Statistical Learning Tools for Heteroskedastic Data

by

Andrew J Henrey

M.Sc., Simon Fraser University, 2012
B.Sc., Simon Fraser University, 2010

Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
Department of Statistics and Actuarial Science
Faculty of Science

© Andrew J Henrey 2016
SIMON FRASER UNIVERSITY
Summer 2016

All rights reserved.
However, in accordance with the *Copyright Act of Canada*, this work may be reproduced without authorization under the conditions for "Fair Dealing." Therefore, limited reproduction of this work for the purposes of private study, research, education, satire, parody, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.

Approval

Name:	**Andrew J Henrey**
Degree:	**Doctor of Philosophy (Statistics)**
Title:	***Statistical Learning Tools for Heteroskedastic Data***
Examining Committee:	**Chair:** Tim Swartz Professor

Tom Loughin
Senior Supervisor
Professor

Hugh Chipman
Supervisor
Adjunct Professor

Rachel Altman
Internal Examiner
Associate Professor

Russell Steele
External Examiner
Associate Professor
Department of Mathematics and Statistics
McGill University

Date Defended: 18 August 2016

Abstract

Many regression procedures are affected by heteroskedasticity, or non-constant variance. A classic solution is to transform the response y and model $h(y)$ instead. Common functions h require a direct relationship between the variance and the mean. Unless the transformation is known in advance, it can be found by applying a model for the variance to the squared residuals from a regression fit. Unfortunately, this approach additionally requires the strong assumption that the regression model for the mean is 'correct', whereas many regression problems involve model uncertainty. Consequently it is undesirable to make the assumption that the mean model can be correctly specified at the outset.

An alternative is to model the mean and variance simultaneously, where it is possible to try different mean models and variance models together in different combinations, and to assess the fit of each combination using a single criterion. We demonstrate this approach in three different problems: unreplicated factorials, regression trees, and random forests.

For the unreplicated factorial problem, we develop a model for joint identification of mean and variance effects that can reliably identify active effects of both types. The joint model is estimated using maximum likelihood, and effect selection is done using a specially derived information criterion (IC). Our method is capable of identifying sensible location-dispersion models that are not considered by methods that rely on sequential estimation of location and dispersion effects.

We take a similar approach to modeling variances in regression trees. We develop an alternative likelihood-based split-selection criterion that has the capacity to account for local variance in the regression in an unstructured manner, and the tree is built using a specially derived IC. Our IC explicitly accounts for the split-selection parameter and our IC also leads to a faster pruning algorithm that does not require crossvalidation. We show that the new approach performs better for mean estimation under heteroskedasticity.

Finally we use these likelihood-based trees as base learners in an ensemble much like a random forest, and improve the random forest procedure itself. First, we show that typical random forests are inefficient at fitting flat mean functions. Our first improvement is the novel $\alpha-$pruning algorithm, which adaptively changes the number of observations in the terminal nodes of the regression trees depending on the flatness. Second, we show that random forests are inefficient at estimating means when the data are heteroskedastic, which we address by using our likelihood-based regression trees as a base learner. This allows explicit variance estimation and improved mean estimation under heteroskedasticity.

Our unifying and novel contribution to these three problems is the specially derived IC. Our solution is to simulate values of the IC for several models and to store these values in a lookup table. With the lookup table, models can be evaluated and compared without needing either crossvalidation or a holdout set. We call this approach the Corrected Heteroskedastic Information Criterion (CHIC) paradigm and we demonstrate that applying the CHIC paradigm is a principled way to model variance in finite sample sizes.

Keywords: Heteroskedasticity, Information Criteria, Random Forest, Regression Tree

Dedication

This work is dedicated to my friends.
I don’t have any friends.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

A common goal in statistics is to model the response y using a set of predictor variables X and then predict new responses given new data. Another typical goal is to determine which predictor variables affect the response. Many algorithms designed to solve these problems are impacted by heteroskedasticity, or non-constant variance [16]. Linear regression, a popular method used to answer both these questions, exhibits suboptimal variability in the fitted mean under heteroskedasticity [16]. A classic approach to mitigate heteroskedasticity is to take a transformation of the responses y [16]. This can be unsatisfactory for a variety of reasons, two being that this might influence the apparent relationship between the mean and the response, and that it is not always clear which transformation should be taken.

Our preferred approach to handling heteroskedasticity is to jointly model the mean and the variance via maximum likelihood. By fitting several models of this type we recast the problem as a model selection problem. Our solution to the model selection problem is to use an information criterion (IC). Akaike ([1]; [2]) developed the Akaike Information Criterion (AIC) which gives an asymptotic result that applies to almost any model. In particular Akaike's result states that the maximized likelihood of a model fit with maximum likelihood on the training data is an overestimate of the fitted likelihood on test data, and the difference is a penalty that increases with the number of estimated parameters. By computing the evaluated maximum likelihood and the corresponding penalty values of a series of models, the AIC can be used as a model selection criterion.

In Chapter 3, we work on a problem of modeling means and variances in unreplicated factorials. The sample sizes are typically very small (our typical case is $n = 16$), and an asymptotic criterion is inappropriate. The 'Corrected AIC', or AICc ([63]; [36]) provides an exact IC under a linear homoskedastic model. We follow the same approach to construct an IC for the linear heteroskedastic model. Unfortunately in the linear heteroskedastic case the formula does not appear to have a closed form. Our approach, which we call the Corrected Heteroskedastic Information Criterion (CHIC), consists of simulating a penalty value for each possible model. We then use these penalty values to do model selection. We apply our approach to unreplicated factorials, providing the first method to our knowledge that can directly compare two arbitrary mean-variance models. We show that our method is a strong improvement over a method recommended in the literature that does not jointly model the mean and variance.

We apply our same CHIC paradigm to the regression tree algorithm of Breiman et al. [11]. Regression trees typically recursively partition the data into two groups, fitting a constant mean function to each group. This results in a variant of the linear homoskedastic model. Ruth and Loughin [57] show that under heteroskedasticity the standard algorithm to create pruned regression trees performs suboptimally. They show the regression tree makes spurious splits in the high variance area of the data, and fails to make useful splits in the low variance area. They determine that this problem stems from an overly aggressive stopping rule. A second problem that we identify is that under heteroskedasticity the tree can produce suboptimal estimates of the mean. Our approach to fixing these issues is to jointly model the mean and the variance through a new tree structure we call the Heteroskedastic Regression Trees (HeaRTs). Our tree makes splits on the mean, the variance, or both the mean and variance simultaneously. The result is a tree that models both the mean and variance as piecewise constants. The tree is both split and pruned using the CHIC approach. We demonstrate superior performance to Breiman et al.'s algorithm under heteroskedasticity, while also improving the runtime.

Ensemble methods are a modern approach to prediction [34]. A popular ensemble is the Random Forest [9]. The Random Forest constructs a set of K regression trees from perturbations of the data. It predicts new observations by first predicting the response of each individual tree, and then averaging the predictions of each tree. As Random Forests are usually constructed from a set of *unpruned* regression trees, it is unclear how heteroskedasticity affects their performance (the results from Ruth and Loughin apply to pruned trees). However, lack of pruning exposes Random Forests to a different problem when the signal-to-noise ratio is low. We show that superior prediction can be obtained by applying some amount of pruning. Furthermore, we show that under heteroskedasticity, jointly modeling the mean and variance can lead to improved mean estimation. Our solution is the Jar of HeaRTs, essentially a Random Forest of HeaRTs plus a development we call α-pruning. We demonstrate our method on real and simulated data and demonstrate a small improvement in mean estimation over the classic Random Forest while keeping the same asymptotic runtime.

Chapter 2

Literature Review

2.1 Linear Regression

A common goal of data analysis is to model a column vector of responses (denoted as y) given some predictor variables (denoted as an n by p matrix X); we call this the training data. Then, we are interested in predicting new responses given a new matrix X, also known as the test data. The response y and predictor variables X can be categorical, ordinal, or continuous. In our work we focus on continuous y and X. A popular and simple method to fit continuous y given X is linear homoskedastic regression [45]. The model is:

$$y = X\beta + \epsilon$$

$$\epsilon \sim N(0, \sigma^2).$$

As usual, β is a $p \times 1$ vector of unknown parameters, ϵ is a $n \times 1$ vector of errors, and σ^2 is an unknown variance. The least squares solution of β given y and X is:

$$\hat{\beta} = (X'X)^{-1}X'y.$$

This model makes several key assumptions. The model assumes a linear mean function $X\beta$, so the model fits better if the true relationship between $E(Y)$ and X is close to linear. If the true relationship between the mean of y and X is discontinuous or curved, this model will not fare as well. Secondly, the model for the errors is $\epsilon \sim N(0, \sigma^2)$: in this model σ is assumed to be constant throughout the space of X. Data that violate this assumption are called heteroskedastic data. When the linear homoskedastic model is applied to heteroskedastic data, the linear homoskedastic model gives unbiased estimates of β but the estimates themselves are excessively variable—the estimator is not efficient [45]. The linear homoskedastic model additionally assumes that the errors are normally distributed and that the observations are independent. We will not be focusing on these two assumptions in our work, and assume they are satisfied.

Instead, our focus is on fitting models when the first two assumptions can be called into question, with a particular emphasis on heteroskedasticity. By relaxing the constant-variance assumption of the linear homoskedastic model, one arrives at the linear heteroskedastic model, which is:

$$y_i = X_i\beta + \epsilon_i$$

$$\epsilon_i \sim N(0, {\sigma_i}^2),$$

where ${\sigma_i}^2$ now represents the variance for a given response y_i. Define the matrix Σ to be a diagonal matrix where the diagonal entries Σ_{ii} are ${\sigma_i}^2$, the variance of y_i. To estimate β under the linear heteroskedastic model, the method of Weighted Least Squares (see e.g., [16]) is used. The Weighted Least Squares solution for β is:

$$\hat{\beta} = (X'\Sigma^{-1}X)^{-1}X'\Sigma^{-1}y.$$

Unfortunately to use Weighted Least Squares, the ${\sigma_i}^2$s must be known.

The method of Feasible Generalized Least Squares (FGLS, [55]) is an adaptation of Weighted Least Squares that can be used to estimate both β and ${\sigma_i}^2$ at the same time. In general FGLS can be used to estimate a variance matrix that has non-zero off-diagonal elements. In our model, however, we continue to assume independence between the observations y and consequently we assume that the off-diagonal elements in Σ are 0. The FGLS algorithm under these assumptions then reduces to:

1. Initialize Σ to be the identity matrix.
2. Estimate $\hat{\beta} = (X'\Sigma^{-1}X)^{-1}X'\Sigma^{-1}y$, using our current estimate of Σ.
3. Estimate Σ_{ii} using $(y_i - X_i\hat{\beta})^2$ for all i, using our current estimate of β.
4. Repeat steps 2-3 until there is no appreciable change in the estimates of β or Σ.

The assumption of linearity is also not always met. Many methods have been developed to adaptively fit the response surface. The common theme among these methods is that they allow the data to determine the shape of the relationship between y and X, rather than specifying a rigid model for the relationship. Some example methods include splines [27], generalized additive models (see, e.g., [34]), nearest neighbor and kernel density estimators (see e.g. [4], [34]) and Regression Trees [11]. Our main focus is on regression trees, mainly due to their use as base learners in ensemble methods like bagging [8] and Random Forests [9].

2.2 Trees

Regression trees [11] are piecewise constant linear models with split points determined adaptively using a "recursive partitioning" algorithm. The algorithm recursively splits data into pairs of smaller and smaller groups, called "nodes", creating a tree structure. Typically, splits are carried out until some stopping criterion is reached on the nodes at the end of the tree, called "terminal nodes." The mean response in each terminal node is estimated from the sample mean of the data within the node. The size of the tree may be reduced ("pruned") if a smaller tree improves prediction accuracy. We describe the tree-building and pruning processes in more detail below.

Breiman et al.'s approach [11] to building a regression tree (which we refer to as RPAB: Recursive Partitioning Algorithm of Breiman et al.) works as follows. We begin by defining the responses y and predictor matrix X, where y is a length n vector and X is a n by p matrix of predictor variables. Assume for the moment that the values within each column of X are continuous without ties, but the method can also handle categorical variables. RPAB begins by finding a splitting rule for the full data, which reside in the root node. A splitting rule consists of a predictor X_j and a rule "is $X_{ij} < c$" for some value c. This creates an indicator function that is evaluated on each observation. Note that many different values of c may result in the same partitioning of the X_{ij}'s. Conventionally only the values of c that fall halfway in between two adjacent values in the sorted list of X_j are considered. This results in $n-1$ possible values of c, though for practical reasons this is usually further reduced. In particular, a 'minimum node size' parameter (which we call m) forces the tree to split at least m observations into each terminal node. The default value of m in the popular `rpart` package in R is 7. Thus, for a given X_j, there are a total of $n+1-2m$ possible values for c. For each of the $p(n+1-2m)$ splits the sample means are estimated and the resulting sum of squares (SSE) is evaluated.

After finding the minimum SSE split for some predictor X_j and some split point c, the algorithm sends all the data that satisfy the rule $X_{ij} < c$ to the left child of the root node of a tree and the data that do not satisfy the rule to the right child. The splitting algorithm is then applied independently to each child. All allowable splits are examined and the best one for each child is chosen. The algorithm continues until a stopping rule is met. A default stopping rule in the `rpart` package is that the best split must reduce at least 1% of the original SSE. Obviously the algorithm will refuse to split if the number of observations in a node is not at least $2m$.

Once a full tree has been created, the sample mean for a terminal node serves as the fitted value for each observation in that node. A new observation is predicted using the sample mean of the terminal node within which it is evaluated to fall.

This splitting procedure can overfit the training data (e.g., [11]; [56]). After the splitting algorithm terminates, the pruning algorithm seeks to find an optimal subtree with respect to some optimality criterion based on out-of-sample prediction. The approach used by Breiman et al. uses crossvalidation to estimate the test error, though there are other methods—a good discussion is given in [66]. Unfortunately, when the data are randomly subdivided and individual trees are fit to subsamples of the data, a different set of splits may be selected. There is therefore no way to measure prediction error on the original tree directly without using a test set. Instead, Breiman et al. develop an algorithm to estimate the optimal *size* of the tree, and then correspondingly restrict the size of the original tree.

Breiman et al.'s pruning algorithm, called cost-complexity pruning, works by assigning a fixed cost $z \in [0, +\infty]$ to each terminal node in a tree. Let T be any tree, $\tilde{T}$ be the set of terminal nodes in T, and let $|\tilde{T}|$ be the cardinality of $\tilde{T}$. Define the "fitness" of a tree to be $Q(T) = SSE(T) + z|\tilde{T}|$, where $SSE(T)$ is the sum of the SSE in the terminal nodes of T.

Let the subtree rooted at h be T_h for any interior node h. The fitness of T_h with $|\tilde{T}_h|$ terminal nodes is defined as: $Q(T_h) = SSE(T_h) + z|\tilde{T}_h|$. The idea is to evaluate this fitness for several values of z on all possible subtrees T_h. For a fixed value of z, Breiman et al. show that there is a unique subtree that gives the best fitness as long as ties are broken by choosing the smaller subtree (the result essentially shows that if two trees have the same fitness, they must be nested). By increasing z from 0 to $+\infty$, Breiman et al. show that a nested sequence of such optimal subtrees trees $T_M \prec ... \prec T_1 \prec T_0$ is obtained (T_0 denotes the full tree and T_M denotes a tree with no splits). Breiman et al.'s method then chooses among these optimal trees based on the best crossvalidation performance. Crossvalidation performance in this context means that the optimal z is estimated via crossvalidation, and then that penalty $\hat{z}$ is applied to the original tree. An optional rule is to find the best crossvalidation performance and then choose the smallest optimal tree with crossvalidation performance within 1 standard error of optimal (called the 1-SE rule). Torgo [66] argues against this rule and we do not employ this option in our analysis.

Breiman et al.'s method of minimizing the SSE on a piecewise constant model is not the only approach to recursive partitioning; there have been many developments on the metric used to split the trees. Alexander and Grimshaw [3] fit linear regression at the terminal nodes instead of piecewise constants. The SUPPORT method given in [18] also uses linear fits at the terminal nodes but additionally forces the estimated mean model to be continuous. MARS [27] also uses the recursive partitioning framework, and provides a continuous and polynomial fit using splines.

2.3 Ensemble Methods

Regression trees, while interpretable, are not the best predictors. Breiman [8] shows a material improvement in efficiency by averaging together the predictions of several trees, where the individual trees are drawn from bootstrap samples of the data. The critical problem Breiman identified is that regression trees are "unstable". In this context, unstable means that if the original data are perturbed in a small way, the resulting model can change drastically. There are several regression tree models that are quite plausible given the data, and furthermore they give materially different predictions. The implication is that the predictions from regression trees are excessively variable. The bootstrap aggregation (bagging) procedure works in the same vein as model averaging [35] - by averaging the predictions from a set of plausible models, Breiman showed the overall averaged predictions are an improvement over the predictions from an individual tree. It is worth noting that bagging does not reduce bias, only variance [34].

Bagging is an example of a general type of prediction model called an 'ensemble'. Ensemble methods for regression construct a set of weak learners and then aggregate the predictions [34]. There are many types of ensembles, including Bagging [8], Boosting [26], Random Forests [9], Bayesian Additive Regression Trees ([20]), and others. Furthermore, ensemble methods have been studied in detail and many variants of the original algorithms exist. Our main focus is on Random Forests. However, because there have been extensions to BART that adapt to heteroskedasticity, we consider this method also. Random Forests and BART aggregate regression trees in a slightly different way.

2.3.1 Random Forests

Random Forests are first proposed by Breiman [9]. Breiman [9] observes that gains in precision of predictions due to bagging are limited due to the fact that prediction errors from individual trees in the ensemble are somewhat correlated. To see why this is a problem, consider the following example from [34]: consider B identically distributed, but correlated random variables with variance σ and positive pairwise correlation ρ. The variance of the average of B random variables as $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$. The term $\frac{1-\rho}{B}\sigma^2$ can always be reduced close to zero with a large enough B. The relevant term is $\rho\sigma^2$. By keeping ρ small, very good predictive performance is attainable. The analogue of this is using an average of B trees created from bootstrap samples of the data.

Like bagging, a random forest is also an average of many regression trees, but Breiman [9] proposes a method for weakening the correlation among the trees' prediction errors. To accomplish this each tree can only use a random subset of the predictor variables at a given split, and each tree is fit on a bootstrap sample of the data instead of the original data. The size of the random subset of predictors is a tuning parameter called r. The subset given to each split is chosen uniformly at random from all $\binom{p}{r}$ possible combinations.

Random Forests possess some unique attributes that make them desirable as a prediction method. First, as each tree is grown on a bootstrap sample of the data, each tree has an implicit holdout set called the "out-of-bag" sample, consisting of approximately 37% of the original sample. This built-in test set provides a direct measure of mean squared prediction error, called the “out-of-bag error”. Breiman proves that this error estimate is unbiased for the test error. As a consequence, the test error can be estimated without needing either cross validation or a holdout set.

Breiman also hypothesized that Random Forests could be used to measure the importance of the variables as predictors of the response. He explains the procedure in the context of classification, although the same principles hold for regression. The idea is to permute the values of a column X_j, and then compute the 'Out-of-bag' error on all observations. Since X_j and Y now have in all likelihood no actual relationship, the difference in performance between the original data and the permuted data should tell us something about how useful X_j was in predicting Y. By ordering the variables in decreasing order of induced error, we have a ranked list of variable importance. Strobl et al. [60] showed that this method actually prefers correlated variables over uncorrelated ones and continuous variables over discrete ones. Additionally they recommend subsampling over bootstrapping. Friedman and Hall [28] show that subsampling 50% of the data without replacement is comparable to bootstrapping the full data. Strobl et al. [59] introduce the conditional inference forest, which they empirically show to have unbiased variable selection. Here unbiased means that their variable selection gives no preference to correlated predictor variables. The consequence is that it seems preferable to default to using 50% subsampling over bootstrapping in the implementation of a random forest.

Finally, Random Forests consist of an aggregation of a set of non-interacting trees (two arbitrary trees in a random forest are generated completely independently from each other). The relevant consequence of this is that the trees can each be computed on a separate core/machine and results aggregated together with minimal effort. The procedure is "embarrassingly parallel", that is, using C cores results in a speedup of roughly C (with minimal overhead). An example of a procedure that is not embarrassingly parallel is boosting, which works by re-weighting the residuals from the prediction of the first $K-1$ trees to create the Kth tree. Since this algorithm is sequential in nature, there is no advantage to using multiple cores.

2.4 Information Criteria

In our work in Chapters 3-5, we need a method for selecting models from a set of candidates with different combinations of explanatory variables and other unusual features. We use information criteria (IC) to help in this selection process.

Kullback and Leibler [39] derived a framework for finding the information lost when a model $g(x, \theta)$ is used to approximate the true distribution $f(x)$ for a random variable X. The well known Kullback-Leibler (KL) information is:

$$\int_x f(x) \ln \frac{f(x)}{g(x, \theta)}.$$

For the purposes of model comparison, the formula is usually split up into two terms:

$$\int_x f(x) \ln f(x) - \int_x f(x) \ln g(x, \theta).$$

By observing that the first term of the formula contains elements that depend only on the truth (not the model, $g(x, \theta)$), we typically seek to evaluate only the second term. We call this the relevant part of the KL information. The relevant part can be expressed as:

$$E_f[\ln(g(x, \theta))], \tag{2.1}$$

where E_f indicates that the expectation is taken under the truth.

Akaike [1] shows under some regularity conditions it is possible to approximate (2.1) with $\ln L(\hat{\theta}|Y)$, the maximized log-likelihood of a model fit via maximum likelihood. Akaike showed that this is a biased estimator, but the asymptotic bias is given by ν, the number of parameters that were estimated by maximum likelihood. Akaike's result does not specify a model for g, which means that AIC can be applied to any model where the regularity conditions apply. For 'historical reasons' [14] information criteria are typically multipled by -2. Akaike's formula is then:

$$AIC(g(x, \theta)) = -2 \ln L(g(x, \theta)) + 2\nu,$$

where $L(g(x))$ is the maximized log-likelihood of $g(x, \theta)$ on the training data.

A nice explanation of the process to derive AIC is found in Cavanaugh [17], which we describe here. As before we let f denote the distribution of the true generating model and we let θ represent the parameters of an approximating model. We also define θ_f to be the parameters of the true model. Cavanaugh begins by writing the relevant part of the KL information as:

$$d(\theta, \theta_f) = E_f\{-2 \ln L(\theta|Y)\}, \tag{2.2}$$

where as before E_f is the expectation under the generating model and $L(\theta|Y)$ is the likelihood of the approximating model. After fitting the model via maximum likelihood, $\hat{\theta}$ is the maximum likelihood estimate of θ. The goal is to evaluate:

$$d(\hat{\theta}, \theta_f) = E_f\{-2 \ln L(\theta|Y)\}\big|_{\theta=\hat{\theta}}. \tag{2.3}$$

Unfortunately we do not know the distribution E_f and thus we cannot estimate this quantity directly. Instead (2.3) is approximated with $-2 \ln L(\hat{\theta}|Y)$, which is the maximized likelihood of the model (a quantity that is possible to evaluate). The asymptotic bias of this estimator is:

$$E_f\{-2 \ln L(\hat{\theta}|Y)\} - E_f\{E_f\{-2 \ln L(\theta|Y)\}\big|_{\theta=\hat{\theta}}\}. \tag{2.4}$$

Akaike estimated this bias to be twice the dimension of θ.

Akaike's result can be corroborated empirically with a simple simulation. Let the true model and fitted model be given as:

$$Y \sim N(\mu, 1),$$

and the true value of $\mu = 0$. The goal is to directly estimate the bias in the maximized log-likelihood when we fit this. Since we are fitting the correct model to these data, we could appeal to the chi-square distribution and obtain an answer mathematically. The point of this setup is to demonstrate the simulation-based approach. The simulation-based approach is useful in more complicated circumstances when we cannot write down a formula for the expected bias.

Evaluating (2.1) by simulation requires knowing $f(x)$, which in this example we do. The simulation approach directly evaluates $\ln L(\hat{\theta}|Y)$ and $\ln L(\theta|Y)\big|_{\theta=\hat{\theta}}$, and takes the difference to approximate the bias (note that we have taken out the factor of -2). We then repeat this process over multiple datasets drawn from f to approximate an expectation under f. Our algorithm for approximating the bias of this model is:

1. We draw a sample of size 1000. We call this sample the training data.

2. We estimate $\hat{\mu}$ from the data using maximum likelihood with the usual estimator $\hat{\mu} = n^{-1} \sum\limits_{i \in train} y_i$. In our setup, σ is known to be 1.

3. We find the maximized log-likelihood on the training data. This amounts to evaluating

$$Q_1 = \sum_{i \in train} \left(-\frac{1}{2} \ln 2\pi - \frac{(y_i - \hat{\mu})^2}{2} \right)$$

   which is an estimate of $\ln L(\hat{\theta}|Y)$.

4. We draw a new dataset from the true model of the same size as the training data. We call this new sample the test data. We evaluate the likelihood on the test data, that is:

$$Q_2 = \sum_{i \in test} \left(-\frac{1}{2} \ln 2\pi - \frac{(y_i - \hat{\mu})^2}{2} \right)$$

   which is an estimate of $E_f\{\ln L(\theta|Y)\big|_{\theta=\hat{\theta}}\}$.

5. The difference between Q_1 and Q_2 is an unbiased estimate of the bias when fitting this model.

6. We repeat these steps several times to obtain the distribution of $Q_1 - Q_2$, and to precisely estimate $E_f(Q_1 - Q_2)$.

In Figure 2.1, we see the distribution of $Q_1 - Q_2$. It is worth noting that, relative to the bias, the amount of variance is relatively large. To estimate the bias using this method we use $100,000$ repetitions of the aforementioned algorithm. The estimated standard error is 0.1. This simulation estimates the bias with a 95% confidence interval of $1.1 + / - 0.2$ (Akaike's approximate result is that the bias should be 1). In our experience it often takes many simulation replicates to approximate the bias to a sufficiently low standard error. The approach we take to approximate IC in nonstandard cases therefore relies heavily on storing computed penalty values in tables. We do on-the-fly penalty value simulation only when necessary (discussed later during Chapter 3).

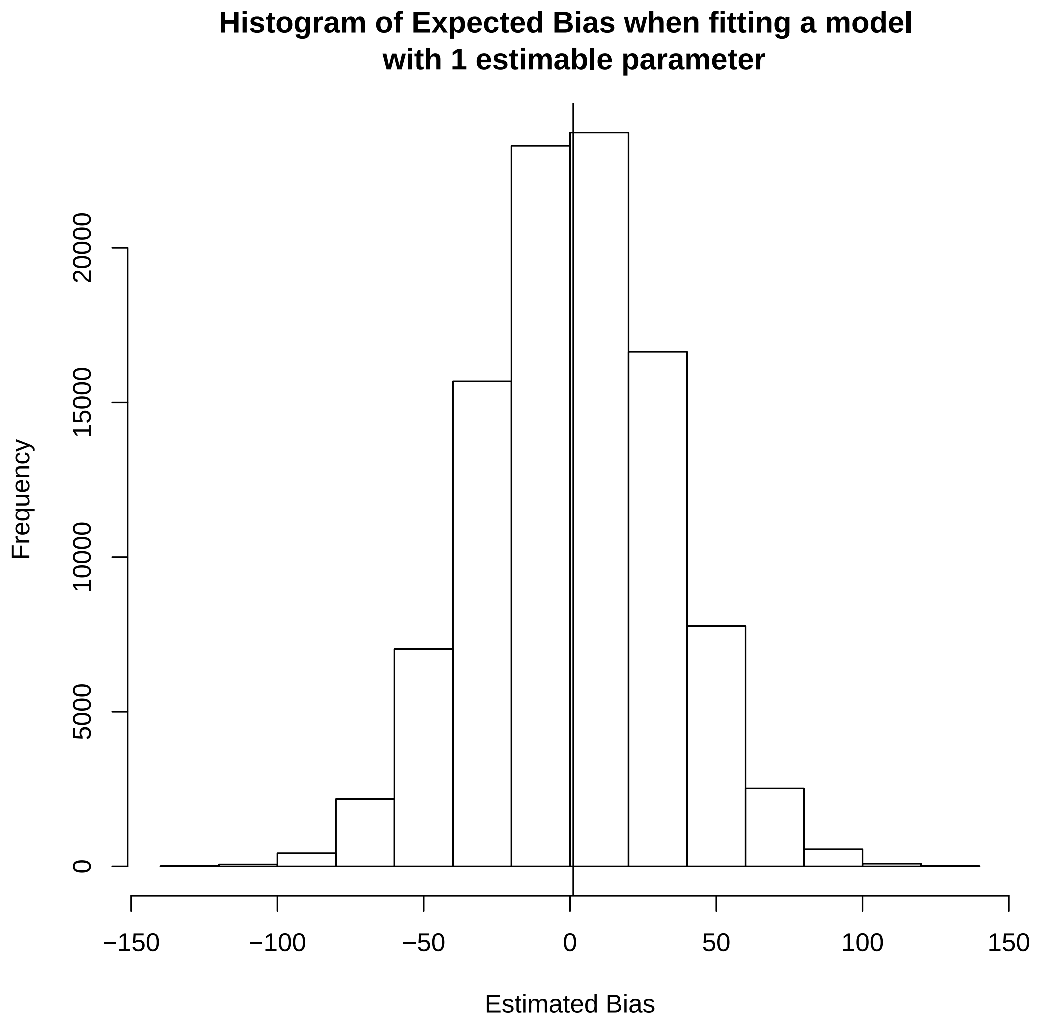


Figure 2.1: Histogram of simulated estimates of bias when using the evaluated log-likelihood on training data as a proxy for the relevant part of the KL information. A vertical line (barely discernable) is shown at the average of the bias estimates, 1.1. The standard error of the location of this line is 0.1.

AIC is unhelpful when samples are small. The AIC penalty approximation works asymptotically, and the true bias can be heavily underestimated by the AIC formula in small samples [13]. There are some developments to extend Akaike's ideas to small samples. In general, the formulations of the small sample bias are model-dependent. Takeuchi [64] develops a criterion (TIC) that works under model misspecification, however Burnham and Anderson [13] point out that relatively large sample sizes are required to estimate the terms to sufficient accuracy.

The small sample version of AIC for linear regression was initially proposed by Sugiura [63]. Hurvich and Tsai [36] named this quantity the "corrected AIC", or AICc. They demonstrated the superiority of AICc over AIC in small samples and additionally recommend it for use on time series models and nonlinear regression. The penalty in AICc has the form $2n/(n-\nu-1)$, where ν is the number of estimated parameters. This is asymptotically equivalent to Akaike's penalty (observe that by setting $n >> \nu$, the denominator is dominated by n, and the fraction approaches 2) but becomes much larger when the number of parameters in a model is an appreciable fraction of the sample size.

To derive the AICc formulation in the linear regression framework, we return to the derivation by Cavanaugh. Cavanaugh assumes the generating model is of the form:

$$y = X\beta_f + \epsilon \qquad\qquad \epsilon \sim N(0, {\sigma_f}^2 I)$$

and the candidate model is of the same form, in particular:

$$y = X\beta + \epsilon \qquad\qquad \epsilon \sim N(0, \sigma^2 I).$$

The log likelihood for the candidate model is given by:

$$\ln L(\theta|Y) = -\frac{n}{2}\ln 2\pi - \frac{n}{2}\ln\sigma^2 - \frac{1}{2\sigma^2}(y - X\beta)'(y - X\beta).$$

where $\theta = (\beta, \sigma^2)$. Now we need to evaluate (2.4) using this form of the likelihood. $E_f\{\ln L(\hat\theta|Y)\}$ can be interpreted as the average maximized log-likelihood of the candidate model over all data sets.

Cavanaugh gives equation (2.5) directly. Here we provide the derivation:

$$E_f\{-2\ln L(\hat\theta|Y)\} = -2E_f\{-\frac{n}{2}\ln 2\pi - \frac{n}{2}\ln\sigma^2 - \frac{1}{2\hat\sigma^2}(y - X\hat\beta)'(y - X\hat\beta)\}$$

$$E_f\{-2\ln L(\hat\theta|Y)\} = n\ln 2\pi + E_f\{n\ln\hat\sigma^2\} + E_f\{\frac{1}{\hat\sigma^2}(n\hat\sigma^2)\}$$

$$E_f\{-2\ln L(\hat\theta|Y)\} = n\ln 2\pi + E_f\{n\ln\hat\sigma^2\} + n \tag{2.5}$$

We turn to evaluating $E_f\{E_f\{-2\ln L(\theta|Y)\}|_{\theta=\hat{\theta}}\}$ on the linear homoskedastic model. First we evaluate the inner expectation (conditional on one dataset drawn from f). Again Cavanaugh simply gives equation (2.6), and we show here the derivation:

$$E_f\{-2\ln L(\theta|Y)\}|_{\theta=\hat{\theta}} = -2E_f\{-\frac{n}{2}\ln 2\pi - \frac{n}{2}\ln\sigma^2 - \frac{1}{2\sigma^2}(y-X\beta)'(y-X\beta)\}|_{\theta=\hat{\theta}}$$

$$E_f\{-2\ln L(\theta|Y)\}|_{\theta=\hat{\theta}} = n\ln 2\pi + n\ln\hat{\sigma}^2 + E_f\left\{\frac{1}{\sigma^2}(Y-X\beta)'(Y-X\beta)\right\}|_{\theta=\hat{\theta}}$$

note that since $(Y-X\beta)'(Y-X\beta)$ is a quadratic form, $E_f\left\{(Y-X\beta)'(Y-X\beta)\right\}|_{\theta=\hat{\theta}} = n{\sigma_f}^2 + (X\beta - X\hat{\beta})'(X\beta - X\hat{\beta})$, and after simplifying we arrive at:

$$E_f\{-2\ln L(\theta|Y)\}|_{\theta=\hat{\theta}} = n\ln 2\pi + n\ln\hat{\sigma}^2 + n\frac{{\sigma_f}^2}{\hat{\sigma}^2} + \frac{1}{\hat{\sigma}^2}(X\beta - X\hat{\beta})'(X\beta - X\hat{\beta}). \quad (2.6)$$

Now we need to take the expectation of (2.6) with respect to E_f, i.e., with respect to the distribution of the MLEs. Cavanaugh shows the third and fourth terms reduce to inverse chi-square random variables (note the caveat that the expectation of the inverse chi-square is not defined if the degrees of freedom is fewer than 2). This yields:

$$E_f\{E_f\{-2\ln L(\theta|Y)\}|_{\theta=\hat{\theta}}\} = E_f\{n\ln\hat{\sigma}^2\} + n\ln 2\pi + \frac{n^2}{n-\nu-2} + \frac{np}{n-\nu-2} \quad (2.7)$$

Taking the difference of (2.7) and (2.5) yields the AICc formulation.

