Senior Thesis in Mathematics

Developing Inference Frameworks for Random Forests Using Bag of Little Bootstraps & Related Methods

Author:
John Bryan

Advisor:
Dr. Johanna Hardin

Submitted to Pomona College in Partial Fulfillment
of the Degree of Bachelor of Arts

May 5, 2016

Abstract

In this thesis we seek to develop statistical inference frameworks for random forests, a widely-used machine learning method. We first explore the possibility of using ideas from the Bag of Little Bootstraps (BLB) to develop a modified implementation of the random forest algorithm which could potentially produce error estimates for random forest predictions. We then explore other approaches for attempting to obtain accurate prediction intervals for new observations. We delve into the specifics of constructing such prediction intervals, first employing naive percentile intervals and assessing their performance. We then propose an alternative method for constructing prediction intervals, based on examining the response variability within the nodes of decision trees. We thoroughly define this method and characterize its performance.

Contents

Chapter 1

Introduction

The Bag of Little Bootstraps (BLB) is a relatively new methodology designed to implement bootstrapping in parallel on large datasets [6]. It uses the bootstrap method, a resampling procedure used for estimating sampling distributions that are not well-characterized by statistical theory [3]. Bootstrapping generates a distribution that closely approximates the shape of the true sampling distribution of some statistic. BLB utilizes parallel computing architectures, subsampling, and a cleverly implemented multinomial random variable to drastically reduce computational costs while preserving the appropriate level of error and statistical correctness.

Machine learning is a broadly defined field that encompasses many aspects of both computer science and statistics. There are an abundance of well-defined machine learning algorithms – both for prediction and classification – but one common critique of many of these methods is that they produce predictions that are not associated with well-defined error bounds. This characteristic stems from a number of distinct differences between machine learning and traditional statistics. In traditional statistical settings we tend to care a great deal about what assumptions are made about our model and data, as well as be concerned with the validity of such assumptions in different situations. In many instances, however, it is in fact the case that some or all such assumptions do not hold. For this reason, machine learning methods take the approach of dropping the insistence on many of these rigorous assumptions when trying to establish a model and predictions. Through dropping these assumptions, machine learning methods typically do very well at predicting. However, one consequence of this aspect of machine learning is that these methods lose the ability to draw clear relationships between the predictions produced and well-defined probability models. This results in machine learning methods not being able to easily fit into the standard inferential frameworks that are employed in traditional statistics. This artifact is why many machine learning methods are thus often unable to produce reliable confidence estimates for predictions in a straightforward and accurate manner.

One such machine learning technique is random forests. Random forests consist of large aggregations of individual decision trees, and they are used for classification and prediction. In this thesis we explore the possibility of combining ideas from BLB with random forests to develop an algorithm which implements random forests on large datasets in a computationally inexpensive manner, while also producing accurate prediction intervals for new observations.

In order to better understand how subsampling and bootstrapping (the fundamental concepts of BLB) will affect random forests once they are implemented in a BLB-like structure, we also seek to thoroughly understand the amount of prediction variability across trees and forests. We explore these aspects of the algorithms through simulation studies that seek to obtain accurate prediction intervals. We define two methods for constructing such intervals – one naive method and another newly-proposed method – and characterize their performance. Ultimately, this thesis seeks to advance the understanding of the feasibility of obtaining accurate prediction intervals for random forests. Secondarily, this thesis explores the possibility of designing a BLB-like random forest algorithm which produces accurate prediction intervals and is optimized for situations involving large data sets.

Chapter 2

Bootstrapping & Bag of Little Bootstraps

2.1 Sampling Distributions

A fundamental concept of statistics is the sampling distribution. A sampling distribution describes how some statistic varies across different sample draws from a population. Suppose we have some population distribution $\mathbf{P}$ from which we obtain data. We can consider drawing samples of size N from this population. Furthermore, for a given sample $\mathbf{X}$ we can evaluate some statistic of interest, $T(\mathbf{X})$. Suppose we could obtain all possible samples of size N that can be drawn from $\mathbf{P}$. We could then compute $T(\mathbf{X})$ for each of these samples. The distribution of these $T(\mathbf{X})$'s would be known as the sampling distribution (Figure 2.1). From this distribution we could understand how the $T(\mathbf{X})$ vary across samples of size N from population $\mathbf{P}$.

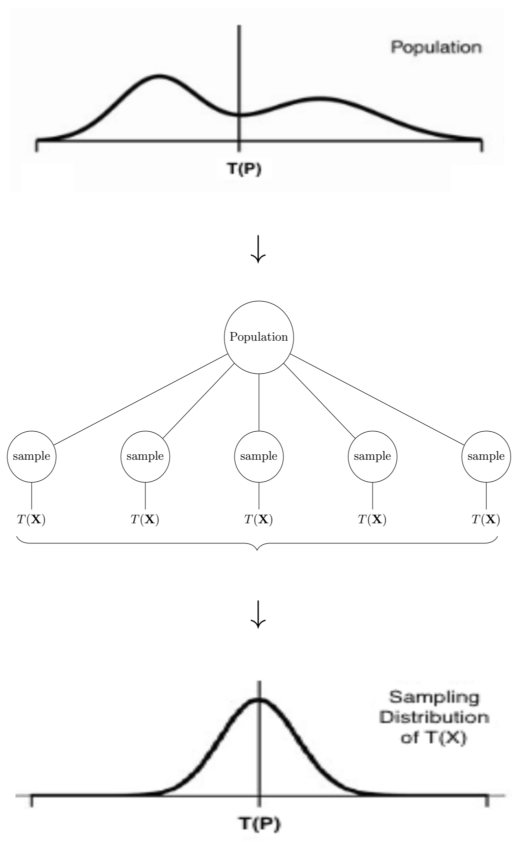


Figure 2.1: Sampling Distribution [4]

In the above figure, note how a non-Gaussian and asymmetrical population distribution produces a sampling distribution that is Gaussian and symmetrical. This result comes from the Central Limit Theorem. For many statistics, the sampling distribution will assume a shape like that seen in the figure above. However, some statistics (such as *correlation*) do not necessarily adhere to this shape, which is when bootstrapping becomes very useful.

