Chapter 1

Literature Review

1.1. Introduction into trees

The early application of decision trees in machine learning started with simple classification or regression tasks, trained on known datasets, to model the relationships between the input and output variables;

- Classification models were responsible for predicting a categorical output.
- Regression models produced a numerical output.

A decision tree takes the appearance of a tree to organise information in a recursive manner. It makes decisions through a series of binary questions that resembles branches. In machine learning, decision trees represent an algorithm containing if-else statements to model patterns 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. These models would not be theoretically refined until computing technologies grew sufficient [1].

Before the conception of decision trees, many of the alternative statistical methods were developed with small datasets in mind, on the assumption of the data being homogeneous, leading to algorithms that focused heavily on only a few parameters to model the influence of a wide range of factors. Applying this to larger datasets further assuming that the data retained its homogeneous structure, would prove to be flawed. datasets are not only subject to being large, but complex as well. This complexity influenced by several factors: high dimensionality; mixtures of data types; non-parametric distribution and non-homogeneity. "The curse of dimensionality", often apparent in these complex datasets, states that the higher the dimensionality, the higher the variance of the data points are. With the progression of computing technologies, datasets with these attributes are no longer a rare occurrence [1].

To combat this, one could reduce the dimensionality of the data. However, the accompanying drawbacks are unfavourable. This led to the demand 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. This brought rise to the refinement of

decision tree learning. The defining characteristic of decision trees being that it does not only produce satisfactory results, it also gives the user thoughtful information and insight into the data it is being applied on. In classification studies, for example, this helps uncover the predictive nature of the issue and helps the user better understand how specific features contribute to the outcome that is being observed [1].

1.1.1. The CART methodology

One of the earliest documented work on the practical as well as the theoretical side of decision trees was published in 1984. L. Breiman, together with J. Friedman, R. Olshen and C. Stone, 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. Their findings would be labelled as the CART (Classification and Regression Trees) software [1]. CART builds a tree through a series of binary questions, which when answered sequentially, would produce a solution to the problem at hand.

A simplistic binary tree is portrayed in Figure 1.1, capable of classifying an individual's sex based on two inputs, their weight and height. Nodes X_1 and X_3 each house a split condition. 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, due to them terminating all possible *paths* (sequence of nodes) that can be followed within the tree, starting at the root node. It is important to note that decisions trees are always constructed in such a manner that when combining the regions each path denotes; the entirety of the instance space is covered [2]. The *size* of a decision tree refers to the amount 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.

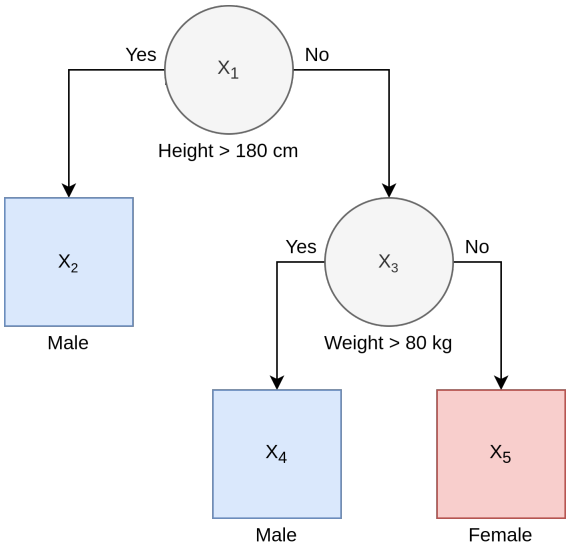

Figure 1.1: A hypothetical two-class tree-structured classifier.

1.1.2. Tree Induction

When CART constructs a decision tree for classification, such as shown in Figure 1.1, from a set of labelled observations, then three fundamental steps are followed:

1. Determining a split.
2. Evaluating a stopping rule.
3. Assigning a class label to each leaf.

Starting at the root node, step 1 is repeated for each subsequent node until step 2 is satisfied for each path within the decision tree. To construct the ideal tree, each split should be chosen such that the descendant subset contains a minimal mixture of classes. Determining the split is done with a splitting function, which compares multiple splits for a node. A metric often used to express such a splitting function is impurity i.e. the larger the impurity, the higher the number of classes present in the subset. CART determines its splits off of the value of a single variable only. At each node the tree algorithm evaluates a split point for each input variable in the dataset, selecting the one split which minimises the impurity function [1].

Once a node is reached where no significant decrease in impurity is produced, then that node becomes 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 would then be assigned as the leaf's label. This completes the initial decision tree growing sequence. The advantages this had to offer over the alternative statistical models included [1]:

1. This process could be applied to both categorical and numerical data. A decision tree can thus be grown from a very complex dataset.
2. Once grown, the application of a decision tree on new incoming data is computationally efficient and interpretable.
3. Decision trees exploit the conditional information present in non-homogeneous data to better model the observations.
4. The most beneficial information for identification is incorporated at each node split to form each subset. Allowing for automatic dimensionality reduction.
5. It proved to be quite robust when subjected to outliers and missclassified points.

In the application of regression, little changes. The tree-structured predictor is still partitioned into subsets via a series of split decisions. Instead of of predicting which class an observation belongs to, the decision tree now predicts the numerical value associated with

a class. Fundamentally, the regression tree predicts this numerical value as a dependent variable, based on a multitude of independent variables. The input variables being either discrete, continuous or even a mixture of the two. Through piecewise approximation is the regression tree able to model the relationship between the input and output variables. Consequently, the leaf nodes of a standard regression tree each have a constant output value [1]. Figure 1.2 illustrates the output of a regression tree, induced on the values $y = sin(x)$ for $x \in (0, 2\pi)$.

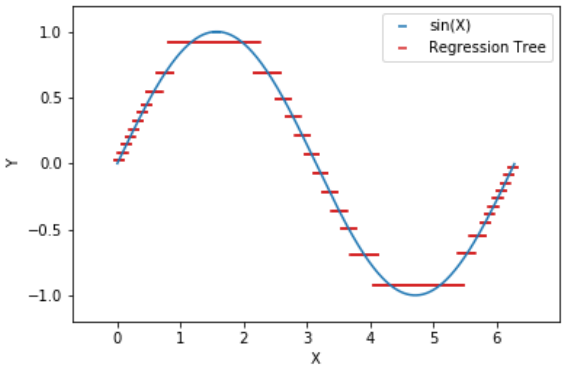

Figure 1.2: A regression tree's piecewise discrete approximation (red) of a continuous sinusoidal function (blue).

