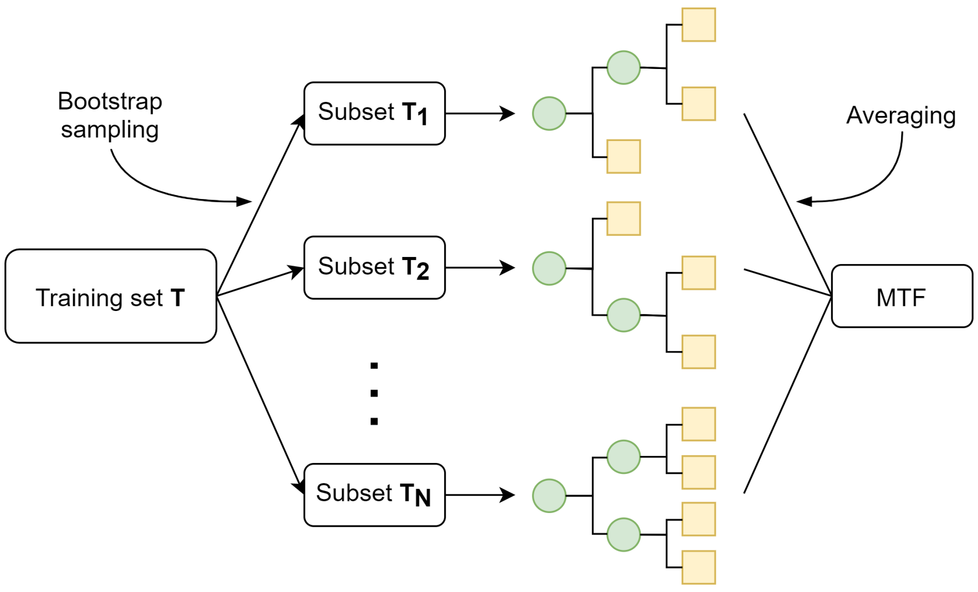


Figure 4.2: Illustration of the MTF bagging and output calculation.

4.2. Hyperparamater tuning

This section discusses the tuning of all remaining hyperparameters applicable to MTF. The hyperparameters are listed and the experimental procedure to follow for the tuning thereof is discussed in Section 4.2.1. Next, Section 4.2.2 analyses the linearity of each dataset on which the performance of MTF is evaluated to tune the maximum polynomial order parameter. Section 4.2.3 follows with the tuning of the maximum tree depth of the base learners of MTF as well as the analysis of computational costs associated with the induction of MTF. Finally, Section 4.2.4 the final hyperparameter of MTF, ensemble size, is tuned for.

4.2.1. Experimental procedure

The GASOPE hyperparameters were previously tuned for in Section 3.2. However, as stated, careful consideration should be given in choosing the maximum order for a given problem. For each unique dataset on which MTF is applied, an analysis of the best performing maximum polynomial order is first performed. The remaining parameters of MTF are:

1. Number of splitting features,
2. minimum leaf samples,
3. maximum depth and
4. ensemble size

During the conceptualisation of random forest, Breiman proposed the number of splitting features for each node equal $\log_2(m+1)$ where m is the dimensionality of the feature space [6]. This value was approximated to $\sqrt{m}$ for classification problems and $m/3$ for regression problems. In general, those values have produced good performances of random forests and have been accepted as the default. For this reason and in favour of minimising computational costs, a value of $m/3$ was chosen for the number of splitting features for MTF.

The next hyperparameter, minimum leaf samples, is dependant on the least-squares algorithm. The Python package used to implement least-squares approximation in GASOPE, SciPy, incorporates the Leven-Marquardt (LM) algorithm to solve the nonlinear least-squares problem, previously described through Equation (3.7). The prerequisite for the implementation of the LM algorithm is that the number of training samples is greater than the number of variables to solve for. Therefore, the minimum leaf samples should always be at least greater than the number of features in the dataset. In the case of GASOPE, the number of leaf samples should be greater than the number of terms used, specifically,

10. However, the polynomials describing a given individual are given the freedom of exceeding the number 10, if doing so results in increased fitness. In most cases, it is rare for the polynomial to grow substantially beyond 10 as the fitness function, described in Equation (3.9), punishes individuals with increased terms. In conclusion, a value of 20 is assigned to the minimum leaf samples to accommodate for an increase in polynomial terms. It is worth noting that minimising the value for the minimum leaf samples allows for an increased number of evaluated splits during training.

To tune the remaining applicable hyperparameters, ten datasets each with a unique composition are selected from the UCI machine learning repository and the L. Torgo Repository [24]. Six of the datasets were present in the testing of GPMCC. The additional four datasets are chosen to expose MTF to big data problems. The ten datasets are listed in Table 4.1 with their attributes.

Table 4.1: Benchmark datasets used for tuning of MTF.

Dataset	Number of features (N = Nominal, C = Continuous)	Number of samples
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

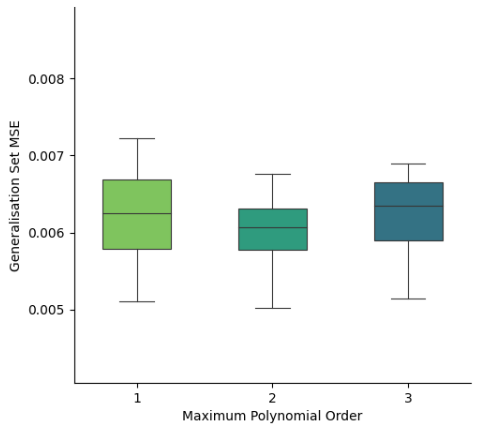
The hyperparameters that require tuning for each dataset are maximum polynomial order, maximum depth and ensemble size. From these hyperparameters, it is clear that computational cost heavily influences the optimal chosen value. For each hyperparameter, increasing its value increases the computational cost of MTF. For the maximum depth as well as maximum polynomial order, an increased value also increases the complexity of a given base learner and thereby contributes to an increased variance error. Although increasing the ensemble size helps reduce this variance error, it also significantly increases the computational cost for each additional base learner that is induced. Therefore, it is beneficial to tune maximum polynomial order and maximum depth first.

Each of the three hyperparameters is associated with different stages of MTF. The maximum polynomial order applies to GASOPE, maximum depth to MT and the ensemble size to MTF as a whole. Therefore, the hyperparameters may be tuned for individually, eliminating the need for a grid search based tuning approach.

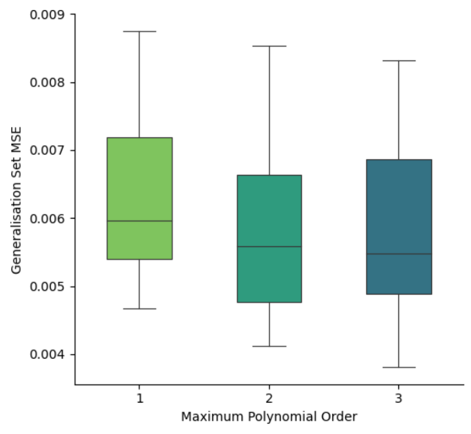
4.2.2. Maximum polynomial order

For maximum polynomial order values of 1, 2 and 3 are compared. Maximum polynomial order tuning serves as an estimation of a dataset's nonlinearity. By inducing GASOPE models of 1, 2 and 3 maximum polynomial order on the entire training set and comparing the results on a generalisation set, the linearity of the dataset is effectively estimated. The polynomial order of the best performing GASOPE instance is thereafter chosen as the maximum polynomial order for MTF.

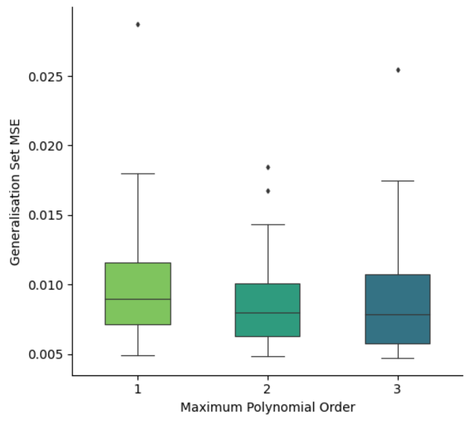
The error metric used to evaluate the performance of GASOPE for a given maximum polynomial order value on a specific dataset is MSE as described in Equation (2.5). For each given hyperparameter value and dataset, GASOPE is trained on a training set and the MSE is calculated on a generalisation set for 30 repeated runs. The training set and generalisation set are split 80:20. Figure 4.3 shows the MSE GASOPE scored on each dataset for varying maximum polynomial orders over the 30 repeated runs.



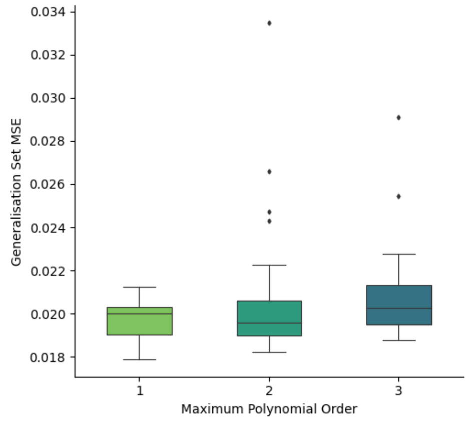
(a) Abalone



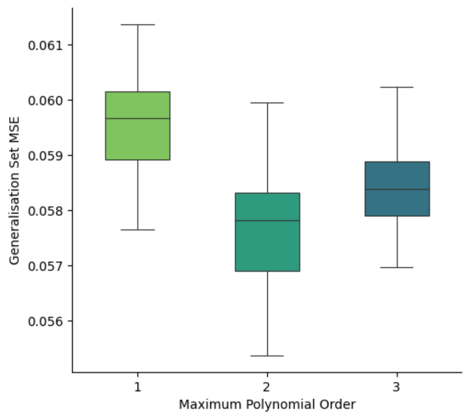
(b) Auto-MPG



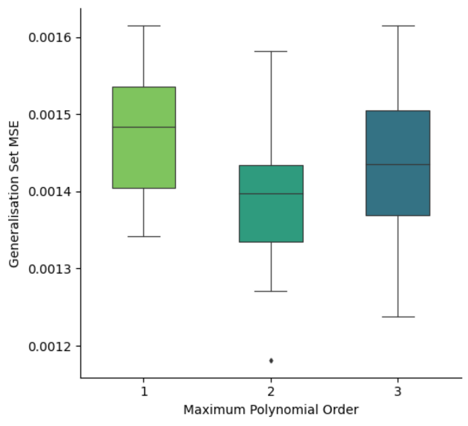
(c) Boston housing



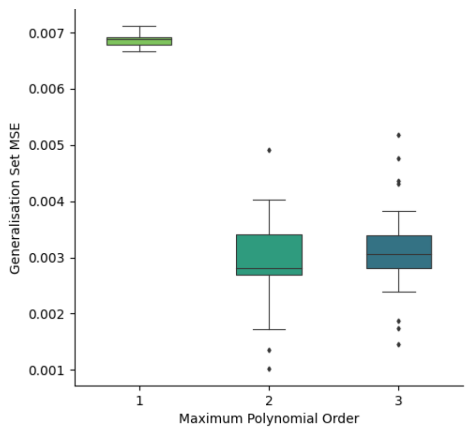
(d) California housing



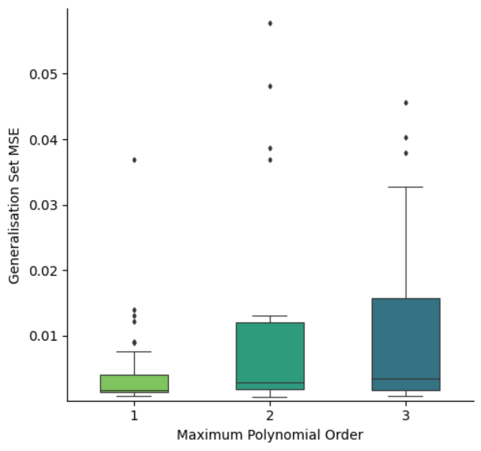
(e) CASP



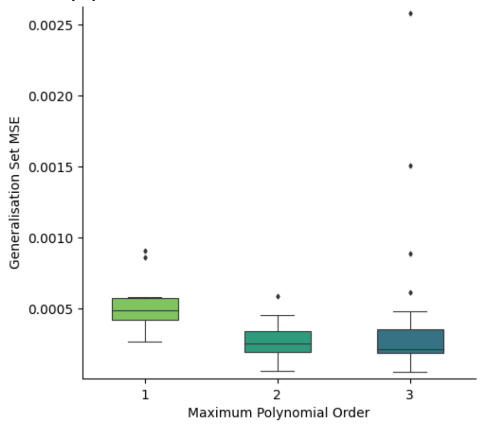
(f) Elevators



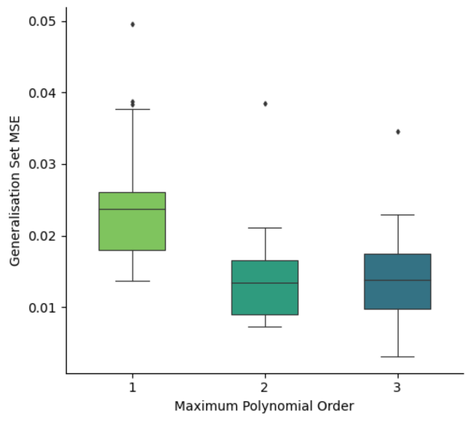
(g) Friedman artificial domain



(h) Machine CPU



(i) MV artificial domain



(j) Servo

Figure 4.3: GMSE of GASOPE for varying degrees of maximum polynomial order on the benchmarking datasets.