Following the same framework as Cavanaugh, we can write the linear model using a diagonal variance Σ and compute the same two terms. This is the linear heteroskedastic model assuming independence between observations. Making these changes results in:

$$E_f\{-2\ln L(\hat{\theta}|Y)\} = n\ln 2\pi + E_f\{\ln|\hat{\Sigma}|\} + n \quad (2.8)$$

and

$$E_f\{-2\ln L(\theta|Y)\}|_{\theta=\hat{\theta}} = n\ln 2\pi + \ln|\hat{\Sigma}| + \mathrm{Tr}(\hat{\Sigma}^{-1}\Sigma_f) + (X\beta - X\hat{\beta})'\hat{\Sigma}^{-1}(X\beta - X\hat{\beta}). \quad (2.9)$$

Evaluating the expectation of (2.9) and taking the difference gives us the bias for the heteroskedastic model:

$$E_f\{-2\ln L(\hat{\theta}|Y)\} - E_f\{E_f\{-2\ln L(\theta|Y)\}|_{\theta=\hat{\theta}}\} = \\ E_f\{\,\mathrm{Tr}(\hat{\Sigma}^{-1}\Sigma_f) + (X\beta - X\hat{\beta})'\hat{\Sigma}^{-1}(X\beta - X\hat{\beta}) - n\}. \quad (2.10)$$

In Chapter 3, when we discuss unreplicated factorials, the number of parameters is often an appreciable fraction of the sample size. AICc is not fully suitable for our problem because infinite or monotone likelihoods can be produced for models where the total number of parameters is considerably fewer than $n-1$. Using AICc on our problem always results in selecting one of the pathological cases of Loughin and Rodríguez [44] as the best model (this is explained further later in this Chapter). Even eliminating these models from consideration is not adequate, because in pilot studies we found that in many data sets simulated from normal distributions, a near-pathological model can be found for which the maximized log-likelihood is much smaller than the penalty adjustment can account for. Essentially the problem is that our model is heteroskedastic linear regression, which does not fit under the AICc framework, and under our setup the bias actually depends on the values of the true parameters. We simulate values from (2.10) instead of computing AICc.

In Chapters 4 and 5, we discuss regression tree models, which are a type of linear regression model, under heteroskedasticity. Making a split on a regression tree involves optimizing over the split location. Many different pairs of piecewise constants are considered, and we choose the one that optimizes some criterion. We choose the criterion to be maximum likelihood, but the split point parameter and our introduction of variance parameters means that we cannot apply AICc. Our solution is to develop our own custom IC by simulating the entire splitting process. We generate data from a null distribution, find the best split, and then approximate the expected bias in the training set likelihood by evaluating (2.10) on each of the three types of splits.

2.5 Heteroskedasticity

Gelfand [30] shows that heteroskedasticity is relatively common in real data sets. She analyzed 42 data sets from Chipman et al. [20], noting that the datasets did not appear to be selected for reasons related to heteroskedasticity. Gelfand performs an analysis on each data set to determine whether it appears to exhibit a variance that changes with the mean. Because these data sets had numerous variables and potential nonlinearity, she first fits a random forest to each data set to remove the mean trend. She then assesses the relationship between the predicted values and the squared residuals, a procedure recommended by Carroll and Ruppert[16]. Two important results emerged from these analyses. First, Gelfand detected heteroskedasticity in 25 of the 42 datasets. Although it is not clear whether this proportion is an unbiased estimate of the true proportion of all data sets that exhibit heteroscedasticity, it is clear that heteroscedasticity is a common problem in real data. The second observation is related to the magnitude of heteroskedasticity. Gelfand computes the ratio between the average absolute residual of the highest 10% of predicted means and the average absolute residual of the smallest 10% of predicted means. We will refer to this quantity in general as the Gelfand ratio. This ratio is roughly measuring how much the error variance changes as a function of the mean. Gelfand showed that on the data sets identified as heteroskedastic, the median ratio was 3.08 with a first quartile at 2.00 and a third quartile at 5.51. We take from this the intuition that these are reasonable amounts of heteroskedasticity to see in real data. Our simulations will typically showcase a standard deviation ratio of 5 : 1.

2.5.1 Effects of Heteroskedasticity on Trees

Heteroskedasticity has been studied in the context of regression trees. Ruth and Loughin [57] identify that the Breiman et al. algorithm for regression trees tends to make poor split selection choices under heteroskedasticity. In particular they use a simple simulation model to gain better understanding of the splitting performance of the popular R implementation of Breiman et al.'s regression tree `rpart`. The `rpart` algorithm uses a default minimum node size m of 7 and a forward stopping rule that requires each split to reduce at least 1% of the overall SSE.

The general setup of the Ruth and Loughin work considers a single predictor variable x and a response y. The response y is related to x through a 10-step increasing mean function; in particular:

$$x = 1, 2, ...1000,$$
$$\mu_x = \lceil \frac{x}{100} \rceil,$$
$$y = \mu_x + \epsilon_x,$$

where μ_x is the mean at x. The model for ϵ_x is

$$\epsilon_x = \begin{cases} N(0, 1^2), x \leq 500 \\ N(0, s^2), x > 500 \end{cases} \quad s = 1, \dots 10$$

Ruth and Loughin demonstrate the critical issue with regression trees under this setup as s gets large. The problem is that the tree prefers to make splits in the high variance area ($x > 0.5$), to the exclusion of splits in the low variance area ($x < 0.5$). This is because the high variance area has more overall SSE to be reduced. Thus `rpart` ignores multiple legitimate splits in the low variance area, which should be objectively quite easy to find because the signal-to-noise ratio in the low variance area is quite a bit higher.

We have mentioned that `rpart` makes a split only if it reduces the overall SSE by at least 1% of the total. Ruth and Loughin demonstrate cases where even if the variance in the low-variance area is set to be 0, the default implementation still cannot find the splits. A solution is to adjust the settings to allow the tree to split fully, requiring no improvement in SSE to continue to make splits. However, this is not enough to fix the problem. The cost-complexity pruning algorithm routinely removes the splits in the low variance area.

2.6 Heteroskedastic Unreplicated Factorials

In many research and quality-improvement endeavors, experiments are run using unreplicated full- or fractional-factorial designs at 2 levels per factor (generically referred to here as 2^k designs, where k is the number of factors when the design is a full factorial; see [69]). These experiments are generally intended to identify factorial effects that influence the mean response. This identification is made difficult by the fact that the natural full model is saturated, but many methods have been proposed to identify mean and variance effects regardless (see [32] for a review of methods).

There may furthermore be an interest in estimating process variance and in determining which factorial effects influence dispersion, particularly in quality improvement settings. While unreplicated experiments are obviously ill-suited for variance estimation, efforts have nonetheless been made to try to extract this information from the data. Box and Meyer [7] developed a seminal procedure to test for “dispersion effects” in data from a 2^k experiment using residuals from a given location model. Several authors have followed this approach by developing improvements or extensions to the Box-Meyer test (e.g., [67]; [5]; [12]; [47]). For a nice review of these procedures see [15].

All of these procedures use the same basic approach of first fitting a selected location model, and then using residuals from the location model to test for dispersion effects. The resulting tests are well known to be sensitive to the starting location model, so that different location models can yield completely different impressions regarding which dispersion effects are important ([53]; [12]; [47] [48]; [54]). In particular, there is potential for confounding to occur between location and dispersion effects: two active location effects that are excluded from the location model can impart a spurious dispersion effect or can mask a real dispersion effect at their interaction ([53], [47]). Thus, it is critical to have the correct model for location before attempting to identify dispersion effects. However, all of the procedures for identifying location models in unreplicated factorial designs are prone to missing small- or moderate-sized real effects [32]. In Section 3.3 we show several studies that have each been analyzed by different authors; in two of these, different “best” models have been identified from the same set of data.

Sequential approaches begin with some method for identifying location effects. They then use the chosen location model to form residuals, which are used for identifying dispersion effects. The methods for identifying the two models are generally completely separate; that is, typical dispersion-effect identification methods assume that the correct location model has been identified, without concern for possible error in the model. We briefly review one procedure for identifying location effects, and three procedures for identifying dispersion effects. These methods are chosen based on performance and apparent popularity.

We present details from the perspective of a full 2^k factorial experiment, although identical results hold for any fractional factorial using $n = 2^k$ runs in the equivalent design. Let $\boldsymbol{W}$ be the $n \times n$ design matrix including all main effects and interactions. We use $+1$ or just $+$ to denote the high level of a factor and -1 or $-$ to represent the low level. Consider a model consisting of p location effects and q dispersion effects, with $p, q = 0, 1, \ldots, n-1$. Denote the corresponding sets of location and dispersion effects by $\mathcal{L}$ and $\mathcal{D}$, respectively, so that a joint location-dispersion model can be represented by $(\mathcal{L}, \mathcal{D})$. Let $\boldsymbol{X} \subseteq \boldsymbol{W}$ be an $n \times (p+1)$ matrix containing the columns corresponding to the effects in $\mathcal{L}$, and $\boldsymbol{U} \subseteq \boldsymbol{W}$ be an $n \times (q+1)$ matrix containing the columns corresponding to the effects in $\mathcal{D}$. Both $\boldsymbol{X}$ and $\boldsymbol{U}$ contain a lead column of ones.

The Lenth test [40] is found by Hamada and Balakrishnan [32] to be in the class of best methods for identifying location effects in unreplicated 2^k factorials, and it is notable for its simplicity. Assume that responses arise from the model

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where $\boldsymbol{Y}$ is the $n \times 1$ vector of responses, $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)'$ is a vector of parameters corresponding to location effects, and $\boldsymbol{\epsilon}$ are i.i.d. errors. We assume that $\boldsymbol{\epsilon} \sim N(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$, where $\boldsymbol{0}$ is an $n \times 1$ vector of zeroes, σ^2 is the error variance, and $\boldsymbol{I}$ is an $n \times n$ identity. Note that the Lenth procedure assumes that no dispersion effects are present. This is typical of location-effect identification methods.

Location effects are estimated using ordinary least squares. Because of the orthogonality of the columns in $\boldsymbol{W}$, estimated location effects from the saturated model $\boldsymbol{X} = \boldsymbol{W}$, $\hat{\beta}_1, \ldots, \hat{\beta}_{n-1}$, are independent when there are no dispersion effects. Thus, the estimates for a particular column of $\boldsymbol{X}$ do not depend on which other columns from $\boldsymbol{W}$ are in $\boldsymbol{X}$. With this in mind, Lenth calculates a "pseudo standard error" (PSE) for the effect estimates as follows. Assuming effect sparsity (see, e.g., [69]), suppose that all estimated effects of median magnitude and lower are not from active effects. Then the PSE is calculated as

$$\text{PSE} = 1.5 * \text{median}_{\{j : |\hat{\beta}_j| < 2.5 s_0\}} |\hat{\beta}_j|,$$

where $s_0 = 1.5 * \text{median}_j |\hat{\beta}_j|$. The statistic $t_j = |\hat{\beta}_j| / \text{PSE}$ is used to test $H_0 : \beta_j = 0$ for each $j = 1, \ldots, n-1$. Lenth suggests comparing t_j to a $t_{(n-1)/3}$ distribution, although Loughin [41] and Ye and Hamada [70] find better critical values by simulation.

Box and Meyer [7] use residuals from a chosen location model, assumed known, to identify active dispersion effects. Let $r_1, \ldots, r_n$ be the residuals from any model fit, $\hat{\boldsymbol{Y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}}$. Assume that $\boldsymbol{U}$ has only one column in it besides the lead column of ones, and suppose that it corresponds to column d of $\boldsymbol{W}$. Let $d+$ and $d-$ be the sets of observations for which $w_{id} = +1$ and $w_{id} = -1$, respectively. Then the Box-Meyer statistic is

$$F_d = \frac{\sum\limits_{i \in d+} r_i^2}{\sum\limits_{i \in d-} r_i^2}.$$

Although this statistic looks like it should have an F distribution, the residuals in the numerator and denominator are not necessarily independent. Thus, the sampling distribution of F_d is not clear.

Bergman and Hynén [5] amend the Box-Meyer test by augmenting the location model in such a way that the residuals contained in $d+$ and $d-$ *are* independent. The augmented model consists of the original model plus all effects formed by the interaction of these effects with d. The Bergman-Hynén test statistic, D_d^{BH}, is structurally identical to F_d, except that it uses the residuals from the augmented location model rather than the original model. Assume that the original location model is correct, so that it contains all active location effects. Also assume that d is the only possible dispersion effect. Then D_d^{BH} has an F distribution with degrees of freedom depending on the number of effects *not* in the augmented location model. Note that this may be different for each d. Pan [53] and McGrath and Lin [47] show that this test can have inflated type I error rate or diminished power when the original location model fails to identify location effects of moderate size. Loughin and Malone [42] describe a testing approach based on D_d^{BH} that provides a measure of safety against this phenomenon.

Harvey [33] proposes a method for estimating dispersion effects in a general linear regression setting. Let $\boldsymbol{u}_i'$, $i = 1, \ldots, n$ be the ith row of $\boldsymbol{U}$. Harvey uses the model

$$\sigma_i^2 = \exp(\boldsymbol{u}_i'\boldsymbol{\delta}), \tag{2.11}$$

where $\boldsymbol{\delta} = (\delta_0, \delta_1, \ldots, \delta_q)$ is a $(q+1) \times 1$ vector of unknown parameters representing dispersion effects. From this model, he writes $\log r_i^2 = \boldsymbol{u}_i'\boldsymbol{\delta} + v_i$, where $v_i = \log(r_i^2/\sigma_i^2)$, and by analogy with a linear model, uses least squares to estimate $\boldsymbol{\delta}$. In the context of a 2^k factorial design, Brenneman and Nair [12] show that this results in

$$\hat{\delta}_d = \log\left(\frac{\prod\limits_{i \in d+} r_i^2}{\prod\limits_{i \in d-} r_i^2}\right)^{1/n} = n^{-1}\left(\sum_{i \in d+} \log r_i^2 - \sum_{i \in d-} \log r_i^2\right).$$

Brenneman and Nair [12] study the bias in several dispersion-effect identification methods. They show that all are biased, with the severity of bias depending on whether an additive or a log-linear model is assumed for the variances. They also show that the bias in the Harvey method is reduced or eliminated when the residuals are computed from the augmented location model used by Bergman and Hynén [5]. They refer to this as the "modified Harvey" method and recommend it for general use because its bias is limited to certain specific cases.

2.7 Maximum Likelihood and Variance Modeling Methods

Regression trees are normally estimated using what amounts to ordinary least squares. However, they have also been recast in a maximum likelihood framework, which then permits their later extension to other models. Su et al. [62] build a regression tree based on the assumption that the data are drawn from a normal distribution with homoskedastic variance structure. They demonstrate that their tree uses the same splits as the RPAB algorithm, and thus the only difference is in the pruning method, which we describe next. For the sake of simplicity, we assume that AIC is to be used in the algorithm, but any IC could be used instead.

Let T_h be the subtree rooted at internal node h, let $\tilde{T}_h$ be the set of terminal nodes in subtree h, let $|\tilde{T}_h|$ be the cardinality of the set of terminal nodes in subtree h, and let $L(T_h)$ be the log-likelihood of model rooted at T_h. They give the following formula for the fitness of a subtree, which is a likelihood based analogue of Breiman et al's cost complexity pruning [62]:

$$AIC(T_h) = -2L(T_h) + 2(|\tilde{T}_h| + 1)$$

This is connected this to AIC by observing the second term is supposed to be related to the number of parameters in the model. In this case, $|\tilde{T}_h|$ is the number of separate means that are fit, and one term is added for the common variance. Although Su et al. observe that the split point, s, is obtained via maximum likelihood, they omit it from the AIC formula. We discuss this omission in detail during Chapter 4. For the time being we use the term 'AIC' to indicate the result of this equation rather than a true AIC that recognized all the parameters in the model. The maximized likelihood is simply a function of the SSE on the training data, and they write:

$$AIC(T_h) = c + n \cdot \ln \sum_{t \in \tilde{T}_h} \sum_{y_i \in t} (y_i - \bar{y}_t)^2 + 2|\tilde{T}_h|$$

where they sum over the terminal nodes with $t \in \tilde{T}_h$ and over the observations inside a given terminal node during $y_i \in t$. Note that c is a constant across all models and discarded for the purposes of model comparison.

Now that an AIC can be evaluated on a given subtree, the approach proceeds as follows. Define the full tree as T_0, and consider subtrees T_h for an internal node h. The goal is to prune the node h such that removing the subtree rooted at node h yields a tree $T_0 - T_h$ with the best AIC. This is equivalent to pruning the subtree T_h with the maximum AIC value. By removing this subtree we arrive at $T_1 = T_0 - T_h$. We use the same approach to find a new weakest link and obtain a sequence of trees : $T_0, T_1 ... T_M$, where T_M is the tree containing no splits.

Choosing between the various trees is considered a model selection problem ([66]). Su et al. choose the best tree based on a holdout set, despite using an IC approach. The mean squared prediction error (MSPE) evaluated on a test set should be an unbiased estimate of each model's true MSPE. However, rather than selecting the model with the best SSE or MSPE on the test set, they strangely opt to add an additional IC penalty to the SSE. IC penalties correct a bias that results from using the maximized log-likelihood from the training set as a measure of the corresponding expected log-likelihood. This bias is not present when the same quantity is evaluated on a test set. Therefore, it is not clear why an additional penalty term is applied in their pruning procedure. The result of this unnecessary correction is that their pruning procedure should systematically favour models that are smaller than they need to be

Finally, Su et al. show some simulations where their method performs better than RPAB and one where it performs worse. It comes as no surprise that under their simulation setups, their method outperforms RPAB on models where the optimal number of splits is small, and performs materially worse when approximating the true mean function requires a large regression tree.

In general, we agree with the principle that a likelihood-based tree-fitting algorithm is an appealing substitute for the standard splitting and pruning approach based on SSE. In Chapter 4 we also use Maximum Likelihood to choose splits among variables, and we also use IC's to determine which subtrees to prune. The difference is in the details: when we model means in the tree, we apply the IC directly on the training data, eliminating the need for a holdout set. We also use a different formula for the IC (ours is simulated), and ours takes into account the splitting process (whereas theirs ignores it in the penalty formula).

2.7.1 Trees for Modeling Variance

Su et al. [61] give an approach for modeling the variance with regression trees. The approach is comparable to the early work on heteroskedastic factorials, in the sense that it assumes that the mean function is correctly specified before working on the variance model. They claim "In practice, variance modeling is a procedure that comes after the mean regression model fitting"; we argue in Chapters 3-5 that modeling the mean and variance simultaneously is preferable.

Their approach consists of creating a regression tree that models the squared residuals after a suitable mean model is found. For their purposes they demonstrate the approach on linear regression, though there is no particular restriction; any mean model suffices. They follow the usual RPAB approach: fit a tree to the squared residuals, construct a nested sequence of subtrees via pruning, and choose the 'best' one via cross validation. An interesting and promising approach they take is that they allow splits to either be of the form "Is $X_j \leq c$", or of the form "Is $\hat{\mu} < c$", where $\hat{\mu}$ represents the estimated mean function. The advantage of this is that this tree can easily model a common type of heteroskedasticity, where the variance is related to the mean [16].

Pruning is done in the spirit of Breiman et al. by defining the objective function $G_\alpha(T_h) = G(T_h) - \alpha|I_h|$, where I_h is the number of interior nodes in some subtree T_h, $\alpha > 0$ is a tuning parameter, and $G(T_h)$ is the sum of the SSE of the squared residuals in the terminal nodes of T_h. They show that for a split, the relevant fraction to consider is $\frac{G(T_h)}{|I_h|}$. By ordering the interior nodes by this quantity (higher being better), one arrives at a nested sequence of trees considered to be optimal for their size.

To choose among the trees of various sizes, they propose to use a holdout set and evaluate the likelihood function. They show that the log-likelihood is, up to an additive constant:

$$L_p = -\frac{1}{2} \sum_{t \in \tilde{T}_h} n_t \ln \hat{\sigma}_t^2$$

where t represents a terminal node. We have argued before that a principled approach is to choose the model that produces the maximum evaluated log likelihood on the test data. Su et al. however propose to choose the best tree via 'BIC'. In particular, they evaluate:

$$-2L_p + \ln(n)(p + |\tilde{T}_h| + 1)$$

for every model, and choose the one with the minimum value. We have argued before that this approach unnecessarily chooses smaller trees.

2.7.2 Bayesian Additive Regression Trees

Bayesian Additive Regression Trees (BART) ([20]) is an alternative, Bayesian approach to the Random Forest. The general concept is to model up the response y as a sum of the predictions from K different trees. In a sense this means that each tree tries to model the residuals from the remaining $K - 1$ trees, much like the generalized additive models [34]. One of the nice features of the Bayesian approach (compared to Random Forests) is the ability to construct credible intervals and predictive intervals. The main drawback of the Bayesian approach is speed, although this issue is mitigated to an extent with a good implementation [37].

Bleich [6] improved BART by incorporating the ability to model variance effects in a procedure he calls HBART. BART explicitly estimates a single variance σ^2 across all observations in the training data. When the data are heteroskedastic, this leads to poor credible and predictive intervals (too wide in the low variance area, too narrow in the high variance area) [6]. BART also suffers from some of the same problems described in Ruth and Loughin [57] in that it prefers to make excessive splits in the high variance area [6]. This problem can corrected by additionally modeling the variance as well as the mean. Bleich's model for the variance is a simple log linear model:

$$\ln {\sigma_i}^2 = \ln \sigma^2 + z_i \gamma$$

where z_i is a column vector of covariates and γ is a row vector of coefficients. Bleich's main argument in favour of modeling the variance is to provide better posterior credible/predictive intervals, not necessarily better mean estimation. However, Bleich does note improved performance in an RMSE sense using a simulated example with a single variable. The Gelfand ratio in his simulation setup is about $\sqrt{270} = 16.4$, which is materially above Gelfand's third quartile of effect size. Bleich further notes that in a different, 3 variable setup where a substantial component of the variance is unrelated to the mean, his HBART method outperforms BART in a RMSE sense. In Chapters 4 and 5 we consider a similar simulation model where the variances of the responses are not directly related to the means.

Chapter 3

Joint Location and Dispersion Modeling for Heteroskedastic Factorials

3.1 Introduction

In many research and quality-improvement endeavors, experiments are run using unreplicated full- or fractional-factorial designs at 2 levels per factor (generically referred to here as 2^k designs, where k is the number of factors when the design is a full factorial; see [69]). These experiments are generally intended to identify factorial effects that influence the mean response. This identification is made difficult by the fact that the natural full model is saturated, but many methods have been proposed to accomplish this goal (see [32] for a review of methods).

There may furthermore be an interest in estimating process variance and in determining which factorial effects influence dispersion, particularly in quality improvement settings. While unreplicated experiments are obviously ill-suited for variance estimation, efforts have nonetheless been made to try to extract this information from the data. Box and Meyer [7] developed a seminal procedure to test for "dispersion effects" in data from a 2^k experiment using residuals from a given location model. Several authors have followed this approach by developing improvements or extensions to the Box-Meyer test (e.g., [67]; [5]; [12]; [47]). For a nice review of these procedures see [15].

All of these procedures use the same basic approach of first fitting a selected location model, and then using residuals from the location model to test for dispersion effects. The resulting tests are well known to be sensitive to the starting location model, so that different location models can yield completely different impressions regarding which dispersion effects are important ([53]; [12]; [47] [48]; [54]). In particular, there is potential for confounding to occur between location and dispersion effects: two active location effects that are excluded from the location model can impart a spurious dispersion effect or can mask a real dispersion effect at their interaction ([53], [47]). Thus, it is critical to have the correct model for location before attempting to identify dispersion effects. However, all of the procedures for identifying location models in unreplicated factorial designs are prone to missing small- or moderate-sized real effects [32]. In Section 3.3 we show several studies that have each been analyzed by different authors; in two of these, different "best" models have been identified from the same set of data.

In all of these previous analyses, location effects are selected based on one criterion and dispersion effects based on another. There is no direct, objective measure that allows one to compare a model containing one set of location and dispersion effects to a model with a different combination of these effects. The methods in this paper address this shortcoming.

As an alternative to sequential model fitting, we propose in Section 3.4 to use a joint location and dispersion model for factor screening. This model results in a single likelihood that is used to estimate location and dispersion effects simultaneously. The maximized likelihood is then used for comparing models and selecting effects through an information criterion. Information criteria whose justification is based on asymptotic approximations are of dubious utility in this problem, where the potential number of parameters is roughly twice the number of observations. We therefore develop an exact criterion in Section 3.4.1 based on the corrected Akaike information criterion (AICc) of Hurvich and Tsai [36]. The form appears somewhat complex, so we simulate the penalty values for different model structures.

The space of all possible models is large, because the presence of dispersion effects can change the ordering of the estimated location effects. When $k \leq 4$, an exhaustive search of the model space is feasible; otherwise we propose using a genetic algorithm to search the space for good-fitting models. Model averaging techniques ([35]; [13]) provide a measure of the certainty associated with the importance of each location and dispersion effect and with each model combination. In Section 3.5 our joint modeling procedure is applied to the examples introduced in Section 3.3. In each case the procedure provides clear, interpretable results regarding which effects are important and which models are best. In one example, we identify a "best" model that had not previously been detected. Finally, in Section 3.6 we present results of a small simulation study comparing the joint modeling approach with a combination of popular location- and dispersion-effect identification techniques. The simulations show that a particular sequential approach for identifying location and dispersion models has slightly inflated type 1 error rate for identifying dispersion effects. The new procedure experiences a substantially smaller type 1 error rate for dispersion effects *while maintaining uniformly better power.*

3.2 Previous Approaches

Sequential approaches begin with some method for identifying location effects. They then use the chosen location model to form residuals, which are used for identifying dispersion effects. The methods for identifying the two models are generally completely separate; that is, typical dispersion-effect identification methods assume that the correct location model has been identified, without concern for possible error in the model. We briefly review one procedure for identifying location effects, and three procedures for identifying dispersion effects. These methods are chosen based on performance and apparent popularity.

We present details from the perspective of a full 2^k factorial experiment, although identical results hold for any fractional factorial using $n = 2^k$ runs in the equivalent design. Let $\boldsymbol{W}$ be the $n \times n$ design matrix including all main effects and interactions. We use $+1$ or just $+$ to denote the high level of a factor and -1 or $-$ to represent the low level. Consider a model consisting of p location effects and q dispersion effects, with $p, q = 0, 1, \ldots, n-1$. Denote the corresponding sets of location and dispersion effects by $\mathcal{L}$ and $\mathcal{D}$, respectively, so that a joint location-dispersion model can be represented by $(\mathcal{L}, \mathcal{D})$. Let $\boldsymbol{X} \subseteq \boldsymbol{W}$ be an $n \times (p+1)$ matrix containing the columns corresponding to the effects in $\mathcal{L}$, and $\boldsymbol{U} \subseteq \boldsymbol{W}$ be an $n \times (q+1)$ matrix containing the columns corresponding to the effects in $\mathcal{D}$. Both $\boldsymbol{X}$ and $\boldsymbol{U}$ contain a lead column of ones.

The Lenth test [40] is found by Hamada and Balakrishnan [32] to be in the class of best methods for identifying location effects in unreplicated 2^k factorials, and it is notable for its simplicity. Assume that responses arise from the model

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where $\boldsymbol{Y}$ is the $n \times 1$ vector of responses, $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)'$ is a vector of parameters corresponding to location effects, and $\boldsymbol{\epsilon}$ are i.i.d. errors. We assume that $\boldsymbol{\epsilon} \sim N(\boldsymbol{0}, \sigma^2\boldsymbol{I})$, where $\boldsymbol{0}$ is an $n \times 1$ vector of zeroes, σ^2 is the error variance, and $\boldsymbol{I}$ is an $n \times n$ identity. Note that the Lenth procedure assumes that no dispersion effects are present. This is typical of location-effect identification methods.

Location effects are estimated using ordinary least squares. Because of the orthogonality of the columns in $\boldsymbol{W}$, estimated location effects from the saturated model $\boldsymbol{X} = \boldsymbol{W}$, $\hat{\beta}_1, \ldots, \hat{\beta}_{n-1}$, are independent when there are no dispersion effects. Thus, the estimates for a particular column of $\boldsymbol{X}$ do not depend on which other columns from $\boldsymbol{W}$ are in $\boldsymbol{X}$. With this in mind, Lenth calculates a "pseudo standard error" (PSE) for the effect estimates as follows. Assuming effect sparsity (see, e.g., [69]), suppose that all estimated effects of median magnitude and lower are not from active effects. Then the PSE is calculated as

$$\text{PSE} = 1.5 * \text{median}_{\{j:|\hat{\beta}_j|<2.5s_0\}}|\hat{\beta}_j|,$$

where $s_0 = 1.5 * \text{median}_j|\hat{\beta}_j|$. The statistic $t_j = |\hat{\beta}_j|/\text{PSE}$ is used to test $H_0 : \beta_j = 0$ for each $j = 1, \ldots, n-1$. Lenth suggests comparing t_j to a $t_{(n-1)/3}$ distribution, although Loughin [41] and Ye and Hamada [70] find better critical values by simulation.

Box and Meyer [7] use residuals from a chosen location model, assumed known, to identify active dispersion effects. Let $r_1, \ldots, r_n$ be the residuals from any model fit, $\hat{\boldsymbol{Y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}}$. Assume that $\boldsymbol{U}$ has only one column in it besides the lead column of ones, and suppose that it corresponds to column d of $\boldsymbol{W}$. Let $d+$ and $d-$ be the sets of observations for which $w_{id} = +1$ and $w_{id} = -1$, respectively. Then the Box-Meyer statistic is

$$F_d = \frac{\sum\limits_{i \in d+} r_i^2}{\sum\limits_{i \in d-} r_i^2}.$$

Although this statistic looks like it should have an F distribution, the residuals in the numerator and denominator are not necessarily independent. Thus, the sampling distribution of F_d is not clear.

Bergman and Hynén [5] amend the Box-Meyer test by augmenting the location model in such a way that the residuals contained in $d+$ and $d-$ *are* independent. The augmented model consists of the original model plus all effects formed by the interaction of these effects with d. The Bergman-Hynén test statistic, D_d^{BH}, is structurally identical to F_d, except that it uses the residuals from the augmented location model rather than the original model. Assume that the original location model is correct, so that it contains all active location effects. Also assume that d is the only possible dispersion effect. Then D_d^{BH} has an F distribution with degrees of freedom depending on the number of effects *not* in the augmented location model. Note that this may be different for each d. Pan [53] and McGrath and Lin [47] show that this test can have inflated type I error rate or diminished power when the original location model fails to identify location effects of moderate size. Loughin and Malone [42] describe a testing approach based on D_d^{BH} that provides a measure of safety against this phenomenon.

Harvey [33] proposes a method for estimating dispersion effects in a general linear regression setting. Let $\boldsymbol{u}_i'$, $i = 1, \ldots, n$ be the ith row of $\boldsymbol{U}$. Harvey uses the model

$$\sigma_i^2 = \exp(\boldsymbol{u}_i'\boldsymbol{\delta}), \tag{3.1}$$

where $\boldsymbol{\delta} = (\delta_0, \delta_1, \ldots, \delta_q)$ is a $(q+1) \times 1$ vector of unknown parameters representing dispersion effects. From this model, he writes $\log r_i^2 = \boldsymbol{u}_i'\boldsymbol{\delta} + v_i$, where $v_i = \log(r_i^2/\sigma_i^2)$, and by analogy with a linear model, uses least squares to estimate $\boldsymbol{\delta}$. In the context of a 2^k

factorial design, Brenneman and Nair [12] show that this results in

$$\hat{\delta}_d = \log \left(\frac{\prod_{i \in d+} r_i^2}{\prod_{i \in d-} r_i^2} \right)^{1/n} = n^{-1} \left(\sum_{i \in d+} \log r_i^2 - \sum_{i \in d-} \log r_i^2 \right).$$

Brenneman and Nair [12] study the bias in several dispersion-effect identification methods. They show that all are biased, with the severity of bias depending on whether an additive or a log-linear model is assumed for the variances. They also show that the bias in the Harvey method is reduced or eliminated when the residuals are computed from the augmented location model used by Bergman and Hynén [5]. They refer to this as the "modified Harvey" method and recommend it for general use because its bias is limited to certain specific cases.

3.3 Examples

We use three examples from the literature to demonstrate some of the model uncertainty that is inherent in sequential approaches to identifying location and dispersion effects in unreplicated factorials. All three examples are 16-run designs in the 2^{k-l} series, where k here is the number of factors and l is the degree of fractionation. These examples have been analyzed multiple times in the literature. Table 3.1 lists papers in which these examples have been analyzed, along with the results from their different approaches. Note that "Lenth/modified Harvey" refers to our re-analysis using the Lenth test for location followed by the modified Harvey test for dispersion. The factor labels used here are those given in first-listed citation for each example.

The first example is the Welding example, a 2^{9-5} design attributed by Box and Meyer [7] to a technical report by Taguchi and Wu. The general consensus among the previous analyses is that factors B and C are active location effects, while C is an active dispersion effect. There is some uncertainty regarding whether factors H and/or J might also be dispersion effects, and two authors have found other location effects besides B and C.

Second is the Injection Molding experiment, a 2^{7-3} experiment given in Montgomery [50]. (The data given there contain four centerpoints that have been subsequently ignored by authors analyzing this example. We also ignore these centerpoints.) Montgomery's original analysis identified the interaction triple A, B, and AB as active location effects, and C as an active dispersion effect. McGrath and Lin [47] recognized that the dispersion effect C lies at the interaction of the next two largest location effects, G and CG. They determined heuristically that the difference between variances at the two levels of C could be largely explained by the product of the fourth and fifth location effects that were missing from Montgomery's model. A more formal analysis in McGrath [46] concludes that the dispersion

Table 3.1: Proposed models resulting from different analyses presented in the literature for three different examples described in Section 3.3. Note that the Lenth/modified Harvey method may be inappropriate for the Dyestuff example, because discreteness can create residuals of exactly zero.

Example	Authors	Loc. Effects	Disp. Effects
Welding	Box and Meyer [7]	B, C	C
	Wang [67]	B, C	C, H, J
	Ferrer and Romero [25]	B, C	C, J
	Bergman and Hynén [5]	B, C	C, H, J
	Nelder and Lee [52]	B, C, J	C; H or J
	Pan [53]	B, C, AC, AH, A, H[a]	—
	McGrath and Lin [47]	B, C	C
	Pan and Taam [54]	B, C	C; H or J
	Loughin and Malone [42]	B, C	C
	Lenth/modified Harvey	B, C	C, H, J
Injection Molding	Montgomery [50]	A, B, AB	C
	McGrath [46]	A, B, AB, G, CG	—
	Loughin and Malone [42]	A, B, AB, G, CG	—
	Lenth/modified Harvey	A, B, AB, G, CG	E
Dyestuff	Bergman and Hynén [5]	D	E
	McGrath and Lin [47]	D	E
	Lenth/modified Harvey	D, AB	C, BC

[a]AC and AH were considered active; main effects of A and H were included to adhere to the marginality principle.

effect disappears upon including these two extra effects in the location model. Loughin and Malone [42] gave a different analysis of these data supporting McGrath's results.