2.2 Bootstrapping

In many instances, it is not possible to obtain the sampling distribution for a statistic of interest. This is primarily due to the calculus being intractable for certain statistics, but can also be due to not having access to all possible samples that could be drawn from our population **P**. Depending on our statistic of interest, however, it may still be possible to understand what the underlying sampling distribution looks like. In the case where our statistic of interest is the sample mean $\bar{X}$, the Central Limit Theorem provides a thorough description of the corresponding sampling distribution. However, the sampling distributions of many statistics are not supported by the same theoretical understanding. In these instances, bootstrapping becomes a very powerful tool to accurately estimate the shape and spread of the sampling distributions of such statistics.

Bootstrapping exhibits a number of useful properties. As discussed, we find that $Var(\text{sampling distribution}) \approx Var(\text{bootstrap distribution})$. In addition, we typically find that any bias (systematic shift from the true parameter θ) present in the original sample will be preserved in the bootstrap distribution. These properties are illustrated through the example below (Figure 2.2).

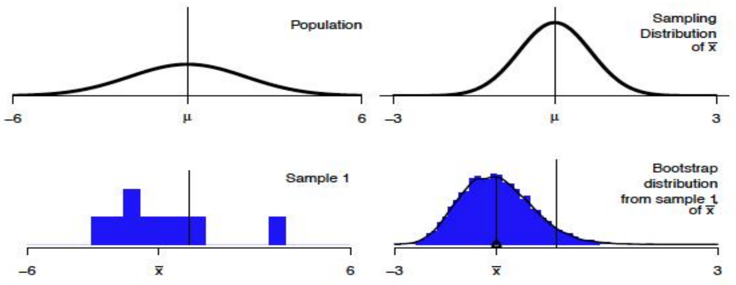


Figure 2.2: Estimating Sampling Distributions Using Bootstrapping [4]

Bootstrapping is performed by repeatedly resampling from some original sample **X**. If we were to generate the sampling distribution, we would begin by repeatedly drawing samples of size N from our population. However, in situations involving bootstrapping we do not have the luxury of repeatedly drawing samples from the population. The intuition becomes that since **X** came from our population of interest, we can treat the data distribution as the best estimate of the population distribution. So then rather than draw new samples, we repeatedly resample with replacement from $\mathbf{X}$ up to size N. It can be shown that on average there will be $0.632N$ unique observations in each bootstrap sample [6]. With each of these resamples, we compute our statistic of interest, which for each bootstrap sample is known as $T(\mathbf{X})^*$. The compilation of these realizations of $T(\mathbf{X})^*$ is known as the bootstrap distribution (Figure 2.3).

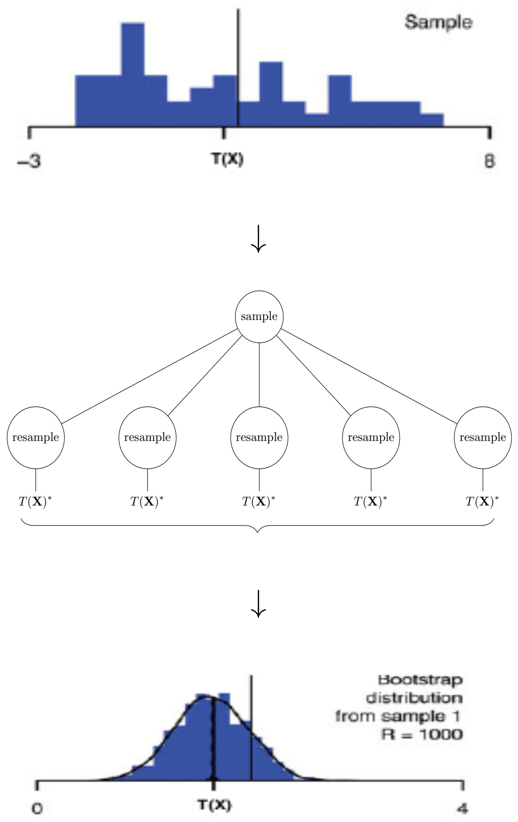


Figure 2.3: Bootstrap Distribution [4]

In the above figure, note how the bootstrap distribution is centered at the original sample statistic$T(\mathbf{X})$. This is in contrast to sampling distributions, which are centered at the population parameter $T(P)$. Once the bootstrap distribution for a statistic of interest is constructed, confidence intervals can be constructed fairly easily. Commonly employed methods in these instances include symmetric percentile intervals, t-intervals, and bootstrap-t intervals.

With percentile intervals, $(1-\alpha)$ level intervals are constructed by letting the intervals endpoints be the $\frac{\alpha}{2}$ and $1-\frac{\alpha}{2}$ quantiles of the bootstrap distribution. For t-intervals, the bootstrap distribution is used to obtain an estimate of the standard error of our statistic. The interval endpoints then become $T(X) \pm t_{\frac{\alpha}{2}} \cdot SE(T(\mathbf{X})^*)$, where $SE(T(\mathbf{X})^*)$ is the standard error estimate obtained from the bootstrap distribution. This type of interval depends on the assumption that the t distribution is an appropriate distribution from which to obtain quantiles. Bootstrap-t intervals go one step further, where they use a second level of bootstrapping to find $\frac{\alpha}{2}$ and $1-\frac{\alpha}{2}$ quantiles for the bootstrap distribution of a standardized statistic. The bootstrap-t intervals are then constructed in an analogous way to the t-intervals, but with the estimated bootstrap quantiles in place of the t quantiles. In this thesis, we exclusively employ symmetric percentile intervals in our early experiments, as well as our own proposed method of obtaining prediction intervals.