The maximum polynomial order which produced the lowest MSE for each dataset is chosen as the final parameter value. For the majority of datasets, a value of two resulted in the lowest MSE. Only the Machine CPU dataset favoured a maximum polynomial order of one. This shows that the datasets chosen for the testing of MTF are all nonlinear except for the Machine CPU dataset. However, datasets are rarely highly nonlinear, only three of the ten datasets favours a maximum polynomial order of three.

It is worth noting that assigning a maximum polynomial order higher than the favourable value is not fatal to the performance of the algorithm as the fitness function of GASOPE penalises higher-order functions that do not significantly improve the error metric when compared to a lower-order function.

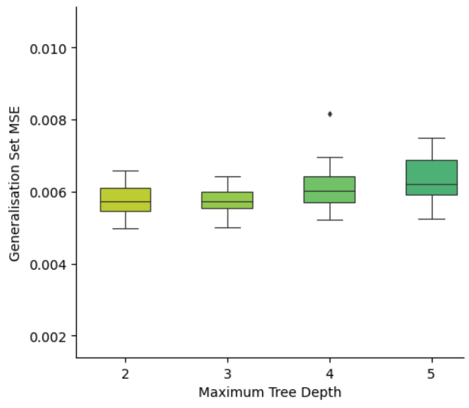
For the cases where there is little discrepancy between the MSE for maximum polynomial orders values of one and two, two is chosen as the preferred value. This is due to the base linear models which are used as a fallback if the output function evolved through GASOPE performs poorly on the given set of data samples.

Appendix 7 lists the values portrayed in Figure 4.3 in full. Additionally, the average computational time of each dataset is listed.

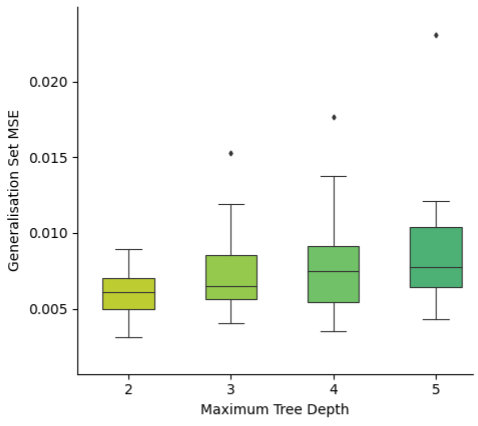
4.2.3. Maximum model tree depth

The maximum depth of MT has a large influence on the computational cost as well as the variance error of MTF. The larger a decision tree is grown, the more likely it is to overfit the training set. In conjunction, the larger a decision tree is grown, the higher the number of leaf nodes are. For MTF, each additional leaf node present in MT requires a GASOPE model to be induced on the data encapsulated by the leaf node. It is therefore beneficial for MTF to cap the depth of a tree at a specific value, to combat high computational costs. An unintended consequence of limiting the depth is a further reduction in the variance error of MTF. In the case of traditional regression trees, prematurely limiting the depth of the tree results in a significant increase in bias error. However, with MTF this increase is minimal due to the already complex composition of GASOPE.

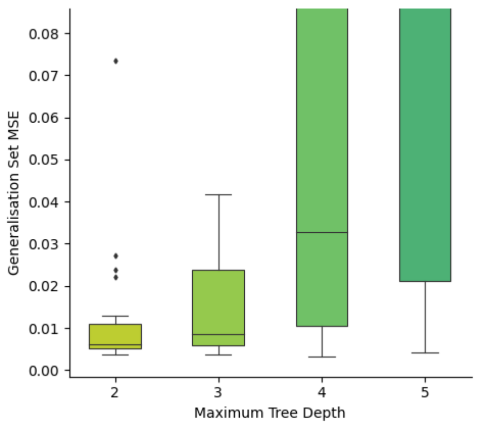
The error metric used to evaluate the performance of the base learners within MTF, MT, for a given maximum depth on a specific dataset is MSE. For each given hyperparameter value and dataset, MTF is trained on a training set and the MSE is calculated on a generalisation set for 30 repeated runs. The training set and generalisation set are split 80:20. The computed MSE on the generalisation set is shown for each dataset in Figure 4.4.



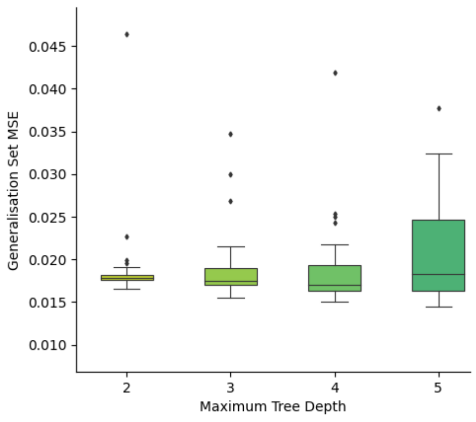
(a) Abalone



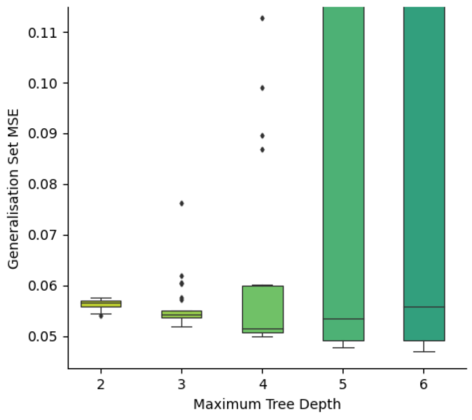
(b) Auto-MPG



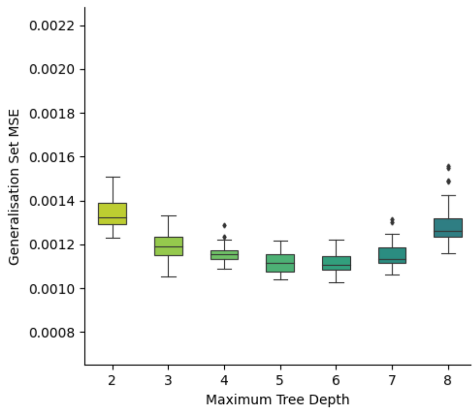
(c) Boston housing



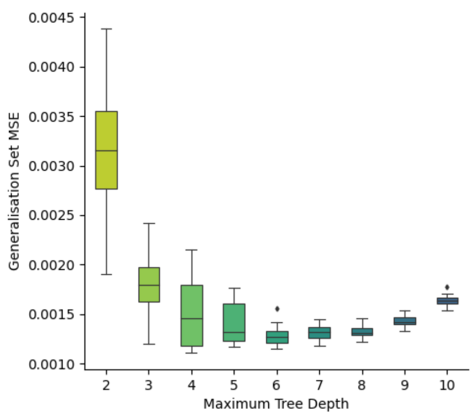
(d) California housing



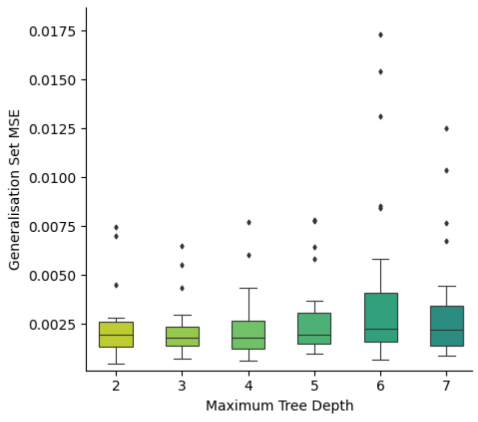
(e) CASP



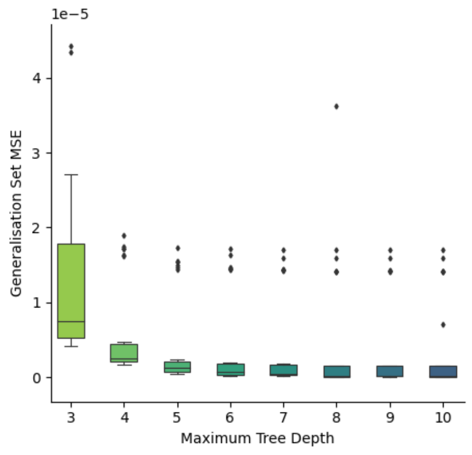
(f) Elevators



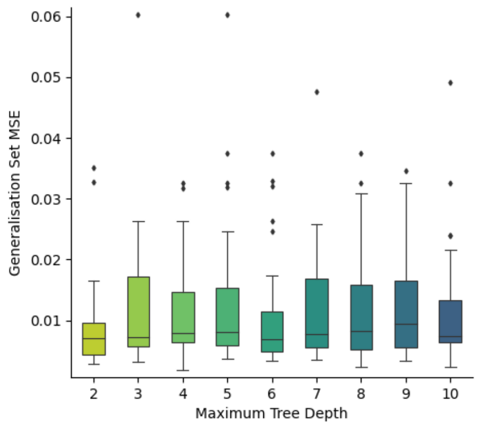
(g) Friedman artificial domain



(h) Machine CPU



(i) MV artificial domain



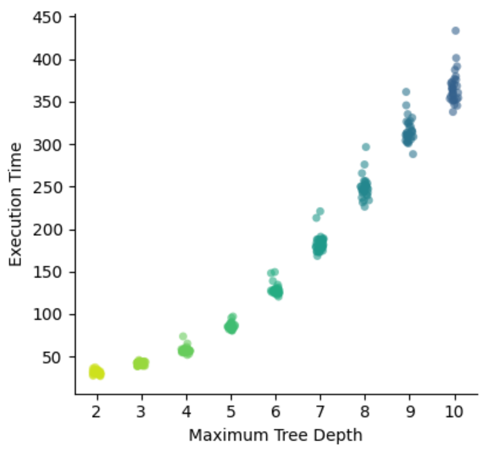
(j) Servo

Figure 4.4: GMSE of MT for varying degrees of maximum tree depth on the benchmarking datasets.

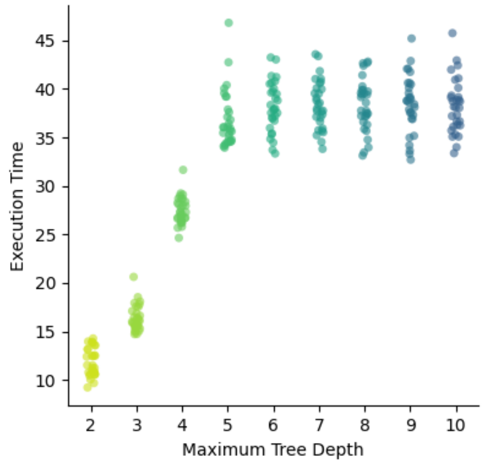
For a single model tree, there is an optimal maximum depth that produces the best performing MSE on the generalisation set. This value differs depending on the data given to the induction algorithm. Allowing trees to grow beyond this value would increase the MSE for a single model tree, however, if multiple model trees are bagged in an ensemble this increase in MSE is reduced. It is still favourable to limit the depth of a single model tree to help reduce the computational cost and allow for more trees to be induced in the ensemble.

For illustrative purposes severe outlier results are omitted in the boxplots of Figure 4.4. Tables 8.1 through 8.10 in Appendix 8 show the complete results. Linear base models were enabled in these results to ensure the model trees were tuned appropriately. Tests were run where linear base models were disabled to show the difference in the tuning results. These results are shown in Appendix 8. Incorporating linear base models brought more consistency to the results.

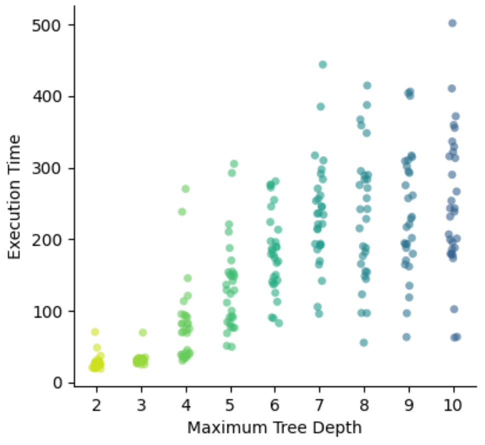
Figure 4.5 shows how the computational cost increased in execution time as the maximum depth of the model tree is increased. In smaller datasets, the execution time is shown to plateau off after a given value. This is explained through the minimum leaf samples value that prohibit the remainder of the dataset to be split on further during the induction process. For example, in Figure 4.5b beyond a maximum tree depth value of six, the execution time no longer increases. Each time a model tree is induced on the dataset, the number of data samples in a node with a depth of six is smaller than twice the minimum leaf samples, prohibiting the node from being split on further by the induction algorithm. For larger datasets, such as Abalone and California Housing, the execution time can be seen growing exponentially, confirming the need to keep the tree depth and consequently computational cost at a minimum.