For regression, CART employs a splitting function that calculates the expected reduction in either variance or absolute deviation for each proposed split. The split that maximises this reduction is then selected. The process repeats for several nodes until a stopping criterion is met. The leaf node output value is thereafter calculated through a specified function, in the case of CART, this would be the average of the sum of squares within the node [1].

1.2. Bias-variance trade-off

The bias-variance dilemma stems from the degree of inaccuracy present in machine learning models. It would be ideal for a machine learning model to have perfect prediction accuracy, but due to models being exposed to unseen data during use, this is not realisable. Therefore, during training, the focus is on mitigating the prediction error, rather than trying to remove it altogether. To better mitigate this prediction error, one must first understand its origin.

Decision trees are regarded as supervised learning methods, as they are trained with the task of mapping an observed output variable to samples of multiple input variables. In the case of a supervised learning algorithm, the criterion used in assessing its accuracy is the ability of the model to generalise to unseen data [3]. This is where the bias-variance trade-off has to be addressed. Both bias and variance are degrees of prediction error present in the model, negatively affecting its accuracy. We define bias as the average difference in prediction a model makes from the target value and variance as the amount of variability or spread a model is subject to when predicting a certain data value [4].

1.2.1. Overfitting and underfitting

Models that are simple in their composition, such as linear models, will struggle to fully capture the intricate patterns in the observed data and as a result, produce an oversimplified model. In turn, the model will exhibit a high bias and subsequently underfit the training data. As for a model that incorporates complex algorithms, it is more vulnerable to noise in the data and likely to capture patterns within the data that are not associated with the desired output. This results in a large variance with overfitting of the training data. Figure 1.3 illustrates both cases together with an ideal model that minimises both errors.

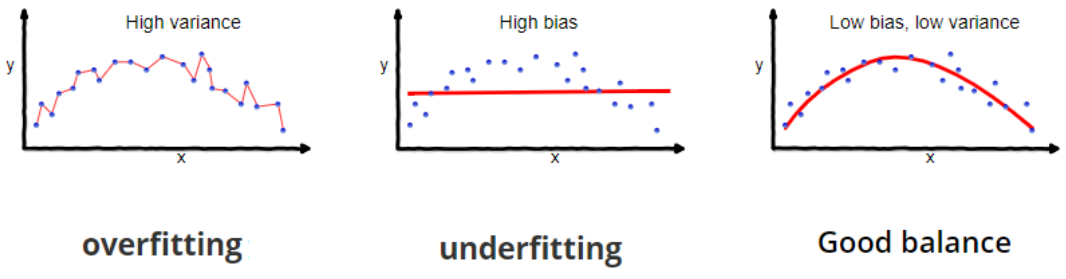

Figure 1.3: Three separate models' fit (red) on a simple regression problem in a two-dimensional space. Reproduced from [4]

It is important to note that a model cannot simultaneously be less complex and more complex. Therefore, bias and variance, being respectively proportional to these factors, are

error functions increasing in opposite directions. This trade-off between bias and variance must be minimised when designing learning algorithms. The strategy of minimising the trade-off and the point of optimal balance between the two errors differ depending on the type of model that is desired.

1.2.2. In the context of decision trees

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 [1]. Trees would often deliver satisfactory results on training sets, but generalised poorly on test sets. During its growing process, classification trees would continuously optimise the classification boundaries adding additional decision nodes, capturing noise in the training set. This meant that trees were prone to overfitting and would 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*. This meant that starting at the leaf nodes and following upwards into the tree, nodes were evaluated through cross-validation or test samples and removed if warranted. Pruning would prove to lower the variance in trees and significantly improve its performance [1].

As for regression trees the constant output values within the leave nodes were subject to underfitting the target value. Figure 1.2 shows how poorly constant values fit the shape of a sinusoidal wave. This poor fit of the target values meant that regression trees exhibited a large bias error.

1.3. Ensemble methods

An alternative method to minimising variance of classification trees is ensemble learning. The premise of ensemble learning is to develop multiple models in unison to provide a better result than an individual model is capable of. The use of several weak learners together proved to form a strong learner. Taking the result of a decision tree ensemble would prove to lower the variance whilst keeping the bias error low. This meant that decision tree ensembles would become a very competitive solution in both classification and regression problems, with various differing implementations developed.

1.3.1. Bagging

One of the simplest tree ensemble methods is bootstrap aggregation, referred to as *bagging* [5]. Several decision trees are induced in parallel, following the aforementioned procedure, each on a subset of the training data. Each subset is determined through a random selection process. The randomness off of which each tree is induced is what brings the variance of the collective model down. For regression applied bagging, the final prediction of the ensemble is derived through averaging the predictions of each individual tree within the ensemble. In the case of classification, the majority vote is taken instead of the average predicted value [6].

1.3.2. Random Forests

Building on the success achieved by bagging, the *random forest* technique was developed as an extension over it; first conceptualised by Breiman [7]. 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 of each individual tree. Input features are randomly selected at each node within the tree to determine the best split for that node, evaluated only on those randomly selected features. This decorrelates the trees within the ensemble model, decreasing the model's variance error. Random forest has the benefit of performing well on higher dimensionality data, due to its dimension reduction-like training technique. It is important to note that the individual trees are left unpruned in a random forest. This means that each tree is *fully grown* and retains the low bias trees exhibit. Random forests were shown to outperform many competing classifiers whilst being robust to overfitting [7].

1.3.3. Boosting

In contrast to bagging is the *boosting* technique. Boosting employs a sequential learning strategy; starting with a simplistic, usually shallow tree and building on it with the goal

of improving the net error. The idea behind boosting is to develop weak learners, one at a time, with each sequential learner focused on the patterns the previous learner was unable to adequately model. This is achieved through assigning weights to each observation. Instances which proved to have high prediction error were given larger weights so that the next learner would be more likely to correctly model it. Finally, combining these learners results in a model that together outperforms its individual parts. Therefore, in bagging each observed feature has an equal probability of being selected, whereas with boosting this selection probability is influenced by the calculated weights [8].