2.3 Bag of Little Bootstraps

As discussed, bootstrapping is a very powerful tool for approximating the variability of a statistic. However, in situations involving large datasets, bootstrapping can quickly become very computationally expensive. This is where the Bag of Little Bootstraps (BLB) becomes very helpful. BLB uses subsampling, parallelized computing, bootstrapping, and the clever implementation of a multinomial random variable to deliver accurate confidence estimates while exhibiting dramatically reduced computational times. In the original BLB implementation, the algorithm constructs confidence intervals for the population mean μ.

2.3.1 Subsampling and Parallelization

The algorithm first randomly selects (without replacement) S subsamples of size $B << N$. A commonly used value is $B = \sqrt{N}$ [6]. Each of these subsamples is then processed on a different parallel processor. Since each processor only has to work with data of a reduced dimension as compared to the full data set, operations are less computationally costly. Within each processor, the algorithm then generates one confidence interval for μ.

2.3.2 Bootstrapping

Within each parallel track, the algorithm bootstraps the corresponding subsample to generate a confidence interval for μ. In order to ensure that these confidence intervals exhibit the correct amount of error, the algorithm needs to work with data of size N rather than of size B. With data of size B, the eventual intervals would be too large. However, it would be counterproductive to generate bootstrap samples of size N through simply sampling N observations with replacement from the B observations in the subsample. This approach would lose the advantage of subsampling the data and would be no less computationally costly than bootstrapping with the full (non-subsampled) data set. The solution to this issue comes from the use of a multinomial random variable to generate bootstrap samples.

2.3.3 Multinomial Random Variable Implementation

In order to bootstrap one of the S subsamples of size B, the algorithm generates a multinomial random variable M that is distributed as $M \sim \text{multinomial}(N, rep(\frac{1}{B}, B))$. Realizations of M are of dimension B and sum to N. We consider the following example:

- $N = 10$
- $B = 5$
- $\mathbf{Y} = (10.2, 11, 16, 9.3, 14)$
- One realization of M: $M = (1, 3, 3, 2, 1)$.

In order to consider this as a bootstrap sample, we treat M as indicating the number of times that each observation occurs in the bootstrap sample. So for the i^{th} observation, the number of times it occurs in the bootstrap sample is the multinomial coefficient m_i, where m_i is the i^{th} element of M. So in the above example, this translates to our bootstrap sample containing:

- 1 occurrence of $y_1 = 10.2$
- 3 occurrences of $y_2 = 11$
- 3 occurrences of $y_3 = 16$

- 2 occurrences of $y_4 = 9.3$
- 1 occurrence of $y_5 = 14$

In constructing the bootstrap sample using a multinomial random variable, we "simulate" sampling to size N while only actually generating a random variable of dimension B.

Within the BLB algorithm, each parallel processor generates R bootstrap samples by drawing R different multinomial random variables. With each multinomial variable, the processor estimates the population mean μ by computing the sample means as

$$\hat{\mu} = \bar{y} = \frac{1}{N}\sum_{j=1}^{B} y_{i,j} \cdot m_j$$

- $y_{i,j}$ is the j^{th} response of the i^{th} subsample
- m_j is the corresponding multinomial coefficient of the j^{th}response

So then for our example from above $\bar{y}$ would be computed as

$$\bar{y} = \frac{1}{N}\sum_{j=1}^{B} y_{i,j} \cdot m_j = \frac{1}{10}((1\cdot 10.2)+(3\cdot 11)+(3\cdot 16)+(2\cdot 9.3)+(1\cdot 14)) = 12.38$$

2.3.4 Confidence Intervals

Within each parallel processor, we now have R realizations of $\bar{y}$. Each parallel processor constructs a single confidence interval to produce S confidence intervals for μ. These confidence intervals can be constructed through various approaches, some of which are percentile intervals, t-intervals, and bootstrap-t intervals. Once the intervals are constructed for each processor, a final interval is obtained by averaging the bounds of the S confidence intervals, or

$$(\frac{1}{S}\sum_{i=1}^{S} l_i, \frac{1}{S}\sum_{i=1}^{S} u_i)$$

The overall algorithm structure is depicted in Figure 2.4 and Algorithm 1.

2.3.5 Notable Characteristics

Through subsampling and the multinomial random variable implementation, the algorithm maintains the correct amount of error while never having to perform computations on the full data set [6]. Parallelization ensures that computational time is further reduced. BLB also has a number of convenient statistical properties. It is a consistent estimator for μ, meaning that as $N, B \to \infty$, the BLB estimator converges in probability to μ [6]. Additionally, the BLB estimator converges to the true value for μ at the same rate as the regular bootstrap, with a convergence rate bounded above by $O(\frac{1}{N})$ [6]. Ultimately, BLB is a very impactful method for effectively implementing bootstrapping on massively large data sets.