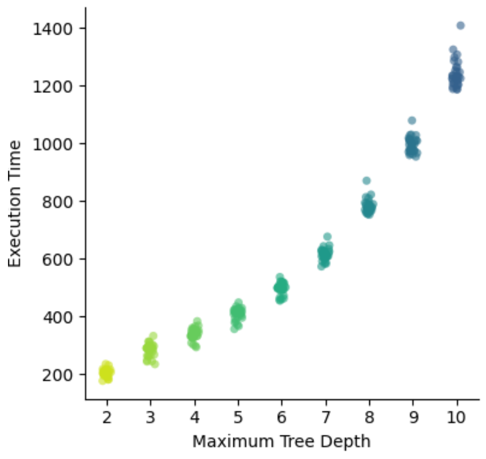
(a) Abalone



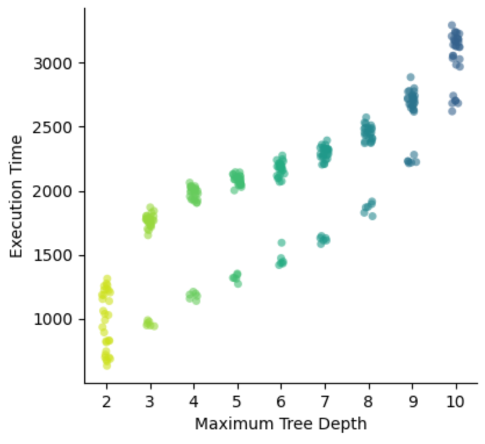
(b) Auto-MPG



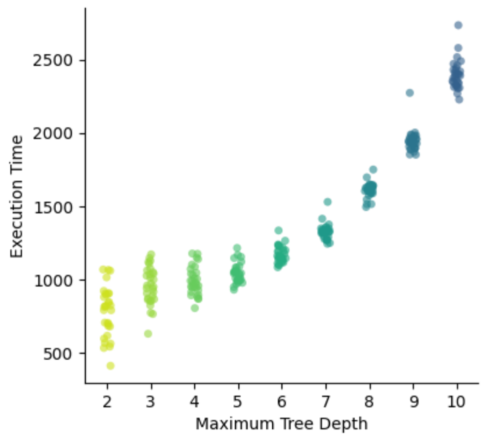
(c) Boston housing



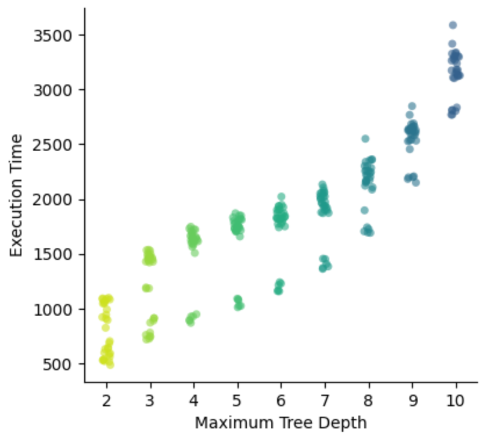
(d) California housing



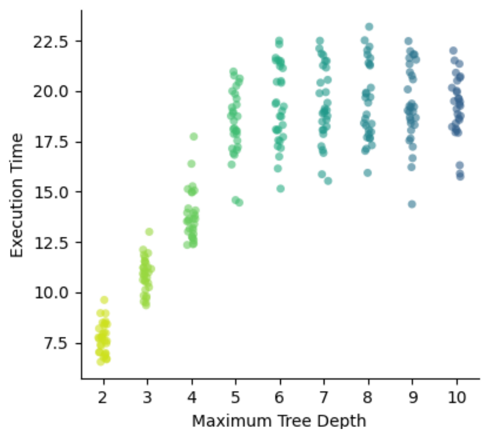
(e) CASP



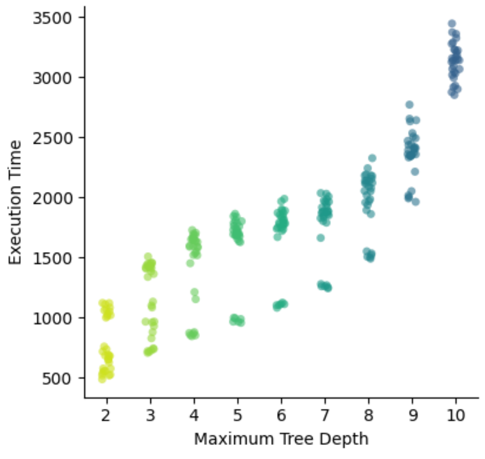
(f) Elevators



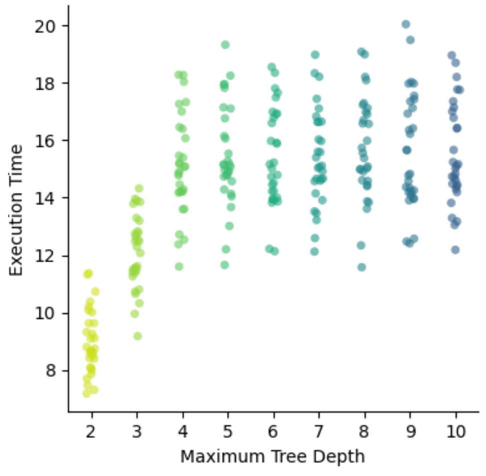
(g) Friedman artificial domain



(h) Machine CPU



(i) MV artificial domain



(j) Servo

Figure 4.5: The computational cost of MT for varying degrees of tree depth on the benchmarking datasets.

4.2.4. Ensemble size

The error metric used to evaluate the performance of MTF for a given set of hyperparameter values on a specific dataset is MSE. For each given hyperparameter value and dataset, MTF is trained on a testing set, pruned using the validation set and the MSE is calculated on a test set for 30 repeated runs. The training set, validation set and test set are split 80:10:10.

The MSE for each consecutively added model tree to the ensemble is portrayed in Figures 4.6 through 4.15. For smaller datasets where computational expenses are more lenient, the ensemble is grown beyond 30, the results are shown in full in Appendix 8.

Take note of the high offset observed in the MSE for the minority of the 30 runs, shown through outliers. This effect can be clearly observed in Figures 4.7 and 4.8 This is due to one or two model trees in the ensemble performing exceptionally poor and biasing the average prediction significantly enough to compromise the ensemble as a whole. As the ensemble is increased, the effective weight such a tree has on the ensemble decreases, however with each additionally induced model tree there is the risk of adding another poorly performing tree. Furthermore, the computational costs associated with growing ensembles large enough to reduce this effect is quite high. It is for this reason that ensemble pruning is implemented in MTF as opposed to increasing the ensemble size. Ensemble pruning is performed via the validation set and retroactively in the induction process of MTF.

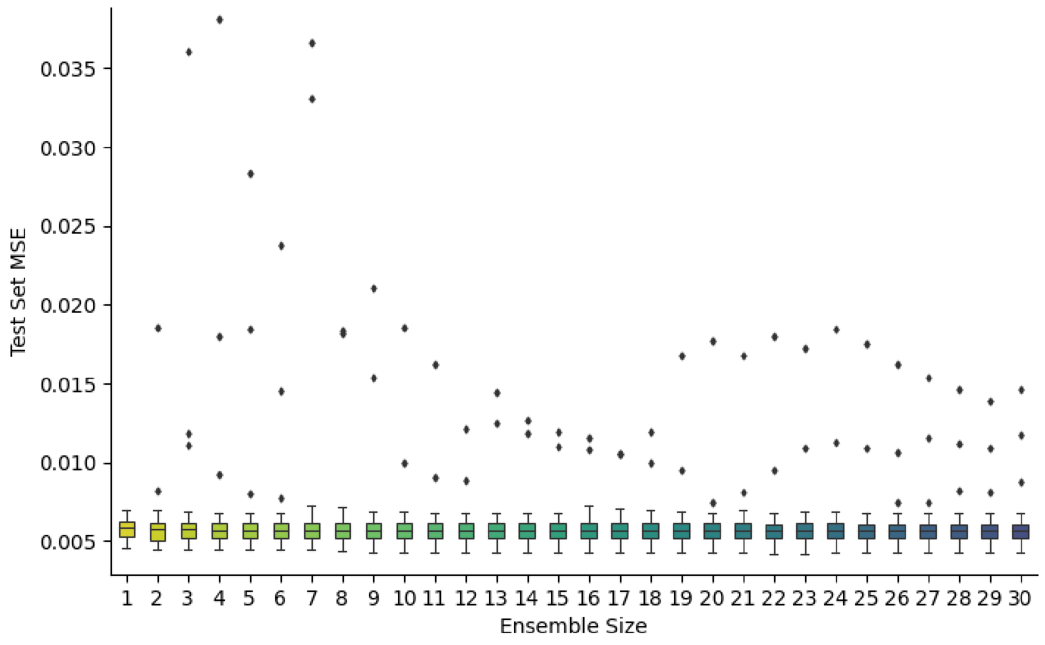


Figure 4.6: Abalone: MSE vs ensemble size of MTF.

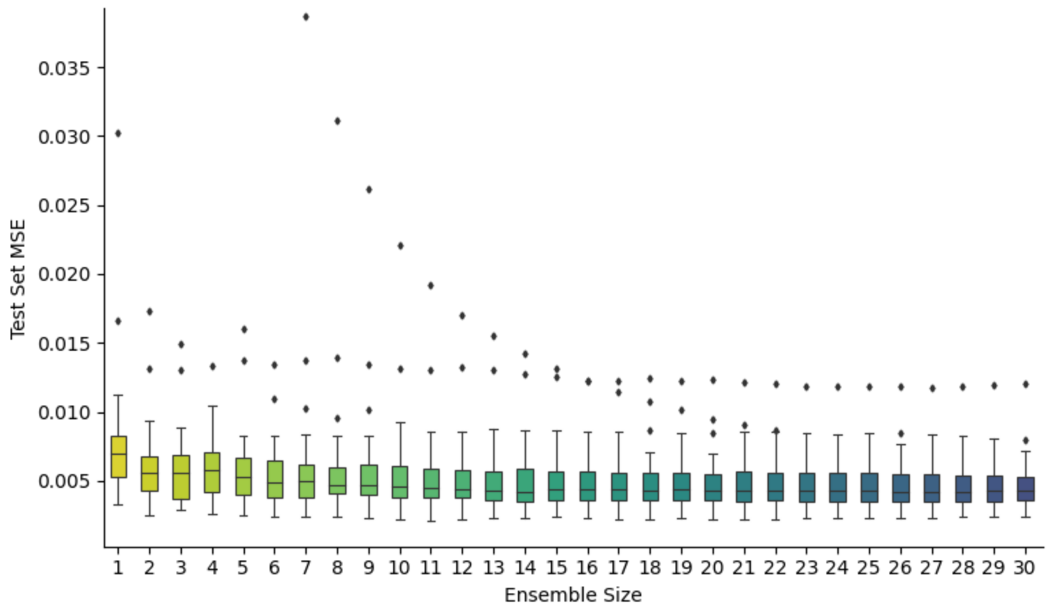


Figure 4.7: Auto-MPG: MSE vs ensemble size of MTF.

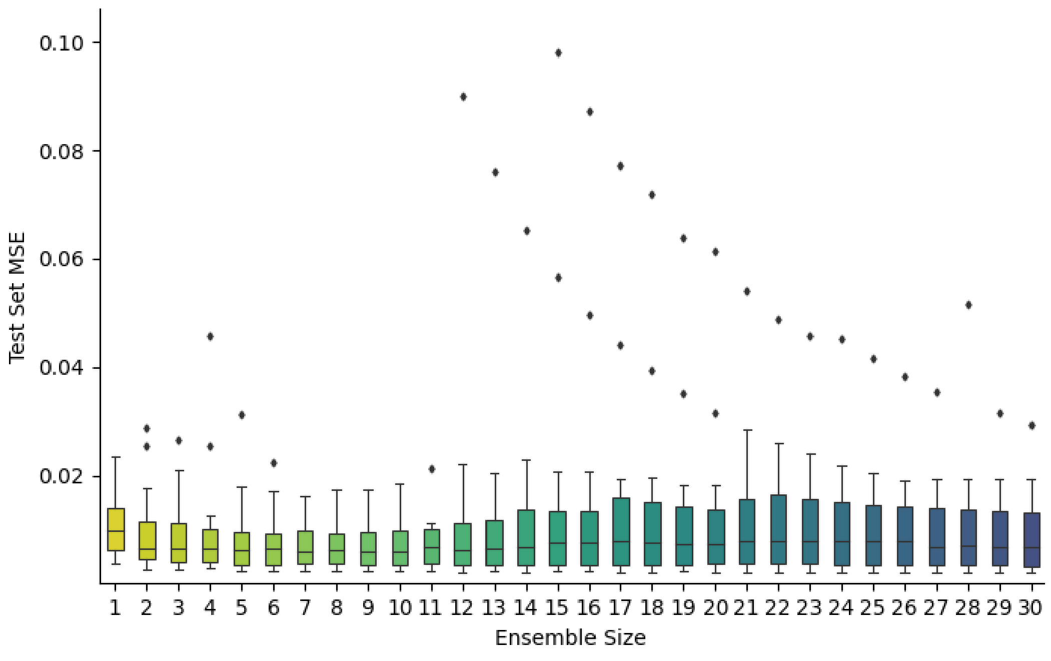


Figure 4.8: Boston Housing: MSE vs ensemble size of MTF.