1.3.4. Ensemble bias and variance

These two techniques have different approaches to addressing the bias-variance trade-off. Because the trees used in boosting are each grown shallow, individually they have a high bias with low variance. Collectively this bias is reduced. Bagging, however, promotes the growth of large trees that have low bias and high variance. The trees are grown with decorrelation in mind, thereby decreasing the variance of the ensemble.

1.4. Model trees

Breiman's early work on improving the robustness of decision trees showed great success concerning the application of decision trees on classification problems [1]. However, Breiman's focus on classification trees [5, 7] 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 1.2. This meant that regression trees, developed through the CART methodology, had limited capability of adequately predicting the target value for continuous classes.

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, such as a linear regression model or neural network. 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 [9].

There was still a need for a learning model capable of approximating non-linear regression and providing its user with insights into the relationship of the data. 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 [10]. Model tree leaves can incorporate any multivariate model to better capture the patterns present in the training data.

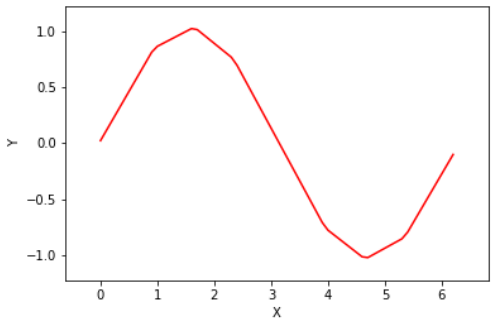

Figure 1.4: The continuous piecewise linear approximation of a sinusoidal wave.

Figure 1.4 illustrates how a model tree, that comprises linear models, would approximate the continuous sinusoidal function $y = sin(x)$ for $x \in (0, 2\pi)$, as opposed to the regression tree in Figure 1.2. The use of linear models over 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 $y = sin(x)$.

1.4.1. 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[1]. The first difference between the two methodologies is its splitting criterion used to evaluate a proposed split. Instead of selecting the split that maximises the reduction in variance or absolute deviation, the M5 induction algorithm selects the split which maximises an expected error reduction at a node, defined by the following formula [10]:

$$\Delta error = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i) \tag{1.1}$$

where T represents the collection of training samples for the data encapsulated by the node, and $sd(T)$ 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 linear models are constructed for non-leaf as well leaf nodes is to accommodate for *pruning* and *smoothing*.

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 [10]. Once the linear model is constructed, each parameter of the linear model is evaluated for simplification of the linear model. If excluding 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 each node has been constructed a simplified linear model, 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 reduce the complexity of the final model, subsequently reducing its variance.

Smoothing is applied when the model is tasked with predicting a test sample. Smoothing prevents discontinuities from forming between the linear models of neighbouring leaf nodes. Smoothing adjusts predicted values of the model tree as follows: starting with the predicted value at the leaf node and passing it on along the path to the root node, the predicted

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

value is 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:

$$y_i = \frac{ny_{i-1} + ky}{n + k} \tag{1.2}$$

where n is the number of training samples at the previous node along the path, y_{i-1} is the adjusted prediction passed on 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.

1.4.2. Early M5 model tree performance

Quinlan compared the M5 model tree to MARS (multivariate adaptive regression spline) as this was a popular regression model that proved effective on non-linear data at the time of the M5 model tree's conception [10]. MARS is similar to M5 as it is also non-parametric[2] 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 would struggle past no more than twenty [10].

1.4.3. 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 skipped over. 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 [9]:

$$w_0 + w_1x_1 + w_2x_2 + ... + w_kx_k \tag{1.3}$$

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. Through experimentation Wang and Witten also deemed the simplification of the linear models compromising to the size of the model tree and opted to omit this induction step, as sometimes much smaller trees were obtained when all features were left in the linear models. The M5' model tree was shown performing better than the original M5 tree on the standard datasets for which results where available [9].

[2] 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.

1.4.4. Shortcomings of model trees comprising linear models

The M5 model has been shown to perform inadequately in comparison to other solutions on datasets that exhibit noisy patterns and highly non-linear relationships between its features. 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[3] 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 overpruning and has been shown to have negative affects on a model tree's performance [11].

A study published in 2016 [12] 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, accuracy 19.1% higher 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 given there are a higher number of input parameters [10].

Quinlan recommended researching the application of non-linear models at the leaf nodes [10], to improve the ability of a model tee 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 in 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 computation 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. There is also the fallacy in expecting the increase in complexity of a model to guarantee an improvement in its performance. Often a simpler approach is the favoured solution to a problem [13]. These factors are hypothesised as contributing to the lack of abundant research on model trees that incorporate non-linear models.

[3]Post-pruning is the 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.

1.5. Non-linear solutions

1.5.1. Kernel Regression

Torgo developed the hybrid regression tree (HTL) that, amongst other proposed models, incorporated kernel regression at its leaf nodes [14]. 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 HTL incorporates is known as a lazy learner as the system only generalizes the training data once a prediction is made[4].

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

$$MSE = \frac{1}{N}\sum_{i}^{N}(y_i - o_i)^2 \tag{1.4}$$

where N is the number of instances over which the MSE is computed, y_i the actual target value of the i^{th} instance, and o_i its predicted value. The predicted value for each training instance is chosen as the average target value in the subtree where the split is proposed[5]. 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 [14] and therefore opts to use CART methodology to induce the tree structure.

The kernel regression model comprises a k-nearest neighbours model and Gaussian kernel function. The kernel regression model calculates predictions using only the training samples most similar to a given query. This similarity based on a distance metric between two instances. The Gaussian kernel function increases the weight, based on the distance metric, of the neighbours the nearer they are. Given the query point $\boldsymbol{q}$, the prediction is calculated through the following function [14]:

$$o(\boldsymbol{q}) = \frac{1}{SK}\sum_{i}^{k} y_i \times K\left(\frac{D(\boldsymbol{x_i}, \boldsymbol{q})}{h}\right) \tag{1.5}$$

with

$$SK = \sum_{i}^{k} K\left(\frac{D(\boldsymbol{x_i}, \boldsymbol{q})}{h}\right) \tag{1.6}$$

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

[5]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.

and

$$K(d) = e^{-d^2} \tag{1.7}$$

where $D(\cdot)$ is a distance function between two vectors, h the bandwidth parameter, and x_i the i^{th} instance of the 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 (1.5) portrays how the computational complexity grows as the value of k increases due to the added terms in both the kernel function and encapsulated distance function. For experimentation, Torgo chose $k = 3$ without specifying the reasoning behind [14].