Last, we consider the Dyestuff data first given in Davies [23] and analyzed for dispersion effects in Bergman and Hynén [5]. The latter authors found a location effect for factor D and a dispersion effect for E, conclusions that were supported by McGrath and Lin [47]. Although previous analysis results are in agreement for this example, we see in Section 3.5 that this does not imply that there is no model uncertainty

All of the analyses cited in Table 3.1 were performed using sequential analysis approaches. Location effects were selected, then *conditional on the location effects*, dispersion effects were identified. (McGrath [46] subsequently reconsiders any identified dispersion effects for possible confounding with two location effects, but the dispersion effect must be initially identified using a sequential analysis.) Thus, all of these analyses are susceptible to errors both due to stochastic uncertainty and due to structural propagation of errors from the location analysis into the dispersion analysis, as discussed by Pan [53], Brenneman and Nair [12], McGrath and Lin [47] [48], and Pan and Taam [54]. McGrath and Lin [47] and Loughin and Malone [42] show that there is information within the data that can distinguish between two location effects and a dispersion effect. The model-selection procedure needs to be able to weigh the value of each interpretation using some objective measure. The analysis approach proposed in the next section does precisely that, and in addition, allows a direct measure of model uncertainty that is novel among techniques for these data.

3.4 Effect Selection using a Joint location and dispersion model

Using the same notation from Section 3.2, a joint location and dispersion model is

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{3.2}$$

where now $\boldsymbol{\epsilon} \sim N(0, \text{Diag}(\exp(\boldsymbol{U}\boldsymbol{\delta})))$, where Diag$(\cdot)$ makes a diagonal matrix out of a vector. The variance structure in this model is the same one used by Harvey [33], Cook and Weisberg [22], [16], and others. It has been studied in the context of 2^k factorial models by Wang [67], who discussed maximum likelihood estimation and derived properties of the estimates, as well as by Nair and Pregibon [51], Engel and Huele [24], and others. All previous work, however, has been done under the assumption that $(\mathcal{L}, \mathcal{D})$ is known or has been correctly identified prior to use of the model. Of course, it is unrealistic to expect that any particular method can achieve this requirement without uncertainty, as the examples in Section 3.3 show.

Parameter estimates $\hat{\boldsymbol{\beta}}$ and $\hat{\boldsymbol{\delta}}$ from (3.2) are found using maximum likelihood as in Harvey [33] and Wang [67]. The log-likelihood is

$$l((\boldsymbol{\beta}, \boldsymbol{\delta}); \boldsymbol{Y}) = -\frac{n}{2} \log 2\pi - \frac{1}{2} \sum_{i=1}^{n} \boldsymbol{u}_i' \boldsymbol{\delta} - \frac{1}{2} \sum_{i=1}^{n} \frac{(y_i - \boldsymbol{x}_i' \boldsymbol{\beta})^2}{\exp(\boldsymbol{u}_i' \boldsymbol{\delta})} \tag{3.3}$$

where y_i is the ith element of $\boldsymbol{Y}$, $\boldsymbol{x}_i'$ is the ith row of $\boldsymbol{X}$, and $\boldsymbol{u}_i'$ is the ith row of $\boldsymbol{U}$. Evaluated at the MLE this becomes

$$l((\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\delta}}); \boldsymbol{Y}) = -\frac{n}{2} (\log(2\pi \hat{\sigma}_0^2) + 1) \tag{3.4}$$

where $\sigma_0^2 = \exp(\delta_0)$ is the variance at the centerpoint of the design space (or when all dispersion effects are inactive). The parameter estimates for $\boldsymbol{\delta}$ have closed form if $q \leq 1$, but otherwise must be estimated using iterative numerical techniques.

Because this model uses a single likelihood for estimating both sets of parameters, model selection criteria such as Akaike's information criterion (AIC) and the Bayesian information criterion (BIC) are available (e.g., [13], [21]). We use the availability of information criteria as the central focus for our joint location- and dispersion-effect identification method, Exhaustive-Search Model Averaging (ESMA) described in detail in the following sections. To outline ESMA, we (1) carry out an exhaustive search of the model space, considering all possible viable combinations of $(\mathcal{L}, \mathcal{D})$; (2) calculate a special form of information criterion on each; and (3) use an approach resembling Bayesian Model Averaging ([35]; [13]) to identify those effects that are more or less likely to belong in the model. We can use this approach to quantify the support in the data for any given model.

3.4.1 Exhaustive search

Our goal is to be able to provide an assessment of the model corresponding to any combination of $(\mathcal{L}, \mathcal{D})$. In principle, this is not difficult to achieve. However, there are several logistical issues that must be addressed in order to complete this task.

First is the sheer size of the problem. In an unreplicated 2^k factorial, there are up to $2^k - 1$ location effects and $2^k - 1$ dispersion effects to consider. When dispersion effects are not considered, the orthogonality and equal standard errors among the location-effect estimates impose an ordering based on the magnitude of the estimates that does not depend on which other effects are in the model. Thus, only models consisting of the effects with the largest estimated magnitudes need to be considered. There are only 2^k such models to consider.

When dispersion effects are added, the location-effect estimates are no longer orthogonal, and their ordering can change depending on which effects of both types are in the model. Thus, there are nominally $2^{2^{k+1}-2}$ different $(\mathcal{L}, \mathcal{D})$ combinations that can be constructed. In most of these models there is no closed-form solution to the likelihood equations, and no obvious way to reduce the computations in the spirit of the leaps-and-bounds algorithm for linear regression [29]. For $k \geq 5$ no exhaustive search can be run under current computational capacity. In these cases, some kind of alternative search procedure, such as a genetic algorithm [49], must be used (see Section 3.4.4 for details).

The second complication is that not all $(\mathcal{L}, \mathcal{D})$ combinations lead to viable models. For one thing, $n = 2^k$, so simultaneously estimating all location and dispersion effects is impossible. However, even models with seemingly viable combined size $p+q+2 < n$ do not always result in valid parameter estimates. Loughin and Rodríguez [44] observe that sets $\mathcal{D}$ of size $q = 1$ cause the last term in the log-likelihood to factor into two separate sums for independent subsets of the data. A saturated location model can be found for one of the sums with only $p = n/2-1$, and fitting this model causes the dispersion effect to go to $\pm\infty$. For example, this occurs for $k = 4$ when $\mathcal{D} = \{A\}$ and $\mathcal{L} = \{B, C, BC, D, BD, CD, BCD\}$, among other cases. Thus models of total size $n/2 + 2$ can be constructed that lead to unbounded likelihoods. Loughin and Rodríguez [44] also observe that sets $\mathcal{D}$ that consist of two effects can combine with certain complementary location models of size $p = n/4 - 1$ to yield a likelihood with multiple monotonically increasing ridges. For example, when $k = 4$, $\mathcal{D} = \{A, B\}$ with $\mathcal{L} = \{C, D, CD\}$ causes this to occur (as does the same $\mathcal{D}$ with $\mathcal{L} = \{AC, AD, ACD\}$, $\{BC, BD, BCD\}$, or $\{ABC, ABD, ABCD\}$). Again, this means that models of a much smaller total size than n can lead to invalid parameter estimates.

Fortunately, the combinations $(\mathcal{L}, \mathcal{D})$ where this can occur are completely predictable *a priori*. Loughin and Rodríguez [44] give a series of criteria that determine whether a given location/dispersion model can be fit to a set of data. Reducing the search to only these models reduces the computational burden considerably.

In our implementation, we further restrict the searchable model space to models satisfying both $p \leq 5$ and $q \leq 5$. This reduces the model space by a significant fraction, and simulations suggest that it has little impact on the results of a search when these restrictions do, in fact, hold. We are not aware of any other procedure that is likely to perform well when a true model does not satisfy some similar sparsity criterion.

Counting the Feasible Models and Computational Tricks We present our approach for counting the models that can be feasibly evaluated in the context of a 2^4 design. Analysing any smaller design is an easy modification, and if the design is any larger (2^5 or more), we are unable to evaluate every model. We begin by demonstrating the 'bitset' paradigm, where we describe a relationship between the factorial structure and base 2. We begin by ordering the columns of X in the following order:

$$\{I, A, B, AB, C, AC, BC, ABC, D, AD, BD, ABD, CD, ACD, BCD, ABCD\}.$$

Furthermore, we assign a 0-indexed number, $0-15$, to the columns in the same order. Notice that a result of this is that 0 represents I, 1 represents A, 2 represents B, 3 represents AB, and so forth up to 15 representing the $ABCD$ interaction. Of further relevance is that C is number 4 and D is number 8. Thus A, B, C, and D are given numbers in powers of two: $1(A) = 2^0$, $2(B) = 2^1$, $4(C) = 2^2$, and $8(D) = 2^3$. As a consequence of this ordering we can quickly compute the column corresponding to the interaction of two columns. This column is computed by taking the bitwise exclusive-or (XOR) of the two columns. For example, the interaction of the ABD effect and the ACD effect is BC. We can do this quickly on a computer by evaluating

11 (ABD) XOR 13 (ACD) $= \{1011\ XOR\ 1101\} = 0110 = 6$ (AC).

The effect of this is that we do not actually need to multiply the entries in the 11th column of X and the 13th column of X to find out the interaction column. This turns out to be useful for reducing runtime as the conditions outlined in Loughin and Rodríguez [44] require us to compute multiple interactions per model.

We now define models in boolean form as a concatenation of the terms in $\mathcal{L}$ and $\mathcal{D}$. We take the additional step of replacing the 16 possible location effects and 16 dispersion effects each with a boolean value (coded as 0/1). As an example, $\mathcal{L} = 1000000000000001$ refers to the location model of $\mathcal{L} = \{ABCD\}$. We always put the intercept of both the location and dispersion model into the model; as a consequence, the $1st$ digit and $17th$ digit is always 1.

An example concatenated model might be:

$$M = 1000000000000001 - 1000000000000000$$

which in our shorthand would refer to the model

$\mathcal{L} = \{ABCD\}$

$\mathcal{D} = \{\emptyset\}$.

The advantage of this representation is that all the models are now representable by a 32-bit integer. This is useful as it is a default datatype in many programming languages, such as **int** in C.

The goal of counting the feasible models reduces to constructing a set S consisting of a list of these 32-bit integers and finding the cardinality. We take a standard backtracking approach, implemented with recursion. We define b and $index$ to represent a partial model M. A partial model consists of defined bits in M plus some undefined bits. An example partial model be $M = 10[uuuuuuuuuuuuu - 1uuuuuuuuuuuuuu]$, where u represents an undefined bit. The interpretation of this partial model is that location effect A is definitely not included in M, and everything else is undecided. We represent this partial model with two values: b contains the values 10 and $index$ contains the value 3—the interpretation of index is the location of the next undecided bit.

The algorithm is:

- Define a function $F(b, index)$. F constructs a set S as a list of integers.
- We initialize $F(0,0)$ to construct the model list.
- To evaluate F, determine whether the model defined by b is a valid model (using the Loughin and Rodríguez conditions).
- If index is 31, we add the entry bits to the set S and we do no further work on this branch.
- If index is less than 31, we evaluate $F(bits, index+1)$ and also $F(bits+2^{index}, index+1)$.
- If b does not give a valid model, then no superset of b will give a valid model (by superset we mean adding any extra location or dispersion terms). We will do no further work on this branch.

After running our algorithm, we find the cardinality of the set S is $1,442,837$ models. We note later that some of these models are actually still too large to be considered as serious models. We have, however, restricted ourselves to ones where we can be sure to avoid the pathology outlined in Loughin and Rodríguez.

3.4.2 Corrected heteroscedastic information criterion

On each model, an information criterion (IC) is computed. Information criteria are generally based on the Kullback-Leibler (KL) information, which is a measure of the discrepancy between a candidate model and the true structure that generated the data (Akaike [1] [2]; [38]). The KL information consists of two terms, one of which is constant for all models and hence is discarded. The other part is -2 times the expectation of the log-likelihood of the candidate model with respect to the true model, evaluated at the MLE. This expectation cannot be computed because the true model is unknown. The maximized log-likelihood—in this case (3.4)—can be used as an estimate of the expectation, but it exhibits bias that grows as the size of the model grows ([1]). Therefore an IC generally consists of -2 times the maximized log likelihood, say l, plus a "penalty" term that adjusts for the bias.

Akaike [1] gives an asymptotic estimate for this bias that works well for most models. The result is the well-known AIC. Hurvich and Tsai [36] develop a small-sample estimate of bias for homoscedastic linear and nonlinear regression models, which they use to form the "corrected AIC" (AICc). Neither of these estimates is suitable for our problem because, as noted above, unbounded or monotone likelihoods can be produced for models where the total number of parameters is considerably less than $n-1$. Using either of these penalties on our problem always results in selecting one of the cases of Loughin and Rodríguez [44] as the best model. Even when we remove these models from consideration, simulations show that pathologically large likelihoods can result from models that "almost" satisfy the Loughin-Rodríguez conditions, yielding bizarre and unrealistic IC values.

We therefore must derive a corrected IC that is appropriate for estimating both location and dispersion effects from the heteroscedastic model (3.2). To do this, we follow the general approach of Hurvich and Tsai [36]. To start, let a "*" superscript indicate the true value of a parameter and for any candidate model $(\mathcal{L}, \mathcal{D})$ a hat indicates the corresponding MLE. We assume (as do Cavanaugh [17] and Hurvich and Tsai [36]) that for any model that is fit, the parameters contained in $(\boldsymbol{\beta}, \boldsymbol{\delta})$ include all of those in $(\boldsymbol{\beta}^*, \boldsymbol{\delta}^*)$. When this is not the case, the bias in $-2l$ may be arbitrarily large in the opposite direction, and hence the model's IC will be much larger than for models that satisfy this assumption. It therefore suffices to consider the true model and fitted model as having parameters of the same dimension, $(p + q + 2)$ with $(\boldsymbol{\beta}^*, \boldsymbol{\delta}^*)$ possibly containing some zero values. Finally, let G be the distribution of $\boldsymbol{Y}$.

The quantity that must be estimated is

$$-2l^E(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\delta}}) \equiv E_G(-2l(\boldsymbol{\beta}, \boldsymbol{\delta}; \boldsymbol{Y}))\Big|_{(\boldsymbol{\beta}, \boldsymbol{\delta}) = (\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\delta}})}.$$

The maximized log-likelihood, $-2\hat{l}(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\delta}}) = -2l(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\delta}}; \boldsymbol{Y})$, is used to estimate this quantity. Thus, the "penalty" that must be calculated is the bias, $B = -2E_G(l^E(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\delta}}) - \hat{l}(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\delta}}))$.

For model (3.2), rewriting (3.4) as

$$-2\hat{l}(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\delta}}) = n(\log 2\pi + 1 + \hat{\delta}_0),$$

it can further be shown that

$$-2l^E(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\delta}}) = n \log 2\pi + n\hat{\delta}_0 + \text{Tr}(\hat{\boldsymbol{\Sigma}}^{-1}\boldsymbol{\Sigma}^*) + (\boldsymbol{X}\boldsymbol{\beta}^* - \boldsymbol{X}\hat{\boldsymbol{\beta}})'\hat{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{X}\boldsymbol{\beta}^* - \boldsymbol{X}\hat{\boldsymbol{\beta}}),$$

where $\boldsymbol{\Sigma} = \text{Diag}(\exp(\boldsymbol{U}\boldsymbol{\delta}))$, with the exponentiation taken element-wise. Thus,

$$B = E_G[\text{Tr}(\hat{\boldsymbol{\Sigma}}^{-1}\boldsymbol{\Sigma}^*) + (\boldsymbol{X}\boldsymbol{\beta}^* - \boldsymbol{X}\hat{\boldsymbol{\beta}})'\hat{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{X}\boldsymbol{\beta}^* - \boldsymbol{X}\hat{\boldsymbol{\beta}}) - n]. \tag{3.5}$$

Unfortunately, except for a few special cases, this does not appear to have a closed-form simplification that does not depend on the true, unknown values of model parameters. We instead perform simulations to estimate the needed expectations, and corroborate these in the cases where a closed-form solution is available.

Before beginning the simulation, we first associate each possible model $(\mathcal{L}, \mathcal{D})$ with a prototype; that is, one model that represents all models whose model matrices are isomorphic under permutation of the rows. For example, it is obvious that the model structure for any single-location-effect model is the same regardless of which location effect is used, so $(\{A\}, \{\emptyset\})$ serves as a prototype for all models of this structure. In models with one location effect and one dispersion effect, the dispersion effect is either on the same factor

Table 3.2: Estimated CHIC penalty values for various common location-dispersion model prototypes in a 2^4 design. A "*" indicates values that may be infinite: simulations yield increasingly large means and standard errors with increasing numbers of runs. The subscript is the standard error from the simulation. A subscript of "$-$" indicates that the value was derived mathematically.

Location	Dispersion model, $\mathcal{D}$				
Model, $\mathcal{L}$	$\emptyset$	A	A,B	A,B,AB	A,B,C
$\emptyset$	4.9_{-}	$10.1_{0.1}$	$17.9_{0.1}$	$42.9_{0.7}$	$31.8_{0.3}$
A	8.0_{-}	12.8_{-}	$25.7_{0.2}$	$54.6_{1.4}$	$58.8_{1.2}$
B	8.0_{-}	$16.9_{0.2}$	$25.7_{0.2}$	$54.6_{1.4}$	$58.8_{1.2}$
A,B	11.6_{-}	$20.0_{0.1}$	$35.3_{0.3}$	$61.0_{2.6}$	$133.3_{5.6}$
A,B,AB	16.0_{-}	$24.1_{0.2}$	$36.3_{0.2}$	64.0_{-}	$190.7_{6.3}$
C	8.0_{-}	$16.9_{0.2}$	$37.7_{0.6}$	*	$58.8_{1.2}$
A,B,C	16.0_{-}	$32.3_{0.2}$	$80.4_{1.6}$	*	$332.9_{17.1}$

as the location effect or on a different one; hence, $(\{A\},\{A\})$ and $(\{A\},\{B\})$ are the two prototypes for this case.

Next, we need to select a true model under which simulations are to be performed. In keeping with the assumption that the true model does not contain nonzero parameters outside of the candidate model, then the only possible true model that is applicable to any candidate model is $(\mathcal{L},\mathcal{D}) = (\{\emptyset\},\{\emptyset\})$. Thus, the simulation for each prototype consists of generating a large number of data sets, each consisting of 2^k independent standard normal observations; fitting the prototype model to each data set; and computing the sum of terms inside the expectation in (3.5). The mean of these results is an estimate of the bias adjustment that is needed to complete the information criterion for this heteroscedastic model. We refer to the new criterion as the "corrected heteroscedastic information criterion" (CHIC):

$$CHIC = -2\hat{l}(\hat{\boldsymbol{\beta}},\hat{\boldsymbol{\delta}}) + B.$$

The number of models under consideration for which CHIC penalties can be computed is too large to provide a comprehensive listing of values. Some estimated values of the bias adjustment are shown in Table 3.2 for certain common models when $k = 4$. Note that exact penalties are available for the cases where $(\mathcal{L},\mathcal{D}) = (\{*\},\{\emptyset\})$, $(\{A\},\{A\})$, or $(\{A,B,AB\},\{A,B,AB\})$, where $*$ here means any location model.

Notice some features about these penalties:

1. The number of simulations used in this table varied depending on the estimated standard error. We started with at least 20,000 in each case and added runs adaptively up to a total of 100,000 as needed when the standard errors were large.

2. Dispersion effects are more expensive than location effects in the sense that adding a dispersion effect to a model generally incurs a larger penalty for the additional parameter than adding a location effect to the same model.

3. The penalties are *much* larger than the AIC penalty ($2(p+q+2)$), the BIC penalty ($\log(16)(p+q+2)$), and generally also the AICc penalty, especially for larger models.

4. The penalties become extremely large before a model becomes saturated in the sense of Loughin and Rodríguez [44]; for example, the estimated penalty for $(\mathcal{L}, \mathcal{D}) = (\{C\}, \{A, B, AB\})$ after 50,000 runs was 582, with a standard error of 148 (recall that model $(\{C, D, CD\}, \{A, B, AB\})$ is "saturated"). This suggests that the underlying expectations in (3.5) may not be finite. We take this as an implication that models with such large penalties are unstable, and therefore we exclude models with estimated penalties greater than 200 in our implementations described below.

3.4.3 Model averaging

The last issue that needs to be resolved is how to use CHIC to select effects and/or models. As with other ICs, we seek models with small values of CHIC. However, it is well understood that exhaustive searches often lead to large numbers of models whose IC values are very close to the minimum value (see, e.g., [13]). It is therefore often a matter of random chance which particular model achieves the minimum IC value. Hoeting et al. [35] give a detailed review of a notion called Bayesian Model Averaging (BMA), wherein BIC values are transformed into approximate posterior probabilities that their respective models are correct, given the data. From the model posterior probabilities, a probability that each parameter is needed in the model can be computed as the sum of probabilities for all models in which the parameter appears. This process is analogous to the fully Bayesian approach to finding model and effect probabilities discussed in Box and Meyer [7].

Burnham and Anderson [13] discuss extending BMA to any information criterion. The exact same calculations are used, but the results are interpreted as "evidence weights" rather than as probabilities in a Bayesian sense. We apply this general model-averaging construct to the joint location-dispersion models using CHIC as follows:

1. Identify the M models to be estimated and calculate CHIC on each model, say CHIC_m, $m = 1, \ldots, M$.

2. Find the minimum CHIC value among all models. Call this value CHIC_0.

3. For each model, compute $\Delta_m = \text{CHIC}_m - \text{CHIC}_0$, and the *model evidence weight*

$$\tau_m = \frac{\exp(-\Delta_m/2)}{\sum_{a=1}^{M} \exp(-\Delta_a/2)}.$$

4. Compute the *effect evidence weight* for effect j as

$$\rho_j = \sum_{m=1}^{M} \tau_m I(\text{Effect } j \text{ is in model } m),$$

   where we let $j = 1, \ldots, 2(n-1)$ index the set of all location effects and dispersion effects, and $I(\cdot)$ is the indicator function.

We refer to this procedure as Exhaustive-Search Model Averaging with Corrected Heteroscedastic Information Criterion (ESMA-CHIC).

Notice that the transformation CHIC$\rightarrow \tau$ is monotone decreasing, so that the smaller a model's CHIC is, the larger its model evidence weight. Furthermore, $\sum_{m=1}^{M} \tau_m = 1$, and if there is a single model whose CHIC is distinctly smaller than the rest, then its evidence weight approaches 1. Thus, evidence weights are interpreted roughly the same as probabilities. Similar interpretations can be applied to each ρ_j. Thus, a $\rho_j \approx 0$ can be interpreted as evidence that effect j is unlikely to influence the process.

3.4.4 Implementation details

For $k \leq 4$ we have implemented ESMA-CHIC subject to the following constraints:

1. Only models with $p \leq 5$ and $q \leq 5$ are considered.

2. Prior to use, a full table of CHIC penalty values was computed using 1000 simulations each.

3. Models with CHIC penalties greater than 200 are not included in the analysis.

The penalties for all models that we consider are computed in advance and stored in a file that is accessed as needed.

For $k > 4$ exhaustive search is not currently feasible, even with the restrictions $p \leq 5, q \leq 5$ and discarding models with penalties above 200. Indeed, the table size for the penalty values would be larger than what we can store in memory. Instead, we have written a genetic algorithm (GA, [49]) to perform the model search.

Our version of the GA proceeds as follows:

1. Seed the GA with M models

2. Evaluate the quality of the models in the GA

3. Sort the models by the quality

4. Keep a fraction of the top models, discard the rest. Replace the trashed models with newly generated models

5. For a total of I steps, we improve the GA by repeating steps $2-4$

Details on the individual steps are given below. Note that the parameter settings below have been tuned for performance in the 2^4 setting. It is possible that other parameter settings would work better in larger experiments.

1. Seed the GA with M models

   In principle, these can be randomly chosen models. In our implementation, we seed the GA with $M = 40$ null models, where none of the location or dispersion effects are present. We rely on mutation, as described below, to slowly populate the models with effects.

2. Evaluate the quality of the models in the GA

   We use the CHIC value, $-2 * loglikelihood + penalty$, as the fitness criterion for each model. Because the program is designed to work with experiments of all sizes, we don't use a table of precomputed penalty values as we do with the exhaustive search. Instead, we estimate the CHIC penalties "on-the-fly". As before, we simulate the expected bias under the null model and take the average bias over the simulations.

   One approach would be to use some fixed number of simulations S to estimate the CHIC value for every model that we try. This is inefficient, however, as there are several models for which we attain a very reasonable standard error of estimation after very few simulations. Instead, we set fixed lower and upper limits on the potential number of simulations, and use a stopping rule based on the ratio of the standard error of the mean penalty estimate to the mean penalty estimate at simulation S.

   Thus, there are thus four tunable parameters in our implementation of the CHIC simulation

   (a) The minimum number of simulations, S_{low}
   (b) The maximum number of simulations, S_{high}
   (c) The lower threshold for the standard error, as a fraction of the mean, T_{low}
   (d) The upper threshold for the standard error, as a fraction of the mean, T_{high}

   The algorithm proceeds for at least S_{low} simulations and necessarily terminates after S_{high} simulations. For any value of simulations S between these two bounds, the algorithm checks if the standard error of the CHIC penalty lies outside either threshold. If the standard error is lower than the lower threshold, the simulation algorithm terminates and returns the estimated mean penalty. If the standard error is higher than the upper threshold, the simulation algorithm terminates and returns a value indicating that this model should be discarded. Otherwise, the simulation algorithm will terminate when it reaches S_{high} simulations. Note that if S_{low} is set too low, it may discard reasonable models.

In any case, the algorithm caches the result and stores it in a hash table indexed by the model name so that we never simulate the penalty for the same model twice. The hash table can be saved locally to speed up computation on different data sets (note that one needs different tables for different values of k).

We use $S_{low} = 50, S_{high} = 1000, T_{low} = 0.03, T_{high} = 0.6$. We found these to be reasonable settings for the 2^4 design. Different tuning parameters may be better for larger setups.

3. Sort the models by the quality

   We sort the models in increasing order of $-2 * likelihood + penalty$. In our implementation of the GA, we allow the possibility that the same model appears multiple times.

4. Keep a fraction of the top models, discard the rest. Replace the discarded models with newly generated models

   We say that the top 50% of the models are "fit" models, and the remaining 50% are "unfit". We discard the unfit models and replace these with newly generated models. To generate a new model, we begin by taking a location model from a randomly selected fit model, and adding a dispersion model from a randomly selected fit model. We splice them together to create a prototype model. The model then undergoes mutation: each location effect can be randomly "flipped" (if it was included, it becomes excluded, or vice-versa) with probability M_L. Each dispersion effect undergoes the same treatment with probability M_D. We generate $M/2$ new candidate models in this way.

   We used $M_L = M_D = 0.04$. This means that more often than not, mutation in at least one bit occurs in experiments with at least 4 factors.

5. For a total of I steps, we improve the GA by repeating steps $2 - 4$

   We used $I = 250$ improvement steps. We record the performance of every model that passes through the GA at some point—this allows the user to do model averaging on, say, the top 100 models ever constructed. The GA can also easily be restarted several times, and the best models over all restarts are easily found in a single table.

3.5 Applications

We apply the ESMA-CHIC procedure to the three examples presented in Table 3.1. The top five models according to CHIC are shown in Tables 3.3–3.5, along with the CHIC values,

Table 3.3: Top five models using ESMA-CHIC and genetic algorithm and for each model listed in Table 3.1 for the Welding data. Evidence weights and model ranks are based on ESMA-CHIC.

Rank	Location Effects	Dispersion Effects	ESMA-CHIC	Model Weight	GA CHIC
1	B C	C	12.3	0.49	14.3
2	B C J "DH"[a]	C	16.0	0.08	
3	B C D	C	16.0	0.08	16.9
4	B C J "CG"[a]	C	17.5	0.04	
5	B C J	C	17.7	0.03	18.0
7	B C	$\emptyset$	19.2	0.02	19.0
34	B C H	$\emptyset$	23.1	0.00	20.1
245	B C	C J	30.0	0.00	
409	B C AC AH A H	$\emptyset$	33.1	0.00	
1094	B C J	C J	52.9	0.00	
>10,000	B C	C H J	299.0	0.00	

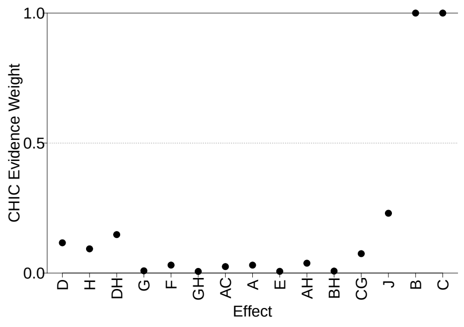
[a]According to Box and Meyer [7], the experimenters did not believe that any of the aliased effects corresponding to these columns of the design matrix—DH=BJ and CG=BF=EH—would be active

Table 3.4: Top five models using ESMA-CHIC and genetic algorithm and for each model listed in Table 3.1 for the Injection Molding data. Evidence weights and model ranks are based on ESMA-CHIC.

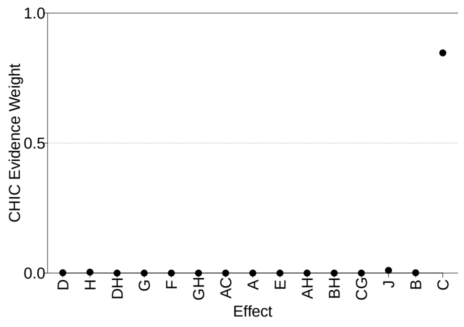
CHIC Rank	Location Effects	Dispersion Effects	ESMA-CHIC	Model Weight	GA CHIC
1	A B AB G CG	$\emptyset$	72.3	1.00	72.5
2	A B AB CG	$\emptyset$	85.6	0.00	86.0
3	A B AB G	$\emptyset$	87.9	0.00	88.3
4	A B AB	$\emptyset$	90.3	0.00	89.7
5	A B AB BC CG	$\emptyset$	90.5	0.00	89.8
277	A B AB	C	116.1	0.00	

evidence weights, and corresponding ranks for any models suggested by past literature that are not among the top five. We also include the top 5 models' CHIC from performing the same analysis using a GA rather than an exhaustive search. Finally, plots of the evidence weights for all effects are shown in Figure 3.1.

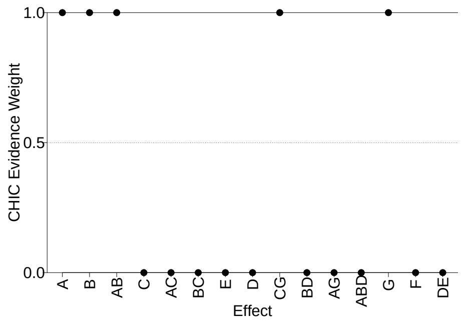
For the Welding example, Table 3.3 shows that the top model is $(\{B, C\}, \{C\})$ as suggested by Box and Meyer [7], McGrath and Lin [47] and Loughin and Malone [42]. This model has 0.49 evidence weight, so the data are not conclusive in their support for this model. However, no other model is a particular challenge for the best; this model is favored by at least 6:1 over any other model. The other top models by ESMA-CHIC all contain $(\{B, C\}, \{C\})$, along with a variety of additional location effects. Models with different dispersion effects do not fare well at all. Based on these results, it is no surprise that Figure



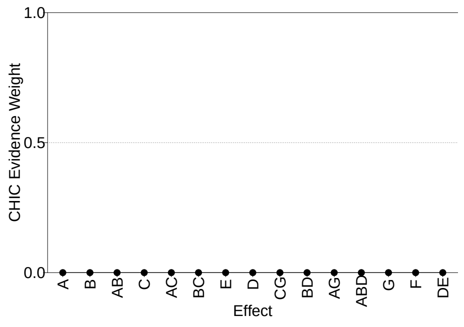
(a) Welding Location Effects



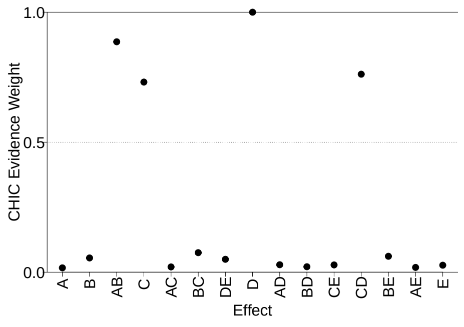
(b) Welding Dispersion Effects



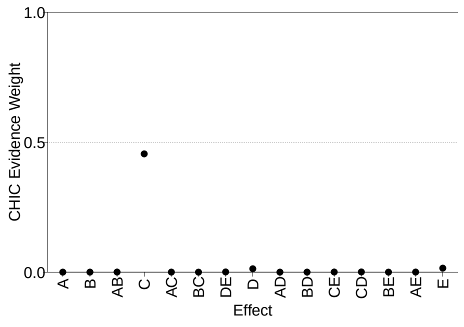
(c) Injection Molding Location Effects



(d) Injection Molding Dispersion Effects



(e) Dyestuff Location Effects



(f) Dyestuff Dispersion Effects

Figure 3.1: Plots of effect evidence weights for three example data sets: Welding (top), Injection Molding (middle), and Dyestuff (bottom). Larger evidence weight indicates greater chance of an active effect.

Table 3.5: Top five models using ESMA-CHIC and genetic algorithm and for each model listed in Table 3.1 for the Dyestuff data. Evidence weights and model ranks are based on ESMA-CHIC.

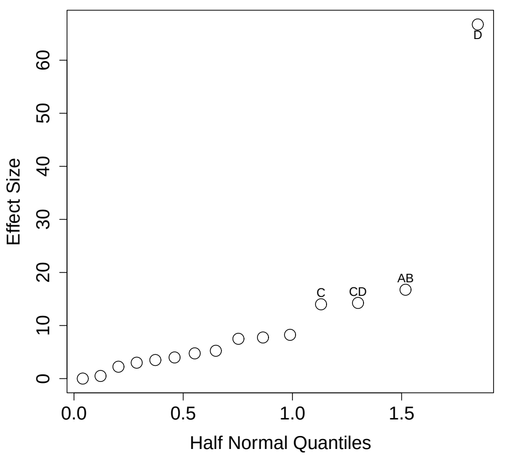
CHIC Rank	Location Effects	Dispersion Effects	ESMA-CHIC	Model Weight	GA CHIC
1	D AB CD C	C	117.5	0.33	119.0
2	D AB CD C	∅	119.5	0.12	120.2
3	D AB CD	C	121.3	0.05	121.2
4	D AB CD C BC	∅	121.7	0.04	121.0
5	D AB CD C BE	∅	122.3	0.03	
8	D AB C	C	122.8	0.02	121.2
15	D	E	124.9	0.01	

3.1a shows location effects B and C with evidence weights that round to 1 and no others that are particularly strong. Most have very little support from the data. Figure 3.1b shows that dispersion effect C has evidence weight 0.85. All other dispersion effects have negligible weight.