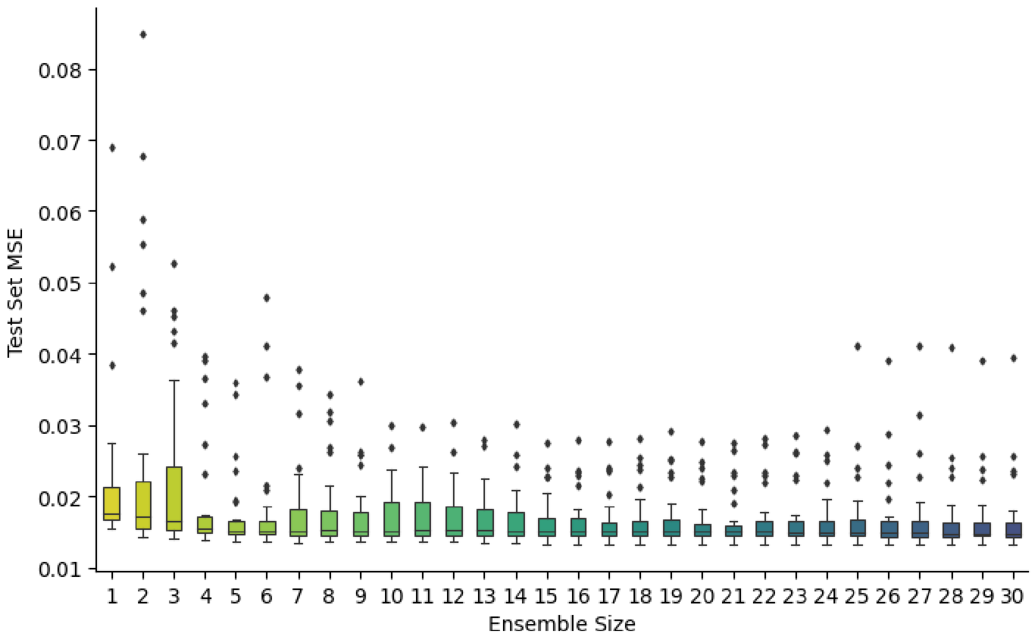


Figure 4.9: California Housing: MSE vs ensemble size of MTF.

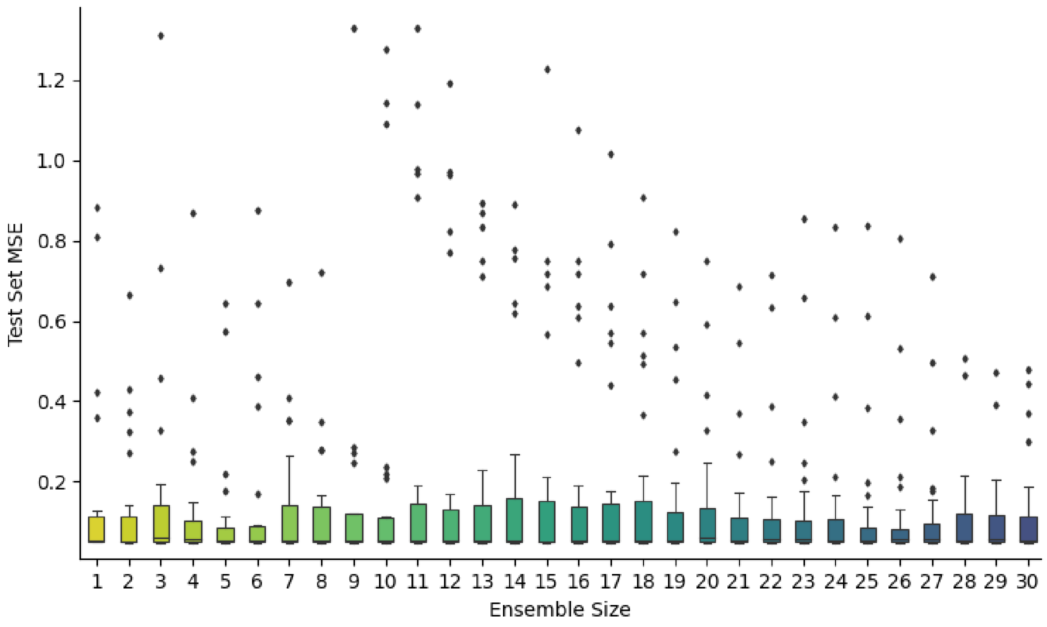


Figure 4.10: CASP: MSE vs ensemble size of MTF.

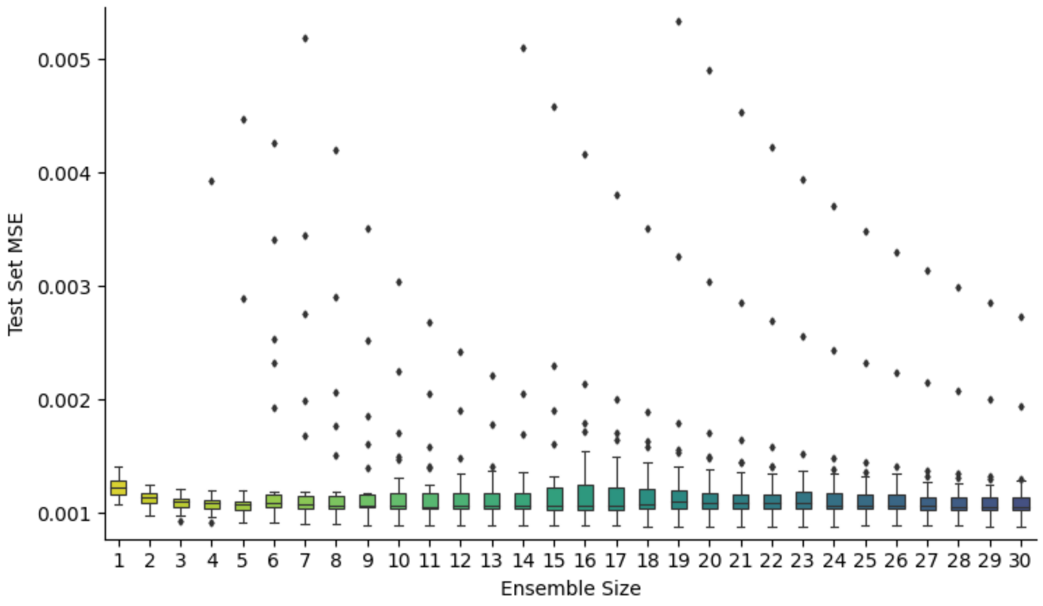


Figure 4.11: Elevators: MSE vs ensemble size of MTF.

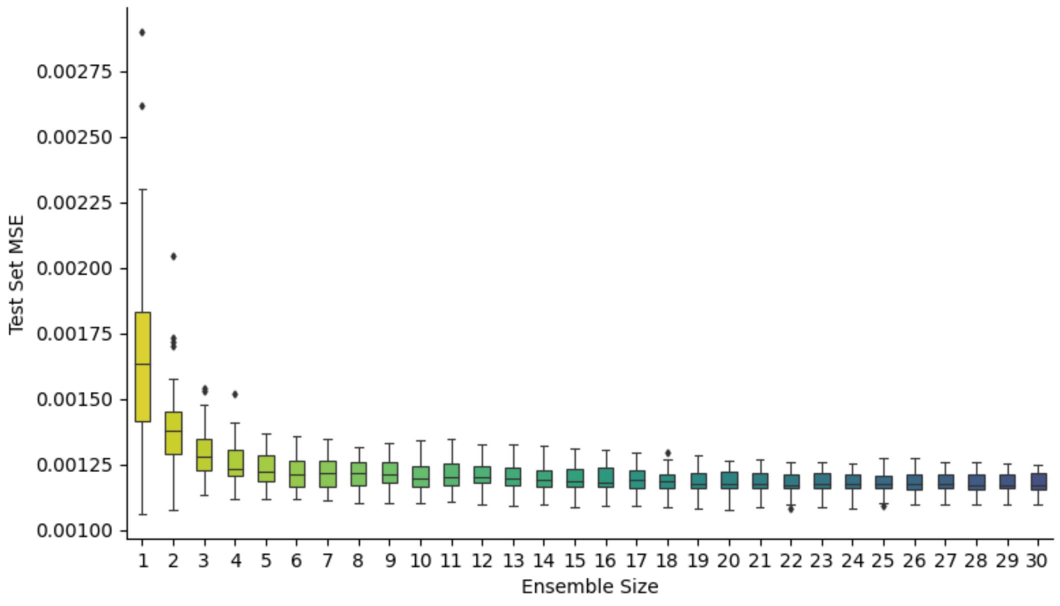


Figure 4.12: Friedman Artificial Domain: MSE vs ensemble size of MTF.

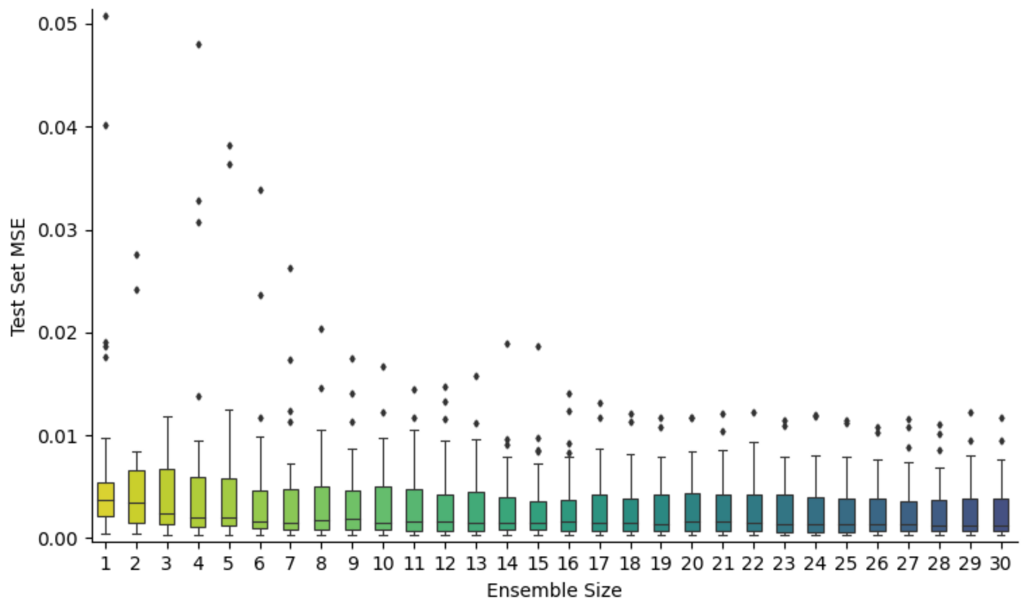


Figure 4.13: Machine CPU: MSE vs ensemble size of MTF.

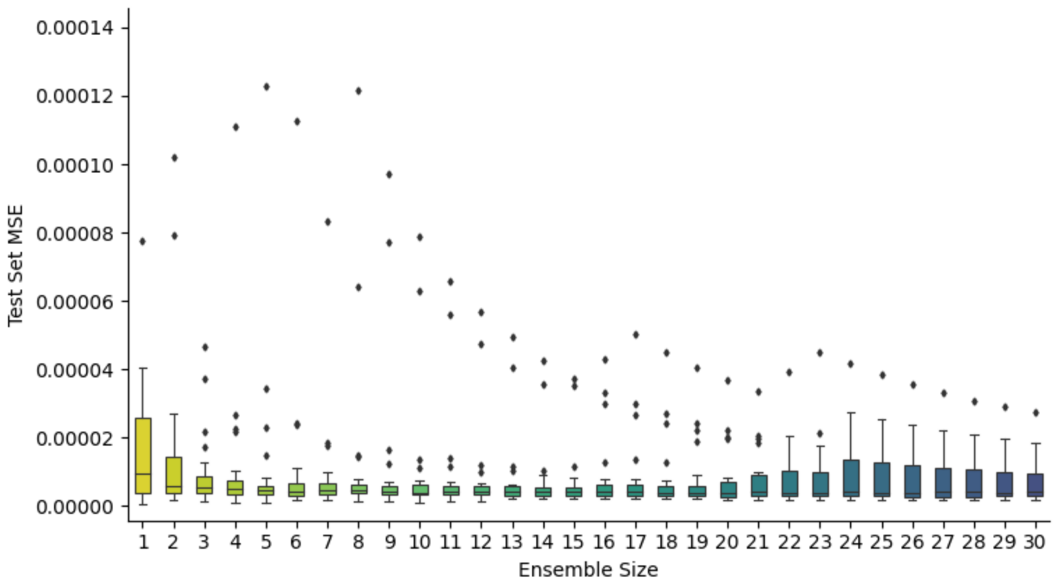


Figure 4.14: MV Artificial Domain: MSE vs ensemble size of MTF.