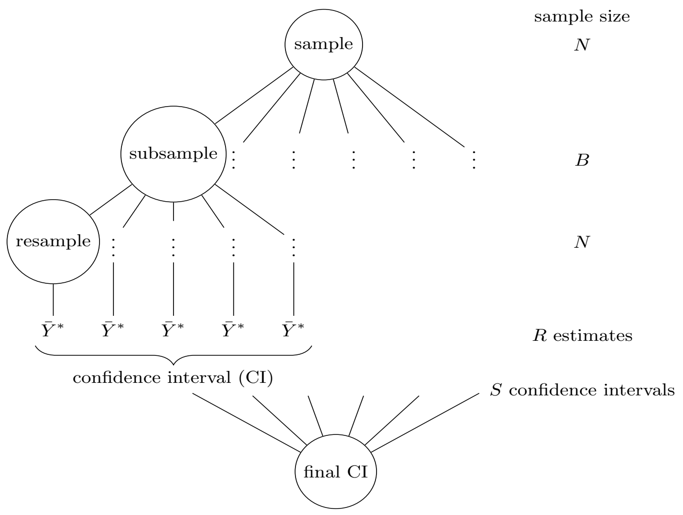


Figure 2.4: Bag of Little Bootstraps [6]

Algorithm 1 Bag of Little Bootstraps [6]

```
Inputs: $N, B, R, S$
Draw $S$ subsamples $\boldsymbol{X_1}, \boldsymbol{X_2}, \ldots, \boldsymbol{X_S}$ of size $B << N$
for $i$ in $1, \ldots, S$ (in parallel) do
    for $r$ in $1, \ldots, R$ do
        Draw $M \sim multinomial(N, \mathrm{rep}(\frac{1}{B}, B))$
        Compute $\bar{y} = \frac{1}{N} \sum_{j=1}^{B} y_{i,j} \cdot m_j$
        $y_{i,j}$ is the $j^{\text{th}}$ response of the $i^{\text{th}}$ subsample.
        $m_j$ is the corresponding multinomial coefficient.
    Obtain confidence interval for $\bar{y}$
    (our implementation of BLB uses the $\frac{\alpha}{2}$ and $1 - \frac{\alpha}{2}$ quantiles).

Average bounds of $S$ confidence intervals
return Single confidence interval for $\bar{y}$.
```

Chapter 3

Decision Trees & CART

3.1 The CART Algorithm

3.1.1 Overview

The classification and regression trees (CART) algorithm is useful for classification or regression problems [2, 5]. The algorithm can classify data or predict responses based on the explanatory components of the observations. CART can be used on data sets with either categorical and numerical explanatory variables, as well is with either categorical and numerical response variables. The algorithm is comprised of two steps, a tree-building step and a tree-pruning step. Here we describe the details of the CART algorithm.

3.1.2 Tree Building

Terminology

In order to explore the construction of classification and regression trees, we must first establish a set of terms with which to refer to different components and characteristics of decision trees. In the context of the diagram below, we define various terms that are relevant to trees (Figure 3.1).

- Root: the top node in a tree, can also refer to the top node of a subtree.
- Level: how far a node is from the root node. In the diagram below, the two daughter nodes are at level 2, and the four terminal nodes are at

level 3. In general, the level is equal to 1+the number of connections between the node of interest and the root node of the tree.

- Daughter: a node that is directly connected to a parent node, and is at a level in the tree that is one lower than the parent.
- Parent: the node that is directly connected to a daughter node and is at a level in the tree that is one higher than the daughter.
- Descendants: All nodes that can be reached from a node by repeatedly moving from parent to daughter nodes.
- Terminal node: a node that has no daughters.
- Branch/Subtree: a subcomponent of the full tree. The node closest to the top of the full tree but still contained in the branch/subtree is considered the root of the branch/subtree. The branch/subtree includes all descendants of the root node.

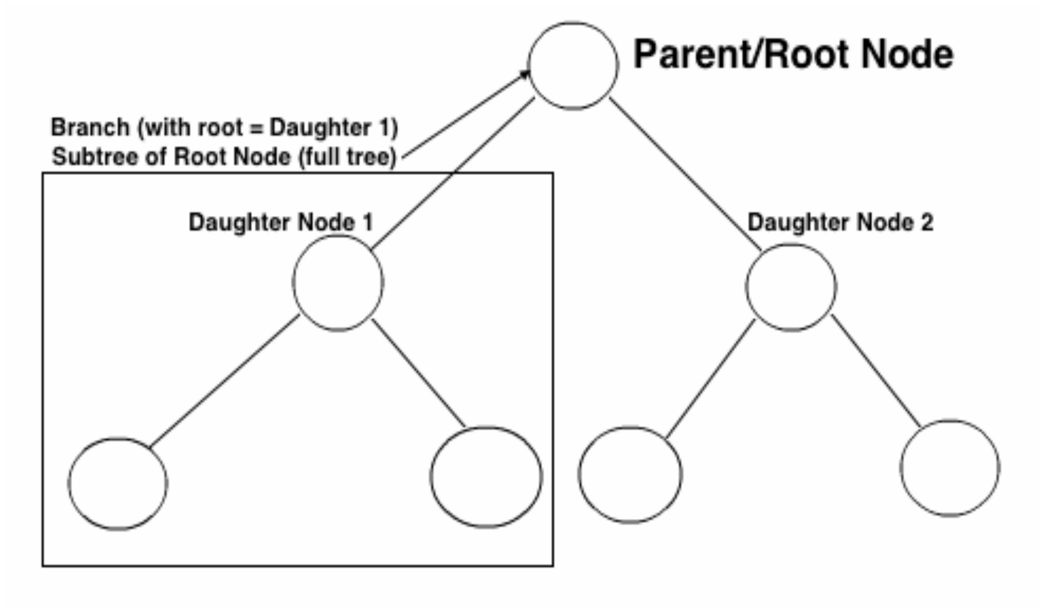

Parent/Root Node
Branch (with root = Daughter 1)
Subtree of Root Node (full tree)
Daughter Node 1
Daughter Node 2

Figure 3.1: Tree Terminology

General Aim

In the CART algorithm, we establish a series of decision nodes to sort the data into increasingly homogenous groups based on the response variable. The intuition is that if we develop nodes that are increasingly similar in their responses/classifications, we can do a better job of predicting/classifying new test data. The exact procedure for determining these nodes is different depending on whether the response variable is numerical or categorical.