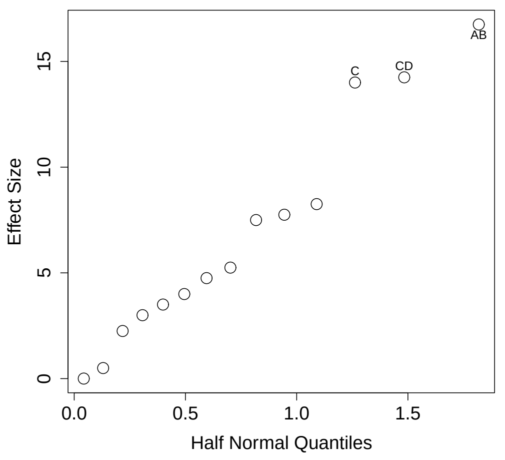
In the Injection Molding example, Table 3.4 shows that the data are conclusive in their preference for the model with five location effects and no dispersion effect, agreeing with McGrath [46] and Loughin and Malone [42]. There is essentially no support for the alternative interpretation of a dispersion effect on C rather than location effects on G and CG. This is noteworthy, as it is the first time that these two competing models have been compared objectively using a single criterion. It serves to highlight one of the fundamental advantages of the ESMA-CHIC approach. Figures 3.1c and 3.1d re-express these results by showing evidence weights of 1 on location effects A, B, AB, G, and CG, and zero for all other effects.

The Dyestuff example represents another interesting problem. Past analyses using different methods by Bergman and Hynén [5] and McGrath and Lin [47] arrive at the same conclusion of a location effect on D and a dispersion effect on E. However, ESMA-CHIC shows that there is considerable uncertainty regarding what model best represents the data, with only 0.33 evidence weight on the top model, $(\{C, D, AB, CD\}, \{C\})$. Despite the model uncertainty, there is considerable agreement in what belongs in the best models. All five of the top models contain location effects D, AB, and CD, and four contain C. On the other hand, it is rather unclear whether the dispersion effect of C is needed, as it appears in only two of the top five models. The previously identified model, $(\{D\}, \{E\})$ is ranked 15th with an evidence weight of only 0.01. According to the data, it is extremely unlikely that this model describes the true process adequately.

Effect evidence weights in Figures 3.1e and 3.1f reflect these observations. Among location effects, while D has evidence weight essentially 1, effects AB, CD, and C have evidence weights ranging from 0.89 to 0.73, suggesting that evidence is generally in their favor but



(a) All effects



(b) All but the largest effect

Figure 3.2: Half-normal plot of location effects for the Dyestuff example. The one extreme point in the plot on the left masks visual detection of potential location effects of smaller magnitude, which are more evident in the plot on the right.

not overwhelmingly so. No other location effects have any appreciable support from the data. Among dispersion effects, only C has non-negligible evidence weight, although it is only 0.46. Note that dispersion effect E has negligible weight, and hence is not considered to be a part of any serious explanation for these data.

Why does such a discrepancy exist between the results of ESMA-CHIC and past analyses? Part of the answer can be seen from a half-normal plot of the location effects for this example, as shown in Figure 3.2. The plot on the left shows that location effect D stands out as an "obviously" active effect, while those for AB, CD, and C are somewhat less obvious. Subjective assessment of this plot could either include or exclude these three effects from a model. However, the choice becomes much clearer when we exclude the obvious outlier and rescale the plot, as shown in Figure 3.2b. Now the three uncertain location effects are seen to stand out a bit more clearly from the line formed by the rest of the points, corroborating their evidence weights.

The belief that E should be a dispersion effect appears to be another example of the location-dispersion confounding issue touched on in Section 3.1 and discussed in more detail in Pan [53], McGrath and Lin [47] [48], and Loughin and Malone [42]. Notice that the two largest "uncertain" location effects, AB and CD, form an interaction triple with E. Thus, it is no surprise that failure to identify AB and CD as location effects should result in possible spurious identification of a dispersion effect at E. Once again we see the immense potential offered by this new ability to compare different combinations of location effects and dispersion effects using a single criterion.

Finally, note that our GA implementation does reasonably well at mimicking the exhaustive search. Across the three examples, it identifies 12 of the 15 top models, including the top model in each case. The additional models that it places among its top five have estimated CHIC values are mostly competitive within the context of the exhaustive search.

3.6 Analyses of Simulated Data

To complement the results obtained on the three examples in the previous section, we assess the performance of ESMA-CHIC using simulated data where the true models are known.

3.6.1 Methods

We generate data sets from model (3.2) for a 2^4 design using selected combinations $(\mathcal{L}, \mathcal{D})$. We perform analyses using either ESMA-CHIC or a two-step procedure consisting of the Lenth location-effect test followed by the modified Harvey dispersion-effect test ([40]; [12]). The Lenth test is chosen due to its simplicity and its reputation for reasonably good performance, even in the presence of dispersion effects ([69], [32], [71]). The modified Harvey method is chosen because it is very similar in structure to the popular Bergman-Hynén test, but it is specifically designed to be compatible with our loglinear model for dispersion effects [12]. We simulate 100 data sets for ESMA-CHIC and 5000 for Lenth/modified Harvey to achieve a balance between simulation error and run time. We summarize the simulation results by computing the average power and type I error rate for detecting location effects and the same measures for detecting dispersion effects. For ESMA-CHIC, effects in the model with the smallest CHIC value are declared active. Note that ESMA-CHIC is not calibrated to achieve any specific Type I error rate, so there is no specific expectation for its performance in this measure. Both parts of the Lenth/modified Harvey test are conducted using a nominal 0.05 level, with the critical values for the Lenth test taken from Loughin [41].

We use all combinations of six true location models and five true dispersion models, for a total of 30 different models. We choose $\mathcal{L} = \{\emptyset\}, \{A\}, \{B\}, \{A, B\}, \{A, B, C\}$, and $\{A, B, AB, C, AC, D\}$ because these represent five fairly typical location-model structures and one somewhat large model. Notice that the last location model has six active effects, which is more than the five that our implementation of ESMA-CHIC is designed to detect. Thus, we have a check on the price we pay for assuming more stringent effect sparsity than is actually present. We pair these location models with $\mathcal{D} = \{\emptyset\}$, $\{A\}$, $\{A, B\}$, $\{A, B, AB\}$, $\{A, B, C\}$.

Each active location effect is set to a level that would be detected with approximately 50% power by a Wald test using the known variance of the contrast under the true dispersion model (McGrath 2003). This level is re-calibrated for each different location model, so that power for all Lenth tests remains at approximately 50% regardless of the model. Each active

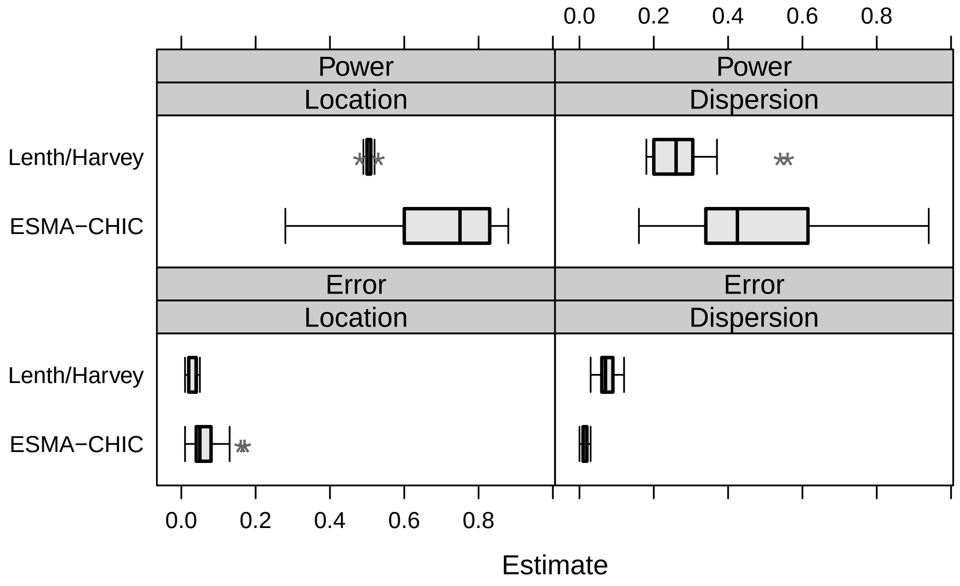


Figure 3.3: Summary of estimated power and type I Error rates from Table 3.6 for detecting location effects and dispersion effects using the ESMA-CHIC procedure (100 simulations) and the Lenth/modified Harvey test (5000 simulations). Note that the Lenth test for location effects has been calibrated to have approximately 0.5 power.

dispersion effect is set to a standard-deviation ratio of 5:1; i.e., the errors at the + level of the effect are multiplied by $\sqrt{5}$, while those at the lower level are divided by $\sqrt{5}$.

3.6.2 Results

Results of the simulations are shown in Table 3.6 and depicted in Figure 3.3. Comparing location-effect detection for the two methods, note that the Lenth test has been calibrated to maintain both its type 1 error rate and its power. The simulations show that this is largely achieved, although the Lenth-test error rate does decrease as model size increases. By comparison, ESMA-CHIC has much more model-dependent type 1 error rates and power for detecting location effects. On average, type 1 error rates are slightly larger than those for the Lenth test, although when there are no dispersion effects they are considerably larger. When the true location model is larger than we assume, ESMA-CHIC has type 1 error rates that drop to around 1–2%.

Power for detecting location effects mirrors the type 1 error rates. ESMA-CHIC detection rates average close to 75%, although they are much lower when the model is too large. In many cases, the power is quite high—over 80%—even when the observed type 1 error rate is at or below 0.05. In contrast, the Lenth test becomes rather conservative as the size of the location model grows and when there are dispersion effects. The re-calibration of the sizes of the active effects allows its power to remain at 50% by increasing the sizes of the

Table 3.6: Estimated power ("Pow") and Type I Error rate ("Err") for detecting location effects ("L") and dispersion effects ("D") using the ESMA-CHIC procedure (100 simulations) and the Lenth/modified Harvey test (5000 simulations).

True Model		Lenth/modified Harvey				ESMA-CHIC			
Loc.	Disp.	PowL	ErrL	PowD	ErrD	PowL	ErrL	PowD	ErrD
∅	∅		0.05		0.06		0.17		0.02
A	∅	0.50	0.04		0.06	0.74	0.17		0.01
B	∅	0.51	0.04		0.06	0.73	0.16		0.01
A,B	∅	0.49	0.04		0.06	0.76	0.13		0.02
A,B,C	∅	0.48	0.03		0.07	0.79	0.11		0.00
A,B,AB,C,AC,D	∅	0.50	0.01		0.10	0.54	0.01		0.02
∅	A		0.05	0.54	0.06		0.08	0.73	0.02
A	A	0.51	0.04	0.56	0.10	0.71	0.07	0.78	0.01
B	A	0.50	0.04	0.28	0.06	0.86	0.10	0.50	0.01
A,B	A	0.50	0.04	0.27	0.07	0.83	0.07	0.66	0.01
A,B,C	A	0.50	0.03	0.20	0.07	0.85	0.09	0.25	0.01
A,B,AB,C,AC,D	A	0.51	0.01	0.32	0.10	0.30	0.02	0.94	0.00
∅	A,B		0.04	0.37	0.06		0.04	0.70	0.02
A	A,B	0.51	0.04	0.29	0.07	0.83	0.05	0.55	0.01
B	A,B	0.50	0.04	0.30	0.07	0.86	0.05	0.60	0.01
A,B	A,B	0.51	0.03	0.19	0.09	0.77	0.06	0.45	0.01
A,B,C	A,B	0.51	0.03	0.18	0.08	0.71	0.06	0.18	0.03
A,B,AB,C,AC,D	A,B	0.51	0.01	0.24	0.11	0.28	0.02	0.49	0.00
∅	A,B,C		0.02	0.26	0.05		0.01	0.63	0.02
A	A,B,C	0.50	0.02	0.20	0.07	0.88	0.04	0.40	0.01
B	A,B,C	0.50	0.02	0.20	0.07	0.87	0.05	0.39	0.01
A,B	A,B,C	0.50	0.02	0.18	0.09	0.75	0.05	0.37	0.01
A,B,C	A,B,C	0.50	0.02	0.20	0.09	0.67	0.06	0.16	0.03
A,B,AB,C,AC,D	A,B,C	0.50	0.01	0.18	0.12	0.28	0.02	0.30	0.01
∅	A,B,AB		0.04	0.34	0.03		0.05	0.48	0.02
A	A,B,AB	0.52	0.05	0.25	0.04	0.76	0.04	0.38	0.01
B	A,B,AB	0.50	0.05	0.25	0.04	0.79	0.05	0.36	0.01
A,B	A,B,AB	0.49	0.04	0.31	0.04	0.45	0.04	0.35	0.01
A,B,C	A,B,AB	0.53	0.04	0.26	0.08	0.60	0.05	0.33	0.00
A,B,AB,C,AC,D	A,B,AB	0.49	0.01	0.26	0.11	0.29	0.02	0.33	0.00

location effects in the larger dispersion models. ESMA-CHIC seems to have less difficulty detecting these large location effects than the Lenth test does.

Following the Lenth test with the modified Harvey test results in slightly larger-than-nominal type 1 error rates in most cases, with the greatest inflation occurring with the very large location model. In that model, there are many location effects that may be missed by the Lenth test, which creates more opportunities for spurious dispersion effects to be detected. On the other hand, ESMA-CHIC has very low type 1 error rates for dispersion effects. It tends to be very conservative in declaring effects active. Despite the lower type 1 error rates, its power is on average much better than the modified Harvey test, in some cases by more than double.

3.7 Discussion and Conclusions

In this paper we have developed the first fully automated analysis procedure for 2^k factorial designs that can identify both location and dispersion effects in a single step. This is a critical advance, as it finally provides an objective approach to choosing between models where location-dispersion confounding may take place. Evidence of its effectiveness in this regard comes from the simulations, where it competently detects moderately sized active effects and avoids detecting spurious effects of both types with very reasonable frequency.

As one reviewer quite correctly points out, we cannot be sure that ESMA-CHIC would enjoy the same relative advantage over Lenth/modified Harvey under all possible simulation settings. However, our settings were chosen based on prior examples and without knowledge of the potential results. In particular, the size of the dispersion effect is not without precedent. Gelfand [30] studies 24 published regression data sets exhibiting significant heteroscedasticity. She measures the ratios of standard deviations between the top 10% and bottom 10% of residuals and finds that their quartiles are 2.0, 3.2, and 5.5. Thus, the ratio of 5 that we used is not particularly extreme relative to heteroscedastic regressions. Furthermore, in the top-ranked model for the Dyestuff data, the MLE of dispersion effect C is $\hat{\delta}_C = -1.61$, leading to a standard deviation ratio of $\exp(|\hat{\delta}_C|) = 4.95$. Similarly, for the top model for the Welding data, we find $\hat{\delta}_C = 1.55$, leading to a standard deviation ratio of $\exp(|\hat{\delta}_C|) = 4.71$. Thus, a standard deviation ratio of 5 is large, but realistic.

The Lenth/modified Harvey approach that we used for comparison was chosen deliberately to be a competitive alternative. In particular, the Harvey test assumes a normal distribution with loglinear dispersion effects, which is the same model that was used for the simulations. Thus, we gave our new method no “home-field advantage.” The main flaw that we anticipated with the Harvey test in this context was its performance in the wake of the inability to correctly glean the location model with complete certainty through the Lenth test. Extended simulation results show that this issue is only one aspect of the whole difficulty with using a sequential testing scheme like Lenth/modified Harvey. In particular, several factors influence whether dispersion effects are spuriously identified by this procedure. For example, in the simulations from $(\{A, B\}, \{A, B\})$, the combination of location and dispersion confounding causes the AB dispersion effect to be falsely detected roughly 40% of the time—more often than either of the two active dispersion effects!

The cause of this is a complicated combination of facets that a sequential process cannot possibly cope with. First, the dispersion effects on A and B induce a complicated correlation structure among the location effects [31]. In our case, the size of the dispersion effects on A and B imply that the estimates of the active location effects on A and B have correlation 0.85. Each is arranged by design to have a marginal detection rate of 0.5 by the Lenth test. Thus, in the vast majority of simulation runs, they are either both detected or both missed, with roughly equal probability on each joint outcome. When both are detected, the modified Harvey test correctly determines that there is no AB dispersion effect. However, when both are missed, their combined effect creates a spurious dispersion effect on AB, and this effect is large enough that the modified Harvey test detects it about 2/3 of the time. This does not happen with ESMA-CHIC, because the model with $\mathcal{D} = \{A, B\}$ can be compared directly to the model with $\mathcal{D} = \{AB\}$, both with and without location effects. This is a clear example of the advantage of joint modeling using a single criterion.

Another huge advantage of the ESMA-CHIC procedure is its ability to provide assessments of uncertainty regarding the importance of the model parameters. Although not true Bayesian posterior probabilities, the evidence weights derived from the CHIC nonetheless convey useful information about the relative importance of effects and models. In addition to providing a listing of top models, the procedure affords an analyst who has a model in mind prior to analysis the opportunity to examine the evidence in its favor relative to other models. This should have great appeal to engineers and other application specialists, who typically know something about the processes that they are investigating and might like an objective assessment of their prior beliefs. Furthermore, the model-averaging aspect also allows one to find more realistic measures of uncertainty on the actual parameter estimates through unconditional variance estimates computed from the multitude of models that have been fit [13]. These variances account for the model-selection uncertainty as well as typical within-model sampling variability.

Unfortunately, an exhaustive search of the model space is a computationally intensive process. As previously stated, we have shortened our computation time considerably (by about 60%) by imposing maxima of five location and five dispersion effects on any model fit. Even with these restrictions, analyzing one data set took an average of about 25 minutes in R on a Quad-core 2.40GHz processor with 32GB of RAM, using 3 of the 4 cores. Some of the code is written in C to improve runtime. Using more cores would improve run time considerably.

The genetic algorithm implementation takes considerably longer to evaluate each model it considers, because we must simulate penalty values for each new model created by the mating phase of the algorithm. Different settings within the GA—mainly the number of generations, number of models per generation, and mutation rate—can affect the number of unique models considered, and hence run time. We tuned our implementation to achieve good results in terms of both speed and power. Using these settings, analysis of a single data set took around 30 minutes on the computer described above for each of the three examples presented in Section 3.5. We re-examined several of the lines from Table 3.6 using the GA and got consistently very comparable results. We then tried the algorithm on simulated data sets from various models from a 2^5 design using very similar settings to those from Section 3.6. We analyzed three example data sets from a selection of models from Table 3.6 using the same default GA tuning parameters as in the 2^4 case. These analyses took about 6 hours per data set (again, this time could be reduced considerably by using more cores). The GA provided very sensible results, generally identifying the majority of the active location and dispersion effects and identifying spurious effects with much lower frequency.

An argument against fully automated model-selection procedures is that they often pay no attention to whether combinations of variables make practical sense together. For example, our algorithm for ESMA-CHIC makes no use of effect heredity or hierarchy. Under the circumstances, however, we do not consider this to be a particular weakness. The model-averaging aspect of ESMA-CHIC allows an analyst to peruse the top models for those that do make sense. If no practical models are highly supported by the data, then this may be a signal either that something was wrong with the experiment or that something unexpected is driving the responses. In either case, we are not aware of any other objective analysis method that would be able to provide a sensible model for the analyst when faced with such data, and indeed would provide far less information regarding the extent to which "sensible" models are *not* supported by the data.

A final criticism is that our procedure is based on a normal model with a multiplicative variance assumption, which means that it may be sensitive to mis-specification of that model. We have not assessed this sensitivity, but instead can point out that practically every other location- or dispersion-testing procedure is based on a variant of the same model (a notable exception is the permutation test of Loughin and Noble [43]). Furthermore, its basis in a parametric model means that there is nothing that limits the procedure to problems in the 2^k series. It could conceivably be developed to apply to other design series or even to much more general regression problems. Of course, this would require deriving or estimating new CHIC penalty values.

Chapter 4

HeaRTs : Heteroskedastic Regression Trees

4.1 Introduction

Regression trees [11] are piecewise constant linear models with split points determined adaptively using a "recursive partitioning" algorithm. The algorithm recursively splits data into smaller and smaller groups, called "nodes", creating a tree structure. Typically, splits are carried out until some stopping criterion is reached on the nodes at the end of the tree, called "terminal nodes." The mean response in each terminal node is usually estimated from the sample mean of the data within the node, although more complicated methods have been developed (see e.g., [18] [27]. The size of the tree may be reduced ("pruned") if a smaller tree improves prediction accuracy.

Regression trees have desirable features. They construct interaction effects automatically, naturally incorporate variable selection, fit the mean function in a flexible manner, and have a relatively fast runtime [11]. Furthermore, the model constructed is highly interpretable. Finally, they are used as base learners in several popular ensemble prediction methods, such as random forests [9], boosted trees [26] and Bayesian additive regression trees[19].

We explained in Chapter 2 that regression trees are a type of piecewise linear regression, estimated by ordinary least squares (OLS). Furthermore OLS estimation is known to be suboptimal in the presence of heteroskedasticity, in the sense that the estimates are inefficient (see e.g., [45]). This is particularly concerning because heteroskedasticity is common in real datasets; Gelfand [30] identifies 25 data sets whose variance changes significantly with the mean in a (non-random) sample of 42 real data sets.

As regression trees have many uses in modern statistical analysis, it is worth studying how they behave in the presence of heteroskedasticity. Ruth and Loughin [57] analyse the performance of the popular `rpart` implementation (in the software language R) of Breiman et al.'s regression trees. They showed under heteroskedasticity, the default settings in the popular R tree-fitting package rpart can cause the algorithm to make splits splits in the high variance area, ignoring the low variance area. Upon changing some default tuning parameters, we can coerce the tree to make splits in the low variance area, but pruning the tree (which is almost universally a good idea, see e.g. [11]), causes these low-variance splits to be pruned away again. We argue in Section 4.8.1 that, all else being equal, modeling the mean function carefully in the low variance area is more important than modeling it carefully in the high variance area. To accomplish this, we introduce HEteroskedAstic Regression TreeS (HeaRTs), a version of a regression tree that can split on the mean, the variance, or simultaneously on the mean and variance. We show that HeaRTs does a better job than `rpart` at modeling the mean in the low-variance area when the data are heteroskedastic.

Another issue with regression trees, not discussed in Ruth and Loughin, is that regression trees do not necessarily use the most efficient estimate of the mean under heteroskedasticity. In particular, when different terminal nodes contain observations with materially different variances, the unweighted average is a worse estimate of the mean than the weighted average, where the weights are the inverse variances. This is the rationale behind weighted least squares (WLS, see [16]), which improves on OLS estimation for heteroscedastic data. Our novel HeaRTs method simultaneously models the mean and variance. As a consequence, we are able to identify when predictors may affect the variance but not the mean, and can adapt the mean estimation accordingly. We see this is particularly useful when the predictors that impact the variance model are different from the predictors that impact the mean model.

The chapter proceeds as follows. We begin by reviewing the standard regression-tree algorithm of Breiman et al. [11] and related efforts to model variances in trees. We then focus on the key mathematical differences between our method and Breiman et al.'s method, in particular on how we make different splits. We introduce 3 different split types, and give a demonstration of how our new splits respond to heteroskedastic data. Following this we discuss how we select which type of split to choose at each node, and introduce the Corrected Heteroskedastic Information Criterion (CHIC) in this context. We then demonstrate how to use CHIC to prune the tree. We then provide a metric that allows us to measure the quality of the model fit. Finally, we show how the HeaRTs procedure performs on some real and simulated data sets, and discuss the effect of both our new pruning method and our new splitting method. We show that our method is both theoretically faster than Breiman et al.'s original pruned regression trees, and in a sense, more accurate.

4.2 Relevant Literature

Breiman et al.'s approach [11] to building a regression tree (which we refer to as RPAB: Recursive Partitioning Algorithm of Breiman et al.) works as follows. We begin by defining the responses y and predictor matrix X, where y is a length n vector and X is a n by p matrix of predictor variables. Assume for the moment that the values within each column of X are continuous without ties, but the method can also handle categorical variables. RPAB begins by finding a splitting rule for the full data, which reside in the root node. A splitting rule consists of a predictor X_j and a rule "is $X_{ij} < c$" for some value c. This creates an indicator function that is evaluated on each observation. Note that many different values of c may result in the same partitioning of the X_{ij}'s. Conventionally only the values of c that fall halfway in between two adjacent values in the sorted list of X_j are considered. This results in $n-1$ possible values of c, though for practical reasons this is usually further reduced. In particular, a 'minimum node size' parameter (which we call m) forces the tree to split at least m observations into each terminal node. The default value of m in the popular `rpart` package in R is 7. Thus, for a given X_j, there are a total of $n+1-2m$ possible values for c. For each of the $p(n+1-2m)$ splits the sample means are estimated and the resulting sum of squares (SSE) is evaluated.

After finding the minimum SSE split for some predictor X_j and some split point c, the algorithm sends all the data that satisfy the rule $X_{ij} < c$ to the left child of the root node of a tree and the data that do not satisfy the rule to the right child. The splitting algorithm is then applied independently to each child. All allowable splits are examined and the best one for each child is chosen. The algorithm continues until a stopping rule is met. A default stopping rule in the `rpart` package is that the best split must reduce at least 1% of the original SSE. Obviously the algorithm will refuse to split if the number of observations in a node is not at least $2m$.

Once a full tree has been created, the sample mean for a terminal node serves as the fitted value for each observation in that node. A new observation is predicted using the sample mean of the terminal node within which it is evaluated to fall.

This splitting procedure can overfit the training data (e.g., [11]; [56]). After the splitting algorithm terminates, the pruning algorithm seeks to find an optimal subtree with respect to some optimality criterion based on out-of-sample prediction. The approach used by Breiman et al. uses crossvalidation to estimate the test error, though there are other methods—a good discussion is given in [66]. Unfortunately, when the data are randomly subdivided and individual trees are fit to subsamples of the data, a different set of splits may be selected. There is therefore no way to measure prediction error on the original tree directly without using a test set. Instead, Breiman et al. develop an algorithm to estimate the optimal *size* of the tree, and then correspondingly restrict the size of the original tree.

Breiman et al.'s pruning algorithm, called cost-complexity pruning, works by assigning a fixed cost $z \in [0, +\infty]$ to each terminal node in a tree. Let T be any tree, $\tilde{T}$ be the set of terminal nodes in T, and let $|\tilde{T}|$ be the cardinality of $\tilde{T}$. Define the "fitness" of a tree to be $Q(T) = SSE(T) + z|\tilde{T}|$, where $SSE(T)$ is the sum of the SSE in the terminal nodes of T.

Let the subtree rooted at h be T_h for any interior node h. The fitness of T_h with $|\tilde{T}_h|$ terminal nodes is defined as: $Q(T_h) = SSE(T_h) + z|\tilde{T}_h|$. The idea is to evaluate this fitness for several values of z on all possible subtrees T_h. For a fixed value of z, Breiman et al. show that there is a unique subtree that gives the best fitness as long as ties are broken by choosing the smaller subtree (the result essentially shows that if two trees have the same fitness, they must be nested). By increasing z from 0 to $+\infty$, Breiman et al. show that a nested sequence of such optimal subtrees trees $T_M \prec ... \prec T_1 \prec T_0$ is obtained (T_0 denotes the full tree and T_M denotes a tree with no splits). Breiman et al.'s method then chooses among these optimal trees based on the best crossvalidation performance. Crossvalidation performance in this context means that the optimal z is estimated via crossvalidation, and then that penalty $\hat{z}$ is applied to the original tree. An optional rule is to find the best crossvalidation performance and then choose the smallest optimal tree with crossvalidation performance within 1 standard error of optimal (called the 1-SE rule). Torgo [66] argues against this rule and we do not employ this option in our analysis.

Breiman et al.'s method of minimizing the SSE on a piecewise constant model is not the only approach to recursive partitioning; there have been many developments on the metric used to split the trees. Alexander and Grimshaw [3] fit linear regression at the terminal nodes instead of piecewise constants. The SUPPORT method given in [18] also uses linear fits at the terminal nodes but additionally forces the estimated mean model to be continuous. MARS [27] also uses the recursive partitioning framework, and provides a continuous and polynomial fit using splines.

Su et al. [62] build a regression tree using likelihood based splits under the assumption that the data are drawn from a normal distribution with homoskedastic variance structure. They demonstrate that their tree uses the same splits as a RPAB algorithm, and thus the only difference is in the pruning method. They test four different information criteria as pruning mechanisms: AIC ([1] [2]), BIC (Schwarz 1978), AICc [36], and RIC (Shi and Tsai, 2002). They conclude that BIC or RIC are preferable for routine use with regression trees, recommending them as superior algorithms when the sample sizes are larger, a claim which is only moderately supported by their simulation study. However, their use of information criteria (IC) for pruning is inconsistent with the expected use of ICs, because they apply the IC penalty to a likelihood evaluated on data that are independent from the set used to fit the tree (see Chapter 2).

Heteroskedasticity has been studied in the context of regression trees. Ruth and Loughin [57] identify that the Brieman et al. algorithm for regression trees tends to make poor split selection choices under heteroskedasticity. In particular they use a simple simulation model to gain better understanding of the splitting performance of the popular R implementation of Breiman et al.'s regression tree `rpart`.

The general setup of the Ruth and Loughin work considers a single predictor variable x and a response y. The response y is related to x through a 10 step increasing mean function; in particular:

$$x = 1, 2, ...1000,$$
$$\mu_x = \lceil \frac{x}{100} \rceil,$$
$$y = \mu_x + \epsilon_x,$$

where μ_x is the mean at x. The model for ϵ_x is

$$\epsilon_x = \begin{cases} N(0, 1^2), x \leq 500 \\ N(0, s^2), x > 500 \end{cases} \quad s = 1, \ldots 10$$

Ruth and Loughin demonstrate the critical issue with regression trees under this setup as s gets large. The problem is that the tree prefers to make splits in the high variance area ($x > 0.5$), to the exclusion of splits in the low variance area ($x < 0.5$). This is because the high variance area has more overall SSE to be reduced. Thus `rpart` ignores multiple legitimate splits in the low variance area, which should be objectively quite easy to find because the signal-to-noise ratio in the low variance area is quite a bit higher.

We have mentioned that the default implementation of regression trees in R will make a split only if it reduces the overall SSE by at least 1% of the total. Ruth and Loughin demonstrate cases where even if the variance in the low-variance area is set to be 0, the default implementation still cannot find the splits. A solution is to adjust the settings to allow the tree to split fully, requiring no improvement in SSE to continue to make splits. However, this is not enough to fix the problem. The cost-complexity pruning algorithm routinely removes the splits in the low-variance area.

There has been some work on modeling the variance with a tree. Su et al. [61] explore the idea of fitting a tree to the squared error residuals from a standard regression model. The rough idea is that if a tree is constructed that models the error residuals, and after pruning still has greater than one terminal node, then the data are not homoskedastic. This is a very clever idea, and the method intuitively rates to be a useful diagnostic for non-standard relationships between the data and the variance. For example, if the mean and variance are unrelated, then a standard diagnostic plot of residuals vs predicted values will show no heteroskedasticity, but their tree might find a relationship between some X_j and the errors. Unfortunately, their approach suffers from two drawbacks. The first is that their overall model for the mean is restricted to be a linear model. The second drawback is again that they require a holdout set to determine the optimal tree size. The method we propose is an improvement on this, allowing for a tree-based model for both the mean and variance, and additionally not requiring a holdout set to prune the tree.

4.3 Adding variance splits to a regression tree

The original RPAB algorithm splits to minimize the sum of squared errors (SSE) in the resulting nodes. Our approach is similar to that of Su et al. [62] in that we base splits on a likelihood criterion rather than SSE. However, to split an interior node we additionally consider a split on the variance. We will assume throughout that the values of X_j are continuous without ties. Categorical and discrete predictors can be dealt with in the same way as Breiman et al. [11].

For a continuous variable, the standard approach is to consider a split of the form $i : X_{ij} < c$ for some constant c, resulting in $n - 1$ possible splits of the data for a given variable. The RPAB algorithm considers every covariate and every split location, computes the SSE if we split the node at that point, and takes the one that produces the minimum SSE in the resulting children. Our approach is the same, except that we change the metric from SSE to likelihood. A split on variable X_j at some value c of the form $i : X_{ij} < c$ partitions the data. The pair X_j, c that define a split are represented by the 'split' parameter S; maximizing over S refers to finding the best split across all predictors. Values in a node that satisfy the rule of any candidate split S are sent to the left child L_S, and values that do not satisfy the rule go to the right child R_S. The number of observations falling into L_S is n_{L_S}, and n_{R_S} is defined analogously. A sum over L_S is taken to be a shorthand for $\Sigma_{i:X_{ij} \leq c}$, and R_S is defined analogously. Finally, $\hat{\mu}_L$ is the model-predicted value in the left child, and $\hat{\mu}_R$ is the model-predicted value in the right child. The SSE criterion is:

$$SSE = \sum_{L_S} (y_i - \hat{\mu}_L)^2 + \sum_{R_S} (y_i - \hat{\mu}_R)^2$$

Our proposed method uses three types of splits: splitting where only the mean changes between the child nodes, splits where only the variance changes between the terminal nodes, or splits where both change.

4.3.1 Splitting on the Mean

When splitting on just the mean, our likelihood-based splitting procedure produces the same splits as the SSE procedure. This is not a surprising result, and has been shown before under various contexts (e.g., it is demonstrated in [62]). We use this opportunity to demonstrate our framework when we split the parent node into two children with means μ_L and μ_R and common variance σ^2. The log-likelihood of the model with a mean split is:

$$l(\mu_L, \mu_R, S, \sigma | X, y) = -\frac{n}{2} \ln 2\pi - \frac{n}{2} \ln \sigma^2 - \sum_{L_S} \frac{(y_i - \mu_L)^2}{2\sigma^2} - \sum_{R_S} \frac{(y_i - \mu_R)^2}{2\sigma^2}$$

It is easy to show that the maximum likelihood estimates are:

$$\hat{\mu}_L = \sum_{L_{\hat{S}}} \frac{y_i}{n_{L_{\hat{S}}}}$$

$$\hat{\mu}_R = \sum_{R_{\hat{S}}} \frac{y_i}{n_{R_{\hat{S}}}}$$

$$\hat{\sigma}^2 = \frac{\sum_{L_{\hat{S}}} (y_i - \hat{\mu}_L)^2 - \sum_{R_{\hat{S}}} (y_i - \hat{\mu}_R)^2}{n},$$

where $\hat{S}$ is the MLE of S; i.e., it is the split location that maximizes $l(\mu_L, \mu_R, S, \sigma|X, y)$. Consequently, the maximized log-likelihood is

$$\hat{l}(\mu_L, \mu_R, S, \sigma|X, y) = -\frac{n}{2}\ln 2\pi - \frac{n}{2}\ln \hat{\sigma}^2 - \frac{n}{2}$$

$$\hat{l}(\mu_L, \mu_R, S, \sigma|X, y) = -\frac{n}{2}\ln 2\pi - \frac{n}{2}\ln \frac{\sum_{L_{\hat{S}}} (y_i - \hat{\mu}_L)^2 - \sum_{R_{\hat{S}}} (y_i - \hat{\mu}_R)^2}{n} - \frac{n}{2}.$$

Clearly this function is maximized at the same split value S as the one that minimizes the SSE. An important corollary of this is that a likelihood tree consisting entirely of mean-only splits produces the same splits as a standard RPAB tree.