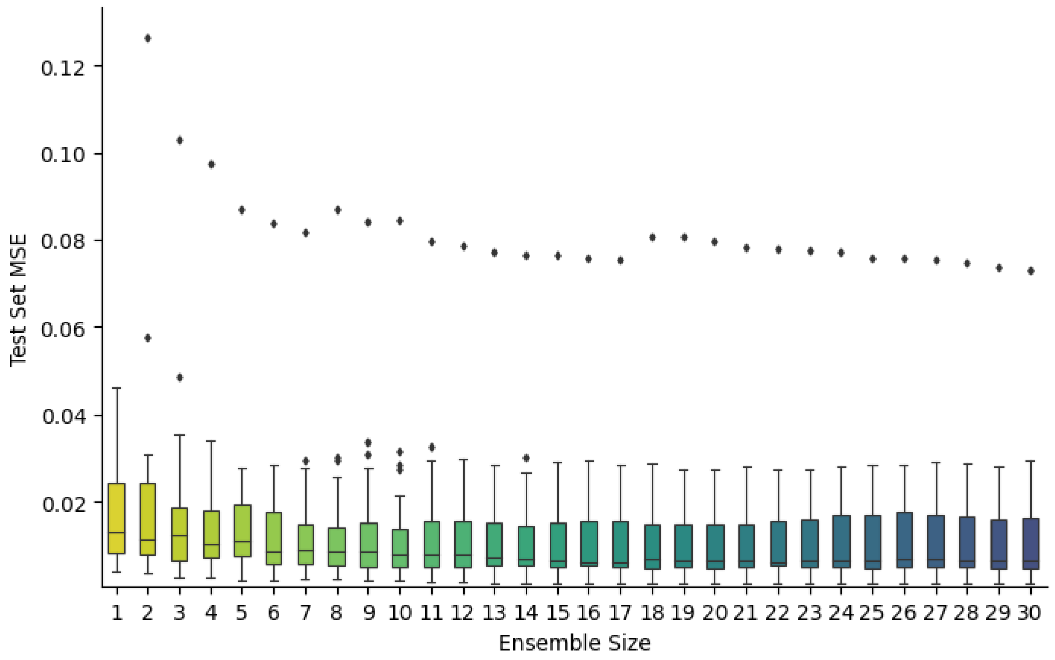


Figure 4.15: Servo: MSE vs ensemble size of MTF.

The results shown in this section confirm that the use of decorrelated model trees in an ensemble effectively reduces the high variance error of MT and produced a lower MSE as an ensemble. The ensemble size at which the MSE starts to converge is lower than initially anticipated. The error in prediction for the majority of datasets plateaus off by an ensemble size of 20, excluding the outlier results which are handled through ensemble pruning. Compared to random forest models which typically start with a minimum ensemble size of 100.

The reasons hypothesised for the smaller required ensemble size compared to other tree-based ensembles such as random forest can be explained through the bias and variance dilemma. The base learners of random forest, which are regression trees with constant values within a leaf node, exhibit higher bias errors compared to model trees due to the reduced complexity. Therefore, a larger number of base learners are required to fit data that tends to be nonlinear in its composition. Model trees, which exhibit extremely low bias errors, when ensembled start with a lower combined bias error and therefore converge quicker with each additional base learner in the ensemble. The number of model trees required to significantly reduce the high variance error is minimal in comparison to the number of regression trees required to reduce a bias error together with the variance error. Decorrelation of decision trees in an ensemble is an extremely effective method to reduce a high variance error. A single model tree already houses enough complexity to adequately model the majority of the datasets, however, multiple trees are still required to help reduce the variance error.

In the conceptualisation of bagging predictors, Breiman suggested for decision tree ensemble methods that incorporate bagging for regression, the ensemble size is optimal at a value around 25 [5]. After this number, the advantages of bagging started to diminish, which is confirmed by the results in this section.

4.2.5. Conclusion

The best performing hyperparameter values for MTF on each of the ten benchmark datasets are listed in Table 4.2 together with the attributes of the dataset.

Table 4.2: Chosen hyperparameter values of MTF for each benchmark dataset.

Dataset	Maximum Polynomial Order	Maximum Tree Depth	Number of features	Number of samples
Abalone	2	3	1N 7C	4,177
Auto-MPG	3	2	3N 4C	398
Boston housing	3	2	0N 13C	506
California housing	2	4	0N 8C	20,460
CASP	2	2	0N 9C	45,730
Elevators	2	6	0N 18C	16,559
Friedman artificial domain	2	6	0N 10C	40,768
Machine CPU	1	3	0N 6C	209
MV artificial domain	2	6	3N 7C	40,768
Servo	3	2	4N 0C	167

Model trees that are confined to a maximum tree depth of two are referred to as stumps. In the case of datasets that are relatively small in size, i.e. less than a thousand samples, it is clear that MTF favours stumps for its base learners. The only exception to this is for Machine CPU where the maximum tree depth which produces the best result is only one greater. For datasets with few training samples, it seems that partitioning the feature space through too many splits result in degraded performance. It is hypothesised that the reason for this is the likeliness of decision trees to overfit data. For larger datasets, there are more training samples to generalise for during induction, which allows for deeper trees to be grown without these consequences. The exception to this is tree stumps being favoured for CASP, possibly due to an increase in noise over other large datasets.

Chapter 5

Experimental Results

In this chapter, the Model Tree Forest (MTF) algorithm is tested against five comparative models on ten benchmarking datasets. The results provide insight into the potential and flexibility of MTF to produce an adequate performance on varying problems. The chapter is outlined as follows. Section 5.1 discusses the empirical method used to ensure testing is performed fairly. The empirical method comprises the selection and tuning of competing models as well as the selection of the datasets on which testing is performed. The hyperparameters of each comparative model are listed and tuned for to ensure the comparative models perform to the best of their ability. Next, the test results are shown in Section 5.2. Multiple error metrics are presented to provide conclusive results as to which models performed best under different circumstances. This section is concluded with the statistical significance of the findings through statistical tests. Finally, the conclusions this chapter brings forth regarding the performance of MTF is discussed in Section 5.3.

5.1. Empirical method

This section discusses the series of experiments designed and followed to evaluate the performance of MTF against other established regression techniques in terms of their predictive accuracy. The datasets used for testing are discussed in Section 5.1.1. Thereafter the selection and tuning of competing models are discussed in Section 5.1.2. MTF does not require any further tuning as this was already discussed in the previous chapter. The empirical method used for the testing of MTF strives to be as elaborate as possible through repetition and variation.

5.1.1. Datasets

To ensure the testing of MTF is thorough, ten datasets each with a unique composition are obtained from the UCI machine learning repository and the L. Torgo Repository [24]. Six datasets from the original testing of GPMCC are selected and an additional four datasets are chosen to expose MTF to big data problems. All datasets contain a single numerical target variable that a given regression model is assigned with predicting based on a set of feature variables that may be nominal and/or continuous.

Large datasets are defined here as datasets comprising more than 10,000 samples. In the original testing of GPMCC, only one large dataset was present from the UCI machine learning repository, namely Elevators. The other two large datasets were artificially created. At the time, there was a lack of sufficient large databases covered by the UCI machine learning repository [18]. For the testing of MTF, there are additional large benchmarking datasets available namely Physicochemical Properties of Protein Tertiary Structure (CASP) and California housing. Two large artificial datasets are also used in the testing of MTF which are the MV artificial domain and Friedman artificial domain. The first was used by Torgo in comparing the performance of PLT and MARS, two favourable regression techniques at the time [23]. The second artificial dataset was used by Friedman in the conceptualisation of MARS as well as the evaluation of bagging predictors [5, 10].

The ten datasets were chosen to ensure all degrees of dimensionality are covered during testing. The number of features ranges from a minimum of 4 to a maximum of 18. tree-based methods are expected to have the advantage of embedded feature selection on the higher dimensional datasets. The datasets have varying degrees of linearity, as shown in the previous chapter through the tuning of the maximum polynomial order of MTF. The difference in the linearity of the datasets ensures the ability of MTF to adapt to varying degrees of complexity in data is tested thoroughly. The ten datasets are listed below in Table 5.1 with their size and dimensionality.

Table 5.1: Datasets used for comparative testing.

Dataset	Number of features (N = Nominal, C = Continuous)	Number of samples
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

Datasets are processed for each test run. Missing values are all dropped, only California housing and Abalone comprised missing values. Feature values are scaled into a range of $[0; 1]$. Finally, the dataset is shuffled randomly and divided into three sets, the training set, validation set and test set in an 80:10:10 ratio. This is repeated a total of 30 times per dataset and each model is tested under this configuration.

5.1.2. Selection and tuning comparative models

The focus during testing is on the comparison of tree-based methods and how the introduction of ensembling and decorrelation can produce improved performance. However, comparative models are not limited to tree-based 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 reduce the problematic high variance error it exhibits?
- Can the decorrelation of model trees within an ensemble further reduce the high variance error?
- Is the complexity of MTF unfavourable for datasets that tend to be more linear in their composition?
- Is MTF able to outperform competing state-of-the-art regression techniques which are catered towards nonlinear problems?

The first question is addressed through a baseline M5 model tree as well as the M5 Ensemble (M5E). MTF will be directly compared to, amongst other models, a baseline model tree and an ensemble thereof to illustrate the effectiveness of ensembling model trees to produce improved performance. The second question is also answered through the use of the M5 and M5E models as M5E incorporates a simple bagging ensemble technique where trees are not decorrelated. This stand opposed to MTF where the base learners in the ensemble are decorrelated through the use of randomised splitting feature sets. The third question is addressed by the addition of RF as well as a feed-forward neural network (NN). Both these models produce favourable results in varying degrees of data complexity if tuned for properly. Neural networks can be tuned by adding or removing nodes and/or hidden layers to increase or decrease their complexity to suit the predictive task at hand. For the final question, a support vector regression (SVR) model is added to the list of comparative models. An SVR with a Gaussian kernel function can effectively model nonlinear problems. Furthermore, the other models are also considered well suited for such problems. This list of comparing models serves as an adequate comparison for MTF on multiple fronts[1].

Each comparative model requires thorough tuning to ensure it performs to the best of its ability under each predictive problem. Therefore, similar to MTF in the previous chapter each model's hyperparameters are tuned individually for each dataset. However, to keep comparisons fair between MTF and the comparing models, the models are not

[1]Due to the availability of the model itself, GPMCC is omitted from testing, however using similar competing models/datasets as the original testing provide a reasonable comparison and still allows for conclusions to be drawn, be it to a limited extent.

trained per each test run. Instead, each model is trained through a k-fold cross-validation approach per dataset before final testing. This ensures that both MTF and the remainder of the models are tuned for under the same set of circumstances i.e. the dataset as a whole before the testing. Each model is therefore exposed to the same inductive biases through the same training samples. Recall from Chapter 4 that MTF was trained on 80% of the training samples, chosen by random, during tuning for 30 repeated runs. If the value of k is chosen as 5 for the cross-validation, each model is likewise trained on 80% of the training samples and also exposed to the dataset in its entirety before testing. It is important to note that this does not affect the model's generalisation ability as during testing a portion of the dataset is still held out for evaluation to expose each model to unseen data. Table 5.2 lists each model and their applicable hyperparameters.

Table 5.2: Hyperparameters of each comparative model.

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

Of the five models, M5 is the only model without any hyperparameters applicable for tuning. As with the testing of GPMCC, M5 is implemented through the Cubist package, a commercially available version of Quinlan's M5 model tree. Additionally, Cubist is also used to implement the ensemble of bagged M5 trees. The size of the ensemble is referred to as the number of committees. In some applications, the predictive accuracy of M5E can be improved by combining a nearest-neighbour technique to the output calculation. This brings rise to the second parameter, neighbours, which also requires tuning. The possible values of neighbours are 1, 3, 5, 7 and 9. The number of committees may range from 1 to 100. Since the two hyperparameters of M5 have a very limited range of values, grid-search optimisation was used without exceeding computational limitations and ensuring the best possible set of hyperparameter values were obtained. The remaining hyperparameters of the comparative models are all tuned using Bayesian optimisation as the search space is too large to cover through grid-search optimisation.

The next comparative model is the NN. Similar to the testing of GASOPE, a simple feed-forward neural network with a single hidden layer is used. The size of the hidden layer is a product of the complexity of the problem at hand and is tuned for each dataset separately. For the activation function, a leaky rectified linear unit is implemented. A leaky ReLU activation function ensures that the NN converges with each training sequence. The NN is trained through Stochastic Gradient Descent (SGD), which requires the specification of a learning rate. Finally, the batch size used for training the NN also influences the quality of the model. To train the NN as best possible, varying learning rates, batch sizes and hidden layer sizes are tuned through Bayesian optimisation. With each training epoch, the error metric on both the training set as well as validation set is measured. Once the error metric converges or overfitting is detected, training is halted. The NN is deemed to overfit once the error on the validation set stops decreasing whilst the error on the training set continues to decrease.