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[6]. Although Torgo mentions the M5 model tree incorporating similar linear models to the linear HTL variant, Torgo does not compare the performance of HTL directly to that of M5 [14].

HTL was further improved on by imposing the kernel regression model, described by Equation (1.5), on the linear model in Equation (1.3). The resulting model now incorporating the weight vector, $\boldsymbol{w}$, of the linear model is described by the equation:

$$o(\boldsymbol{q}) = \boldsymbol{w}\boldsymbol{q} - \frac{1}{SK}\sum_{i}^{k} e_i \times K\left(\frac{D(\boldsymbol{x_i}, \boldsymbol{q})}{h}\right) \tag{1.8}$$

where e_i is the error of the linear model calculated as $e_i = \boldsymbol{w}\boldsymbol{x_i} - y_i$. This form 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 [15].

PLT was compared to MARS as well as a commercial version of M5, Cubist, on 12 separate benchmarking datasets producing highly competitive results, though Torgo described the use of PLT as being computationally heavy for the same reasons HTL is considered computationally expensive [15].

Torgo recommends 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 [15].

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

1.5.2. 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 [16]:

$$f(\boldsymbol{x}) = \sum_{\tau=0,\Sigma_{j=1}^{m}\lambda_j=\tau}^{n} \left(w_{(\lambda_1,\lambda_2,\ldots,\lambda_m)} \prod_{q=1}^{m} x_q^{\lambda_q} \right) \tag{1.9}$$

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 instances is already computationally demanding. Genetic algorithms, however, can be implemented to perform these tasks that are otherwise considered computationally too expensive [16].

A genetic algorithm is a heuristic search method that replicates natural selection mechanisms to evolve an optimal solution from multiple candidates [17]. "Survival of the fittest" is simulated across numerous generations, each generation having multiple candidate solutions [18]. The fitness function of a genetic algorithm dictates which candidate solutions correspond to better solutions in the problem domain [2]. Only the candidates deemed fit, are allowed to reproduce or 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 optimal 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 [16]. 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. 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 getting 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 [19]. This is due to genetic approaches seeking the 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 instance.

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 trees. Potgieter and Engelbrecht attributed the slow computational speed to the recursive procedures that perform fitness evaluation, crossover, and mutation of genetic candidates [2].

1.6. Model tree ensembles

The use of model trees in an ensemble is something not widely researched. The challenges of developing ensembles of model trees are the increased computation cost and the difficulty of interpreting the models [18]. However, these challenges are rewarded by the improved performance ensemble approaches have to offer. There are various approaches to ensemble modelling, those applicable in the context of this paper were discussed in Section 1.3.

Authors Aleksovski and Pfahringer are two of the few who have applied ensemble approaches to model trees for regression problems, with each author incorporating a unique approach [11, 20]. 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. Introducing randomness into an ensemble increases the diversity present in the learners that comprise the ensemble, allowing for greater prediction accuracy [7].

1.6.1. Pfahringer's Random Model Trees

Pfahringer modified the M5 algorithm to grow balanced trees within an ensemble approach (RMT) largely based on Breiman's Random Forest with little deviation other than the use of model trees as base learners. 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). 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 [20].

1.6.2. Aleksovski's Model Tree Ensembles

Aleksovski was tasked with developing models for dynamic systems, whose data was not evenly distributed in the feature space. Aleksovski, therefore, opted not to incorporate balanced trees for base learners as to avoid poor approximations of the data [11]. 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 subsets of randomly selected features, with replacement[7], are created for each tree in the ensemble. The standard induction of

[7]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.

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 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 used instead to combat this [11]. 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 instance, $o(\boldsymbol{x})$, is accordingly calculated as:

$$o(\boldsymbol{x}) = \mu(x_j, c, \alpha) f_1(\boldsymbol{x}) + \mu(x_j, c, \alpha) f_2(\boldsymbol{x}) \tag{1.10}$$

with

$$\mu(x_j, c, \alpha) = \frac{1}{1 + e^{-\alpha(x_j - c)}} \tag{1.11}$$

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. It is worth noting 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 it demanded [11].

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 the ensemble is evaluated both with and without each tree[8] it comprises. If a particular tree contributes to the increase of the prediction error, it is removed from the ensemble. The prediction of the ensemble is calculated through the uniformly weighted average of its trees.

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 produce a low variance error [11].

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

Chapter 2

GASOPE

2.1. Original GASOPE

The Model Tree Forest (MTF) proposed in this paper will employ a replication of GASOPE in the leaf nodes of the model trees that make up the forest. The original implementation of GASOPE was heavily influenced by computational expenses and developed with techniques of minimising its computational demand [16]. 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 when computing the fitness of an individual. This significantly reduced the time taken for the genetic algorithm to evolve the final individual. The hall-of-fame was incorporated to ensure that the subsampling of the training data for each generation did not hinder the genetic algorithm. Once the maximum number of generations has been reached, the individual that performed best over all generations was chosen, as apposed to the individual that performed best in the last generation. An individual was found to not always present the true nature of the dataset, particularly for extremely noisy datasets, because new subsamples of the dataset was used at each generation [16].

Since GASOPE was first conceptualised, computational limits have since become more lenient. Consequently the replicated version of GASOPE, used in MTF, does not require the same three stage process to minimise computational demand and instead can utilise the dataset in its entirety, ensuring the best individual is evolved through the generations of the genetic algorithm. The replication of GASOPE, was developed in Python using libraries Numpy and Scikit-learn. All over aspects of the original genetic algorithm was maintained and is discussed further in this section.