4.3.2 Splitting on the Mean and Variance simultaneously

We return to the initial likelihood formulation with a slightly different model. This time, we require a separate mean and a separate variance in both the left child and right child:

$$l(\mu_L, \mu_R, S, \sigma_L, \sigma_R|X, y) = \\ -\frac{n_{L_S} + n_{R_S}}{2}\ln 2\pi - \frac{n_{L_S}}{2}\ln \sigma_L^2 - \frac{n_{R_S}}{2}\ln \sigma_R^2 - \sum_{L_S}\frac{(y_i - \mu_L)^2}{2\sigma_L^2} - \sum_{R_S}\frac{(y_i - \mu_R)^2}{2\sigma_R^2}. \quad (4.1)$$

By rearranging slightly, we obtain

$$l(\mu_L, \mu_R, S, \sigma|X, y) = \\ \Big(-\frac{n_{L_S}}{2}\ln 2\pi - \frac{n_{L_S}}{2}\ln {\sigma_L}^2 - \sum_{L_S}\frac{(y_i - \mu_L)^2}{2{\sigma_L}^2}\Big) + \Big(-\frac{n_{R_S}}{2}\ln 2\pi - \frac{n_R}{2}\ln {\sigma_R}^2 - \sum_{R_S}\frac{(y_i - \mu_R)^2}{2{\sigma_R}^2}\Big) \quad (4.2)$$

Given the optimal split location $\hat{S}$, the MLEs for the other parameters are:

$$\hat{\mu}_L = \sum_{L_{\hat{S}}} \frac{y_i}{n_{L_{\hat{S}}}}$$

$$\hat{\sigma}_L^2 = \sum_{L_{\hat{S}}} \frac{(y_i - \hat{\mu}_L)^2}{n_{L_{\hat{S}}}}$$

$$\hat{\mu}_R = \sum_{R_{\hat{S}}} \frac{y_i}{n_{R_{\hat{S}}}}$$

$$\hat{\sigma}_R^2 = \sum_{R_{\hat{S}}} \frac{(y_i - \hat{\mu}_R)^2}{n_{R_{\hat{S}}}}$$

Consequently, we have essentially split the data completely into two non-interacting pieces. Both the left child and right child estimate their respective means and variances using the usual MLEs on a single data set. Also note that the MLE's for the means μ_L and μ_R are the same as in the single mean split.

4.3.3 Splitting on the Variance

We return to the initial likelihood formulation with another slightly different model. This time, we estimate two separate variances and one common mean in the left child and right child, leading to the log-likelihood

$$l(\mu, S, \sigma_L, \sigma_R | X, y) = -\frac{n_L + n_R}{2} \ln 2\pi - \frac{n_L}{2} \ln {\sigma_L}^2 - \frac{n_R}{2} \ln {\sigma_R}^2 - \sum_L \frac{(y_i - \mu)^2}{2{\sigma_L}^2} - \sum_R \frac{(y_i - \mu)^2}{2{\sigma_R}^2}$$

Although the form of these likelihood equations is similar to the previous ones, it turns out that this model has no closed form for the estimates of μ, σ_L, σ_R. These estimates can be found iteratively. Our preferred optimization method is the method of feasible generalized least squares (FGLS, [55]).

For a variance split on n observations, the goal is to estimate an $n \times 1$ column vector $\hat{\mu}$ and an $n \times n$ diagonal matrix Σ. Throughout we define τ_i to be the inverse of the element Σ_{ii}, that is $\tau_i = \frac{1}{\Sigma_{ii}}$. For clarity, Σ_{ii} is either be the value ${\sigma_L}^2$ or ${\sigma_R}^2$, depending on whether observation i falls into the left or right child. The FGLS algorithm is then:

1. Initialize $\hat{\Sigma}$ to be the identity matrix.
2. Estimate $\hat{\mu} = \frac{\sum \tau_i y_i}{\sum \hat{\tau}_i}$, using our current estimate of $\hat{\tau}_i = \frac{1}{\hat{\Sigma}_{ii}}$.
3. Estimate $\hat{\sigma}_L^2 = \sum_L \frac{(y_i - \hat{\mu})^2}{n_{L_S}}$ using our current estimate of $\hat{\mu}$.
4. Estimate $\hat{\sigma}_R^2$ using $\sum_R \frac{(y_i - \hat{\mu})^2}{n_{R_S}}$ using our current estimate of $\hat{\mu}$.

5. Repeat steps 2-4 until there is no appreciable change in the estimates of $\hat{\mu}$ or $\hat{\Sigma}$.

4.3.4 Computational Efficiency of a Series of Splits

One of the major advantages of a regression tree is that it can be built quickly. Now that we have defined the new splits that we will add, we show that we can compute our new splits in the same asymptotic complexity as the original algorithm, so our algorithm also enjoys the same speed capabilities. We consider the case where a node has n_T observations. Assuming for the moment that the values of each X_j are distinct, we have $O(n_T p)$ different places we can split. Naively, the computational cost of the split must be at least $O(n_T)$, to evaluate terms involving sums over all the local data. The naive algorithm would be at least $O({n_T}^2 p)$ to find the best split for a given node.

There is an improvement to be made to the efficiency of this algorithm. The usual approach is to sort the values of X_j for every j , which incurs a cost of $O(n_T p \log_2 n_T)$. The approach then changes depending on the type of split:

Fast Splits on the Mean

The relevant MLEs are:

$$\hat{\mu}_L = \sum_{L_{\hat{S}}} \frac{y_i}{n_{L_{\hat{S}}}}$$

$$\hat{\mu}_R = \sum_{R_{\hat{S}}} \frac{y_i}{n_{R_{\hat{S}}}}$$

$$\hat{\sigma}^2 = \frac{\sum_L (y_i - \hat{\mu_L})^2 + \sum_R (y_i - \hat{\mu_R})^2}{n_T}$$

To quickly compute $\hat{\mu}_L$, we need only n_L and $\sum_L y_i$. We split on some variable X_j, and since we have sorted X_j, we consider splits in increasing order of the values of X_j. As a consequence, n_L and $\sum_L y_i$ change in very predictable patterns. In particular, when we move the split point by exactly 1 observation (where the corresponding response is y_k), we can update ${n_{L_S}}^* = n_{L_S} + 1$ and $\sum_{L_S} {y_i}^* = \sum_{L_S} y_i + y_k$. Instead of needing to recompute the sum over n_L observations, we can compute the new value of $\hat{\mu}_L$ in $O(1)$.

Computing $\hat{\mu}_R$ is straightforward in an analogous way. Once we have $\hat{\mu}_L$ and $\hat{\mu}_R$, we need to compute $\hat{\sigma}^2$. In particular, we need to quickly update $\sum_{L_S} (y_i - \hat{\mu_L})^2$ (and analogously , $\sum_{L_S} (y_i - \hat{\mu_L})^2$). This is computed easily using the well known relation

$$\sum_{L_S} (y_i - \hat{\mu_L})^2 = \sum_{L_S} {y_i}^2 - n_{L_S} \hat{\mu_L}^2.$$

This formulation allows us to exploit the structure again. It is easy to maintain a running total of $\sum_L {y_i}^2$, and we are already able to compute n_{L_S} and $\hat{\mu}_L$ easily. Similarly we keep

Table 4.1: Runtime summary for all mean or mean-variance splits on a node with n_T nodes

Operation	Time Spent
Sorting $X_j's$	$O(n_T p \log_2 n_T)$
Evaluating the First Split for each of p predictors	$O(n_T p)$
Evaluating the Remaining Splits for each of p predictors	$O(n_T p)$
Total Complexity	$O(n_T p \log_2 n_T)$

track of $\sum_{R_S} {y_i}^2$, n_{R_S} and $\hat{\mu}_R$. These 6 quantities are sufficient to estimate $\hat{\sigma}^2$. Finally, observing that the evaluated likelihood depends only on $\hat{\sigma}^2$, we are successful in evaluating every split for a given variable in a total of $O(n_T)$. Summarizing, the computational complexity of each step in the algorithm is shown in Table 4.1. Since the most expensive operation is sorting the X_j's, the total runtime is $O(n_T p \log_2 n_T)$, which is much better than the naive approach of $O(n_T^2 p)$.

Fast Splits on the Mean and Variance

We will proceed using the same approach of sorting the relevant columns in X before we start. Recall the MLEs for the Mean-Variance split:

$$\hat{\mu}_L = \sum_{L_{\hat{S}}} \frac{y_i}{n_{L_{\hat{S}}}}$$

$$\hat{\sigma}_L^2 = \sum_{L\hat{S}} \frac{(y_i - \mu_L)^2}{n_{L_{\hat{S}}}}$$

$$\hat{\mu}_R = \sum_{R_{\hat{S}}} \frac{y_i}{n_{R_{\hat{S}}}}$$

$$\hat{\sigma}_R^2 = \sum_{R_{\hat{S}}} \frac{(y_i - \mu_R)^2}{n_{R_{\hat{S}}}}$$

We can maintain estimators of $\hat{\mu}_L$ and $\hat{\mu}_R$ in the same way that we did during the case for mean splits. Finding $\hat{\sigma_L}^2$ and $\hat{\sigma_R}^2$ also requires running sums of $\sum_{L_{\hat{S}}} {y_i}^2$ and $\sum_{R_{\hat{S}}} {y_i}^2$. As a consequence, splitting on both the mean and the variance have the same time complexity as the split on just the mean. The summary of the complexity of this algorithm is therefore also found in Table 4.1.

Fast Splits on the Variance

Unlike the previous two splits, which behave nearly identically, variance splits require an iterative optimization routine. Let us assume that the number of iterations required is bounded above by some constant c. The cost of evaluating the objective function is then

Table 4.2: Runtime summary for all variance splits on a node with n_T nodes

Operation	Time Spent
Sorting $X_j^\prime s$	$O(n_T p \log_2 n_T)$
Evaluating the First Split for each of p predictors	$O(n_T p)$
Evaluating the Remaining Splits for each of p predictors	$O({n_T}^2 p)$
Total Complexity	$O({n_T}^2 p)$

Table 4.3: Runtime summary for a single variance split on a node with n_T nodes

Operation	Time Spent
Sorting $X_j^\prime s$	0
Evaluating one split	$O(n_T)$
Total Complexity	$O(n_T)$

$O(cn_T) = O(n_T)$ (constants are excluded from the asymptotic runtime). However, we cannot compute the estimates for successive splits by a simple updating rule. Consequently the $O(n_T)$ optimization routine must be run for each of the $O(np)$ splits. A summary of the complexity of this split type is in Table 4.2

In addition to this worse theoretical runtime, we also have the practical issue that each split takes c times longer than the other splits. Even for moderate values of n_T, we can expect that evaluating all variance splits is prohibitively costly. We do, however, observe that evaluating one variance split is not costly compared to the time taken when evaluating all mean splits and all mean variance splits. We therefore compromise between evaluating all variance splits and evaluating none: we evaluate the variance split at the point where the mean-variance split is considered optimal. This significantly reduces the runtime, and lets us try to find a more parsimonious model. Also worth mentioning is that we no longer need to sort columns of X_j since we are just evaluating a single split.

4.4 Example: A Single Split on Heteroskedastic Data

We refer to the algorithm described in the previous section as 'HeaRTs': Heteroskedastic Regression Trees. While HeaRTs provides the opportunity to estimate variances that standard regression trees do not, HeaRTs also can be used to improve mean estimation in heteroskedastic settings. This happens because the sample mean is not the most efficient estimator of the population mean in a heteroscedastic setting. Rather, a weighted mean $\hat{\mu} = \frac{\sum \tau_i y_i}{\sum \tau_i}$, where τ_i is the precision (inverse variance) of observation i, is known to be a better estimate.

However, in order to compute this estimate, the variances for each observation must be known [16]. This information is generally not available, which leads to the FGLS algorithm described in Section 4.3.3 However, FGLS does not necessarily lead to more efficient estimates than OLS in heteroscedastic settings, because the variance estimation adds variability to the mean estimation in amounts that depend on the quality of the variance estimation (see [65], [58]). In this section, we show how HeaRTs can improve on mean estimation using 50 simulated data sets from a simple model. In Section 4.9 we show that improvement is often attained in real examples.

For our setup, we use $n = 1000$ points and a single predictor X_1. The true model for Y is:

$$Y = \begin{cases} N(0, 1^2), x_1 \leq 0.5 \\ N(0, 4^2), x_1 > 0.5 \end{cases} .$$

A dataset from this model is shown in the top panel of Figure 4.1.

We ignore split selection for this example, and simply place the split at $x_1 = 0.5$. In practice, HeaRTs will usually place a variance split near 0.5. We will also compare the results to the model that makes no split, which is what we expect from RPAB since the mean function is constant. We evaluate the quality of the splits using two error metrics. The first is the classical Mean Square Error (MSE), and the second is the Weighted Mean Square Error (WMSE).

$$MSE = \sqrt{\sum (\mu_i - \hat{y}_i)^2}$$

$$WMSE = \sqrt{\sum \frac{(\mu_i - \hat{y}_i)^2}{{\sigma_i}^2}}$$

where we use μ_i to denote the true value of the mean function at some value of x_1, and $\hat{y}_i$ is the model estimated value. We offer some discussion later as to why we prefer the second metric, but for the time being we provide both. Both statistics are computed on a test set of size 1000. We repeat the modeling process on 50 different training and test sets, and average the results. The results are shown in Table 4.4. In this setup, the mean-only split and the mean-variance split result in the same estimated means. We therefore have three setups to consider: no split, mean split, and variance split.

The direct comparison between no split and the mean split is fairly clear. Making no split tends to be less efficient when modeling Y when $X_1 \leq 0.5$, because the estimate of the mean using all 1000 points is 'contaminated' with points that have excess variability. Cutting them out gives a better estimate of the mean. However, making no split provides an improved model for Y when $x_1 > 0.5$, because here we're estimating Y with points that have, on average, lower variability. Instead of just using the 500 observations where $X > 0.5$, which have average variance of 16, the points used to estimate $Y|X > 0.5$ are estimated with all 1000 points with an average variance of 8.5.

It is intuitively appealing that the mean split performs 'equally well' on the RWMSE metric on $X_1 \leq 0.5$ and $X_1 > 0.5$. This is reasonable, as $\hat{\mu}_L$ and $\hat{\mu}_R$ tend on average to be

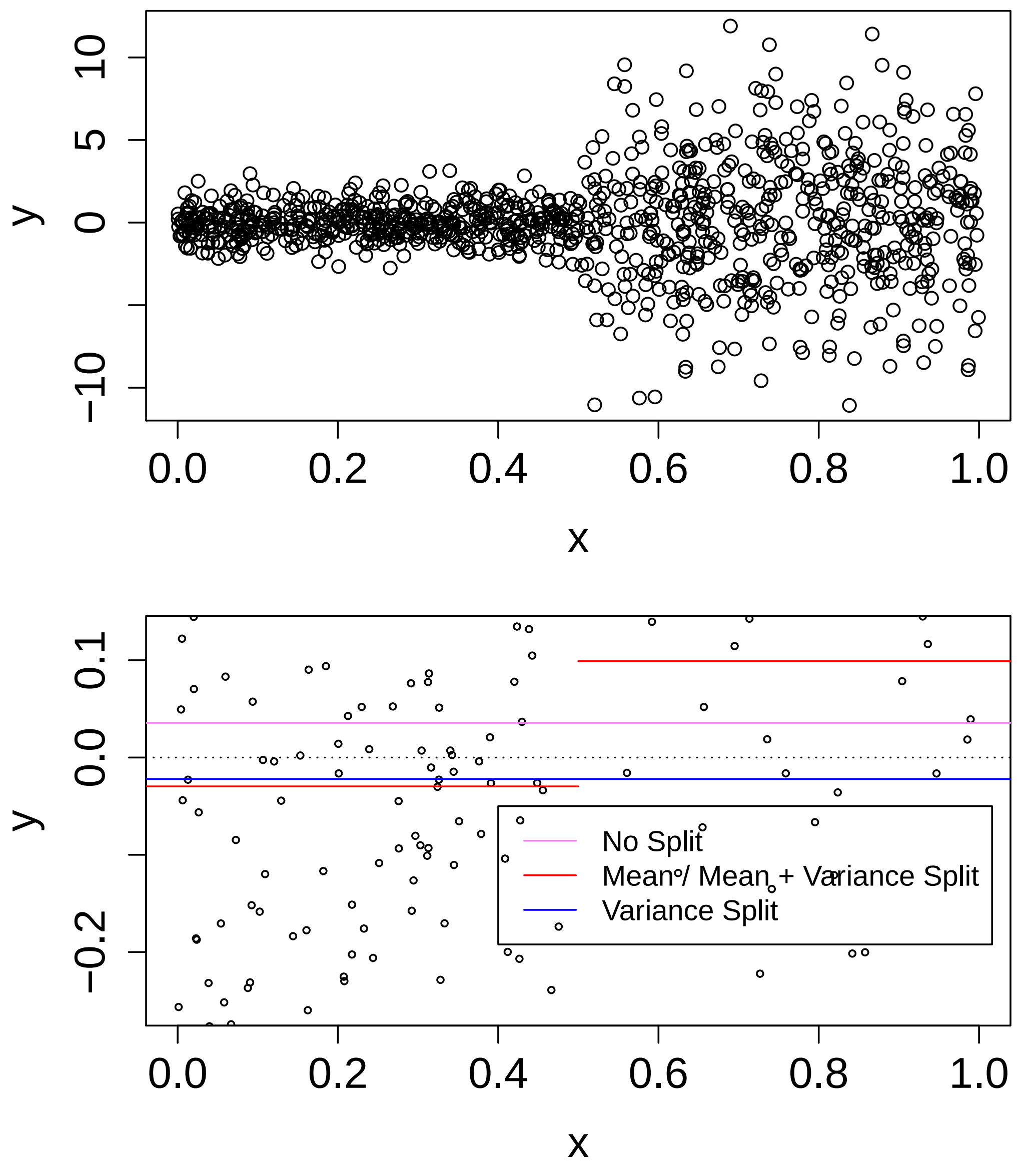


Figure 4.1: Plot of toy setup analysing one split. Top pane shows one dataset, centered at $y = 0$ with significantly higher variance when $X_1 > 0.5$. The bottom pane shows the resulting mean models given by the various types of splits, with a thin line at $y = 0$ for reference. Worth noting is that the variance split uses a weighted average of the two mean estimates, and tends to be a more precise estimate of the mean than simply averaging across all the data with uniform weight.

about the same number of standard deviations away from the true mean. Recall that the split is placed at $X_1 = 0.5$, so $\hat{\mu}_L$ is estimated from $N(0, 1)$ data, and $\hat{\mu}_R$ is estimated from $N(0, 4)$ data. Finally, it is clear from this table that for this setup, the variance split gives much better mean estimates than its competitors.

Table 4.4: MSE and WMSE on the single split example for three types of split.

Split Type	MSE $x_1 \leq 0.5$	MSE $x_1 > 0.5$	Total MSE
No Split	0.0049	0.0049	0.0098
Mean or Mean-Variance Split	0.0010	0.0180	0.0190
Variance Split	0.0002	0.0002	0.0004

Split Type	WMSE $x_1 \leq 0.5$	WMSE $x_1 > 0.5$	Total WMSE
No Split	0.0049	0.0003	0.0052
Mean or Mean-Variance Split	0.0010	0.0010	0.0020
Variance Split	0.0002	0.0000	0.0002

4.5 Choosing the best type of split at a given node

Kullback and Leibler [39] derived a framework for finding the information lost when a model $g(x, \theta)$ is used to approximate the true distribution $f(x)$ for a random variable X. The well known Kullback-Leibler (KL) information is:

$$\int_x f(x) \ln \frac{f(x)}{g(x, \theta)}.$$

For the purposes of model comparison, the formula is usually split up into two terms:

$$\int_x f(x) \ln f(x) - \int_x f(x) \ln g(x, \theta).$$

By observing that the first term of the formula contains elements that depend only on the truth (not the model, $g(x, \theta)$), we typically seek to evaluate only the second term. We call this the relevant part of the KL information. The relevant part can be expressed as:

$$E_f[\ln(g(x, \theta))], \tag{4.3}$$

where E_f indicates that the expectation is taken under the truth.

Akaike [1] shows under some regularity conditions it is possible to approximate (4.3) with $\ln L(\hat{\theta}|Y)$, the maximized log-likelihood of a model fit via maximum likelihood. Akaike showed that this is a biased estimator, but the asymptotic bias is given by ν, the number of parameters that were estimated by maximum likelihood. Akaike's result does not specify a model for g, which means that AIC can be applied to any model where the regularity conditions apply. For 'historical reasons' [14] information criteria are typically multipled by -2. Akaike's formula is then:

$$AIC(g(x, \theta)) = -2 \ln L(g(x, \theta)) + 2\nu,$$

where $L(g(x))$ is the maximized log-likelihood of $g(x, \theta)$ on the training data.

AIC is unhelpful when samples are small. The AIC penalty approximation works asymptotically, and the true bias can be heavily underestimated by the AIC formula in small samples [13]. There are some developments to extend Akaike's ideas to small samples. In general, the formulations of the small sample bias are model-dependent. Takeuchi [64] develops a criterion (TIC) that works under model misspecification, however Burnham and Anderson [13] point out that relatively large sample sizes are required to estimate the terms to sufficient accuracy.

The small sample version of AIC for linear regression was initially proposed by Sugiura [63]. Hurvich and Tsai [36] named this quantity the "corrected AIC", or AICc. They demonstrated the superiority of AICc over AIC in small samples and additionally recommend it for use on time series models and nonlinear regression. The penalty in AICc has the form $2n/(n-\nu-1)$, where ν is the number of estimated parameters. This is asymptotically equivalent to Akaike's penalty (observe that by setting $n >> \nu$, the denominator is dominated by n, and the fraction approaches 2) but becomes much larger when the number of parameters in a model is an appreciable fraction of the sample size.

We showed in Chapter 2 that for a heteroskedastic linear model, the relevant part of the KL information is

$$E_f\{-2\ln L(\hat{\theta}|X,y)\} - E_f\{E_f\{-2\ln L(\theta|X,y)\}|_{\theta=\hat{\theta}}\} = \\ E_f\{\operatorname{Tr}(\hat{\Sigma}^{-1}\Sigma_f) + (X\beta - X\hat{\beta})'\hat{\Sigma}^{-1}(X\beta - X\hat{\beta}) - n\}. \quad (4.4)$$

where Σ is a diagonal matrix representing the variance of y and μ is a column vector corresponding to the means of y. In the context of evaluating the split of a terminal node, the parameters μ and Σ each have two possible forms. For variance splits, μ is a vector of constants, while for mean or mean-variance splits, μ contains n_{L_S} elements μ_{L_S} and n_{R_S} elements μ_{R_S}. Similarly, for mean splits, the diagonal elements of Σ are all equal, whereas for variance or mean-variance splits, Σ contains n_{L_S} elements ${\sigma_{L_S}}^2$ and n_{R_S} elements ${\sigma_{R_S}}^2$.

Note, however, that this form is conditional on knowing the split location in advance. In reality, S is another parameter that is estimated jointly with μ and Σ through maximum likelihood, and the IC bias needs to account for this. Unfortunately, the process by which this is done makes it unlikely that the bias correction, $E_f\{-2\ln L(\hat{\theta}|Y)\} - E_f\{E_f\{-2\ln L(\theta|Y)\}|_{\theta=\hat{\theta}}\}$, exists in closed form.

We therefore approximate this quantity using simulation. We generate data from a null distribution where the true model requires no split to be made:

$$X_1, ..., X_p \overset{iid}{\sim} U(0,1)$$

$$y \sim N(0,1)$$

and find the best mean split, mean-variance split, and variance split. Our algorithm computes $p(n-2m+1)$ different mean and mean-variance splits, and a single variance split (chosen at the same point as the best mean-variance split).

After finding the best split of each category, we then evaluate the right hand side of (4.4), conditional on the generated columns $X_1, ..., X_p$. By simulating many datasets drawn from the null distribution and averaging the results from (4.4), we obtain an estimate of the penalty term for each of the three split types. The penalty depends on split type, n, p, and the minimum node size parameter m. We fix $m = 20$, a setting that was found to be fairly good, and simulate the IC for each type of split and a variety of n, p cases with $20,000$ simulations each. The penalty values are found in Table 4.5, Table 4.6, and Table 4.7. We also compute (not shown) an alternate table for $m = 10$, which tends to be a better value on small data sets.

There are a few important points to make regarding these tables. First, note that p, the number of predictors being searched for the optimal fit, has an extremely strong effect on the IC. However, note that the number of parameters ν being estimated by the model is constant within a table (consider for example the mean split, where we fit two means and one variance no matter the value of p.). Most IC's would therefore use the same penalty for different values of p. Thus it is apparent that increasing the number of locations at which the log-likelihood is evaluated to find the optimal split point exerts upward pressure on the bias associated with using the maximum log-likelihood to estimate (4.4). Our IC takes this into account and correctly penalizes the search procedure. Note also that unlike AICc, the penalties seem to increase with n for fixed p. In the AICc framework, as n increases (holding the ν fixed) we expect the penalty term to approach 2ν from above. Again, our IC accounts for the fact that the splitting algorithm evaluates every possible split point and chooses the best. As n increases, there are more potential splits. As a result the best overall split is even more likely than usual to be 'chasing the errors' in the training set instead of the true mean function, and the expected bias increases with n.

Table 4.5: Penalty Table for a Mean Split. Penalties were estimated using $20,000$ trials.

		p					
		1	2	4	8	16	32
	50	8.9	11.1	13.7	16.8	20.2	23.6
	100	11.0	13.8	16.8	19.7	23.0	26.5
	200	12.3	14.9	17.8	21.0	23.9	27.3
n	400	13.4	15.7	18.8	21.8	25.2	28.1
	800	13.9	17.0	19.9	22.5	25.6	28.7
	1600	14.7	17.3	20.2	23.8	26.3	29.3
	3200	15.8	18.4	21.0	24.2	27.2	29.9
	6400	17.2	18.8	20.6	25.0	28.1	31.2
	12800	15.4	18.5	21.9	24.2	27.8	31.4

Table 4.6: Penalty Table for a Variance Split. Penalties were estimated using $20,000$ trials.

		p					
		1	2	4	8	16	32
n	50	7.8	9.2	10.9	12.6	14.4	16.2
	100	8.9	10.7	12.4	13.9	15.5	17.4
	200	9.6	11.4	12.9	14.8	16.2	17.9
	400	10.6	11.7	13.5	15.3	17.3	18.6
	800	10.9	12.8	14.4	15.7	17.4	19.0
	1600	11.5	12.8	14.5	16.9	17.9	19.4
	3200	12.2	13.8	15.1	16.9	18.4	19.7
	6400	13.5	13.9	14.3	17.4	19.0	20.8
	12800	10.8	12.6	14.8	15.7	17.9	20.0

Table 4.7: Penalty Table for a Mean-Variance Split. Penalties were estimated using $20,000$ trials.

		p					
		1	2	4	8	16	32
n	50	12.8	16.1	19.9	24.1	28.5	33.1
	100	16.1	20.0	24.2	28.2	32.6	37.4
	200	17.8	21.6	25.7	30.0	34.2	38.9
	400	19.3	22.7	26.9	31.1	35.7	39.8
	800	20.0	24.1	28.0	31.8	36.0	40.3
	1600	20.9	24.4	28.4	33.1	36.5	40.7
	3200	22.0	25.5	29.2	33.3	37.3	41.0
	6400	23.5	26.0	28.6	33.9	37.9	42.0
	12800	21.7	25.6	29.9	33.0	37.4	41.9

When querying the table with values that fall between the rows, we use bilinear interpolation. For values that fall outside, we use the closest point in the table. Particularly for large values of p a more appropriate algorithm should be used. In our work we have not analysed values of p much larger than 32 or n larger than 12800, so a better rule was not necessary. Now that we have a penalty for each model, choosing which split type to use is easy. We simply compute the penalized log-likelihood for each of the three splits, and choose the one that produces the maximum penalized likelihood.

Some tree building procedures have a stopping rule that requires a split to be sufficiently good to actually be made (e.g. in the implementation of RPAB in R, the default setting is that a split must reduce the overall SSE by at least 1% to be considered). We use no stopping rule of this form. Our algorithm builds the tree until every node contains fewer than $2m$ observations.

4.6 Pruning and the Accumulated Information Algorithm

On a given data set, the more splits a tree makes, the fewer observations remain in each terminal node to estimate means and variances. Naturally using fewer observations leads to higher variance of these estimates, but having more terminal nodes means that the tree can estimate means and variances with less bias. Conversely, smaller trees (with fewer splits) have more observations in the terminal nodes, and can produce estimates that are less variable but potentially more biased. We typically wish to find an optimal balance between bias and variance, which may reside in a tree partway between the root node and the full tree.

The usual approach to finding an optimal tree is to first build a large tree, and then remove some of the splits in the tree. 'Pruning' is the name given to the second step of removing nodes from the tree. Essentially pruning is imparted to minimize the probability of overfitting the training data ([66]). Breiman et al. [11] suggested using either a hold out set or cross-validation to estimate the quality of a set of trees with varying 'complexity' (i.e., size), and to pick the tree that performs the best on the test data.

Our method differs from previous pruning approaches in that *our pruning doesn't use a tuning parameter*. Our pruning algorithm simply finds the model that has the best IC. There is no input from the user to be considered, so there is only one model (the best one) to find. An alternative could be to work in the style of Su et al.[62], and keep a holdout set to evaluate a series of models. We could pick the best model as the one with the maximumized log-likelihood (not the AIC, just the likelihood) on the holdout set. Our method essentially does this, but *without needing a holdout set*.

Our pruning algorithm, like our tree-building algorithm, is recursive. We therefore illustrate the pruning paradigm in the context of a three-node tree, with a parent node P, the left child L, and the right child R. All pruning decisions are based on three-node subtrees. The goal is to recursively choose between two models: one for the parent node and one for the two child nodes.

The model fit at the parent node is a normal model $N(\mu, \sigma^2)$, which is fit to all the data. This is compared to the best model found by the tree, which is a mean split, a variance split, or a mean-variance split. For a mean split the model for the left child is $N(\mu_L, \sigma^2)$ and for the right child is $N(\mu_R, \sigma^2)$. For a variance split the model for the left child is $N(\mu, {\sigma_L}^2)$ and the model for the right child is $N(\mu, {\sigma_R}^2)$. For a mean variance split the model for the left child is $N(\mu_L, {\sigma_L}^2)$ and the model for the right child is $N(\mu_R, {\sigma_R}^2)$. Initially we obtain the evaluated likelihood of each model on the data in the node, and initially refer tot his as the 'information' in the node. We denote the information of the left child as I_L, the right child's information as I_R, and the parent's information as I_P. We create an IC by adding the appropriate penalty value to the information. Finally, the IC penalty value for the model fit at the parent node is B_P, and the IC penalty value for the model fit to the children is B_S.

The model for the parent node has no split, so penalty is given by the AICc (we use the version that does not multiply the log-likelihoods by 2):

$$B_P = p + \frac{p(p+1)}{n_T - p - 1}.$$

The penalty for the children is the relevant CHIC table evaluated at n_T and p, found in 4.5,4.6, or 4.7. We denote this penalty as $B_S = CHIC[n_T, p]$.

The general approach is to retain a split if:

$$I_P - B_P < I_L + I_R - B_S$$

and otherwise, we prune the split. When we retain a split, the intuition is that the parent's true value lies in its children. We therefore replace the parent's information value I_P with the information of the children. In particular, we set:

$$I_{Pnew} = max(I_{Pold}, I_L + I_R - B_S + B_P).$$

We apply this procedure recursively starting from the bottom of the tree and working up until all splits have been evaluated and either pruned or retained.

4.7 Final Estimation of the Tree Parameters

Once a typical RPAB tree has been split and then pruned, the model parameter estimates can be obtained by taking the sample mean at every terminal node. Our method, which is unique in that it is able to share estimated parameters across different nodes in the tree, needs to estimate a final set of parameters through a slightly more complicated mechanism.

Our estimation step has two stages. First, we determine which parameters are shared across the nodes of the tree. Second, we estimate the parameters. Since means and variances can be shared separately across the nodes of the tree, we introduce the 'painting algorithm' to determine which terminal nodes share means and which terminal nodes share variances.

The painting algorithm for the mean function is:

1. Begin with an arbitrary 'colour' for the root node
2. For every split, paint the children with either: a) the same colour as the parent or b) two new colours

   a) If the split type is a variance split, paint the children with the same colour as the parent

b) If the split type is a mean split or a mean-variance split, paint the children with two new colours

3. Beginning at the root and painting downwards, paint the mean function for the entire tree

The painting algorithm for the variance function is:

1. Begin with an arbitrary ‘colour’ for the root node
2. For every split, paint the children with either: a) the same colour as the parent or b) two new colours

   a) If the split type is a mean split, paint the children with the same colour as the parent

   b) If the split type is a variance split or a mean-variance split, paint the children with two new colours

3. By beginning at the root and painting downwards, paint the variance function for the entire tree

Once we have painted the entire tree, the terminal nodes have the following properties: some share a common mean, some terminal nodes share a common variance, and some terminal nodes share no parameters with any other node.

Consider a group of g terminal nodes that share a common variance. Each node has a different colour for the mean function. The goal is to first estimate g separate means, and then construct the pooled variance estimator. Number the means μ_1 through μ_g, the terminal nodes t_1 through t_g, and the sizes of the terminal nodes n_1 through n_g. The MLE for the means is given by:

$$\mu_i = \frac{1}{n_i} \sum_{j \in t_i} y_j.$$

After estimating the means, the MLE for the pooled variance is:

$$\sigma^2 = \frac{\sum_{j \in t_1} (y_j - \mu_1)^2 + \sum_{j \in t_2} (y_j - \mu_2)^2 + ... \sum_{j \in t_g} (y_j - \mu_g)^2}{n_1 + n_2 ... n_g}$$

For a given group of g nodes that share a common mean but have different variances, there is no closed form for the MLE's of μ and ${\sigma_1}^2, {\sigma_2}^2 ... {\sigma_g}^2$. Instead, we estimate them using the FGLS algorithm [55].

Finally, for any node that shares no parameters with another node, there is no need to re-estimate the parameters using data from other nodes. Both the mean and variance are easily estimated using the usual MLEs for a normal model.

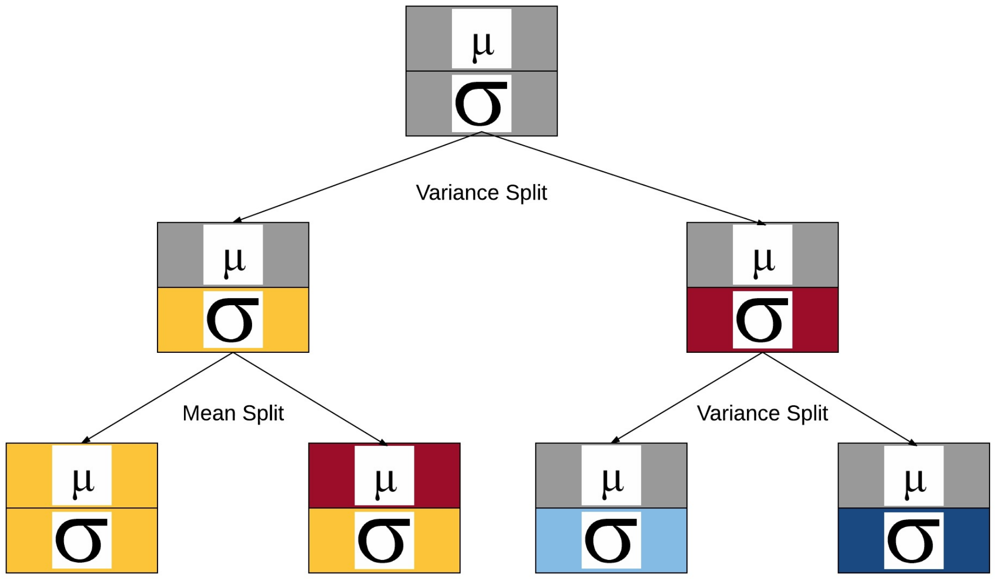


Figure 4.2: An illustration of the painting algorithm.