The final tree-based comparative model, RF, has the highest number of hyperparameters to tune for. The hyperparameters are ensemble size, minimum leaf samples, maximum tree depth, splitting features. All of which are also present in MTF. Finally, the parameters of the SVR model are the values for C and gamma, similar to the testing of GASOPE in Chapter 3.

5.2. Experimental Results

The results of testing are listed and discussed in this section. For each of the six models, two error metrics are compared on the training and test set of each dataset. The error metrics are root mean square error (RMSE) and mean absolute error (MAE). The choice of error metrics and the results thereof are discussed in Section 5.2.1. Thereafter, Section 5.2.2 evaluates the statistical significance of the results through the non-parametric Friedman test together with the post-hoc pairwise Bonferroni-Dunn test.

5.2.1. Comparative model performance

Each dataset was split into training, validation and test sets. Optimisation of model hyperparameters was performed before testing. The training set was used to train each model. The validation set was only utilised by the NN and MTF. The NN employs the validation set for overfit detection during training whilst MTF utilises the validation set post-training to prune the ensemble as a whole. The test set is used to expose each model to unseen data, to evaluate the generalisation capabilities of a given model. The datasets were split 80:10:10. All tests were repeated 30 times over, with the dataset being shuffled for each test run.

The error metrics used for the evaluation of the performance of a given model are RMSE MAE. Although RMSE is the favoured metric for model evaluation, MAE provides valuable insight into the performance of model's without penalising outlier results as heavily. For each test run the RMSE and MAE values are computed on both the training and test set. Comparing both the training and test set error metrics allows for the identification of overfitting. Models with a relatively low training set error and higher test set error are likely to overfit the data at hand. Only the test set error metrics are used for statistical tests in the following section.

Tables 5.3 and 5.4 lists the results of all six models on every dataset over the 30 repeated tests through the median of the error metric values. The median values are listed instead of mean values due to a minority of outlier results heavily skewing the mean values thereby degrading the interpretability. The mean values are available in Appendix 10 in Tables 10.1 through 10.4. It is important to note that the use of median values instead of mean values for the portrayal of the test results does not influence of the outcomes of the statistical test. The statistical tests take the results of each run into account.

Comparing the performance of each model according to the RMSE and MAE scores shows that there is a large discrepancy between the two error metrics for tree-based models. M5 and M5E are shown producing the worst performance by a considerable margin according to RMSE. However, evaluating the models according to their MAE scores, both M5 and M5E are considerably more competitive. This is explained through the poor extrapolation qualities that model trees exhibit [18]. RMSE penalises outlier results more severely than MAE, that leads to the poorer RMSE scores for M5 and M5E compared to the other models which extrapolate better. For the non-parametric statistical tests, both MAE and RMSE values are used during the rankings of the results.

The the poor extrapolation qualities of the model tree methods are considerably reduced in MTF. MTF can produce competitive scores on both MAE as well as RMSE. Furthermore, evaluation of the RMSE scores shows that RF is the best performing model for six out of the ten datasets. It is reinforced through these results that RF does not suffer from the same extrapolation problems as model trees as outlier values are quarantined into leaf nodes during induction. This is possible as RF can have a minimum leaf value of one, whereas model trees require multiple samples within a leaf node to properly fit a multivariate model describing the leaf's output. RF thereby efficiently isolates the outlier values without degrading the performance of the model.

Table 5.3: Comparison of all six models on the first five datasets.

Dataset	Model	Training median		Test median	
		RMSE	MAE	RMSE	MAE
Abalone	M5	0.074835	0.052176	0.076114	0.053630
	M5E	0.074961	0.052110	0.075983	0.053379
	MTF	0.005510	0.052189	0.005676	0.053613
	NN	0.005487	0.052538	0.005629	0.053534
	RF	0.004102	0.044790	0.005744	0.053495
	SVR	0.005241	0.056161	0.006326	0.059169
Auto-MPG	M5	0.070832	0.050789	0.081180	0.058785
	M5E	0.065755	0.047523	0.077815	0.057379
	MTF	0.003760	0.044383	0.004644	0.051011
	NN	0.004668	0.050355	0.005193	0.052833
	RF	0.000737	0.018788	0.004832	0.050543
	SVR	0.004764	0.052498	0.004990	0.054921
Boston housing	M5	0.059977	0.042526	0.076849	0.054649
	M5E	0.051886	0.037964	0.066695	0.047304
	MTF	0.005374	0.044690	0.006150	0.051419
	NN	0.004325	0.047758	0.006021	0.052548
	RF	0.000686	0.017692	0.004069	0.044799
	SVR	0.028366	0.114317	0.029819	0.113600
California housing	M5	0.089878	0.057831	0.102522	0.064730
	M5E	0.082529	0.052641	0.093324	0.058811
	MTF	0.014213	0.082872	0.014836	0.084378
	NN	0.014873	0.085670	0.014858	0.086167
	RF	0.001409	0.024932	0.010467	0.068102
	SVR	0.012889	0.079686	0.013704	0.081421
CASP	M5	0.196568	0.136587	0.208861	0.144950
	M5E	0.175114	0.123580	0.187242	0.131827
	MTF	0.054299	0.190272	0.054908	0.190900
	NN	0.049117	0.174477	0.049652	0.175320
	RF	0.003776	0.041792	0.027135	0.113007
	SVR	0.015286	0.092911	0.031738	0.128393

Table 5.4: Comparison of all six models on the second five datasets.

Dataset	Model	Training median		Test median	
		RMSE	MAE	RMSE	MAE
Elevators	M5	0.031972	0.023461	0.033125	0.024294
	M5E	0.031422	0.023088	0.032382	0.023529
	MTF	0.001831	0.022883	0.003405	0.025486
	NN	0.000934	0.022519	0.000950	0.022596
	RF	0.000229	0.010387	0.001589	0.027721
	SVR	0.008210	0.081129	0.012114	0.093229
Friedman artificial domain	M5	0.032618	0.025897	0.033605	0.026728
	M5E	0.031495	0.025049	0.032159	0.025617
	MTF	0.001074	0.025992	0.001110	0.026485
	NN	0.001089	0.026271	0.001105	0.026423
	RF	0.000306	0.013420	0.001614	0.031792
	SVR	0.010444	0.088321	0.024795	0.128195
Machine	M5	0.027888	0.017556	0.044771	0.028120
	M5E	0.025303	0.016318	0.040548	0.025280
	MTF	0.000744	0.015702	0.001402	0.024449
	NN	0.001706	0.031165	0.002045	0.030528
	RF	0.000408	0.009469	0.000702	0.017116
	SVR	0.014072	0.082980	0.006432	0.075289
MV artificial domain	M5	3.003e-04	1.333e-04	3.337e-04	1.367e-04
	M5E	2.257e-04	1.222e-04	2.470e-04	1.243e-04
	MTF	1.999e-06	4.061e-04	2.732e-06	4.295e-04
	NN	6.220e-07	6.629e-04	6.449e-07	6.669e-04
	RF	4.036e-07	2.830e-04	1.535e-06	6.789e-04
	SVR	1.805e-03	3.520e-02	1.826e-03	3.529e-02
Servo	M5	0.045259	0.025987	0.055805	0.037230
	M5E	0.045259	0.025987	0.055805	0.037230
	MTF	0.003817	0.034379	0.004534	0.042299
	NN	0.008128	0.065783	0.006947	0.067827
	RF	0.000772	0.012659	0.003838	0.031874
	SVR	0.009689	0.071009	0.006620	0.066440

5.2.2. Statistical significance

Statistical tests are performed on the test results to determine which differences in performance are considered significant. The statistical test used to first reject the null-hypothesis (All models are equal in their performance based on the results) is the non-parametric Friedman test. Once the null-hypothesis is rejected, pairwise tests are performed to determine models produced results that are statically dissimilar. The post-hoc test performed is the Bonferroni-Dunn test.

To perform the Friedman test, each test run out of the total 30 are ranked twice. Once according to RMSE scores and once more according to MAE scores. This is performed individually for each dataset as well as all ten datasets combined. At a significance level of 5%, each of the eleven Friedman tests resulted in a rejection of the null-hypothesis and post-hoc tests performed accordingly.

The results of the post-hoc tests are portrayed by Figures 5.1 through 5.11. The critical diagrams are interpreted as follows, each model is placed on a scale of one to six according to their average performing rank over the 30 tests. The black bars group models which are **not** considered statistically dissimilar. For example, Figure 5.1 shows that on average, NN was ranked the best performing model whilst SVR was the worst. Furthermore, NN, MTF and RF are considered to produce results that do not sufficiently distinguish them. The same applies to SVR, M5 and M5E. However, MTF, NN and RF are considered to be statistically better performing models than SVR, M5 and M5E. For the

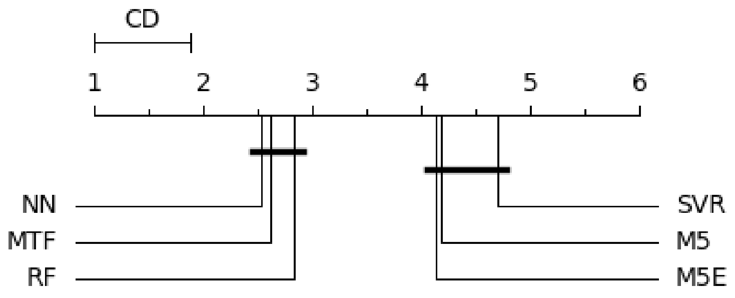


Figure 5.1: Critical difference diagram on Abalone.

remaining critical diagrams, notice the difference in the performance of MTF compared to the more simple model tree ensemble of M5E. MTF is shown to produce the same or better performance over M5E for eight out of the ten datasets, four of which are statistically better. RF produced one of, if not the best, ranked results for eight datasets. The RF inspired modifications brought into MTF to improve its performance over a single model tree and bagged ensemble of model trees are shown to produce a significantly increased performance. However, RF still matches or often outperforms MTF for the majority of cases. The only case where MTF outperforms RF is on the Friedman artificial domain dataset, which is a large dataset with a nonlinear composition. RF outperforms MTF on Boston housing, California housing, CASP and Servo. It is worth noting that MTF produced very competitive results on Machine CPU, a piecewise linear dataset on which GPMCC did not have favourable results [18]. This portrays the flexibility of MTF to produce adequate results on both linear as well as nonlinear data, given that it is properly tuned to the problem at hand.

As expected, M5E is shown to either match the performance of a single M5 model tree or considerably increase its performance for all cases. This illustrates that the flaws of model trees, which include their tendency to overfit, are addressed through the use of ensemble methods. Other flaws, such as poor extrapolation, can further be improved on by the use of decorrelated base learners in an ensemble.

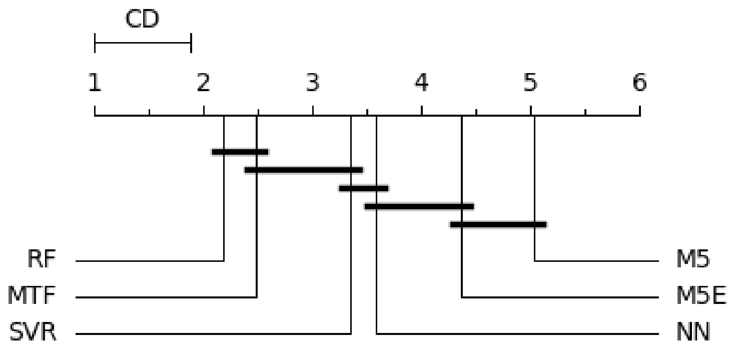


Figure 5.2: Critical difference diagram of Auto-MPG.

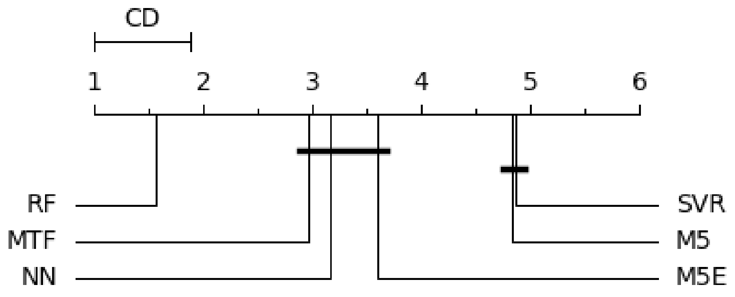


Figure 5.3: Critical difference diagram of Boston housing.

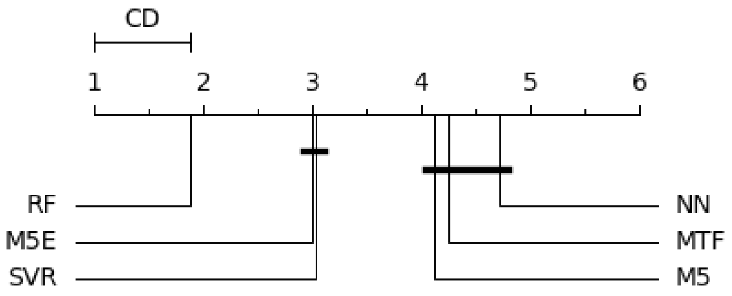


Figure 5.4: Critical difference diagram of California housing.