Numerical Response

For data with a numerical response, we use the within sum of squares metric (WSS_t) to assess the homogeneity in the response variable for observations at node t. We compute this as

$$WSS_t = \sum_{i=1}^{N_t} (y_{i,t} - \bar{y}_t)^2$$

$$\bar{y}_t = \frac{1}{N_t} \sum_{i=1}^{N_t} y_{i,t}$$

where $\bar{y}_t$ is the average response at node t, N_t is the number of observations at node t, and $y_{i,t}$ is the response of the i^{th} response at node t.

A high WSS_t indicates that the data at that node are relatively heterogenous in their responses, while a low WSS_t indicates that they are relatively homogenous in their responses. We want to progress toward ending up with nodes that contain data which are homogenous in their response, since this characteristic will better allow us to produce predictions at the different nodes. As such, we repeatedly consider different ways that we could take the data in a certain root node – referred to as the parent node – and make a binary split of the data within one of the explanatory variables. This splitting criterion in the explanatory variables is referred to as a decision point. Such a split produces two subsets of the data that was originally found at the parent node, and each of these two subsets comprise two new daughter nodes. There are a number of possible splitting criteria that we could choose. In order to determine the best one, we seek a splitting criterion which maximizes the drop in WSS – our metric for assessing spread – between the parent node and the two daughter nodes D_1, D_2. This reduction in WSS is calculated as

$$\Delta WSS = WSS_{parent} - \sum WSS_{daughter}$$

For a numeric explanatory variable, we consider all possible cutoffs for splitting the data into two daughter nodes. So if we consider some decision point $s \in (\min(\mathbf{X}), \max(\mathbf{X}))$ for splitting the data at the parent node cutoff s for a explanatory variable x_i, we divide the data in the parent node into the two daughter nodes: $D_1 = \{\mathbf{X} | x_i < s\}, D_2 = \{\mathbf{X} | x_i \geq s\}$. The splitting cutoff s that produces the greatest reduction in WSS is determined to be the best split.

If we are dealing with a numerical explanatory variable, it becomes clear that there will be some interval, $[d_l, d_u]$, such that there are an infinite number of decision criteria that would result in equal reductions in WSS because they result in identical splitting of our training data set. The precise decision criterion is determined as the median of the two bounds of this interval. For a categorical explanatory variable, we consider all possible groupings of the different explanatory levels into the daughter nodes D_1 and D_2, and assess the reduction in WSS for each split. Similar to the case of numeric explanatory variables, the split that produces the greatest reduction in WSS is chosen as the best split.

Categorical Response

For data with a categorical response, we can no longer employ the mean response nor the WSS as meaningful statistics. In their place, CART uses the concept of impurity. The impurity assesses the homogeneity of the responses of data at a specific node. Similar to using WSS as an assessment of homogeneity, CART can compare the drop in impurity for different daughter nodes. Different possible daughter node groupings of the data are created in an identical fashion as that discussed above for the numerical response case. We introduce the Gini index as a way of quantifying impurity. The Gini index is defined as

$$G(p_{k,t}) = p_{k,t}(1 - p_{k,t})$$

where $G(p_{k,t})$ is the Gini index at node t with category k (for the response), and $p_{k,t}$ is the proportion of observations at node t that are included in category k. The intuition is that if a node contains data that is entirely (or close to entirely) the same in their response category, then that node is a very pure node. This will result in $p_{k,t} \approx 1$, and $G(p_{k,t}) \approx 0$. Similarly, if $p_{k,t} \approx 0$, then $G(p_{k,t}) \approx 0$. The Gini index lives in the interval $[0, 1]$, and a Gini index closer to 0 indicates a more pure node. In this way the Gini index is used as a measure of impurity, and we can evaluate the impurity at a node t by summing all the Gini indices for the different categories present at node t, calculated as

$$I(t) = \sum_{k=1}^{K} G(p_{k,t})$$

where K is the total number of response categories.

Similar to cases where numerical responses are encountered, the CART algorithm considers all possible splits at a given node across all explanatory variables. The best split is determined in an analogous manner to cases with numerical responses – by assessing the drop in impurity between a parent node and potential daughter nodes. This is calculated as

$$\Delta I = p(t_0)I(t_0) - (p(t_1)I(t_1) + p(t_2)I(t_2))$$

where t_0 is the parent node, t_1, t_2 are the daughter nodes, and $p(t)$ is the proportion of observations that are at that node, with respect to the entire data set. Weighting impurity reductions by the proportion of data at a specific node ensures that we give appropriate weight to the homogeneity of a node based on how much data is there. For instance, in terms of making splits, we should give more weight to a highly homogenous node with 100 observations than a node that is perhaps more homogenous but only has 5 observations. After assessing the impurity reduction across all possible splits, the best split is made by CART based on maximizing ΔI.

Tree Building Process

The CART algorithm continues to make additional nodes based on maximizing ΔWSS or ΔI (depending on whether the response is numerical or categorical) for each new node. The algorithm continues to do this until it reaches some stopping criterion. This could be a variety of things, but one common stopping criterion is some minimum number of observations that must be contained in a node. Once a terminal node contains fewer than this minimum number of observations, the CART algorithm will no longer consider that node for additional splitting. When all terminal nodes meet this criterion, the CART algorithm stops building the tree.

Assessing Risk

As a tree is built using CART, we can evaluate the risk at each terminal node. The risk gives an indication of how well the decision tree fits the data. The way that CART computes risk is different based on whether the response variable of the dataset is numeric or categorical. If the response is numeric, the risk at a node t is simply the WSS at that node.

$$R(t) = WSS_t = \sum_{i=1}^{N_t} (y_{i,t} - \bar{y}_t)^2$$

When we are dealing with a categorical response variable, we calculate risk as

$$R(T) = \sum_{i=1}^{N_t} I(y_{i,t} \neq k_{t,\text{majority}})$$

where $k_{t,\text{majority}}$ is the majority class at node t.