Figure 4.2 gives an illustration of the painting algorithm. We begin with the mean model. Starting at the root, we paint the root grey. We then perform a variance split. We retain the parent's grey for the mean and paint the children grey. The left child is considered next. Here we make a mean split, and the children's means are painted two new colours (yellow and red). Finally we consider the right child of the root. Here we perform another variance split, and these children's means are also painted grey.

We now consider the variance model. Again we begin at the root, painting grey. Since the root performs a variance split, we paint the children's variances two new colours (yellow and red). We then split the left child. Here we perform a mean split and paint the children's variances yellow. Finally we split the right child of the root with a variance split, and paint the children with two new colours (light and dark blue).

Once we have painted the tree, we observe that the two leftmost terminal nodes share a colour for the variance (yellow), and the two rightmost terminal nodes share a colour for the mean (grey). As a consequence, these parameters are shared across the terminal nodes.

4.8 Simulated Examples

Having described our modifications to the RPAB algorithm, we demonstrate our technique on a set of toy examples. We begin by showing some toy examples in detail to demonstrate the advantages of our new methodology. In particular we identify two flaws in the RPAB algorithm under heteroskedasticity. The first issue is that RPAB can have terminal nodes where the observations making up the terminal node have significantly different variances. Under this scenario, RPAB uses an inefficient estimator for the mean of the terminal node. The second flaw is defined by Ruth and Loughin [57]. Ruth and Loughin demonstrate that if a variable X_j affects both the mean and the variance, then the splits that RPAB creates prefer to be on the high variance data instead of the low variance data. We show that our HeaRTs algorithm can overcome both flaws.

4.8.1 Metrics for model performance

Before discussing our simulation setups, we need a method to evaluate how well a model fits a heteroskedastic data set. The KL Divergence [39] is a popular method for assessing the quality of fit of a model g to the truth f:

$$\phi = \int_x f(x) \ln \frac{f(x)}{g(x,\theta)} = \int_x f(x) \ln f(x) - \int_x f(x) \ln g(x,\theta).$$

For model comparison purposes we ignore the first term and focus on the second. We approximate the integral using test data, which we assume to follow the same distribution as the training data. The test data has n_{test} points y_1 through y_n, and we have:

$$\hat{\phi} \approx -http://ideone.com/\sum_{i=1}^{n} \frac{1}{n_{test}} \ln g(y_i, \hat{\theta}).$$

For the HeaRTs method, $g(y,\theta)$ is simply the normal density function with μ_i's and σ_i's estimated from the training data. The RPAB method, however, is not based on an explicit model. We cannot use the KL divergence without making further assumptions about the model that RPAB is based upon. Considering that the RPAB produces the exact same tree as HeaRTs does if we allow only mean splits, it seems reasonable to evaluate both algorithms using a metric based on a normal model. Specifically, we assume that the test set can be constructed as $y_i \sim N(\mu_i, {\sigma_i}^2), i = 1, ..., n$, where σ_i^2 is known. When doing simulations, ${\sigma_i}^2$ is known a priori and the true value is used. On real data it is impossible to know the true value of ${\sigma_i}^2$, so we use the predicted variances from the HeaRTs algorithm as a substitute.

Under this further assumption, we can now evaluate:

$$\hat{\phi} \approx \frac{-1}{n_{test}} \sum_{i=1}^{i=n} \Big(-\frac{\ln 2\pi}{2} - \frac{\ln {\sigma_i}^2}{2} - \frac{(y_i - \hat{\mu}_i)^2}{2{\sigma_i}^2} \Big)$$

Under this setup, $\frac{\ln 2\pi}{2}$ and $\frac{\ln {\sigma_i}^2}{2}$ are constant terms and are not necessary to evaluate for model comparison purposes. Thus the only relevant term is:

$$\frac{1}{n_{test}}\sum\frac{(y_i - \hat{\mu}_i)^2}{{\sigma_i}^2}.$$

We have dropped the factor of 2 for convenience and we seek to minimize this term. We call this criterion the "Weighted Mean Square Error" (WMSE). For the purposes of the next sections, we produce both the Root Mean Square Error (RMSE) and the Root Weighted Mean Square Error (RWMSE). These quantities are computed by taking the square root of the MSE and WMSE, respectively.

4.8.2 Example: Variance Splits

Having defined how the models are assessed, we continue by using an example to highlight some of the differences between our method and RPAB. We show two setups. In the first setup, we will show that HeaRTs and RPAB produce different splits when the variance is unrelated to the mean. In the second example, we show the ability of HeaRTs to overcome the problems described in Ruth and Loughin [57].

In our first example, data are heteroskedastic with a variance model that is unrelated to the mean model. We fit RPAB with default settings (RPAB Default), RPAB with no forward stopping rule (RPAB Full Split), RPAB with no stopping rule plus pruning (RPAB Full Split + Pruning), and HeaRTs.

The model is:

$$X_1 ... X_5 \overset{iid}{\sim} U(0,1)$$

$$\mu_i = 4(\mathbb{1}_{X_{i2}>0.5} + \mathbb{1}_{X_{i3}>0.5} + \mathbb{1}_{X_{i4}>0.5} + \mathbb{1}_{X_{i5}>0.5})$$

$$Y_i = \mu_i + \epsilon_i$$

$$\epsilon_i \sim N(0, 1 + 4(\mathbb{1}_{X_{i1}>0.5})),$$

where $\mathbb{1}$ is the indicator function. Figure 4.3 shows the plot of the data in four summary perspectives. The top left plot shows how y relates to X_1 (demonstrating the degree of heteroskedasticity). Next, in the bottom left, we show how y relates to X_2. Note the step function at 0.5 and also that the data do not appear heteroskedastic in this dimension. Third in the top right we show the relationship between y and $\sum_{j=2...5}\mathbb{1}_{X_j>0.5}$, demonstrating the true mean function. Finally in the bottom right we show a plot of the "Theoretical Residual Plot". This is a plot of the data, where the horizontal axis is the evaluated true mean function and the vertical axis is $y_i - \sum_{j=2...5} \mathbb{1}_{X_j>0.5}$. We call these two terms the theoretical mean and theoretical residual, respectively. It is worth noting that this typical diagnostic plot to assess heteroskedasticity would obviously fail under our setup.

Recall that one of the main observations of Ruth and Loughin [57] was that RPAB prefers to make splits in the high variance area of the data. As the heteroskedasticity is unrelated to the mean, this setup does not cause the problems for RPAB described in Ruth and Loughin. The RPAB tree makes splits on the variables related to the mean, and makes few splits on X_1. The performance difference, then, between RPAB and HeaRTs is that HeaRTs makes variance splits on X_1. The metrics are tabulated in Table 4.8.

Note that the HeaRTs model, which explicitly minimizes RWMSE, does not necessarily provide the best RMSE. In this setup HeaRTs typically starts by making a variance split near the root of the tree, and then making several more mean splits on each half. As a consequence, there are no shared parameters across the terminal nodes in the tree. Thus the low variance means are estimated precisely, but the high variance means are imprecisely estimated. HeaRTs thinks that this is a worthwhile tradeoff, as the most important (lowest variance) points are estimated precisely. In an unweighted average, however, the estimated means from HeaRTs are further from the truth than RPAB means.

Table 4.8: Performance on the setup with the variance unrelated to the mean. Averages over 50 runs

Method	RMSE	RWMSE
RPAB - Default	1.26	0.91
RPAB - Full Split	1.48	0.96
RPAB - Full Split + Pruning	1.06	0.74
HeaRTs	1.58	0.68

In Figure 4.4 we show a predicted vs actual plot for the four different methods of fitting these data. The horizontal axis represents the predicted values and the vertical axis represents the mean of the actual values in the test set for a given predicted observation. The vertical bar shows $\pm 1\sigma^*$, where σ^* is the average true standard deviation of the values at that prediction. The main observation is that HeaRTs partitions its predictions into groups where the variance within terminal nodes is relatively homogenous, either high or low, whereas RPAB does not. This is shown by the varying lengths of the vertical lines in the HeaRTs plot, whereas the vertical lines in the other plots are relatively heterogeneous. This is expected—since the variance is unrelated to the mean, RPAB has no particular preference to split on X_1.

In our second example, we show a setup comparable to the examples in Ruth and Loughin using one of their moderate variance settings. Like them we use $n = 1000$, and we consider the model

$$X_1 \sim U(0,1)$$

$$Y_i = \lceil 10x_{i1} \rceil + \epsilon_i$$

$$\epsilon_i \sim N(0, 1 + 4(\mathbb{1}_{X_{i1}>0.5})).$$

In Figure 4.5, we see a clear demonstration of why variance splits can be useful for individual regression trees. We first look at the HeaRTs model, which does more or less what we would hope. For values of X_1 less than 0.5, the HeaRTs model splits this data set 6 times, one split in excess of the optimal. For higher values of X_1, the HeaRTs algorithm finds only 3 splits because the high variance makes the changes in the true mean function harder to locate. Next, the default unpruned RPAB algorithm makes too few splits, particularly in the low variance region. Large prediction errors are worse in a RWMSE sense if they occur in the low variance regions. RPAB with no pruning and full splitting obviously makes too many splits, and the result is a model that has excess variance (observe that the average distance between these points and the blue line is considerably higher over the space of X_1 than for the previous two fits). Finally, the pruned model strips away most of the valuable mean splits in the low variance portion of the data, while leaving relatively unhelpful mean splits in the high variance region: precisely the opposite of what should be done! The metrics are computed and shown in Table 4.9. We see that the HeaRTs algorithm is a clear improvement over all 3 versions of RPAB both in the variance-weighted and unweighted metrics.

What is interesting to note is that RPAB will make some splits on the variance 'by accident' if the relationship between the predictors and the mean is related to the relationship between the predictors and the variance. In particular, in the case where the variance is related to the mean, RPAB will make several of the same splits as HeaRTs. This is in contrast to the previous setup where RPAB clearly made no splits on variance, as the terminal nodes all had the same variance.

Table 4.9: Performance on Ruth and Loughin Setup - Averages over 50 runs

Method	RMSE	RWMSE
RPAB - Default	0.80	0.52
RPAB - Full Split	1.12	0.32
RPAB - Full Split + Pruning	0.82	0.38
HeaRTs	0.61	0.21

4.9 Real Data Examples

We compare RPAB and HeaRTs on the same 42 datasets as in Chipman et al. [20]. The datasets are regression setups, with n varying from 96 to 6906. Categorical variables are coded as individual 0/1 columns. The regressions have between 3 and 28 continuous variables and 0 to 6 categorical variables. Gelfand [30] showed that many of these datasets are heteroskedastic. An important difference between our setup and that of Chipman et al. is that we do **not** use a variance-stabilizing transformation of the response Y.

To assess the quality of the fit on a given data set, we split the data 90%/10%, constructing a training set and test set respectively. The two methods (RPAB and HeaRTs) are fit to the training data and the fit is evaluated on the test data. Previously we showed three versions of RPAB: as we found that RPAB with no forward stopping rule and pruning seems to perform best, we use that version of RPAB. To reduce the variability due to the randomness of the splitting procedure, the data are resplit 50 times and the metrics (RMSE and RWMSE) are averaged over all 50 resplits.

Figure 4.6 shows two boxplots of the ratio of the metrics between the two methods. For each dataset, we compute:

$$\frac{RMSE_{RPAB}}{RMSE_{HeaRTs}}$$

and

$$\frac{RWMSE_{RPAB}}{RWMSE_{HeaRTs}}.$$

Values above 1 indicate that HeaRTs performs numerically better on this dataset.

We find that the two methods perform comparably on real data in terms of RMSE. However, on heteroskedastic data, the HeaRTs method has an obvious advantage with respect to RWMSE. It is worth noting that a limitation of our approach is in order to compute RWMSE, we need a variance model. We chose to use estimated variances based on the fitted HeaRTs model. It is conceivable that this might lead to an outcome that favours the HeaRTs algorithm. However, note that HeaRTs does not perform better in a RWMSE sense on the datasets Gelfand found to be homoskedastic. We view this as evidence that HeaRTs is not spuriously estimating variances and inflating the RWMSE metric in its favour. An important, additional advantage of our method is that since our tuning algorithm does not require crossvalidation, it runs significantly faster. All else being equal, if k fold cross validation is used to find the best cost complexity parameter, our pruning method is a full factor of k faster.

4.10 Conclusion

We propose a novel method that combines mean and variance splits in a regression tree. We also introduce a new method of pruning that does not require crossvalidation, and as a consequence is significantly faster than tuning via CV. We show that our method, while not necessarily reducing RMSE, improves the quality of the model fit on real data in the low variance portion of the data. Furthermore, our pruning algorithm executes much faster than one based on sample-reuse, such as the cost-complexity pruning of Breiman et al [11]. Thus, we can offer a faster algorithm for regression trees that holds its own on homoscedastic data while showing a clear improvement over its main competitor on heteroscedastic data.

Although our focus was on modeling the mean, particularly in the low variance area of the data, HeaRTs could potentially be used in other contexts. HeaRTs naturally constructs an implicit variance model. One potential use of this would be to construct predictive intervals for new observations. When modeling heteroskedastic data, it seems reasonable to expect that HeaRTs would give better predictive intervals than a model that assumes homoskedasticity. In Chapter 3 we discuss models where one of the objectives of the modeling process is to understand the relationship between the predictors and the variance of the response. HeaRTs can address this also, with all the advantages that a regression tree offers (the HeaRTs tree is easy to interpret, automatically detects interactions, and can model non-linear effects). Finally HeaRTs can potentially be used as a method for detecting heteroskedasticity in datasets. By fitting a pruned HeaRTs model to a dataset, if the HeaRTs model still has variance splits, it could be reasonable to conclude that the data are heteroskedastic.

There is an important limitation to consider regarding the simulated penalties in CHIC. Our computation of the penalties under the assumed 'null model' could be incorrect if the data are distributed considerably differently from the assumed simulation. In particular, we assume that the entries in X are distributed independently and uniformly, and that the y's are distributed as normal random variables. If the actual entries in X were correlated or discrete, the CHIC penalties would be incorrect (in this case they would be conservative, in the sense that the penalties will be too large and smaller models will be preferred). If the true ys are actually drawn from a distribution quite unlike the normal, then the penalties may be off in either direction. This can be fixed by simulating penalties under different scenarios, but the drawback is that simulating penalties takes a considerable amount of time.

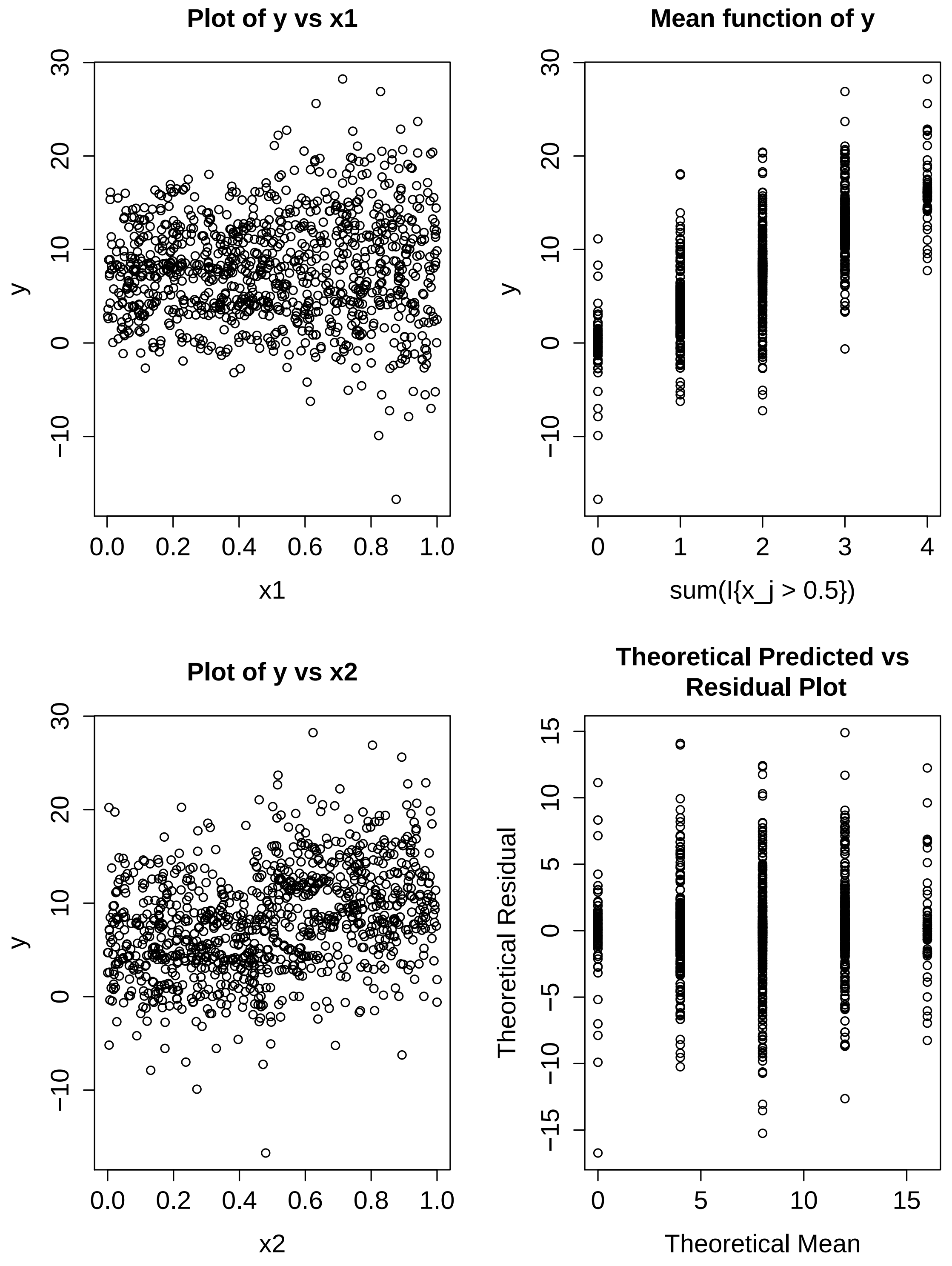


Figure 4.3: Plot of the data distribution where X_2 through X_5 are related to the mean and X_1 is related to the variance. The top left panel shows the heteroskedastic relationship between X_1 and y. The bottom left panel shows the relationship between X_2 and y, and the top right panel shows the relationship between the true mean function and y. The bottom right panel shows a residual vs predicted value plot, used to assess heteroskedasticity. Here we plot as a 'residual' $y_i - \mu_i$ vs the theoretical mean, μ_i.

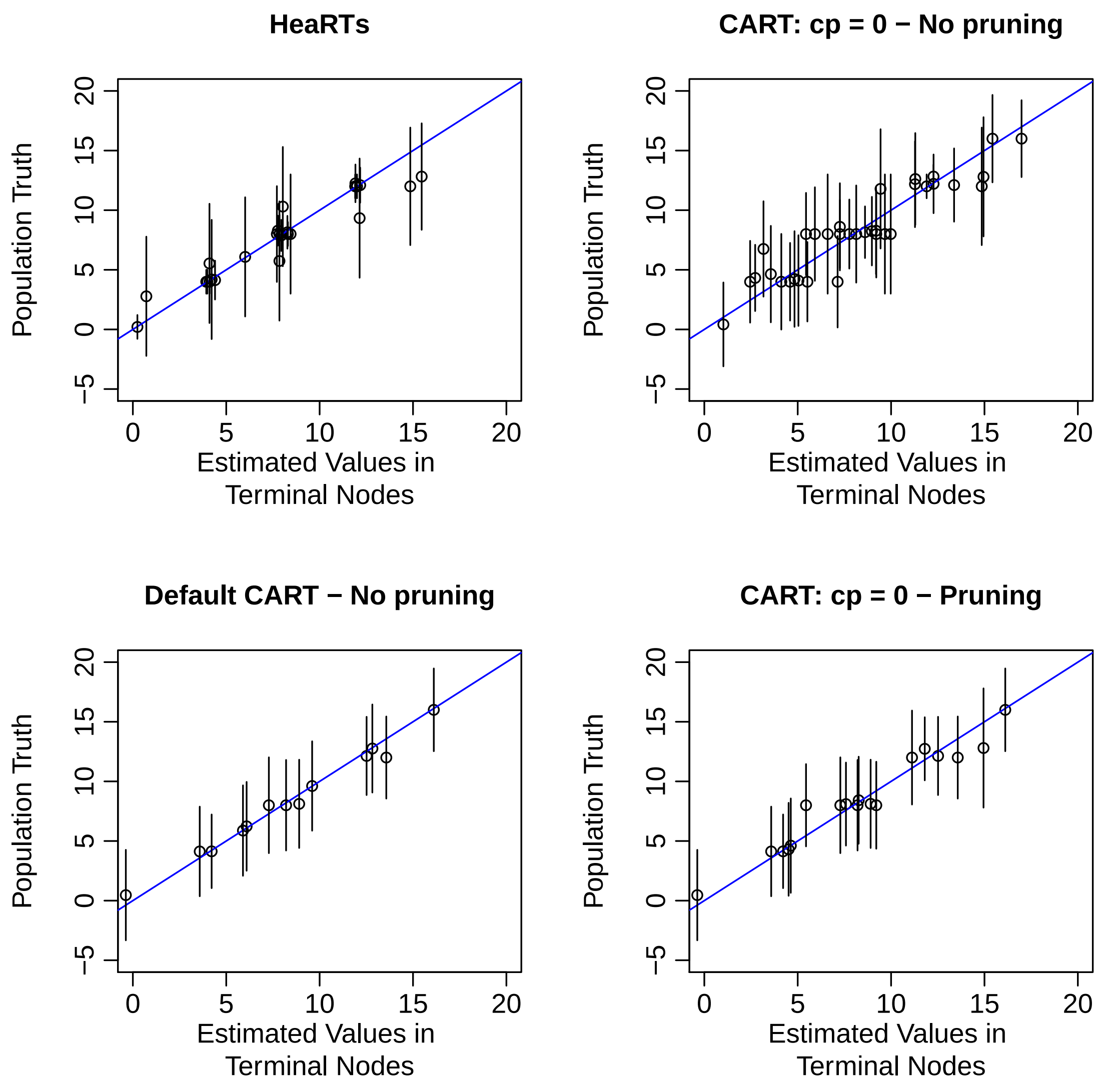


Figure 4.4: Population Parameters vs Terminal Nodes, plotted for four setups. The x-axis represents distinct terminal nodes in a tree. The y-axis represents the average true mean in the terminal node: e.g. if a node contains the subset $0.47 < X_2 < 0.51$, then 75% of the y's have a true mean of 6 and 25% of the y's have a true mean of 10, so the displayed average is 7. The vertical bars at each point represent ± 1 true standard deviation, again taking a weighted average of variances if the terminal node crosses the boundary $X_1 = 0.5$

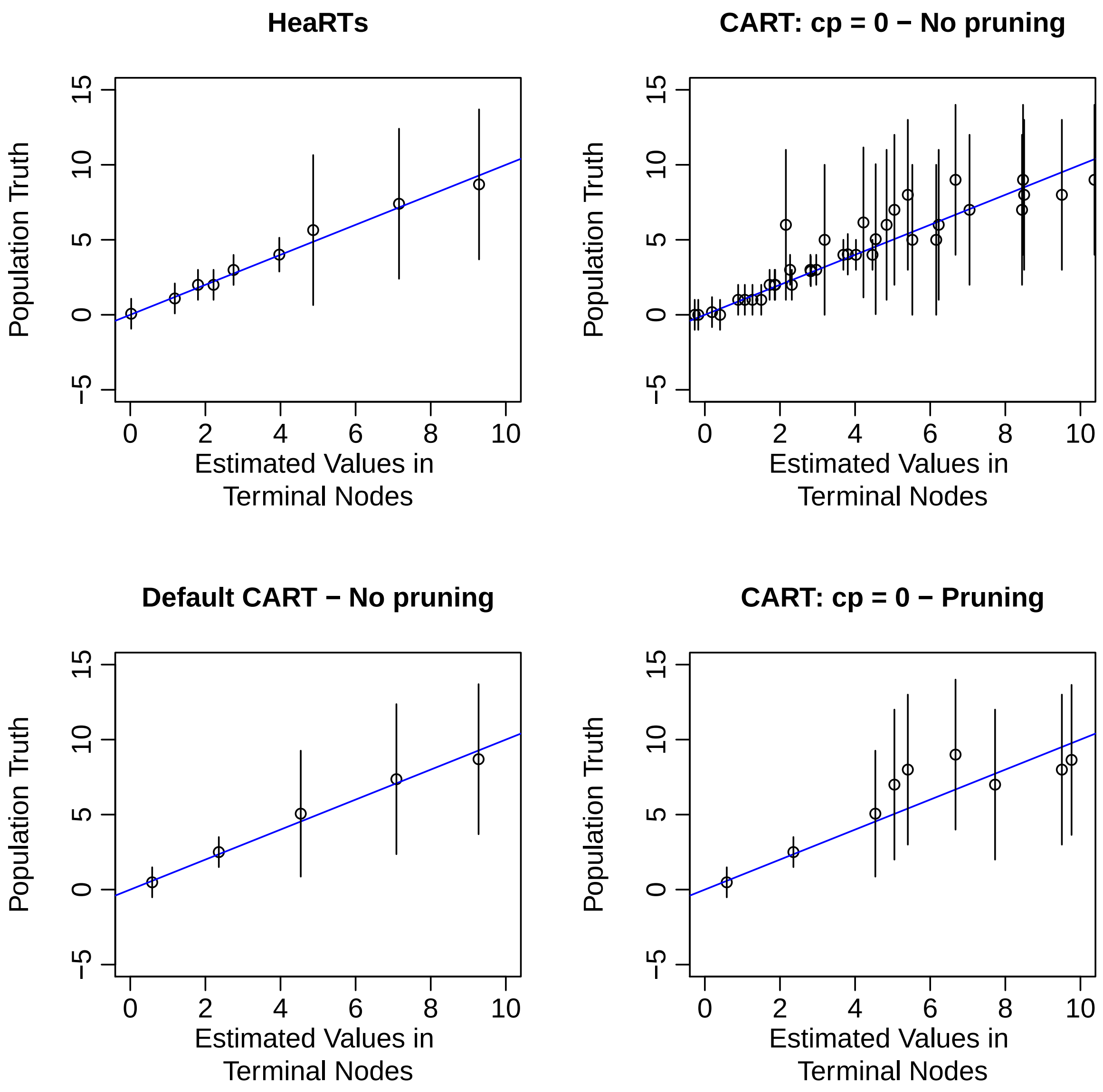


Figure 4.5: Population Parameters vs Terminal Nodes, plotted for four setups under the univariate design of Ruth and Loughin. The horizontal axis represents distinct terminal nodes in a tree. The vertical axis represents the average true mean in the terminal node: e.g. if a node contains the subset $0.17 < X_1 < 0.21$, 75% of the y's have a true mean of 1, and 25% of the y's have a true mean of 2 - the displayed average is 1.75. The vertical bars at each point represent $+/-$ 1 true standard deviation, again taking a weighted average of variances if the terminal node crosses the boundary $X_1 = 0.5$

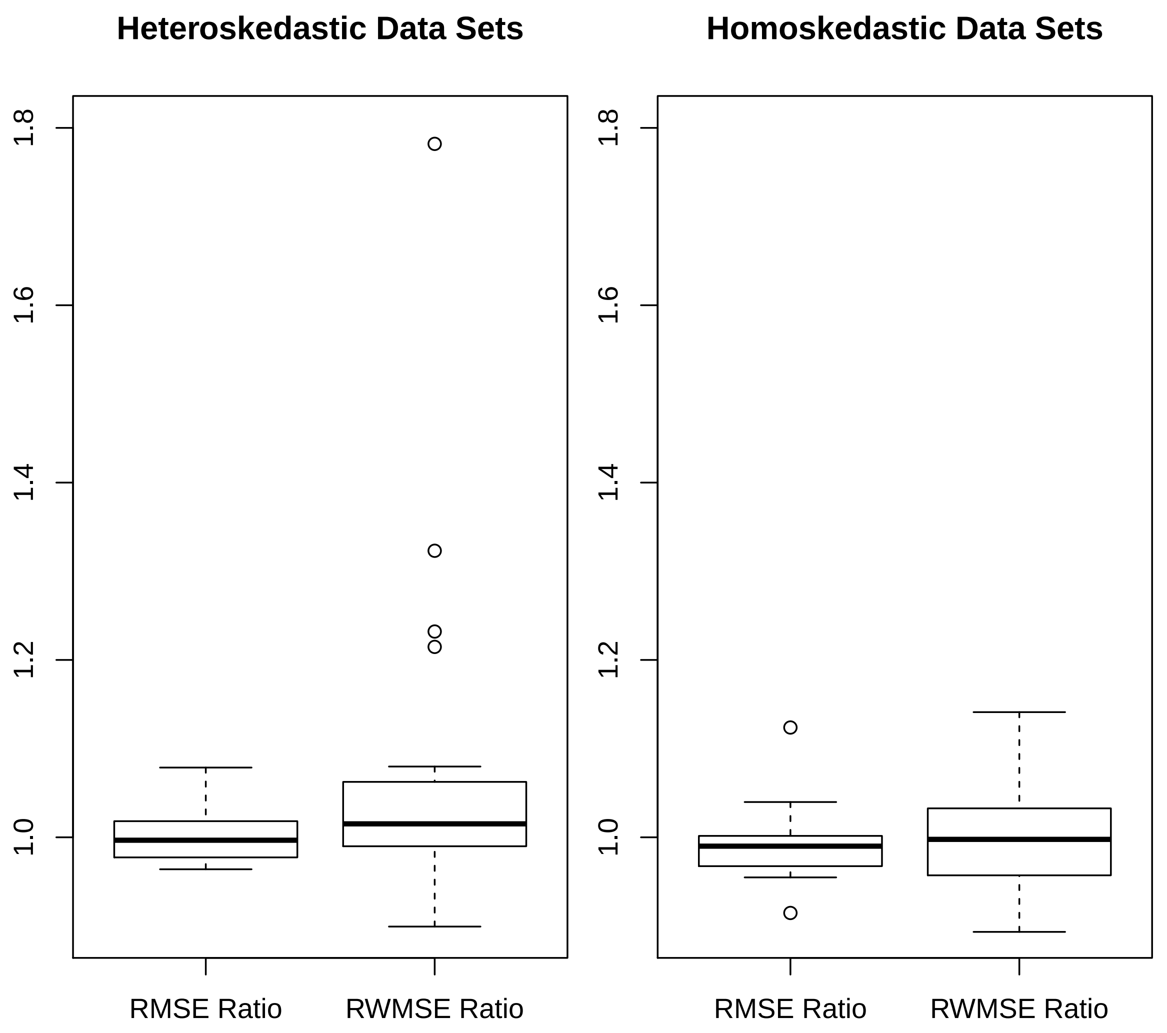


Figure 4.6: 42 Real Data Sets from Chipman et al. Ratios computed are based on averages of 50 sets of 90/10 subsamples. Ratios greater than 1 are in favour of HeaRTs. On the homoskedastic datasets, RPAB and HeaRTs perform similarly. On heteroskedastic datasets, again the RMSE is similar, but HeaRTs improves upon the RWMSE of RPAB by a mean of 6.7% and median of 1.5%. Note that HeaRTs does not appear to give itself an advantage in a RWMSE sense by estimating the variance and then using it in the RWMSE formula.

Chapter 5

Jar of HeaRTs : An ensemble of Heteroskedastic Regression Trees

5.1 Introduction

Ensemble methods have been around at least since bagging [8], boosting [26], and arcing [10]. Roughly speaking, an ensemble method aggregates the predictions (via an average or weighted average) of a set of models, rather than relying on a single model. A popular ensemble method is the random forest [9]. In the context of regression, random forests are an aggregation of regression trees [11]. The regression trees are trained on bootstrap samples of the data and are trained on random subsets of predictors in each split in order to de-correlate their individual predictions. There are many versions of the random forest, including a notable Bayesian variant called Bayesian Additive Regression Trees (BART, [20]).

Although random forests have been improved and studied in many ways, there is a notable gap in the study of heteroskedasticity with random forests. Recently, BART has received some attention in this area, including the work of Bleich [6] who incorporated a variance model into BART. There is currently no analogue for random forests.

We demonstrate how we combine the base learner HeaRTs into an ensemble, which we call the "Jar of HeaRTs". The key feature is that HeaRTs naturally models variance and consequently the ensemble is also able to model variance. We show some situations where the variance modeling is helpful compared to default random forests. We also show that default random forests perform poorly on flat and elbow functions, and we introduce a technique we call $alpha-$pruning that improves the fit.

5.2 Relevant Literature

Typical regression trees work by recursively splitting data into two groups, and predicting the response using just the data inside the group. The method was popularized by Breiman et al. [11], who proposed splitting the data using the Sum of Squared Errors (SSE) criterion. We refer to the regression trees of Breiman as Recursive PArtitioning of Breiman et al. (RPAB). There have been many developments on the metrics used to split the trees. Su [62] developed maximum likelihood regression trees, showing that splits using the likelihood are equivalent to SSE splits, although they prune the tree differently. Regression trees, although interpretable, are considered to be unstable predictors [9] in the sense that if the data are perturbed slightly, a completely different estimated mean structure is proposed. Approaches like bagging [8] are intended to overcome this variability.

Random forests were first proposed by Breiman [9]. Random forests are an improvement upon the bagging procedure. The additional observation is that the improvement obtained by averaging over many trees is directly related to the correlation of the trees. The lower correlation, the better. An illustration of this idea is given in [34]. Consider a set of K identically distributed, but correlated random variables with variance σ and positive pairwise correlation ρ. The analogue of this is using an average K trees created from bootstrap samples of the data. They give the variance of the average of the K random variables as $\rho\sigma^2 + \frac{1-\rho}{K}\sigma^2$. The term $\frac{1-\rho}{K}\sigma^2$ can always be reduced close to zero with a large enough K. The relevant term is $\rho\sigma^2$. By keeping ρ small, very good predictive performance is attainable.

It is therefore desirable to de-correlate the predictions from individual trees. To accomplish this the regression trees fit to the data are different from classic trees in two ways. The first change is that each tree is grown on its own bootstrap sample of the data, instead of the original data, just like bagging. The second change is that each tree can only use a random subset of the predictor variables at a given split. The size of this subset is a tuning parameter called r. The subset given to each split is chosen uniformly at random from all $\binom{p}{r}$ possible combinations.