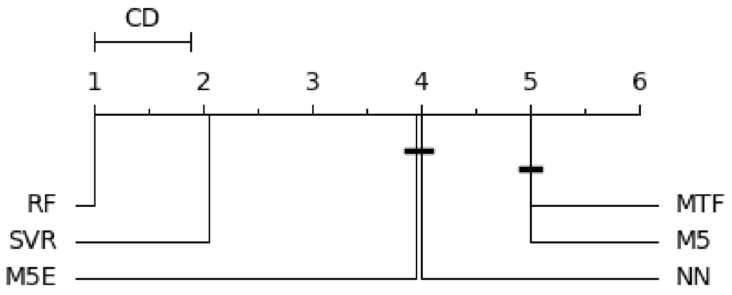


Figure 5.5: Critical difference diagram of CASP.

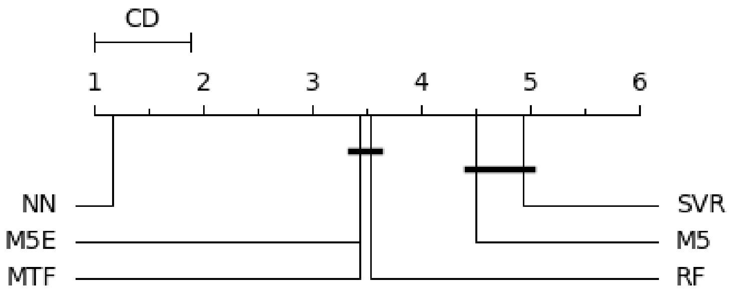


Figure 5.6: Critical difference diagram of Elevators.

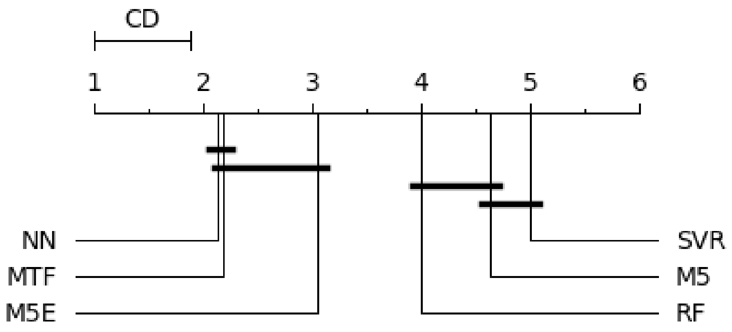


Figure 5.7: Critical difference diagram of Friedman artificial domain.

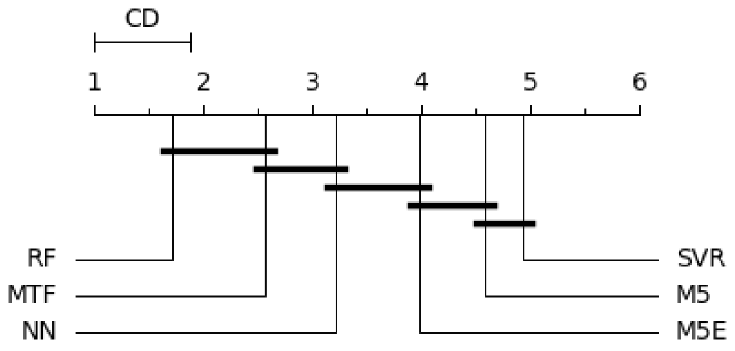


Figure 5.8: Critical difference diagram of Machine CPU.

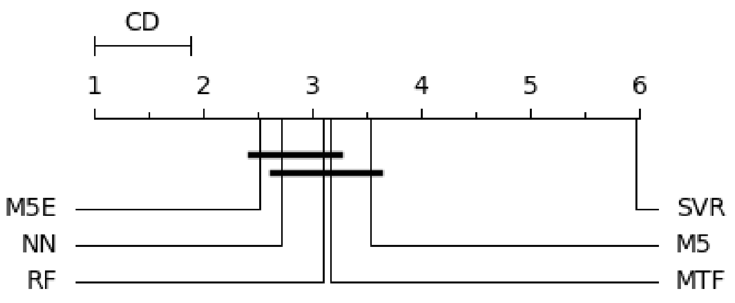


Figure 5.9: Critical difference diagram of MV artificial domain.

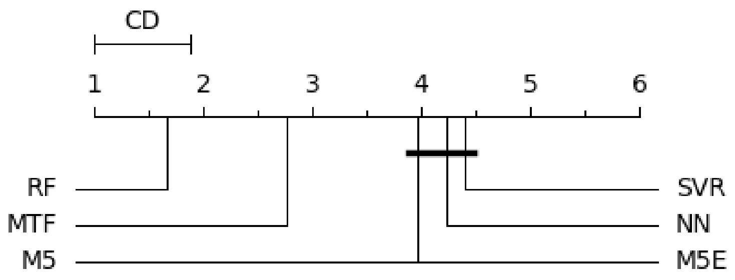


Figure 5.10: Critical difference diagram of Servo.

The three datasets which comprised more nonlinear relationships are Auto-MPG, Boston housing and Servo, as shown through the tuning of MTF in Section 4.2.2. All three of these datasets are considered small datasets. For Auto-MPG and Boston housing, MTF was tied for the best-ranked model. For Servo, MTF was ranked second best. Compared to NN and SVR, which are very popular solutions to nonlinear problems, MTF outperformed them significantly four out of the six possible times and matched them for the remainder.

Finally, Figure 5.11 shows the combined ranking for all ten datasets. MTF and NN produced very comparable results across all test cases. RF was a clear best performer, whilst SVR failed to improve on the results a single model tree achieved.

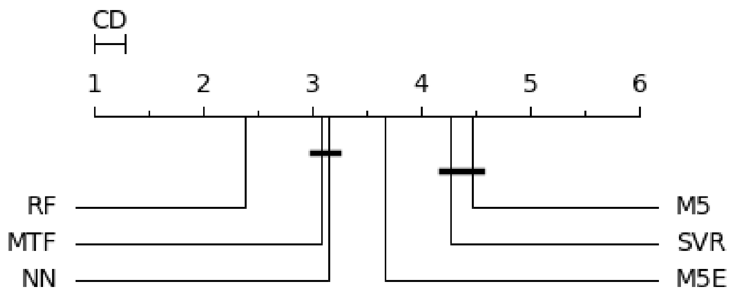


Figure 5.11: Critical difference diagram on the combined testing runs of all ten datasets.

5.3. Conclusion

This chapter aimed to investigate the advantages ensemble learning can offer model trees. Additionally, the performance of model trees and model tree ensembles are placed into perspective against other state-of-the-art regression techniques. Recall the questions posed in Section 5.1.2. It was showed that the bagging of M5 models guarantees an improvement in the performance of the base learner, a model tree. Furthermore, the decorrelation of model trees within an ensemble (MTF) significantly increased its robustness to some of the flaws which model trees and previous ensembles thereof exhibited. These flaws included poor extrapolation and a tendency to overfit data as evident through the high variance error these models inherit. The incorporation of RF inspired techniques within model tree ensembles resulted in MTF being able to compete with a NN and outperform an SVR. Finally, MTF was also shown producing competitive performance on data that was very linear in its composition. This proved 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.

The test results brought fourth insights into the ideal configuration of a model tree depending on the problem at hand. In the case of small datasets, model trees with a single split, i.e. stumps, as base learners in an ensemble tended to perform better than the competing tree-based models which incorporated deeper trees. Recall that MTF was tuned to induce stumps for Auto-MPG, Boston housing and Servo. In smaller sized datasets, there are fewer possible splits for tree-based approaches to follow. For model trees, the number of splits is even less due to the higher number of minimum leaf samples required to constitute a split. By inducing deeper trees on a smaller pool of viable splits, decision trees are expected to overfit the dataset. This is because by consecutively reducing the number of samples in a node, the leaf node is left with fewer observations to induce the output model on, making it more susceptible to overfitting the noise present in these training instances. MTF solves this issue through the use of stumps, minimising the splits required to describe the relationships of the dataset and ensuring there are a larger number of samples in a given leaf node. Additionally, due to the increased complexity of the higher-order polynomial models within the leaf nodes of MTF, MTF can divide the feature space a single time and still adequately model the intricate patterns within the data.

For large datasets where MTF was given the freedom of inducing deep trees, namely Friedman artificial domain and MV artificial domain, MTF was tied for the best performing model. With each consecutive split, the likelihood of a node encapsulating a large number of samples is higher than in the case of a small dataset. Thereby the higher-order polynomial functions in the leaf nodes of the model tree are given sufficient training instances to model the output off of. In addition to this, the feature space of the data is divided in such a manner that different relationships within the data are distinguishable and modelled separately from each other. Whereas with stumps, the feature space is only divided a single time per base learner in the ensemble. The two large datasets, California housing and CASP, where MTF produced poor results are cases where the maximum tree depth was limited to four and two respectively. It is hypothesised that these two datasets have a significant increase in noise over the other large datasets which MTF is unable to handle as effectively as the comparative models. This would explain the tuning results in the previous chapter, which showed deeper trees producing error metrics with exceptionally large variability. Furthermore, the comparative models less complex in their composition, i.e. RF and SVR, as opposed to MTF, M5E and NN were the considerably best ranking models on these datasets. The other two large datasets where MTF produced competitive performance are artificially induced and as a result, have significantly less noise compared to California housing and CASP. The linear base models of MTF were expected to improve performance in cases with noise, but additional measurements are required to increase the robustness of MTF to noise. MTF is highly complex due to the higher-order polynomial functions used to predict the output. Therefore, it is recommended that noisy data be pre-processed before MTF is exposed to it.

Chapter 6

Conclusion

6.1. Summary

TBD

6.2. Future Research

TBD

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] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025510005554

[3] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's razor," *Information Processing Letters*, vol. 24, no. 6, pp. 377 – 380, 1987. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0020019087901141

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

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

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

[7] 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.

[8] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B978044450673350110X

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

[10] J. H. Friedman, "Multivariate Adaptive Regression Splines," *The Annals of Statistics*, vol. 19, no. 1, pp. 1 – 67, 1991. [Online]. Available: https://doi.org/10.1214/aos/1176347963

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

[12] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, ser. Springer Series in Statistics. Springer New York, 2013. [Online]. Available: https://books.google.co.za/books?id=yPfZBwAAQBAJ

[13] 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.

[14] G. Louppe, "Understanding random forests: From theory to practice," *arXiv preprint arXiv:1407.7502*, 2014.

[15] G. Moisen, "Classification and regression trees," *In: Jørgensen, Sven Erik; Fath, Brian D.(Editor-in-Chief). Encyclopedia of Ecology, volume 1. Oxford, UK: Elsevier. p. 582-588.*, pp. 582–588, 2008.

[16] 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.

[17] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0096300306010708

[18] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417407003673

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

[20] 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.

[21] S. Singh, "Understanding the Bias-Variance Tradeoff," 2018. [Online]. Available: https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229. [Accessed: 22- Mar- 2020].

[22] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 2951–2959. [Online]. Available: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf

[23] L. Torgo, "Partial linear trees." in *International Conference on Machine Learning*, 2000, pp. 1007–1014.

[24] ——, "Regression datasets," [Online]. Available: https://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html. [Accessed: 15- Aug- 2020].

[25] ——, "Functional models for regression tree leaves," in *International Conference on Machine Learning*, vol. 97, 1997, pp. 385–393.

[26] Y. Wang and I. Witten, "Induction of model trees for predicting continuous classes," *Induction of Model Trees for Predicting Continuous Classes*, 1997.

Chapter 7

DB linearity

Table 7.1: Tuning results of GASOPE for varying maximum polynomial orders on each dataset.