Regardless of whether the response variable is numeric or categorical, we can compute a total risk for a decision tree T based on summing the risks of all terminal nodes in the tree.

$$R(T) = \sum_{j=1}^{|T|} R(t_j)$$

where t_j is the j^{th} terminal node and $|T|$ is the total number of terminal nodes for the decision tree.

3.1.3 Pruning the Tree

When the CART algorithm builds the initial tree, it seeks to add branches and nodes so as to account for as much variability as possible. It does this by continuing to add branches and nodes as long as doing so decrease WSS or I, increasing the homogeneity of the terminal nodes. One way we can assess the quality of a tree (in terms of fitting the data) is by the total risk of the tree. However, this is not always desirable. In only considering risk reduction and with no regard for how many new branches and nodes the algorithm adds, CART runs the risk of over-fitting the data, and growing the tree to account for white noise rather than actual structure. Consequently, once CART has built the initial tree, it has to undergo a pruning process to generate the best final subtree.

Selecting Against Excessive Complexity

As mentioned previously, we don't want to solely consider risk reduction when determining our best decision tree model, due to the high likelihood of over-fitting. As such, the CART algorithm relies on a complexity parameter $\alpha > 0$ in order to quantify the cost of adding additional nodes and branches in the tree. The α parameter acts as a sort of filter, such that as α increases, branches and node additions must result in even greater risk reductions to be included in the tree. We introduce the assessment of $cost - complexity$ as a meaningful way for comparing different possible trees. We can calculate the $cost - complexity$ of a tree T as

$$R_\alpha(T) = R(T) + \alpha|T|$$

where $R(T)$ is the total risk of the tree T, and $|T|$ is the number of terminal nodes in tree T.

Similar to computing the $cost - complexity$ of a decision tree, we can compute the $cost - complexity$ of a node as well as of a branch. This is computed, respectively, as

$$\text{Node: } R_\alpha(t) = R(t) + \alpha * (1)$$

$$\text{Branch: } R_\alpha(T_t) = R(T_t) + \alpha|T_t|$$

where T_t is the branch with node t as its root and $|T_t|$ is the number of terminal nodes found in the branch T_t. Consider if $\alpha = 0$. Due to the fact that branches reduce risk more than nodes – and with $\alpha = 0$ the cost-complexity is unchanged by the size of the node/branch – then $R_0(T_t) < R_0(t)$. However, branches carry greater complexity than nodes, so as we increase the complexity cost parameter α, $R_0(T_t)$ increases at a faster rate than $R_0(t)$. Thus, at some point α' it will be the case that $R_{\alpha'}(T_t) = R_{\alpha'}(t)$. Rearranging we find that

$$\alpha' = \frac{R(t) - R(T_t)}{|T_t| - 1}$$

At this point, with the $cost - complexity$ equal for the branch and the node that it originates from, it makes more sense to prune the branch rather than the node, in favor of a smaller tree. We want a smaller tree so as to protect against the possibility of over-fitting the data. We find the weakest link, or the best branch to prune, by finding the node t that minimizes α'_1 – the lowest α' that results in a node and branch where $R_{\alpha'}(T_t) = R_{\alpha'}(t)$ – and then remove the branch originating from that node, which we refer to as the branch $T(\alpha_1)$. We then repeat this process with subsequently larger α' values, each time identifying the next node t at which $R_{\alpha'_i}(T_t) = R_{\alpha'_i}(t)$ using the smallest $\alpha'_i > \alpha'_{i-1}$. At each step of this process, we identify the next weakest link, and we prune the corresponding branch. We continue this process until α'_n, at which point $T_{\alpha'_n}$ is a tree with only the root node t_0. We end up with a series of n intervals:

$$\begin{aligned} I_0 &= [0, \alpha'_1) \\ I_1 &= [\alpha'_1, \alpha'_2) \\ &\vdots \\ I_n &= [\alpha'_n, \infty) \end{aligned}$$

such that for each interval, there is one optimal tree for that collection of α values. This gives us a corresponding sequence of trees $T_{\alpha'_1}, T_{\alpha'_2}, \ldots, T_{\alpha'_n}$. We then need to determine which α' produces the optimal tree overall.

3.2 K-Fold Cross Validation

In order to find the optimal α' pruning parameter, we engage in K-fold cross-validation. We begin by randomly dividing the training data into K folds, or subsets of equal size. For each of the α' values and for each of the K folds, we grow and prune a tree based on a data set that consists of the training data but with 1 of the K folds excluded. We then find the risks of all of these trees, and then average across the K folds. This gives us a good picture of which of the α' values produces the lowest risk on a variety of random subsets of our training data. Based on these results, we choose our optimal α' so as to minimize the average risk. We then obtain the corresponding decision tree for our chosen α'. The overall algorithm structure is depicted in Algorithm 2.

Algorithm 2 K-Fold Cross-Validation [5]

```
**Inputs:** $K$
Split the training data into $K$ folds (subsets) of equal size
**for** $k$ in $1, \dots, K$ **do**
    Grow a decision tree on the training data, with the $k^{\text{th}}$ fold excluded
    Prune the tree with each of $\alpha'_1, \dots, \alpha'_n$ to get $T_{\alpha'_1,k}, T_{\alpha'_2,k}, \dots, T_{\alpha'_n,k}$
    Compute $R(T)_k$ for each of $T_{\alpha'_1,k}, T_{\alpha'_2,k}, \dots, T_{\alpha'_n,k}$
For each $\alpha'_j$, find $\bar{R}(T) = \frac{1}{K} \sum_{k=1}^{K} R(T)_k$
Choose the optimal pruning parameter $\alpha'_{\text{optimal}}$ to be the $\alpha'_j$ that minimizes $\bar{R}(T)$
**return** Grow final tree on full training data and prune using $\alpha'_{\text{optimal}}$
```

Chapter 4

Bagging & Random Forests

4.1 Introduction

Individual trees produced by the CART algorithm are often highly variable in their structure. In addition, they also exhibit a tendency to overfit the training data that are inputted [5]. Various methods exist to help stabilize the highly variable nature of CART trees. Two of these methods are bagging and random forests.