Random Forests possess some unique attributes that make them desirable as a prediction method. First, as each tree is grown on a bootstrap sample of the data, each tree has an implicit holdout set called the "out-of-bag" sample, consisting of approximately 37% of the original sample. This built-in test set provides a direct measure of mean squared prediction error, called the “out-of-bag error”. Breiman proves that this error estimate is unbiased for the test error. As a consequence, the test error can be estimated without needing either cross validation or a holdout set.

Finally, random forests consist of an aggregation of a set of non-interacting trees. The relevant consequence of this is that the trees can each be built on a separate core/machine and results aggregated together with minimal effort. The procedure is “embarrassingly parallel", that is, using C cores results in a speedup of a factor of roughly C (with minimal overhead). An example of a procedure which is not embarrassingly parallel is boosting, which works by re-weighting the residuals from the prediction of the first $K-1$ trees to create the Kth tree. Since this algorithm is sequential in nature, there is no advantage to using multiple cores.

Breiman's complete forest building algorithm is as follows:

- For each of K trees, grow a regression tree on a bootstrap sample of the data.
- When growing a regression tree, at every node, we pick a random subset of size r from the p predictors.
- Find the optimal split on these r predictors and make the split
- Stop making splits on any node with no greater than m responses inside the node
- Apply no pruning to the trees
- Once all the trees are built, the predicted responses are obtained by averaging the predictions from each individual tree

The intuition behind the approach is that the trees are built to have highly variable but unbiased prediction of the mean surface. By averaging several trees together, the variability is lowered and the unbiasedness remains. The effectiveness of the approach depends on the correlation between the trees. If the prediction errors from the trees are highly correlated, averaging over many of them does little to improve variability. Breiman writes that with enough trees, the prediction error of the forest is no more than ρ times the prediction error of an individual tree (where ρ denotes the average correlation of the trees).

The `randomForest` package in R is a common tool for fitting random forests. In this package there are several ways to control the sizes of the individual trees. One method is through the tuning parameter called "minimum node size", which we call m. The tuning parameter m is defined as the smallest node size where we can possibly make a split. Another method to control tree size is through the "maximum nodes" parameter, which we call $|T_0|$. This parameter controls a forward stopping rule, which causes a tree to cease to make splits after reaching $|T_0|$ terminal nodes. It is evident from the source code that the splits created by the `randomForest` are created in the 'Breadth first' paradigm. That is to say, the tree splits in levels. Define the level of the root node to be 0, and the level of a child is the level of the parent $+1$. Every level is fully split before considering splits at the next level. As a consequence, these trees with early stopping are balanced (they may not, however, be optimal in the minimum SSE sense).

In our work we refer to a 'default' random forest, which we mean to be the default settings of the implementation of `randomForest`. In particular, the package uses a default setting of $m = 5$, $r = \lfloor p/3 \rfloor$, and $K = 500$. To re-iterate an interesting point of the previous paragraph, this random forest implementation routinely splits off some terminal nodes of size 1 under the default settings.

5.3 Problems with random forests on flat functions

It is usually recommended to prune individual trees when they are used as predictors ([11], [56], many others). Random Forests, however, are typically created as an average of unpruned trees. Tuning the size of the minimum nodes in a random forest is usually thought to have relatively low effect (e.g., [34]). Random Forests do, however, exhibit a bizarre pathology: default random forests are terrible at predicting very flat surfaces. In fact, if the response surface is relatively flat, random forests can do much worse than RPAB.

To demonstrate this, we show a setup where:

$$X_1 ... X_5 \overset{iid}{\sim} U(0,1)$$

$$Y \sim N(0,1). \tag{5.1}$$

We fit three models to training sets of size 1000: default random forest, pruned RPAB, and default linear regression. We then compute the Root Mean Square Error (RMSE) on a test set of size 1000. For observation i in the test set, each model computes a predicted mean $\hat{\mu}_i$. In this setup the test set mean is always 0, so the RMSE is calculated using:

$$RMSE = \frac{1}{1000} \sum_{i=1}^{i=n} {\mu_i}^2$$

Finally, we average the predictions over 50 runs. The RMSE's are tabulated in Table 5.1.

Table 5.1: Performance of random forest for estimating means from a flat function. Numbers are averages over 50 simulations from model (5.1), and have standard error ≤ 0.01.

Method	RMSE
Random Forest	1.04
RPAB - Full Split + Pruning	0.13
Linear Regression	0.35

Clearly this is not the quality of performance we expect from a random forest! Intuitively, it is surprising that an average of regression trees is so much **worse** than a single regression tree. The problem is the random forest are too variable. It is worth noting that the number of trees is not the problem here. The problem is that the trees are grown too large, so that the flat surface is being predicted by lots of means from small samples. Furthermore because the trees are still somewhat correlated, the variance of the average prediction remains high. One attempt to fix this is to tune the minimum node size parameter m. The results of tuning m are given in Figure 5.1. We see large improvement by tuning m on this flat function.

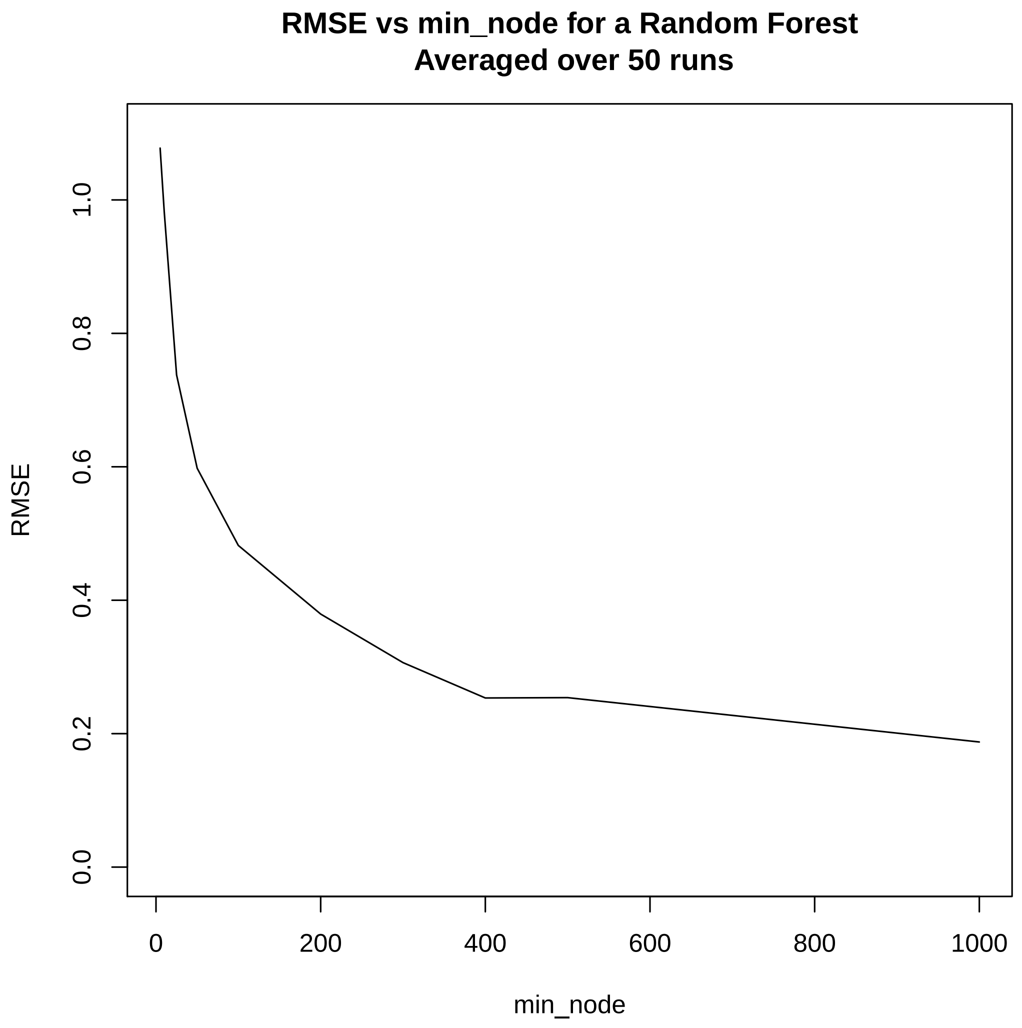


Figure 5.1: Plot of RMSE vs nodesize under a model with no real effects. Smaller trees (larger nodesize) gives significantly better performance, contrary to the usual claims in the literature. Even with $nodesize = n$, the random forests still makes splits and has a RMSE (0.18) worse than a tree from RPAB (0.13).

This behaviour can also be found by tuning the other optional parameter given in the `randomForest` package, called 'maxNodes'. We show the results of tuning the maxNodes parameter in Figure 5.2.

Clearly flatness is a problem for random forests. We now try to determine to what extent flatness is a problem. To vary the flatness we simulate using the truth as a linear

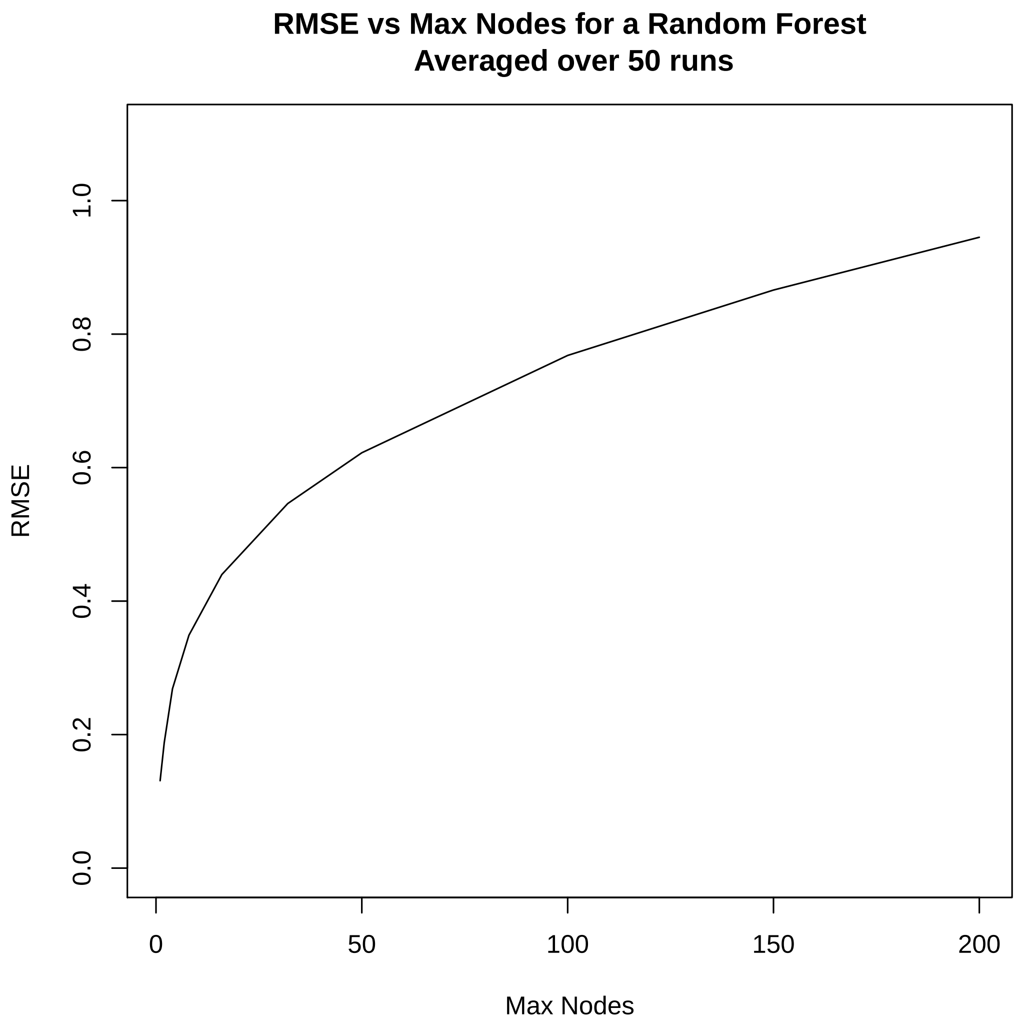


Figure 5.2: Plot of RMSE vs Max Nodes under a model with no real effects. Smaller trees (smaller maxNodes) gives significantly better performance, contrary to the usual claims in the literature. Setting $maxNodes = 1$ is the only way to attain the same RMSE performance (0.13) as RPAB.

model with varying slope. The idea is to try to find the point at which the default m setting (of 5) is acceptable.

$$X_1 ... X_5 \overset{iid}{\sim} U(0,1)$$

$$Y \sim N(X\beta, 1)$$

We set $\beta = \{c, c, c, c, c\}^T$ for some values of c, and for each value of c we determine the optimal nodesize. We plot the relationship between slope and average optimal nodesize in

Figure 5.3, and also show the relationship between optimal nodesize and the theoretical R^2 (the theoretical R^2 is the average R^2 when fitting the true, linear model). We note that on low signal-to-noise relationships, it appears useful to tune the nodesize to be larger than the default (5), and when the signal-to-noise ratio is very high, a smaller nodesize is preferable.

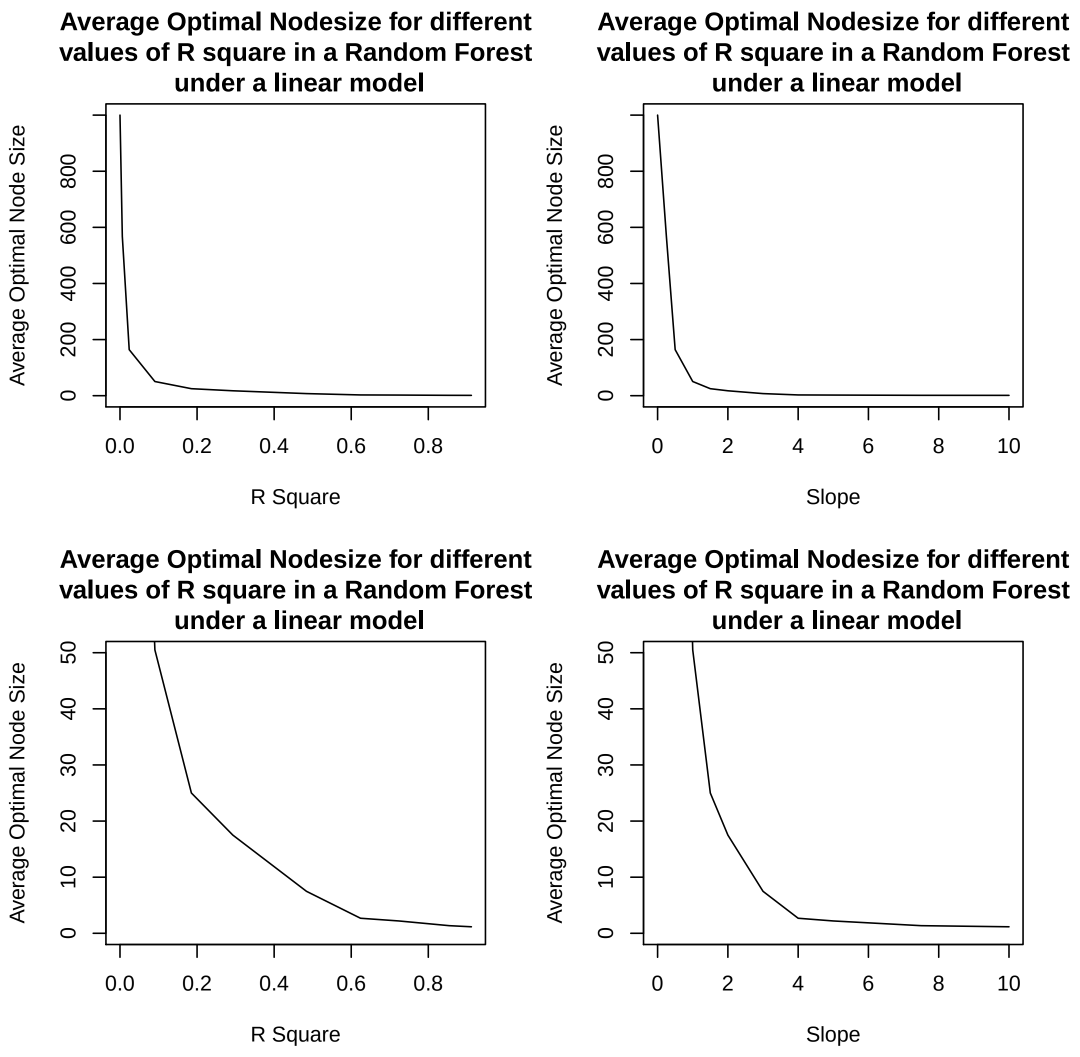


Figure 5.3: The plots on the left hand side show the relationship between average optimal nodesize and the true R^2 under a true linear model. Plots on the right hand side show the relationship between average optimal nodesize and slope. Top plots are identical to their respective bottom plot, the only difference is the y-axis. Top plots are shown with $y \in [0, 1000]$, bottom plots are 'zoomed in' to $y \in [0, 50]$. We see that the default value of nodesize (5) relies on the signal-to-noise ratio (as measured by R^2) being appreciably higher than 0.

The tuning parameter m controls a particular bias-variance tradeoff in a random forest. Averaging over more observations in the terminal nodes leads to mean estimates with lower variability , but the mean estimates can be biased if the mean function is non-constant across the observations in the terminal nodes. Averaging over fewer observations in the terminal nodes allows us to capture a rapidly changing mean function, but the variability of the terminal nodes is higher.

Tuning m, furthermore, can be a blunt instrument. The problem is that there is no particular guarantee that the true mean function exhibits the same amount of flatness across the entire space of X. An example to illustrate this is an elbow function which contains both a section of flatness and a section with some positive slope. Setting m to be small results in low bias in the sloped portion of the surface, but excessive variability in the flat portion. Conversely, setting m large results in low variability in the flat portion but excessive bias in the sloped portion.

We show the attempts of a random forest to fit an elbow function on the following setup:

$$X_1 ... X_5 \overset{iid}{\sim} U(0,1)$$

$$Y \sim N(10(max(0.5, X_1) - 0.5), 1).$$

We fit random forests with varying m to these datasets. Two example plots are shown in Figure 5.4 and Figure 5.5. It turns out that there is no particularly satisfactory value of m for this function—the random forest has either excess variance in the low end (Figure 5.4) or excess bias in the high end (Figure 5.5). We later demonstrate a new pruning approach that fixes this issue.

5.4 Constructing a Jar

We follow the same approach as Breiman [9] when constructing our ensemble of HeaRTs (Chapter 4). We call this ensemble the Jar of HeaRTs. Breiman's original random forest models only the mean, not the variance, and as such we make some small modifications to our algorithm to incorporate this. Furthermore we add in a step to improve performance on flat functions.

- For each of K trees, we grow a regression tree on a 50% subsample of the data. We have not found subsampling to provide different performance compared to bootstrapping, and subsampling seems to be a preferred default [59].

- When growing a regression tree, at every node, we pick a random subset of size r from the p predictors.

- We find optimal split among the candidate predictors, using all 3 different types of split (Mean, Mean Variance, and Variance).

- We pick the best split using the CHIC criterion.

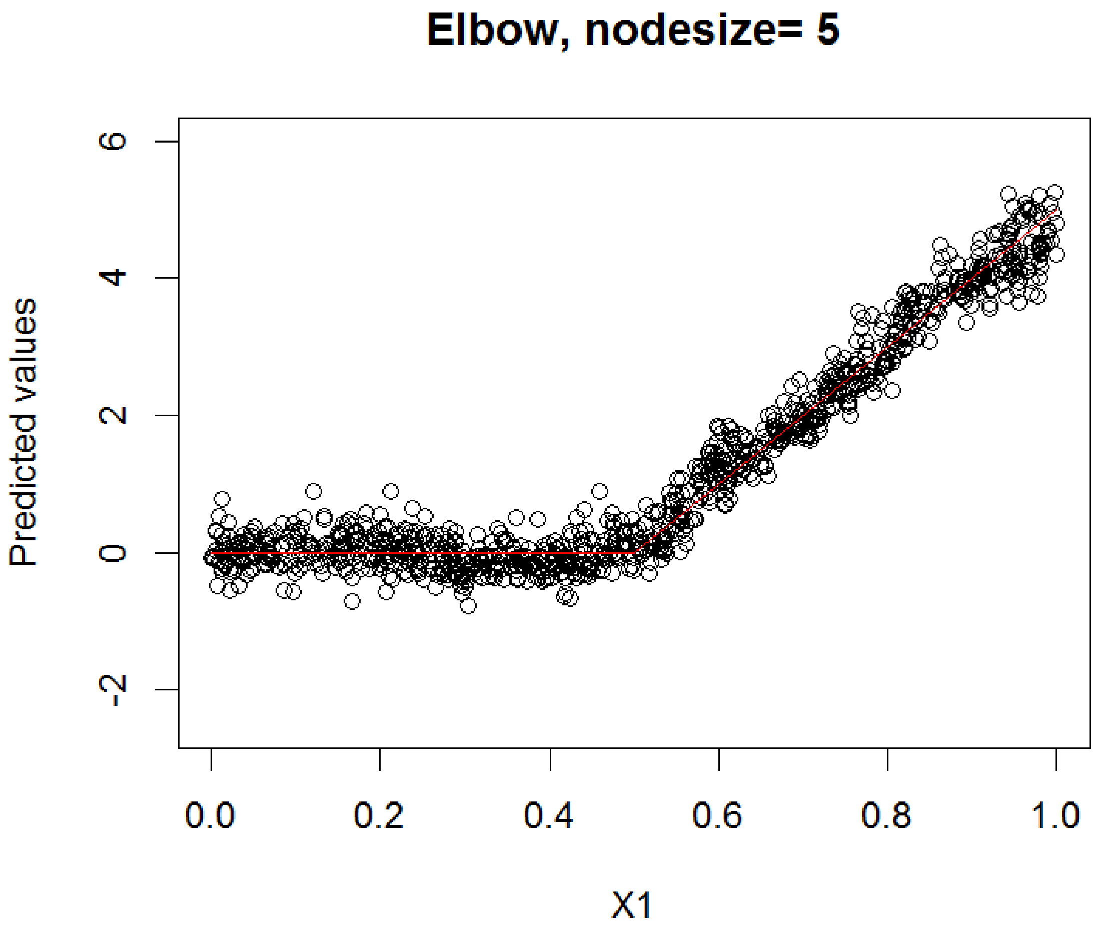


Figure 5.4: A plot of the predicted values by a random forest with $m = 5$, as a function of X_1, on the elbow function. The true mean function is drawn in red. The problem with these predictions is that there is excess variability. The estimates of the mean are relatively unbiased here: compare to Figure 5.5

- After constructing all the trees, we re-estimate the final parameters in the Jar of HeaRTs (more details found in Section 5.6).
- We apply a pruning technique we call $\alpha-$pruning (more details found in Section 5.5) to help improve the fit on flat functions. Each time we apply pruning we re-estimate the final parameters of the Jar.

If we turn off the ability to make variance and mean-variance splits in our method, and do not use the $\alpha-$pruning technique, we are left with a standard random forest. One additional detail in our Jar implementation is that we do not want to make variance or mean-variance splits and be left with one observation in a terminal node - this would make it impossible

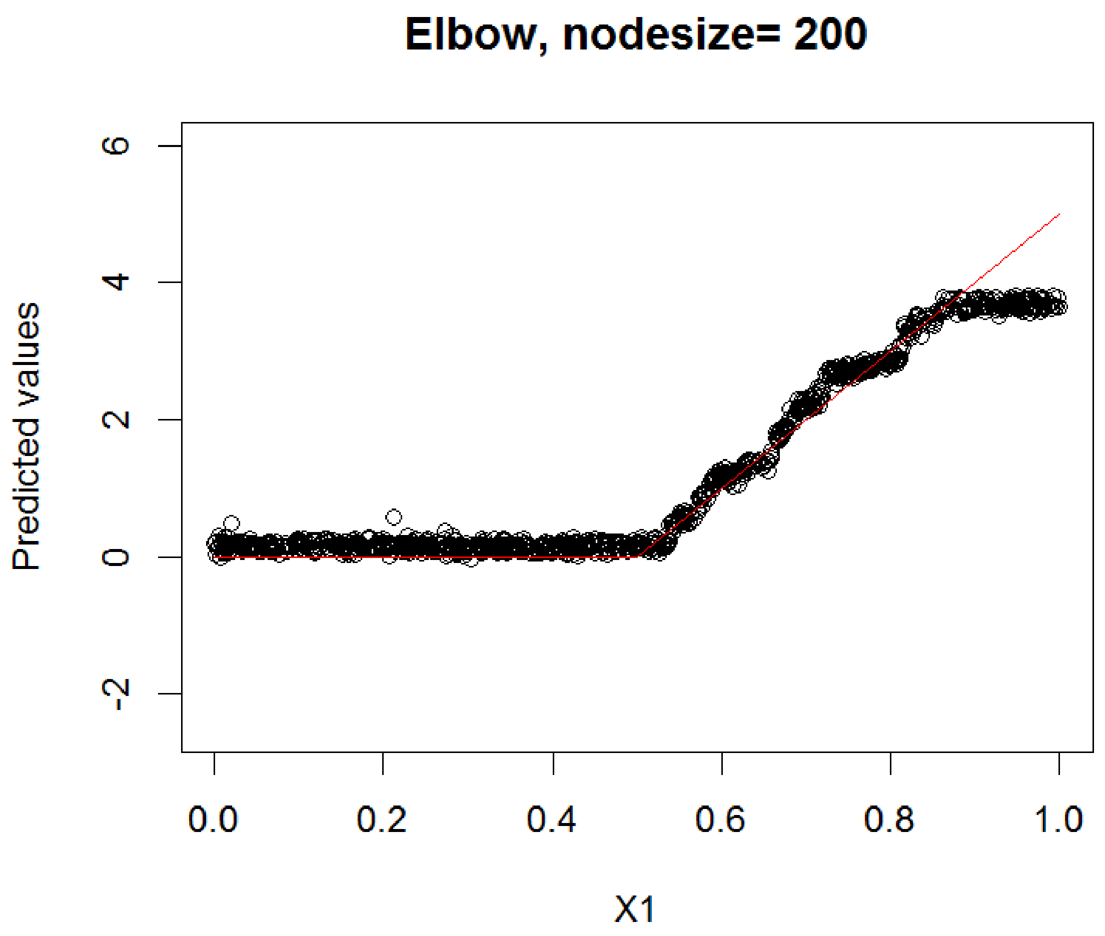


Figure 5.5: A plot of the predicted values by a random forest with $m = 200$, as a function of X_1, on the elbow function. The true mean function is drawn in red. The problem with these predictions is that the estimates of the true mean are biased (too high in the low end, and too low at the high end). The variability of these estimates is much lower than Figure 5.4

to optimize the likelihood. We impose a hard constraint that any split involving a variance must have at least 7 observations in both terminal nodes. If the node contains fewer than 14 observations, only mean splits are considered. The minimum size to split on the mean is the same tuning parameter as in the original forest algorithm.

5.5 $\alpha-$Pruning

Now that we have demonstrated the need for flexible node sizes in a random forest, we introduce the $\alpha-$pruning algorithm on the Jar of HeaRTs. This form of pruning seeks to automatically tune the trees to the best possible size, as required within the context of the entire ensemble. The rough approach is to prune the HeaRTs using the CHIC penalties, which is a very fast approach (requiring only one pass through the data). However, pruning using the computed CHIC penalties for a split is usually going to excessively prune. This is because the penalties are computed under the assumption that only one tree was being used to estimate the mean, rather than the average of many trees. Thus, the penalties balance the bias-variance tradeoff differently from what is needed within an ensemble, where it is better to take on some extra variability in exchange for eliminating bias. Consequently, we typically want to use a lower penalty, allowing trees to be larger and less biased than usual.

Our solution to this is to relax the pruning criterion that is normally used by HeaRTs. Recall from Chapter 4 that for each internal node we fit a model to the parent node and compare the model to the model found by the best split. The model fit at the parent node is a normal model with a single mean and variance, $N(\mu, \sigma^2)$. This is compared to the best model found by the tree, which is a mean split, a variance split, or a mean variance split. The model for the children is also a normal model with either two means and one variance, in case of a mean split, one mean and two variances if the split is on variance, or two means and two variances if the split is on both.

Initially we obtain the maximized likelihood of each model on the data in the node, and this is the initial value of the 'information'. We refer to the information of the left child as I_L, the right child's information is I_R, and the parent's information is I_P. Finally, the IC penalty value for the model fit at the parent node is B_P, and the IC penalty value for the model fit to the children is B_S.

The information is a biased estimate of the true quality of the model on test data. The model for the parent node has no split, so penalty is given by the AICc (we use the version that does not multiply the log-likelihoods by 2):

$$B_P = p + \frac{p(p+1)}{n_T - p - 1}.$$

The penalty for the children is the relevant CHIC table evaluated at n_T and p, found in 4.5,4.7, or 4.7. We denote this penalty as $B_S = CHIC[n_T, p]$.

The general approach is to retain a split if:

$$I_P - B_P < I_L + I_R - B_S$$

and otherwise, we prune the split. When we retain a split, the intuition is that the parent's true value lies in its children. We therefore replace the parent's information value I_P with the information of the children. In particular, we set:

$$I_{Pnew} = max(I_{Pold}, I_L + I_R - B_S + B_P).$$

We apply this procedure recursively starting from the bottom of the tree and working up until all splits have been evaluated and either pruned or retained.

However, when creating a forest of trees, it is preferable to have higher variance, low bias trees (see e.g.,[34]). Our approach of finding the optimal bias-variance tradeoff for a single tree using CHIC produces trees with a mixture of bias and variance. Since the ensemble will reduce some of the variance (but not bias), it is preferable to grow larger trees. This means that the CHIC values need to be reduced somewhat. The $\alpha-$pruning approach is to retain a split if:

$$I_P - \alpha B_P < I_L + I_R - \alpha B_S,$$

where $\alpha \in [0, 1]$ is a tuning parameter. Note that setting $\alpha = 0$ uses full trees with no pruning, and setting $\alpha = 1$ gives trees using the regular CHIC penalties. The updating equation for I_{Pnew} is now:

$$I_{Pnew} = max(I_{Pold}, I_L + I_R - \alpha(B_S + B_P)).$$

It is natural to assume that by adding a tuning parameter the algorithm will be slower. However with the right approach, it is easy to either try a few values of α (or of m) with no additional cost. The idea is to prune each tree at a fixed value of α and find the α that minimizes a test-set error rate. First, recall that Breiman et al. showed that the OOB error is unbiased for the test set error. Analogously, we use the OOB likelihood as our error rate - the OOB likelihood is an unbiased estimate of the test-set likelihood. Secondly, consider the cost of building the forest. Building a single tree is known to have cost of $O(np \log_2 n)$ (see, e.g. [68]), and we build K trees, so the total cost of building a forest is $O(Knp \log_2 n)$. Now, we compute how long it takes to evaluate the OOB error. For one observation, we evaluate the error on $O(K)$ trees (usually about 37% of them) at an average height of $O(log_2 n)$ per tree, so the total cost to evaluate the OOB error of all n observations is $O(nK \log_2 n)$. Pruning the trees using the CHIC criterion takes less time: for each split made, the CHIC criterion does an $O(1)$ calculation, and thus prunes in $O(n)$ time per tree for a total pruning time of $O(nK)$.

The consequence of this is that the expensive part of tuning α or m is the computation of the OOB error. We have shown that the cost of building the tree is $Knp \log_2 n$ and the cost of evaluating the OOB error is $nK \log_2 n$. As long as we don't evaluate too many different values of α or m, the runtime will not be increased substantially. As an illustration, evaluating p different values of alpha would result in approximately twice the runtime.

The current implementation of `randomForest` in R does not allow for m to be tuned as efficiently. Since the forest has to be re-fit with a new m every time to tune, the implementation suffers a slowdown of a factor equal to the number of different values of m that are tried. However, with a good implementation, it is possible to change the m to any higher value $m*$ at no additional cost. We tune m using the OOB error. To compute the OOB error, for every observation in the training data we drop the observation through every OOB tree until it reaches the corresponding terminal node. To change m to a new value $m*$ in this procedure, the only necessary change to this algorithm is to terminate on any node that contains no more than $m*$ observations. Thus in the same vein as $\alpha-$pruning, m can be tuned without needing to resort to crossvalidation. What is required is to simply use a smart 'predict' function that can take m as a parameter.

5.6 Estimating Final Parameters of the Jar

The random forest algorithm constructs the final estimates by estimating means in the individual trees and then aggregating means across the trees. Our Jar of HeaRTs seeks to do the same, but the process is a little more complicated. To begin, the naive estimates for the means and variances in the Jar of HeaRTs are unsatisfactory. The naive mean estimator, which takes a simple unweighted average of the individual predictions from each tree, does not use the variance information and is inefficient. The naive estimate of the variance is also poor. Large trees naturally tend to overfit the data, and as a consequence we would under-estimate the variance.

Therefore, we cannot simply average the means and variances across all K trees' terminal nodes for a given x. In particular, individual terminal nodes are not meant to provide the best possible estimates of the respective means, because the trees are built using suboptimal choices of variables and not pruned to an optimal level for prediction from single trees. Therefore, the variances in these terminal nodes contain both the natural error variance of the process and the error from mis-estimating the means.

Our solution relies on using a different residual to estimate the global variance. Terminal node variances in a given tree are based on squared 'local' residuals, where local residuals are differences between an observation and the mean estimate in its terminal node. Local residuals change from tree to tree, so an observation's contribution to the variance also changes from tree to tree. We define instead a 'global residual', which is the difference between an observation and its OOB mean. We base our variance estimation on the global residual.

We deconstruct the parameters in the Jar into four separate components. The general idea is that we will fix three of them and estimate the fourth conditional on the other 3, in the same vein as the Feasible Generalized Least Squares (FGLS,[55]) approach to optimizing variance. The four groups are:

- Mean models within individual trees (each tree will have several different mean models)
- OOB means for each data point in the training data

- Variance models within individual trees (each tree will have several different variance models)
- Variance estimates for each data point in the training data

We initialize the algorithm with all means at 0 and all variances at 1. We then iterate through the following steps.

5.6.1 Part 1 - Estimation of means in trees

Consider the first tree, T_1. T_1 has some mean splits, some variance splits, and some mean-variance splits. We use the painting algorithm on the tree (Chapter 4) to determine which means should be estimated together. This splits the data into a set of G groups, with sizes $n_1, n_2 ... n_G$. In group g, the first goal is to estimate a single mean to model all n_g observations that fall into the(se) terminal node(s)—recall that the painting algorithm may determine that several terminal nodes share a common mean.

Consider the an arbitrary group of nodes that share the same mean. For each observation we use the current estimate of ${\sigma_i}^2$ and corresponding precision $\tau_i = \frac{1}{{\sigma_i}^2}$, and the estimator of the mean is given by:

$$\hat{\mu} = \frac{\sum\limits_{t=1}^{n_i} y_t \hat{\tau}_t}{\sum\limits_{t=1}^{n_i} \hat{\tau}_t}$$

We repeat this for all G groups to estimate the means of the first tree. We repeat the same algorithm to estimate the means in the mean groups for all K trees. This is the same as the first step of the HeaRTs algorithm (see Chapter 4).