2.1.1. Discrete least-squares approximation

A structurally optimal function is evolved in two phases. The first phase, determines terms that make up the function. The second phase approximates the coefficients of these terms which result in the best fit of the data at hand. These coefficients are obtained through discrete least-squares approximation. For a arbitrary set of M data points $\{(\boldsymbol{a_1}, b_1), \ldots, (\boldsymbol{a_m}, b_m)\}$, the least squares error is minimised.

$$\epsilon = \sum_{i=1}^{m} \left[b_i - b_i'\right]^2 \tag{2.1}$$

where b_i' is the predicted output of a function of the form:

$$b_i' = \sum_{\tau=0, \Sigma_{j=1}^{m} \lambda_j = \tau}^{n} \left(r_{(\lambda_1, \lambda_2, \ldots, \lambda_m)} \prod_{q=1}^{m} a_{i,q}^{\lambda_q} \right) \tag{2.2}$$

An extrapolated example of Equation 2.2 would be:

$$b_i' = r_{(0,0)} + r_{(1,0)} a_{i,1} + r_{(0,1)} a_{i,2} + r_{(1,1)} a_{i,1} a_{i,2} + r_{(2,0)} a_{i,1}^2 + r_{(0,2)} a_{i,2}^2 \tag{2.3}$$

The coefficients are thus determined by solving the linear system:

$$\mathbf{b} \approx A\mathbf{r} \tag{2.4}$$

where $\mathbf{r}^{\mathrm{T}} = \begin{bmatrix} r_0 & r_1 & \cdots & r_n \end{bmatrix}$, which is rewritten as

$$\left(A^{\mathrm{T}} A\right) \mathbf{r} = A^{\mathrm{T}} \mathbf{b} \tag{2.5}$$

and has no exact solution. Matrix A is of the form:

$$A = \begin{bmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^n \\ 1 & a_2 & a_2^2 & \cdots & a_2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & a_m & a_m^2 & \cdots & a_m^n \end{bmatrix} \tag{2.6}$$

and vector $\mathbf{b}^{\mathrm{T}} = \begin{bmatrix} b_0 & b_1 & \cdots & b_n \end{bmatrix}$. Before this system can be solved, the structure of the polynomial needs to be fixed. As discussed in Section 1.5.2 there are too many combinations of possible terms to compute each. The genetic algorithm is assigned the task of evolving the optimal set of polynomial terms. Thereafter the set of terms is subject to discrete least-squares approximation, which produces the complete output function with coefficients.

2.1.2. Representation

Individual is a representation of a poly. The formula of a multivariate polynomial function incorporating all possible terms can be expressed as [16] follows, each individual is made up of a set I of unique, term-coefficient mappings:

$$I = \{(t_0 \to r_0), ..., (t_n \to r_n)\} \tag{2.7}$$

where $r_\xi, \xi \in \{0, \ldots, n-1\}$ is a real-valued coefficient. Each term is as follows:

$$T = \{(a_1 \to \lambda_1), ..., (a_m \to \lambda_m)\} \tag{2.8}$$

variable-order pairs.

2.1.3. Initialisation

For each individual in the population the initialisation process is described by Algorithm 1. A random amount of variable-order pair is repeatedly selected without replacement for each term-coefficient mapping, up to the maximum number of terms.

Algorithm 1: Pseudocode for the initialisation of a GASOPE chromosome.

```
Set I_w = {}.
while |I_w| < p do
    Set T_ξ = {}.
    Select e ∈ {0, ..., 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 ≤ g ≤ m randomly from the set of m inputs.
        if |T_ξ| < |T_ξ ∪ {(g → f)}| then
            e := e/{f}, i.e. remove f from the available orders.
            Set T_ξ = T_ξ ∪ {g → f}.
        end
    end
    Set I_w = I_w ∪ {T_ξ → 0}.
end
```

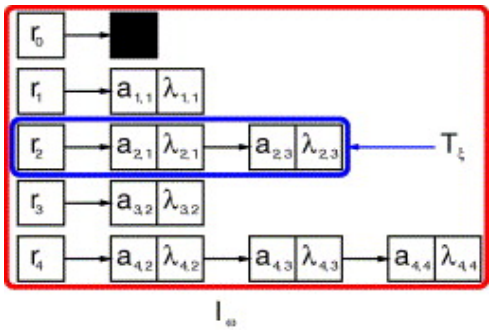

Figure 2.1: Illustration of the GASOPE initialisation algorithm, reproduced from [16].

2.1.4. Mutation operators

The mutation operators are responsible for injecting new genetic material into randomly selected individuals of the population. These mutation operators ensures that a larger search space is covered by the genetic algorithm, preventing the possibility of the algorithm falling into a local minima. The four mutation operators are: shrink, expand, peturb and reinitialise. Each operator helps optimise the structure of the polynomial by either making adjustments to variable-order pairs or an entire term-coefficient pair.

The first mutation operator, *shrink*, is very simple in its execution. Arbitrarily, one term-coefficient pair is selected from the set I_w and removed.

Algorithm 2: Pseudocode for the shrink operator.

Select $T_\xi \in I_w$ randomly from the set of terms I_w.
Set $I_w = I_w / \{T_\xi\}$.

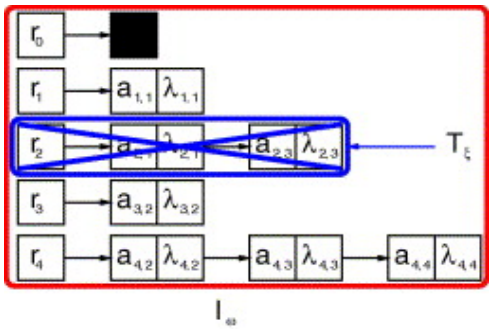

Figure 2.2: Illustration of the GASOPE shrink algorithm, reproduced from [16].

The second operator, *expand*, adds a new term-coefficient pair to the set I_w as described in Algorithm 3. In comparison to the original implementation, *expand* is executed on an individual even if the individual already contains the maximum number of allowed terms.

Algorithm 3: Psuedocode for the expand operator.

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**
    $e \coloneqq e/\{f\}$, i.e. remove f from the available orders.
    Set $T_\xi = T_\xi \cup \{g \to f\}$.
  end
end
Set $I_w = I_w \cup \{T_\xi \to 0\}$.