4.2 Bagging

Predictions produced from a single CART tree can be highly variable and less accurate than we would like. One improvement to this issue with CART is bootstrap aggregating, commonly referred to as bagging. In bagging we bootstrap our training set many times, and then create a CART tree with each bootstrap resample (Figure 4.1). When we want to obtain predictions for some new observation (with a continuous response), we obtain predictions from each individual tree, and then average them for our final prediction. Through this method, we obtain predictions that are much more stable and consistent than if they were solely based on a single decision tree.

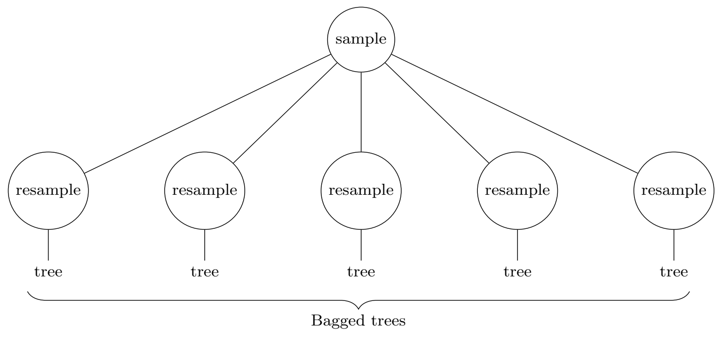


Figure 4.1: Bagging

4.3 Random Forests

4.3.1 Description

Our aim in bagging is to stabilize the highly variable nature that is inherent in individual decision trees, as well as to reduce the tendency of individual trees to overfit the data. Random forests have the same goal, and share many commonalities with bagging. Similar to bagging, random forests are constructed by repeatedly bootstrapping the original training data set and building an individual decision tree with each bootstrap sample. However, random forest also rely on an additional variability measure in order to avoid over-fitting of the data.

When trees are built within the random forest algorithm, node-splitting is determined using the same measures as CART, such as maximizing ΔRSS between parent and daughter nodes. However, in considering potential splits for a node, only a randomly selected subset of size m of the explanatory variables are considered for splitting. This is a parameter in the random forest algorithm, and has been studied extensively. If we have p explanatory variables in our training data, a very common value for m is $m = \sqrt{p}$. By choosing nodes based on a subset of explanatory variables, a greater variety of tree structures are introduced to the random forest.

An additional distinction between CART decision trees and random forests is that random forest trees are not pruned. The aim of pruning in CART is to protect against over-fitting the data. However, the bootstrapping and subsetting of explanatory variables in tree-building that are characteristic of random forests make it so that pruning is not necessary in random forests. In an identical manner to bagging, we obtain predictions from forests by obtaining predictions from each individual tree, and then averaging them for the final prediction (Figure 4.2).

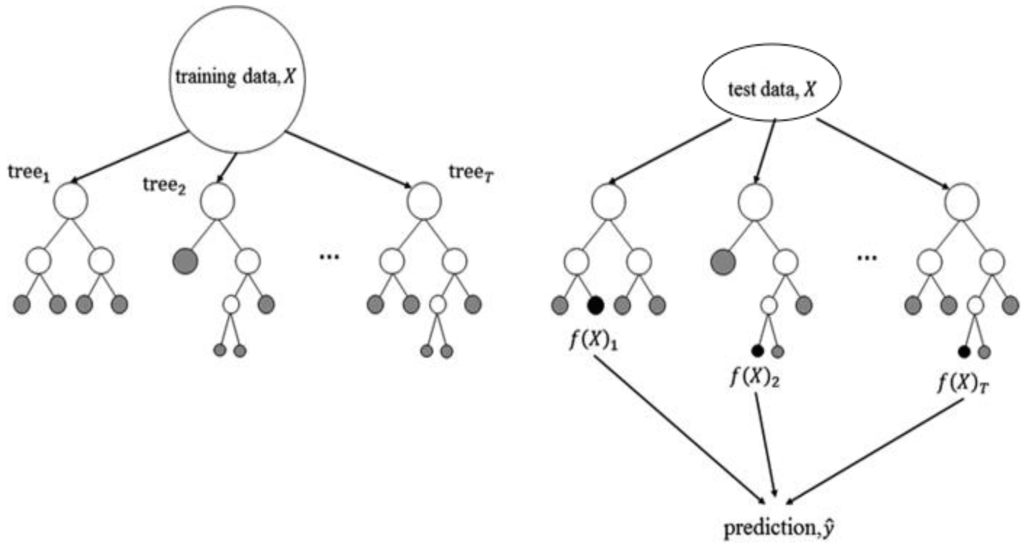


Figure 4.2: Obtaining Predictions from Random Forests [7]

Chapter 5

Merging BLB and Random Forests

Given that one critique of random forests and other machine learning methods is that the predictions they produce are not accompanied by error estimates, we might consider trying to implement a BLB-like structure for random forests as a way to get prediction error estimates in a computationally inexpensive manner.