5.6.2 Part 2 - Estimation of global means and residuals

We will estimate the OOB mean for each of the n observations, denoted as $\tilde{\mu}_i$. The point of doing this is to construct unbiased estimates of the residuals, which will be used to estimate variances. We continue to use the current estimates of σ_i and τ_i.

Consider the first observation. For each of the j trees where this observation falls out of bag we find the estimates of the mean and precision for each tree. For example, the mean estimates created in step 1 of this algorithm will be what we use as the mean estimate for the trees. We will call the means $\hat{\mu}_1, ...\hat{\mu}_j$ and precisions $\hat{\tau}_1, ...\hat{\tau}_j$.

The estimate for the OOB mean is:

$$\tilde{\mu}_i = \frac{\sum\limits_{t=1}^{j} \mu_t \hat{\tau}_t}{\sum\limits_{t=1}^{j} \hat{\tau}_t}$$

And the residual, r_i , is:

$$r_i = y_i - \tilde{\mu}_i$$

5.6.3 Part 3 - Estimation of tree variance

To estimate the variance, ideally we would know the true mean model. Since this is impossible to obtain, the next best thing is to use the model predicted means. In the random forest setting, we can get an approximately unbiased estimate of the mean for each data point by using the OOB mean estimate [9]. For each tree, the variance model is estimated as an average of the squared global residuals.

Consider the first tree T_1, which has some mean splits, some variance splits, and some mean-variance splits. We use the painting algorithm to determine which variances should be estimated together. This splits the data into a set of G groups, with sizes $n_1, n_2...n_g$. For each of these G groups, the goal is to estimate a single variance to model all n_g observations that fall into the(se) terminal node(s).

Consider the first group of observations. Each observation has an estimated global residual r_t. The estimate of the variance is therefore:

$$\hat{\sigma}^2 = \frac{\sum\limits_{t=1}^{n_i} {r_t}^2}{n_i}$$

We estimate each tree variance in this way, for every group within a tree and for every tree in the Jar of HeaRTs.

5.6.4 Part 4 - Estimation of data variances

We now focus on estimating the variance of a particular observation in the data set. By averaging the estimated variances from the trees where the data fall out of bag, we get an estimate of the variance of each datapoint. The main use of this is in step 1, when we estimate the individual tree means.

Consider the first observation. For each of the j trees where this observation falls out of bag we find the estimate of the variance and also the number of observations that contributed to the estimate. We call these $\sigma_1...\sigma_j$ and $n_1...n_j$.

The overall estimate of variance for a point is given by:

$$\sigma^2 = \frac{\sum\limits_{i=1}^{i=j} {\sigma_i}^2 n_i}{\sum\limits_{i=1}^{i=j} n_i}$$

Finally, we repeat steps $1-4$ until convergence. We use a stopping rule that each mean and variance estimate must have an absolute or relative change no more than 10^{-6} to stop the algorithm. Generally this takes very few iterations (fewer than 20) to accomplish. Convergence is guaranteed with the usual FGLS properties [55].

It is worth mentioning that this estimation procedure adds negligibly to the runtime of the Jar of HeaRTs. Estimating one pass of the means and variances is done in a loop that only considers each observation once during each phase. The runtime is $O(n)$ per iteration, and the number of iterations is quite small.

5.7 Simulated Examples

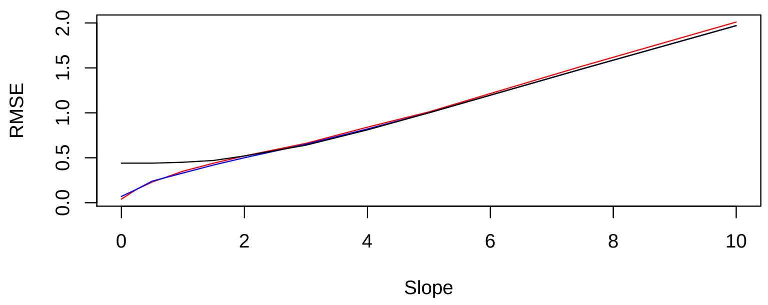
We now turn to see how well the $\alpha-$pruning algorithm with the Jar solves the nodesize problems that the random forest has. Furthermore, we show a setup where we attain superior mean estimation by additionally modeling the variance.

We show three simulation setups. First, we demonstrate empirically that the $\alpha-$pruning algorithm does as good a job as nodesize tuning on linear functions. Next, we demonstrate the flexibility of $\alpha-$pruning to model the elbow function. The $\alpha-$pruning algorithm removes splits in the flat section but keeps splits in the section with a slope. Finally, we show the ability of the Jar to model variance, and in turn achieve superior mean estimation on a setup where the variance is unrelated to the mean.

Figure 5.6 shows the performance of 3 algorithms (Default random forests, random forests with nodesize tuning, and Jar of HeaRTs with no variance splits) on the linear setup with varying slopes (see Equation (5.1)). The variance splits in the Jar are turned off so that we can see how the pruning approach performs by itself. We see that the $\alpha-$pruning of the Jar more or less mimics the nodesize tuning done for a random forest. Both are preferable to the random forest default when the mean function is flat. We see an appreciable difference in RMSE when the R^2 is less than 0.1. It appears that for datasets with sizable signal-to-noise, the default nodesize is a reasonable choice.

In our second setup, we explore more closely how the Jar and the random forest perform on the elbow function. First, we test several values of nodesize on the random forest, and also the $\alpha-$pruned Jar at the optimal value of α. Then we compute the RMSE on test data. Finally, we average the results over 50 trials. The results are tabulated in Table 5.3. For the random forest, a nodesize of 50 is found to be the best balance of mean and variance, with a RMSE of 0.19. The Jar, however, improves upon this by about 10%, with a RMSE of 0.17.

To see why the Jar can outperform nodesize tuning on the elbow, we look at the individual splits that were made in each tree. We bin each split into one of ten categories - a split can be made on any variable X_1 to X_5, and it can be made either above 0.5 or below. We then tabulate across all the trees which splits were made. The methods are: a



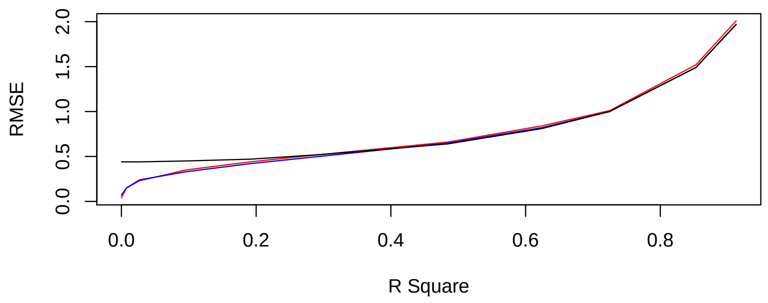


Figure 5.6: Plot of RMSE on a linear model by 3 methods. Top plot shows RMSE vs Slope and bottom plot shows theoretical RMSE vs R-square. The black curve represents a default random forest. The blue curve is the random forest with nodesize tuning. The red curve is the Jar with $\alpha-$pruning.

Table 5.2: Performance of the random forest on the elbow function for various values of nodesize

Node Size	RMSE
5	0.28
10	0.27
20	0.22
50	0.19
100	0.22
200	0.31
300	0.45
400	0.62
500	0.61
1000	0.88

default forest (with nodesize 5), a forest with nodesize 20, a Forest with nodesize 50, and a $\alpha-$pruned Jar with initial nodesize 5. Recall that only splits in the region $X_1 > 0.5$ are useful splits. It is worth noting that the Jar initially had the same distribution of splits as a default forest, but the $\alpha-$ pruning mechanism removes some splits. The split distributions are found in table 5.3.

Table 5.3: Split distribution for four methods on the Elbow Function, fit with $r = 1$ (recall that r is the number of randomly selected predictors for each split). Only X_1 is active, and only splits where $X_1 > 0.5$ are pertinent to the true mean function.

Default forest			$\alpha-$pruned Jar		
Predictor	Split < 0.5	Split > 0.5	Predictor	Split < 0.5	Split > 0.5
0	4410	6140	0	195	2266
1	4362	3882	1	331	289
2	4475	3892	2	328	285
3	4034	4093	3	289	268
4	4352	4014	4	329	256
Random forest - 20			Random forest - 50		
Predictor	Split < 0.5	Split > 0.5	Predictor	Split < 0.5	Split > 0.5
0	1405	2798	0	617	1583
1	1208	1374	1	384	375
2	910	1115	2	422	553
3	1154	1166	3	390	503
4	1131	1153	4	337	437

There are two main insights gleaned from this table. First, we compare the default forest to the $\alpha-$pruned Jar. In the areas where there are no true splits (all categories except the top right bin), $\alpha-$pruning retains just 6% of the splits. In contrast, $\alpha-$pruning retains 37% of the splits in the meaningful quadrant. This has improved the proportion of useful splits from 14% to 47%, a substantial increase. The random forest with nodesize 20 has 21% useful splits, and the random forest with nodesize 50 has 28% useful splits. Also worth observing is that the $\alpha-$pruned Jar is strictly more efficient than the optimal random forest (with nodesize 50)—the pruned Jar contains fewer irrelevant splits and more useful splits. The upshot is that the Jar of HeaRTs allows larger nodesizes on the flat surface and smaller nodesizes on the sloped surface.

In our final simulated example, we demonstrate the efficiency of modeling the variance for improving mean estimation. In our toy example, the true variance model is unrelated to the true mean model. The full model is:

$$X_1 ... X_5 \overset{iid}{\sim} U(0,1)$$

$$Y_i \sim X_{i2} + X_{i3} + X_{i4} + X_{i5} + \epsilon_i$$

$$\epsilon_i = \begin{cases} N(0,1), X_{i1} < 0.5 \\ N(0,6), X_{i1} > 0.5 \end{cases}$$

To separate the idea of $\alpha-$pruning from variance estimation, we directly compare our method (using pruning) with variance splits turned on to our method with variance splits turned off (also using pruning). The metrics (RMSE and RWMSE, see Chapter 4) are tabulated in Table 5.4. An illustration of the Jar's variance model for X_1 is given in Figure 5.7. The upshot of this toy simulation is that we can further improve mean estimation (over and above α-pruning or m tuning) on relatively flat mean functions (the theoretical R-square of this setup is 0.17) by modeling the variance. The mean estimation is improved because the Jar of HeaRTs uses a weighted mean instead of an unweighted mean.

Table 5.4: Comparing the Jar of HeaRTs with and without Variance splits. $\alpha-$pruning is used on both. Results are averages over 50 datasets.

	RMSE	RWMSE
Variance Splits	0.429	0.307
No Variance Splits	0.499	0.360

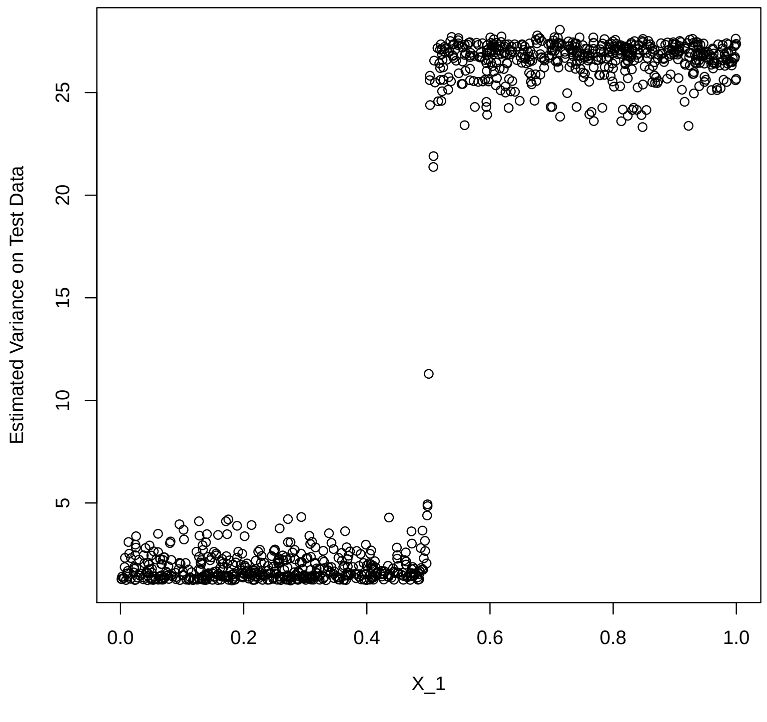


Figure 5.7: Plot of estimated variance on test data vs X_1. The true model has a variance of 1 for $X_1 < 0.5$ and a variance of 25 for $X_1 > 0.5$. The Jar is able to estimate the variance to a reasonable degree. As a consequence it has superior mean estimation.

5.8 Real Data Examples

We compare our Jar of HeaRTs procedure and the random forest on the same 42 datasets as in Chipman et al. [20]. The datasets are regression problems, with n varying from 96 to 6906. Categorical predictor variables are coded as individual 0/1 columns. The regressions have between 3 and 28 continuous variables and 0 to 6 categorical variables. Gelfand [30] shows that many of these datasets are heteroskedastic. An important difference between our study and that of Chipman et al. is that we do **not** use a variance-stabilizing transformation of the response y.

To assess the quality of the fit on a given data set, we split the data 90%/10%, constructing a training set and test set. We fit and evaluate three methods. We consider a default random forest, a Jar of HeaRTs with $\alpha-$pruning but no variance modeling (no variance or mean-variance splits), and finally the full Jar of HeaRTs. Each model is fit on the training data and the predictions are evaluated on the test data. To reduce the variability due to the randomness of the splitting procedure, the data are resplit 50 times and the metrics (RMSE and RWMSE) are averaged over all 50 resplits.

For each metric on each dataset we identify the best performing method of the three. We then compute the ratio of each method's metric against the best, resulting in a value that is necessarily at least 1. We then present these summary ratios as 6 boxplots in Figure 5.8. We see a slight improvement in RMSE by $\alpha-$pruning the random forest. In terms of RWMSE, we see that the Jar of HeaRTs is the best performer.

We point out that the methods (random forest, $\alpha-$pruned Forest, and the Jar of HeaRTs) all share the same asymptotic runtime. As a consequence, it seems reasonable to choose among the methods based strictly on performance. For datasets where heteroskedasticity is known or thought to exist, we recommend the use of the Jar of HeaRTs. Even if heteroskedasticity turns out to be negligible, the Jar provides reasonable mean estimation. If there is significant heteroskedasticity, the Jar rates to be the best method. If heteroskedasticity is known to not exist, it is possible to remove the variance model from the Jar for slightly improved mean estimation.

5.9 Conclusion

Although random forests are a popular and very good ensemble method, they are not flawless. We demonstrate that random forests perform poorly on flat functions with low signal-to-noise ratio. This can be rectified by tuning the nodesize parameter m, a tuning parameter that is often ignored. On functions containing both flatness and signal, we show that tuning m is not enough. Our solution is the novel $\alpha-$pruning method, which essentially allows larger nodesizes on the flat surface and smaller nodesizes on the sloped surface.

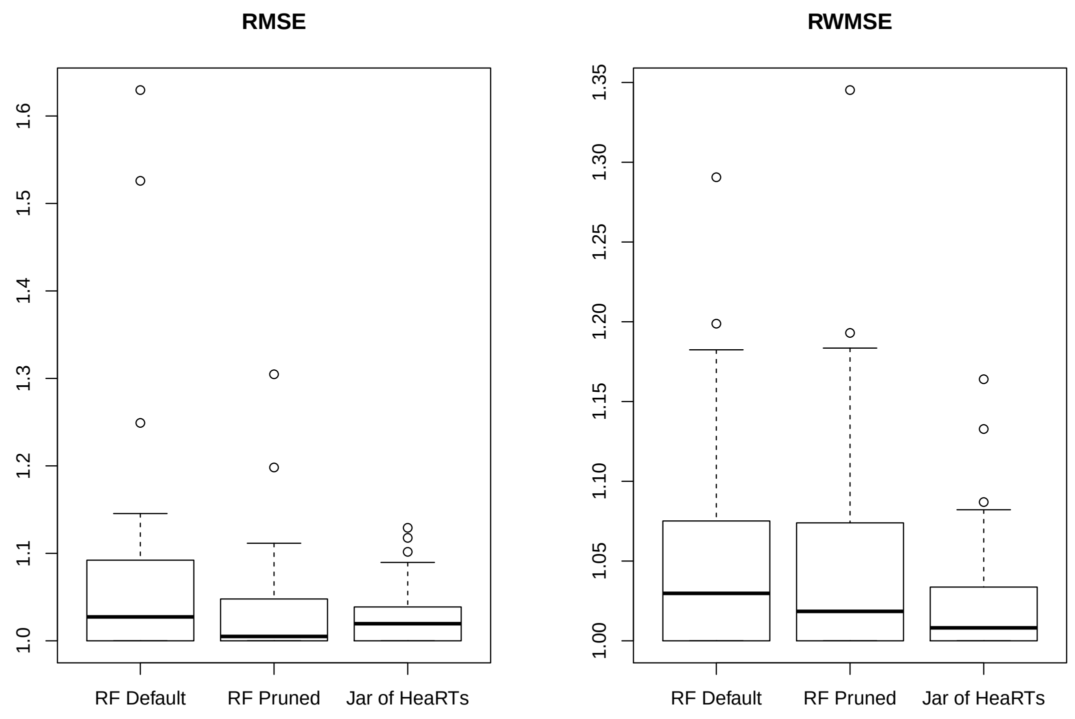


Figure 5.8: Boxplots of random forest, Pruned random forest, and Jar of HeaRTs on the Chipman bakeoff data. Left pane shows the performance of the methods in terms of RMSE. We see that applying $\alpha-$pruning to the random forest tends to be a slight improvement over the default. On the right pane, when measuring RWMSE, we see that the full Jar of HeaRTs is the best performing method.

Random Forests are based on regression trees, which implicitly assume homoskedasticity. It is plausible that we can achieve superior mean estimation on heteroskedastic data with a model that explicitly accounts for heteroskedasticity. We extend the HeaRTs procedure to an random forest style ensemble method which we call the Jar of HeaRTs. By modeling the variance, we show that we can improve the mean estimation when the variance is unrelated to the mean. We test our two improvements to the random forest on simulated and real datasets, and show that a small performance gain is achievable.

Although our focus is on modeling the mean, the Jar of HeaRTs could potentially be used in other contexts. The Jar of HeaRTs provides explicit estimates of the variance, which can be useful in some contexts. Variance estimates could be used to construct predictive intervals for new observations, as was done in Bleich's work [6]. When modeling heteroskedastic data, it seems reasonable to expect that the Jar of HeaRTs would give better predictive intervals than the random forest model. In Chapter 3 we discussed models where one of the objectives of the modeling process was to understand the relationship between the predictors and the variance of the response. The Jar of HeaRTs can address this also, using the variable importance ideas from Breiman's work [9] and the additional work by Strobl [60]. Finally the Jar of HeaRTs can potentially be used as a method for detecting heteroskedasticity in datasets. By fitting a Jar of HeaRTs model to a dataset and looking at the number of variance or mean-variance splits made across the forest, this produces a nonparametric heteroskedasticity test.

Chapter 6

Conclusion and Closing Thoughts

Throughout this dissertation, the goal has been to demonstrate the applicability of the CHIC paradigm to variance modeling under a few different contexts. In short, the paradigm is that we approach a joint mean-and-variance modeling problem as a model selection problem. The problem is that choosing between different variance models when also faced with small n can be challenging. Our approach is to tabulate a series of penalty values for each model and fit joint mean-variance models using maximum likelihood. The penalty values serve as an approximation to the bias incurred when the maximized log-likelihood is used to estimate its expectation. The corrected heteroskedastic information criterion (CHIC), which is the maximized log-likelihood $-$ the simulated penalty, is used to choose between multiple models. Because CHIC depends on the unknown values of the model parameters, our procedure computes penalties under a common null model. We find empirically that this simplification seems to be reasonable.

In Chapter 3, we specifically looked at the unreplicated factorial problem, typically of size 2^4. In this case, we were able to enumerate all the models, evaluate them using CHIC, and optionally, averaged the results. We found our ESMA-CHIC algorithm to provide considerably better results than a recommended method in the literature. When the design grows in size (e.g. a 2^5 design or larger), we show that the number of models grows super-exponentially, and evaluating the models via brute force is infeasible. We resort to a Genetic Algorithm which searches a random subspace of models, evaluating CHIC penalties on the fly (we can no longer tabulate a long table of penalties - the model space is too big). We show that our GA does a reasonable job of finding the models found by our exhaustive search. The code is available online as an easily runable script in R, at:
`http://amstat.tandfonline.com/doi/suppl/10.1080/00401706.2015.1114024?scroll=top`

In Chapter 4 we switch gears to the modern statistical learning methods and the recursive partitioning framework. Ruth and Loughin [57] demonstrated that the Recursive Partitioning of Breiman et al. (RPAB) can behave unsatisfactorily when the predictors are related to both the mean and the variance. We further show an issue with RPAB under a model where the mean and variance are unrelated. To solve this, we develop Heteroskedastic Regression Trees (HeaRTs) using the CHIC paradigm. We view the splitting and optimization procedure as a maximum likelihood problem, and develop a table of CHIC penalty values to assess the quality of a split. We also use the CHIC values to decide whether to split the model on the mean, on the variance or on both the mean and variance. We show that our trees are better able to handle heteroskedastic data, and in particular provide superior mean estimation in regions where the variance is low.

In Chapter 5 we extend the HeaRTs to an ensemble method which we call the Jar of HeaRTs. We introduce a new method, which we call $\alpha-$pruning for improving the fit of a random forest to flat functions with low signal-to-noise ratio. We also demonstrate the oft-overlooked importance of tuning the nodesize on such functions. We further show that $\alpha-$pruning is superior to nodesize tuning on an elbow-type function. Finally, we demonstrate the importance of the variance-modeling aspect of HeaRTs as a base learner for the Jar of HeaRTs ensemble. By modeling the variance, we show that we can improve the mean estimation when the variance is unrelated to the mean.

In both Chapter 4 and 5 we evaluated our method only in the context of mean estimation. The HeaRTs and Jar of HeaRTs provide a variance model, which can be used for multiple purposes. In particular, the variance model can be used to construct predictive intervals, which we expect to be more accurate on heteroskedastic datasets. Bleich [6] found that HBART was an improvement over BART for constructing predictive intervals on heteroskedastic data, and we assume that HeaRTs and the Jar of HeaRTs would also construct

We propose the CHIC paradigm as an attractive solution to model selection under heteroskedasticity. The main drawback is the need to precompute a large table of simulated values. Once the table is computed, however, CHIC can be used in the same way as any other information criterion, allowing model selection without either a holdout set or crossvalidation. As data sets become more complicated to model, and computing time is becoming a more valuable resource, being faster is surely better.

6.1 Speculation, future work, and potential improvements

In retrospect, choosing use a single tree to estimate the mean and variance may not be ideal. A difficulty that can arise is illustrated by a simple example, where the response y is related to a single predictor X_1. Suppose the mean function is constant within the following intervals in X_1 but different between them, e.g. $E(y|X_1 \in \{0, 0.5\}) = 0$ and $E(y|X_1 \in \{0.5, 1.0\}) = 10$. Furthermore the variance model is: $\sigma^2|X_1 \in \{0, 0.7\} = 1$ and $\sigma^2|X_1 \in \{0.7, 1.0\} = 25$. It is reasonable to split on the variance at or near $X_1 = 0.7$. After making this split, we travel to the left branch, and in the low variance area we need a mean split at $X_1 = 0.5$. What's the problem?

The problem is that the estimate of the mean in the range $X_1 \in \{0.7, 1.0\}$ is actually relatively poor, as it is estimated only from data with relatively high variance. The means in the range $\{0, 0.5\}$ and $\{0.5, 0.7\}$ are estimated from low-variance data and rate to be relatively precise. The problem is the tree has estimated two separate means that are actually estimating the same quantity without recombining them. We saw this in Chapter 4, where HeaRTs sacrificed RMSE in the high variance area in exchange for better RWMSE. We suspect that this is the cause. In this example, the estimation of the mean where $X_1 \in \{0.7, 1.0\}$ is in the high variance area, and consequently the RWMSE is not affected much. However, the RMSE deteriorates.

One approach for improvement might be to postprocess the tree. By looking for predicted means that are relatively similar and looking for similar values of X_{ij} across the terminal nodes, it may be possible to fix this issue by recombining some nodes and re-estimating. An alternative approach that has potential is to use two trees: a tree for means and a tree for variances. The proposed algorithm proceeds as follows. Begin by fitting the mean tree in the same way as RPAB, and prune it with CHIC. Then, fit a variance tree to the squared residuals from this model. After pruning the variance tree, throw away the original mean tree. Build a new mean tree treating the variances as fixed and known, pruning again with CHIC. By iterating back and forth between fitting a mean and a variance model, and stopping after a fixed number of iterations, it should be possible to keep the runtime within a fixed constant of the runtime of RPAB. The advantage of this approach is that the variance model and mean model are kept separate, and the issue described would not exist.

A helpful addition to the CHIC approach for HeaRTs and random forests would be to model the tables of CHIC values. Currently the table can handle n as large as 12800 and p as large as 32. However, the table currently offers no good extrapolation to larger n or p. By using a model, the table could be extended and extrapolated without the expensive upfront simulation cost. Extending this table for large datasets (Big Data) with n in the millions would almost certainly require an approximation formula for the CHIC penalty values.

Bibliography

[1] H Akaike. Information theory and an extension of the maximum likelihood principle. *Second International Symposium on Information Theory*, pages 267–281, 1973.

[2] H Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

[3] W P Alexander and S D Grimshaw. Treed regression. *Journal of Computational and Graphical Statistics*, 5(2):156–175, 1996.

[4] N S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[5] B Bergman and A Hynén. Dispersion effects from 7 unreplicated designs in the 2 k-p series. *Technometrics*, 39(2):191–198, 1997.

[6] J Bleich. *Extensions and applications of ensemble-of-trees methods in machine learning.* PhD thesis, University of Pennsylvania, 2015.

[7] G E P Box and R D Meyer. Dispersion effects from fractional designs. *Technometrics*, 28(1):19–27, 1986.

[8] L Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[9] L Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[10] L Breiman et al. Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, 26(3):801–849, 1998.

[11] L Breiman, J Friedman, R A Olshen, and C J Stone. *Classification and Regression Trees.* Chapman and Hall, New York, 1984.

[12] W A Brenneman and V N Nair. Methods for identifying dispersion effects in unreplicated factorial experiments: a critical analysis and proposed strategies. *Technometrics*, 43(4):388–405, 2001.

[13] K P Burnham and D R Anderson. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach.* Springer, 2002.

[14] K P Burnham and D R Anderson. Multimodel inference understanding aic and bic in model selection. *Sociological methods & research*, 33(2):261–304, 2004.

[15] D Bursztyn and D M Steinberg. Screening experiments for dispersion effects. In A Dean and S Lewis, editors, *Screening*, pages 21–47. Springer, 2006.

[16] R J Carroll and D Ruppert. *Transformation and weighting in regression*, volume 30. CRC Press, 1988.

[17] J E Cavanaugh. Unifying the derivations for the akaike and corrected akaike information criteria. *Statistics & Probability Letters*, 33(2):201–208, 1997.

[18] P Chaudhuri, M C Huang, W Y Loh, and R Yao. Piecewise-polynomial regression trees. *Statistica Sinica*, 4(1):143–167, 1994.

[19] H A Chipman, E I George, and R E McCulloch. Bayesian ensemble learning. *Advances in Neural Information Processing Systems*, 19:265—272, 2007.

[20] H A Chipman, E I George, and R E McCulloch. Bart: Bayesian additive regression trees. *The Annals of Applied Statistics*, pages 266–298, 2010.

[21] G Claeskens and N L Hjort. *Model selection and model averaging*, volume 330. Cambridge University Press Cambridge, 2008.

[22] R D Cook and S Weisberg. Diagnostics for heteroscedasticity in regression. *Biometrika*, 70(1):1–10, 1983.

[23] O L Davies and G E P Box. Design and analysis of industrial experiments. 1963.

[24] J Engel and A F Huele. A generalized linear modeling approach to robust design. *Technometrics*, 38(4):365–373, 1996.

[25] A J Ferrer and R Romero. Small samples estimation of dispersion effects from unreplicated data. *Communications in Statistics-Simulation and Computation*, 22(4):975–995, 1993.

[26] Y Freund and R E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[27] J H Friedman. Multivariate adaptive regression splines. *The annals of statistics*, 19:1–67, 1991.

[28] J H Friedman and P Hall. On bagging and nonlinear estimation. *Journal of statistical planning and inference*, 137(3):669–683, 2007.

[29] G M Furnival and R W Wilson. Regressions by leaps and bounds. *Technometrics*, 42(1):69–79, 2000.

[30] S Gelfand. Understanding the impact of heteroscedasticity on the predictive ability of modern regression methods. Master's thesis, Simon Fraser University, Canada, 2015.

[31] J M Grego, J F Lewis, and T A Craney. Quantile plots for mean effects in the presence of variance effects for 2^{k-p} designs. *Communications in Statistics-Simulation and Computation*, 29(4):1109–1133, 2000.

[32] M Hamada and N Balakrishnan. Analyzing unreplicated factorial experiments: a review with some new proposals. *Statistica Sinica*, pages 1–28, 1998.

[33] A C Harvey. Estimating regression models with multiplicative heteroscedasticity. *Econometrica: Journal of the Econometric Society*, pages 461–465, 1976.

[34] T Hastie, R Tibshirani, and J Friedman. *The elements of statistical learning: data mining, inference and prediction.* Springer, 2 edition, 2009.

[35] J A Hoeting, D Madigan, A E Raftery, and C T Volinsky. Bayesian model averaging: a tutorial. *Statistical science*, pages 382–401, 1999.

[36] C M Hurvich and C Tsai. Regression and time series model selection in small samples. *Biometrika*, 76(2):297–307, 1989.

[37] A Kapelner and J Bleich. bartmachine: Machine learning with bayesian additive regression trees. *arXiv preprint arXiv:1312.2171*, 2013.

[38] S Konishi and G Kitagawa. *Information criteria and statistical modeling.* Springer Science & Business Media, 2008.

[39] S Kullback and R A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[40] R V Lenth. Quick and easy analysis of unreplicated factorials. *Technometrics*, 31(4):469–473, 1989.

[41] T M Loughin. Calibration of the lenth test for unreplicated factorial designs. *Journal of Quality Technology*, 30(2):171—175, 1998.

[42] T M Loughin and C J Malone. An adaptation of the bergman-hynen test for dispersion effects in unreplicated two-level factorial designs when the location model may be incorrect. *Journal of Quality Technology*, 45(4):350—359, 2013.

[43] T M Loughin and W Noble. A permutation test for effects in an unreplicated factorial design. *Technometrics*, 39(2):180–190, 1997.

[44] T M Loughin and J E Rodríguez. Computational issues with fitting joint location/dispersion models in unreplicated 2^k factorials. *Computational Statistics & Data Analysis*, 55(1):491–497, 2011.

[45] J Neter M Kutner, C Nachtsheim and W Li. *Applied Linear Statistical Models.* McGraw-Hill Irwin, 2004.

[46] R N McGrath. Separating location and dispersion effects in unreplicated fractional factorial designs. *Journal of quality technology*, 35(3):306—316, 2003.

[47] R N McGrath and D K J Lin. Confounding of location and dispersion effects in unreplicated fractional factorials. *Journal of Quality Technology*, 33(2):129—139, 2001.

[48] R N McGrath and D K J Lin. Testing multiple dispersion effects in unreplicated fractional factorial designs. *Technometrics*, 43(4):406–414, 2001.

[49] Z Michalewicz. *Genetic algorithms+ data structures= evolution programs.* Springer, New York, 1998.

[50] D C Montgomery. Using fractional factorial designs for robust process development. *Quality Engineering*, 3(2):193–205, 1990.

[51] V N Nair and D Pregibon. Analyzing dispersion effects from replicated factorial experiments. *Technometrics*, 30(3):247–257, 1988.

[52] J A Nelder and Y Lee. Generalized linear models for the analysis of taguchi-type experiments. *Applied stochastic models and data analysis*, 7(1):107–120, 1991.

[53] G Pan. The impact of unidentified location effects on dispersion-effects identification from unreplicated factorial designs. *Technometrics*, 41(4):313–326, 1999.

[54] G Pan and W Taam. On generalized linear model method for detecting dispersion effects in unreplicated factorial designs. *Journal of Statistical Computation and Simulation*, 72(6):431–450, 2002.

[55] R W Parks. Efficient estimation of a system of regression equations when disturbances are both serially and contemporaneously correlated. *Journal of the American Statistical Association*, 62(318):500–509, 1967.

[56] J R Quinlan. C4. 5: Programs for empirical learning, 1993.

[57] W Ruth and T M Loughin. The Effect of Heteroscedasticity on Regression Trees. *arXiv:1606.05273 [stat.ML]*, June 2016.

[58] A Saha, A Havenner, and H Talpaz. Stochastic production function estimation: small sample properties of ml versus fgls. *Applied Economics*, 29(4):459–469, 1997.

[59] C Strobl, A Boulesteix, T Kneib, T Augustin, and A Zeileis. Conditional variable importance for random forests. *BMC bioinformatics*, 9(1):1, 2008.

[60] C Strobl, A Boulesteix, A Zeileis, and T Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC bioinformatics*, 8(1):1, 2007.

[61] X Su, C Tsai, and X Yan. Treed variance. *Journal of Computational and Graphical Statistics*, 15(2):356–371, 2006.

[62] X Su, M Wang, and J Fan. Maximum likelihood regression trees. *Journal of Computational and Graphical Statistics*, 13(3):586–598, 2004.

[63] N Sugiura. Further analysts of the data by akaike's information criterion and the finite corrections: Further analysts of the data by akaike's. *Communications in Statistics-Theory and Methods*, 7(1):13–26, 1978.

[64] K Takeuchi. Distribution of information statistics and criteria for adequacy of models. *Math. Sci*, 153:12–18, 1976.

[65] W E Taylor. Small sample properties of a class of two stage aitken estimators. *Econometrica: Journal of the Econometric Society*, pages 497–508, 1977.

[66] L Torgo. *Inductive learning of tree-based regression models*. PhD thesis, University of Porto, Portugal, 1999.

[67] P C Wang. Test for dispersion effects from orthogonal arrays. *Computational Statistics & Data Analysis*, 8(1):109–117, 1989.

[68] I H Witten and E Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.

[69] C F J Wu and M Hamada. Experiments: Planning. *Analysis, and Parameter Design Optimization. Wiley*, 2000.

[70] K Q Ye and M Hamada. Critical values of the lenth method for unreplicated factorial designs. *Journal of Quality Technology*, 32(1):57–66, 2000.

[71] Y Zhang. A comparison of location effect identification methods for unreplicated fractional factorials in the presence of dispersion effects. Master's thesis, Simon Fraser University, Canada, 2010.