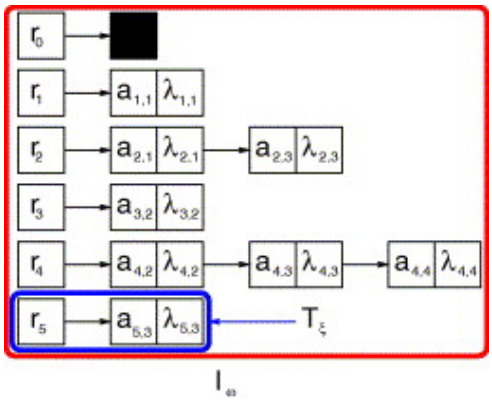

Figure 2.3: Illustration of the GASOPE expand algorithm, reproduced from [16].

The third mutation operator, *peturb*, is a particularly intricate operator focused on altering a randomly selected variable-order pair within one of the terms in I_w. When peturb is performed on an individual, one of three possible adjustments are made: a new variable-order pair is added to the term; an existing variable-order pair is removed from the term or a vriable-order pair is adjusted. This process is described through Algorithm 4. As apposed 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: Pseudocode for the peturb operator.

Select $T_\xi \in I_w$ randomly from the set of terms in I_w.
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_ξ, as portrayed by the crossed out variable-order pair in Figure 2.4.
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 2.4.
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 2.4.
end

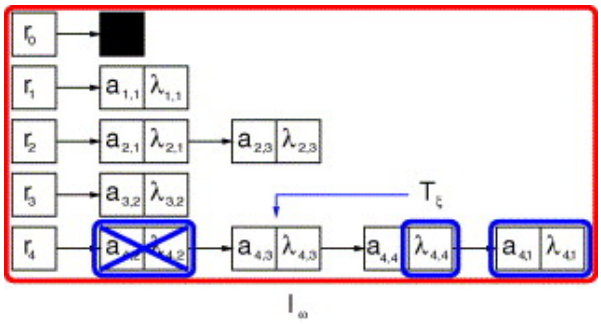

Figure 2.4: Illustration of the GASOPE peturb algorithm, reproduced from [16].

The fourth and final mutation operator, *reinitialise*, is a re-invocation of the initialisation process described fully in Section 2.1.3.

2.1.5. Crossover operator

The crossover operator is responsible for transferring over favourable genetic material from one generation of individuals to the next. The crossover operator thus narrows the search space with each generation. Two unique individuals are selected for crossover and their term-coefficent mappings are evaluated before randomly chosen to construct a new chromosone. Term-coefficient pairs that are present in both individuals, have a larger probability of being chosen. A ratio of 80:20 is chosen as this was found to result in newly

generated individuals to roughly equal the length of its longer parent [16]. Algorithm 5 describes how this ratio is implemented to generate a new individual. Contradictory to the original implementation of GASOPE, newly generated individuals may exceed the maximum amount of terms.

Algorithm 5: Pseudocode 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

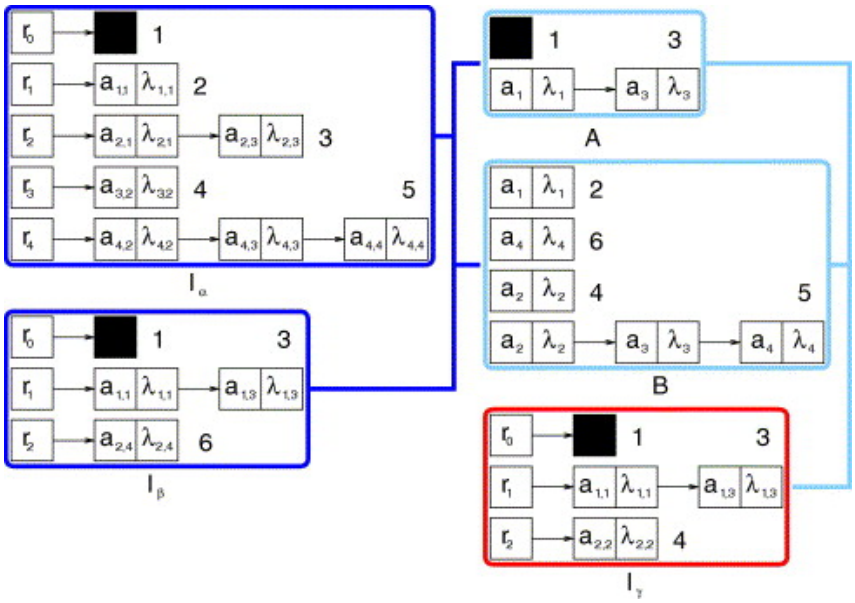

Figure 2.5: Illustration of the GASOPE crossover algorithm, reproduced from [16].

2.1.6. Fitness function

The fitness function dictates an individuals rank amongst all other individuals in the population. This rank determined by how well the individual solves the problem at hand. With 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, similar to the *adjusted coefficient of determination*, is defined as [16]:

$$R_a^2 = 1 - \frac{\sum_{i=1}^{N} \left(b_i - b'_{I_\omega,i}\right)^2}{\sum_{i=1}^{N} \left(b_i - \bar{b}\right)^2} \cdot \frac{s-1}{s-k} \tag{2.9}$$

where N is the sample size, b_i is the target output and $b_{I_\omega,i}$ the output predicted by individual I_w for pattern i. The model complexity is defined by k:

$$k = \sum_{\xi=1}^{|I_\omega|} \sum_{\tau=1}^{|T_\xi|} \lambda_{\xi,\tau} \tag{2.10}$$

where $\lambda_{\xi,\tau}$ is the order of a variable-order pair within a term-coefficient mapping T_ξ of individual I_w. k thereby penalised 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.

Before Equation (2.9) can be computed for each individual in the population, the coefficients are estimated through discrete least-squares approximation. The A Matrix from Equation (2.6) is first completed using the combination of terms each term-coefficient mapping incorporates. With vector $\boldsymbol{b}$ containing the target output for each pattern, the linear system in Equation (2.5) is reduced, producing vector $\boldsymbol{r}$ which is populated with the coefficient of each term-coefficient mapping.