Dataset	Order	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
	1	5.945e-03	1.959e-08	6.237e-03	2.659e-07	15.06
Abalone	2	5.718e-03	1.492e-08	7.750e-03	4.450e-05	21.10
	3	5.827e-03	3.535e-08	8.432e-02	1.361e-01	66.85
	1	5.390e-03	1.508e-07	6.307e-03	1.413e-06	5.455
Auto MPG	2	4.794e-03	1.045e-07	5.750e-03	1.485e-06	6.274
	3	4.802e-03	1.041e-07	5.743e-03	1.459e-06	8.012
	1	7.726e-03	1.135e-06	1.013e-02	2.226e-05	8.442
Boston Housing	2	6.106e-03	5.772e-07	8.938e-03	1.166e-05	12.02
	3	6.211e-03	4.155e-07	9.117e-03	2.059e-05	21.09
	1	1.967e-02	1.065e-07	1.976e-02	6.810e-07	58.18
California Housing	2	1.935e-02	2.956e-07	2.057e-02	9.817e-06	88.87
	3	2.003e-02	8.046e-07	2.081e-02	4.441e-06	389.4
	1	5.961e-02	2.673e-07	5.955e-02	9.002e-07	152.2
CASP	2	5.771e-02	1.681e-07	5.764e-02	1.062e-06	240.5
	3	5.839e-02	3.028e-07	5.841e-02	5.956e-07	1089
	1	1.439e-03	6.562e-10	1.475e-03	6.150e-09	101.3
Elevators	2	1.353e-03	2.664e-09	1.386e-03	7.509e-09	136.5
	3	1.413e-03	4.727e-09	1.439e-03	8.932e-09	571.2
	1	6.862e-03	7.178e-10	6.871e-03	1.161e-08	73.95
Friedman Artificial Domain	2	2.882e-03	6.531e-07	2.877e-03	6.371e-07	128.8
	3	3.101e-03	6.495e-07	3.115e-03	6.606e-07	707.9
	1	9.247e-04	3.028e-08	4.762e-03	5.160e-05	4.408
Machine CPU	2	7.023e-04	2.122e-08	1.313e-02	5.458e-04	5.729
	3	6.972e-04	1.334e-08	1.306e-01	3.897e-01	8.072
	1	1.413e-03	4.426e-06	1.418e-03	4.553e-06	92.00
MV Artificial Domain	2	2.533e-04	1.614e-08	2.530e-04	1.626e-08	128.2
	3	3.856e-04	2.773e-07	3.803e-04	2.538e-07	703.6
	1	1.808e-02	8.123e-06	2.415e-02	7.287e-05	9.630
Servo	2	8.770e-03	2.556e-06	1.391e-02	3.795e-05	15.32
	3	8.514e-03	2.474e-06	1.409e-02	3.852e-05	16.95

Chapter 8

MT maximum tree depth

This is some appendix.

Table 8.1: 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 8.2: 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 8.3: 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 8.4: 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 8.5: 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 8.6: 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 8.7: 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 8.8: Tuning results of MT for varying tree depth on Machine CPU.

Baseline comparison	Depth	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
Disabled	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
	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
Enabled	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
	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 8.9: Tuning results of MT for varying tree depth on MV artificial domain.

Baseline comparison	Depth	$\overline{TMSE}$	σ_{TMSE}	$\overline{GMSE}$	σ_{GMSE}	$\bar{t}$
Disabled	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
	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
Enabled	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
	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 8.10: 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

Chapter 9

MTF ensemble size

This is some other appendix.

Table 9.1: Tuning results of varying ensemble size for MTF on Abalone.

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
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 9.2: 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 9.3: 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 9.4: 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 9.5: 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 9.6: 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 9.7: 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 9.8: 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 9.9: 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 9.10: 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

Chapter 10

Mean test results

Table 10.1: Comparison of RMSE scores of all six models on the first five datasets.

		Training set		Test set	
Dataset	Model	$\overline{RMSE}$	σ_{RMSE}	$\overline{RMSE}$	σ_{RMSE}
Abalone	M5	7.487e-02	7.439e-04	7.655e-02	4.053e-03
	M5E	7.497e-02	6.412e-04	7.638e-02	3.951e-03
	MTF	5.716e-03	5.872e-04	7.967e-03	7.312e-03
	NN	5.521e-03	2.383e-04	5.687e-03	6.901e-04
	RF	4.110e-03	7.358e-05	5.750e-03	6.201e-04
	SVR	5.245e-03	9.294e-05	6.242e-03	5.103e-04
Auto-MPG	M5	6.969e-02	4.315e-03	8.265e-02	1.538e-02
	M5E	6.717e-02	3.143e-03	7.931e-02	1.313e-02
	MTF	3.727e-03	3.145e-04	5.334e-03	2.256e-03
	NN	4.668e-03	5.110e-04	5.649e-03	1.740e-03
	RF	7.234e-04	5.819e-05	5.130e-03	2.328e-03
	SVR	4.714e-03	3.182e-04	5.452e-03	1.955e-03
Boston housing	M5	6.126e-02	7.321e-03	8.553e-02	2.507e-02
	M5E	5.285e-02	2.909e-03	7.295e-02	1.975e-02
	MTF	2.325e+00	1.041e+01	2.053e+00	7.667e+00
	NN	4.795e-03	1.745e-03	7.327e-03	4.759e-03
	RF	6.921e-04	4.475e-05	4.765e-03	2.173e-03
	SVR	2.843e-02	1.141e-03	2.792e-02	7.549e-03
California housing	M5	8.994e-02	8.781e-04	1.027e-01	3.937e-03
	M5E	8.256e-02	5.523e-04	9.378e-02	3.744e-03
	MTF	1.425e-02	3.474e-04	1.562e-02	3.743e-03
	NN	1.474e-02	5.709e-04	1.502e-02	8.810e-04
	RF	1.411e-03	1.571e-05	1.043e-02	7.687e-04
	SVR	1.287e-02	1.397e-04	1.382e-02	9.041e-04
CASP	M5	1.964e-01	7.460e-04	2.083e-01	2.600e-03
	M5E	1.751e-01	5.113e-04	1.873e-01	2.092e-03
	MTF	5.467e-02	9.813e-04	5.509e-02	1.827e-03
	NN	4.927e-02	1.092e-03	4.988e-02	1.719e-03
	RF	3.778e-03	2.425e-05	2.711e-02	8.643e-04
	SVR	1.528e-02	8.673e-05	3.162e-02	9.892e-04

Table 10.2: Comparison of RMSE scores of all six models on the second five datasets.

		Training set		Test set	
Dataset	Model	$\overline{RMSE}$	σ_{RMSE}	$\overline{RMSE}$	σ_{RMSE}
Elevators	M5	3.202e-02	4.119e-04	3.356e-02	1.241e-03
	M5E	3.144e-02	2.533e-04	3.250e-02	9.241e-04
	MTF	2.789e-03	2.625e-03	8.441e-03	1.492e-02
	NN	9.488e-04	6.544e-05	9.688e-04	8.033e-05
	RF	2.288e-04	2.842e-06	1.598e-03	1.171e-04
	SVR	8.208e-03	4.832e-05	1.211e-02	5.740e-04
Friedman artificial domain	M5	3.264e-02	1.635e-04	3.352e-02	2.892e-04
	M5E	3.148e-02	1.272e-04	3.214e-02	3.339e-04
	MTF	1.074e-03	1.808e-05	1.114e-03	3.768e-05
	NN	1.100e-03	4.336e-05	1.111e-03	5.084e-05
	RF	3.063e-04	1.647e-06	1.623e-03	4.707e-05
	SVR	1.045e-02	2.861e-05	2.471e-02	4.413e-04
Machine	M5	2.884e-02	5.599e-03	5.565e-02	3.267e-02
	M5E	2.515e-02	2.403e-03	4.656e-02	2.191e-02
	MTF	1.651e-02	6.284e-02	1.985e-01	1.072e+00
	NN	1.707e-03	5.629e-04	2.708e-03	2.510e-03
	RF	4.056e-04	6.040e-05	1.794e-03	2.780e-03
	SVR	1.303e-02	1.796e-03	1.179e-02	1.315e-02
MV artificial domain	M5	3.036e-04	1.534e-05	3.425e-04	6.922e-05
	M5E	2.270e-04	1.576e-05	2.504e-04	3.419e-05
	MTF	1.674e-04	6.898e-04	1.845e-03	8.016e-03
	NN	2.426e-06	3.992e-06	2.435e-06	4.001e-06
	RF	4.078e-07	6.813e-08	1.821e-06	8.785e-07
	SVR	1.808e-03	2.984e-05	1.820e-03	3.893e-05
Servo	M5	4.667e-02	1.423e-02	6.216e-02	2.263e-02
	M5E	4.667e-02	1.423e-02	6.216e-02	2.263e-02
	MTF	3.802e-03	7.877e-04	8.851e-03	1.094e-02
	NN	7.742e-03	1.821e-03	1.055e-02	9.743e-03
	RF	7.555e-04	1.177e-04	5.824e-03	6.729e-03
	SVR	9.512e-03	1.352e-03	1.101e-02	1.021e-02

Table 10.3: Comparison of MAE scores of all six models on the first five datasets.

		Training set		Test set	
Dataset	Model	$\overline{MAE}$	σ_{RMSE}	$\overline{MAE}$	σ_{RMSE}
Abalone	M5	5.219e-02	5.356e-04	5.362e-02	2.982e-03
	M5E	5.213e-02	4.366e-04	5.336e-02	2.863e-03
	MTF	5.226e-02	5.886e-04	5.441e-02	3.696e-03
	NN	5.285e-02	1.163e-03	5.372e-02	2.687e-03
	RF	4.479e-02	3.839e-04	5.367e-02	1.995e-03
	SVR	5.623e-02	5.492e-04	5.952e-02	1.973e-03
Auto-MPG	M5	5.036e-02	2.912e-03	5.928e-02	1.098e-02
	M5E	4.806e-02	1.981e-03	5.609e-02	8.850e-03
	MTF	4.417e-02	1.608e-03	5.087e-02	8.830e-03
	NN	4.994e-02	2.678e-03	5.431e-02	8.388e-03
	RF	1.881e-02	6.244e-04	4.885e-02	9.064e-03
	SVR	5.264e-02	1.327e-03	5.546e-02	8.823e-03
Boston housing	M5	4.335e-02	3.814e-03	5.615e-02	9.564e-03
	M5E	3.819e-02	1.625e-03	4.852e-02	8.228e-03
	MTF	7.043e-02	7.255e-02	1.107e-01	2.004e-01
	NN	4.911e-02	8.328e-03	5.710e-02	1.202e-02
	RF	1.771e-02	4.883e-04	4.622e-02	7.893e-03
	SVR	1.142e-01	2.200e-03	1.157e-01	1.311e-02
California housing	M5	5.789e-02	4.628e-04	6.507e-02	1.809e-03
	M5E	5.262e-02	3.429e-04	5.895e-02	1.770e-03
	MTF	8.292e-02	6.408e-04	8.427e-02	2.005e-03
	NN	8.547e-02	2.272e-03	8.618e-02	2.198e-03
	RF	2.497e-02	1.359e-04	6.774e-02	1.806e-03
	SVR	7.969e-02	3.699e-04	8.170e-02	1.794e-03
CASP	M5	1.367e-01	9.483e-04	1.448e-01	1.921e-03
	M5E	1.236e-01	4.015e-04	1.319e-01	1.531e-03
	MTF	1.904e-01	7.091e-04	1.910e-01	1.991e-03
	NN	1.750e-01	3.913e-03	1.759e-01	4.199e-03
	RF	4.178e-02	1.414e-04	1.129e-01	1.960e-03
	SVR	9.292e-02	2.236e-04	1.283e-01	2.022e-03

Table 10.4: Comparison of MAE scores of all six models on the second five datasets.

		Training set		Test set	
Dataset	Model	$\overline{MAE}$	σ_{RMSE}	$\overline{MAE}$	σ_{RMSE}
Elevators	M5	2.352e-02	2.097e-04	2.443e-02	5.393e-04
	M5E	2.305e-02	1.801e-04	2.370e-02	5.343e-04
	MTF	2.296e-02	4.652e-04	2.601e-02	2.240e-03
	NN	2.278e-02	1.025e-03	2.299e-02	1.148e-03
	RF	1.038e-02	5.122e-05	2.774e-02	7.475e-04
	SVR	8.113e-02	1.756e-04	9.329e-02	1.475e-03
Friedman artificial domain	M5	2.590e-02	1.281e-04	2.668e-02	2.380e-04
	M5E	2.503e-02	1.070e-04	2.558e-02	2.829e-04
	MTF	2.602e-02	2.043e-04	2.651e-02	4.477e-04
	NN	2.640e-02	5.235e-04	2.654e-02	6.117e-04
	RF	1.342e-02	4.193e-05	3.177e-02	4.490e-04
	SVR	8.831e-02	1.119e-04	1.280e-01	1.322e-03
Machine	M5	1.785e-02	2.572e-03	3.031e-02	1.275e-02
	M5E	1.629e-02	1.218e-03	2.661e-02	9.131e-03
	MTF	2.046e-02	1.862e-02	4.924e-02	1.361e-01
	NN	2.860e-02	5.488e-03	3.194e-02	1.060e-02
	RF	9.362e-03	6.196e-04	2.075e-02	9.790e-03
	SVR	8.210e-02	3.151e-03	8.145e-02	2.035e-02
MV artificial domain	M5	1.340e-04	2.736e-06	1.366e-04	4.974e-06
	M5E	1.214e-04	3.275e-06	1.239e-04	4.322e-06
	MTF	4.290e-04	1.163e-04	6.569e-04	8.871e-04
	NN	1.017e-03	1.033e-03	1.019e-03	1.034e-03
	RF	2.836e-04	2.629e-06	6.775e-04	1.805e-05
	SVR	3.523e-02	3.504e-04	3.525e-02	4.152e-04
Servo	M5	2.782e-02	7.868e-03	3.854e-02	1.061e-02
	M5E	2.782e-02	7.868e-03	3.854e-02	1.061e-02
	MTF	3.428e-02	3.759e-03	4.676e-02	1.521e-02
	NN	6.096e-02	9.931e-03	6.972e-02	1.851e-02
	RF	1.240e-02	1.130e-03	3.419e-02	1.482e-02
	SVR	7.038e-02	2.739e-03	7.271e-02	1.615e-02