2.1.7. Process

Algorithm 6 presents the complete genetic algorithm incorporating the aforementioned mutation and crossover operators. After the specified amount of generations have been surpassed the individual with the highest ranking fitness score is the final solution.

Algorithm 6: Process [16]

Let $g = 0$ be the generation counter.
Initialise a population P_g of N individuals:
$P_g = \{P_{g,n} | n = 1, ..., N\}$
while $g < G$, *where* G *is the maximum number of generations* **do**
  for *each individual* $P_{g,n}$ *in* P_g **do**
    Perform least square optimisation to determine the coefficients of $P_{g,n}$.
    Calculate the fitness of $P_{g,n}$ from Equation (2.9).
  end
  Let $P'_g \subset P_g$ be the top $x\%$ of the individuals selected for elitism.
  $P_{g+1} \coloneqq P'_g$
  Let $P''_g \subset P_g$ be the top $y\%$ of the individuals selected for crossover.
  Set $K =$ mutation rate$\times N$.
  while $|P_{g+1}| < (N - K)$ **do**
    Select two unique individuals I_α and I_β randomly from P''_g.
    Perform crossover between I_α and I_β to produce I_γ
    $P_{g+1} \coloneqq P_{g+1} \cup \{I_\gamma\}$
  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.
    $P_{g+1} \coloneqq P_{g+1} \cup \{I_m\}$
    $k \coloneqq k + 1$
  end
  $P_g \coloneqq P_{g+1}$
end

2.2. Testing

Section heading chapeter The following section discusses testing of GASOPE. The original experimental procedure is replicated under the same hyperparameters...

2.2.1. Original procedure

A subset of functions from the original testing procedure was used to test GASOPE. Table 2.2 lists these four functions with their function definitions.

The following aspects were chosen to be the same as with the original tests. 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 2.1. All functions are injected with uniformly generated noise over the interval $[-1, 1]$.

Table 2.1: GASOPE hyperparameter descriptions.

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 2.2: Functions.

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	$f(x_0, x_1) = x_0^5 - 5x_0^3 + 4x_0 + x_1^5 - 5x_1^3 + 4x_1; \{x_0, x_1\} \in [-2, 2]$

Table 2.3 lists the original results of GASOPE, reproduced from [16], together with the results obtained from the newer rendition of GASOPE, simply referred to as GASOPE from henceforth. These results' statistical significance evaluated, however from observation there are no notably large discrepancies.

Table 2.3: OG vs PyG w noise

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

2.2.2. Testing on higher dimensional problems

Shortcomings of the original testing of GASOPE include the lack of higher dimensional datasets and sounded motivation to the choice of hyperparameters. Only in the testing of GPMCC did an alteration of GASOPE interact with higher dimensional problems. GPMCC also incorporated several other complications of the genetic algorithm. One should also note that the chosen hyperparameters in Table 2.1 of GASOPE were evaluated on the one- and two-dimensional functions and are not necessarily adequate for higher dimensional problems. GASOPE is therefore tuned and tested on a higher dimensional problem to ensure that it is the optimal model to employ at the leaves of MTF.

There are many openly available UCI (University of California at Irvine) datasets to perform this analysis on. 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) [2]. However, with the appropriate tuning of GASOPE's hyperparameters, these results can be improved on. Evaluation of the dataset is done under 10-fold cross validation.

Bayesian optimisation is used for the tuning of GASPOPE's hyperparameters. Although it is not as exhaustive as a gridsearch, is less time-consuming for large parameter spaces [21]. Before Bayesian optimisation is applied, however, the computational expense of certain hyperparameters is first evaluated. The following parameters are considered to improve the results as their values are enlarged, but beyond a certain value the improvement in the error metric converges:

1. Population size,

2. the number of generations evolved for and

3. the maximum terms of a individual.

Maximum number of terms is the single hyperparameter that has the highest contribution to computational cost. Therefore it should be chosen as small as possible. This parameter is only applicable to the initialisation of an individual and GASOPE does reward the minimisation of the amount of terms of each individual with each generation. The maximum number of terms should be chosen in such a manner that the initial search space is adequately covered. From GASOPE's application in GPMCC, a value of 10 was found to be sufficient in higher dimensional problems. The best performing individual was found to often comprise less than 10 terms [2].

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 repetitions. From the box plots that portray the distribution of the RMSE on the validation set of each outcome for 30 repeated runs over the varying population sizes in Figure 2.6, is is clear that for very small population sizes the RMSE value tends to fluctuate. The smallest population size that produced consistent RMSE values was that of 30 individuals. Figure 2.7 shows how the average RMSE value over the 30 repeated runs, decreases with each generation that is evolved for. Note that for population sizes smaller than 30, the average training RMSE was higher than those with populations sizes above 30.

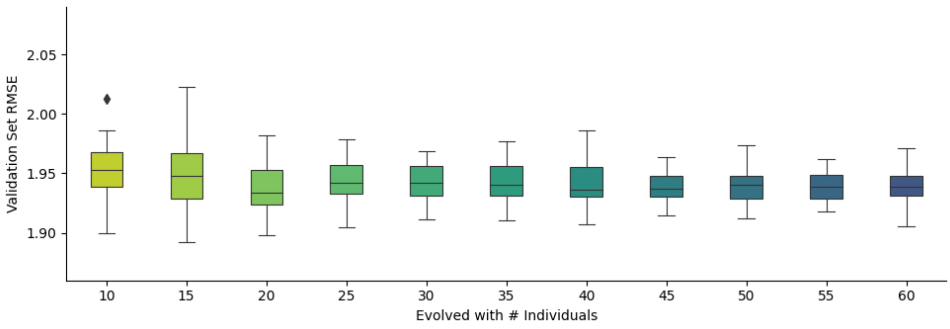

Figure 2.6: Pop size vs RMSE

The final parameter influencing computational costs is the amount of generations for which GASOPE is evolved. Figure 2.8 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 value tends to decrease as well as fluctuate less. However this tendency starts dropping off after 100 generations, where it starts to converge. The ideal value for the number of generations is subsequently chosen to be 100.

The maximum order hyperparameter can be considered unique to each case GASOPE is applied to. Allowing polynomials to incorporate large orders when the dataset itself does not exhibit highly nonlinear tendencies results in GASOPE imposing non-linear

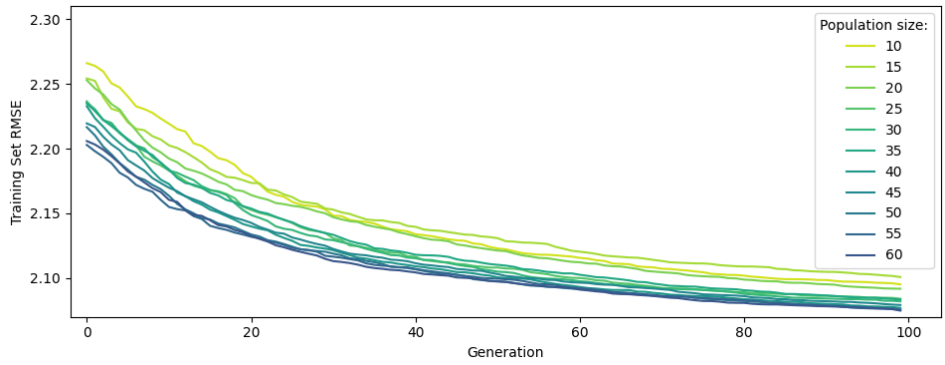

Figure 2.7: Pop size vs RMSE

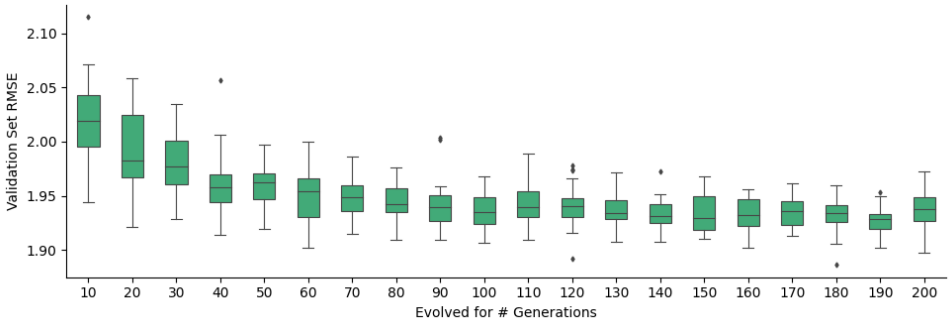

Figure 2.8: Generations vs RMSE.

relationships that are not prevalent in the data. This is what causes a degradation in performance of GASOPE in the case of the Abalone dataset. It is also worth noting that very rarely did the polynomials produced by GPMCC ever exceed the order three [2]. However, specifying an order to small is also faulty. To portray the effect the maximum order parameter can have three different instances of GASOPE are compared with each respective maximum order, o, set to 1,2 and 3.

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

1. Crossover rate
2. Mutation rate
3. Elitis rate

The objective function used by Bayesian optimisation to evaluate a set of hyperparameters is chosen to be the RMSE on the validation set. This error is minimised by Bayesian optimisation. Table 2.4 lists the final parameters for each GASOPE instance.

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 dataset. For the linear regression model there are no hyperparameters to tune. For the SVR model two hyperparameters are tuned, also

Table 2.4: 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

using Bayesian optimisation. These are the *gamma* and C values. *Gamma* refers to the coefficient of the Gaussian kernel function and C a regularisation parameter. Thereby ensuring that the SVR is performing to its best capabilities.

2.2.3. Results

Table 2.5 presents the results obtained from 30 repeated tests for each model together with the results GPMCC obtained on the Abalone dataset [2]. 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 [22]. 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 statistically determine whether all the results can be considered 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 [23].

Table 2.5: 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 the case of this test, that would be M5's linear regression model) [24]. Figure 2.9 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.

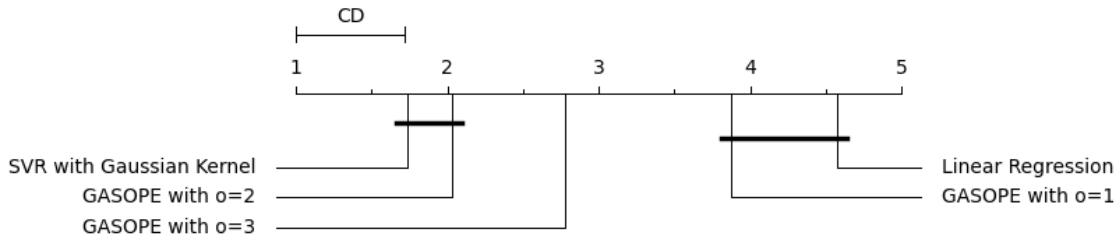

Figure 2.9: Criticial difference diagram for results obtained in Table 2.5.

It is therefore evident from Figure 2.9 that the results of SVR and GASOPE with $o = 2$ are statistically equivalent. The same goes for linear regression and GASOPE with $o = 1$. Which is expected as GASOPE with $o = 1$ 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.

Bibliography

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

[2] 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

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

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

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

[6] A. Nagpal, "Decision Tree Ensebmles- Bagging and Boosting," 2017. [Online]. Available: https://towardsdatascience.com/decision-tree-ensembles-bagging-and-boosting-266a8ba60fd9. [Accessed: 23- Mar- 2020].

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

[8] 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.

[9] Y. Wang and I. Witten, "Induction of model trees for predicting continuous classes," *Induction of Model Trees for Predicting Continuous Classes*, 01 1997.

[10] J. R. Quinlan *et al.*, "Learning with continuous classes," in *5th Australian joint conference on artificial intelligence*, vol. 92. World Scientific, 1992, pp. 343–348.

[11] 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.

[12] 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.

[13] 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

[14] L. Torgo, "Functional models for regression tree leaves," in *ICML*, vol. 97. Citeseer, 1997, pp. 385–393.

[15] ——, "Partial linear trees." in *ICML*, 01 2000, pp. 1007–1014.

[16] 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

[17] 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

[18] 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.

[19] 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

[20] 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.

[21] 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

[22] 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.

[23] S. García, A. D. Benítez, F. Herrera, and A. Fernández, "Statistical comparisons by means of non-parametric tests: a case study on genetic based machine learning," *algorithms*, vol. 13, p. 18, 2007.

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