5.1 Translating BLB to Random Forests

5.1.1 Overall Structure

We propose a structure that is very similar to BLB. The algorithm will randomly select (without replacement) S subsamples of size $B << N$. Similar to BLB, a typical value for B will be $B = \sqrt{N}$. Each of these subsamples will be processed on a different parallel processor. We expect that similarly to BLB, working with data of a reduced dimension will lead to a meaningful reduction in computational time as compared to working with the full data set. Within each processor, we will construct a random forest. As discussed previously, in the random forest algorithm a collection of decision trees are generated, with each tree being built from a new bootstrap sample of the original inputed data.

At this point in BLB, in order to ensure the correct amount of error, bootstrap samples of size N are generated by sampling with replacement R

times from the subsample of size B. We need to translate this bootstrapping – from size B up to size N – to the random forest setting. In the context of random forests, we propose sampling generating bootstrap samples, and then using each of those bootstrap samples to grow a different tree within the random forest.

However, we also must recall that the bootstrapping step of BLB is where a multinomial random variable is cleverly placed. The use of this multinomial random variable is what enables the algorithm to only ever work with data of dimension B rather than of dimension N. We consider how to implement the multinomial random to the BLB-like random forest structure in an analogous and equally effective way.

Implementing the Multinomial Random Variable

In the original BLB algorithm, the clever use of the multinomial random variable occurs when computing the sample mean for each bootstrap sample. The way the sample mean is calculated "resembles" computing the sample mean on data of size N, when in actuality the computation is done with data of size B. In order to implement a similar step in the BLB-like random forest structure, we propose a change to the node-splitting step in tree-building. The modifications in Table 5.1 depict such an implementation, which incorporates the multinomial random variable into this node-splitting component of the tree-building that occurs within random forests at some node t.

	$randomForest$	$randomForest$ BLB
Mean	$\bar{y}_t = \frac{1}{N_t}\sum_{i=1}^{N_t} y_i$	$\bar{y}_t = \frac{1}{N_t}\sum_{i=1}^{N_t} m_i \cdot y_i$
Sum of Squares	$WSS_t = \sum_{i=1}^{N_t}(y_i - \bar{y}_t)^2$	$WSS_t = \sum_{i=1}^{N_t} m_i \cdot (y_i - \bar{y}_t)^2$

Table 5.1: Incorporating a Multinomial Random Variable into Tree-Building

- N_t denotes the number of training observations at node t.
    - N_t would be expected to be smaller on average in the $randomForest$ BLB case since trees are built with training data of size $B << N$.

Prediction Interval Construction

Once the random forests are built using the implementation of the multinomial random variable, we obtain predictions for our test observations from the different forests. Within each of the S parallel tracks, we obtain R predictions for each test observation. For the i^{th} observation, we have predictions $\hat{\mathbf{Y}}_{\mathbf{i}} = (\hat{y}_{i,1}, \hat{y}_{i,1}, \ldots, \hat{y}_{i,R})$. We then construct prediction intervals centered at $\frac{1}{R}\sum_{j=1}^{R}\hat{y}_{i,j}$. It remains to be seen how exactly the algorithm will effectively determine how far to extend from this center for the interval bounds. As a naive method, we propose simply setting the endpoints of the interval to be the $\frac{\alpha}{2}$ and $1-\frac{\alpha}{2}$ quantiles of the collection of R predictions. We explore the question of effectively establishing interval endpoints in detail later on in this thesis.

Once confidence intervals are constructed from each parallel track, for each observation we have S intervals of the form $(l_{i,j}, u_{i,j}), j = 1, \ldots, S, i = 1, \ldots, N_{\text{test}}$. In a similar manner to BLB, the final intervals for each test observation are obtained by simply averaging the endpoints of the S intervals, or $\left(\frac{1}{S}\sum_{j=1}^{S} l_{i,j}, \frac{1}{S}\sum_{j=1}^{S} u_{i,j}\right)$. The complete structure of this proposed BLB-like random forest implementation is depicted in Figure 5.1.

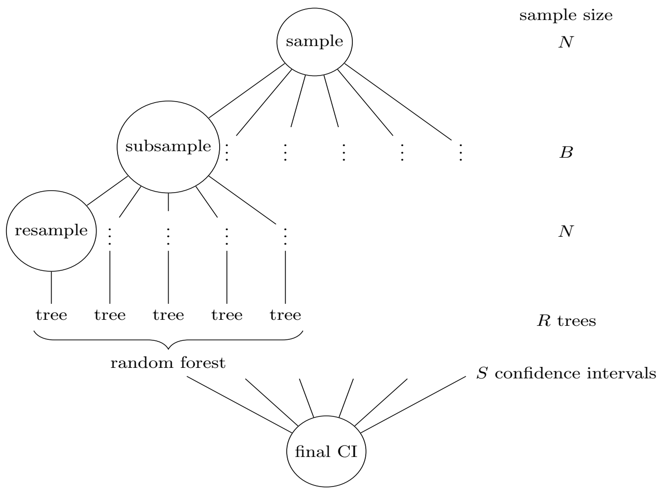


Figure 5.1: BLB-like Implementation of Random Forests

Chapter 6

Simulation Studies – Prediction Intervals from CART & RFs

6.1 Motivation

As discussed previously, we eventually hope to implement a BLB-like structure for random forests. One of the major motivations behind such an initiative is to be able to obtain reliable prediction error estimates on the predictions that we obtain from random forests. In the original BLB algorithm, the focus was on obtaining confidence intervals for μ rather than prediction intervals for predictions coming from a model. The former captures the average $(1-\alpha)\%$ of the time, whereas the latter captures the individual observations $(1-\alpha)\%$ of the time. In the random forest setting, we are instead interested in prediction intervals. This changes the way that we think about constructing accurate intervals. Additionally, before we can feel confident in saying a BLB-like structure for random forests produces accurate prediction intervals, it is important that we ask a number of preliminary questions:

1. How do we estimate prediction variability in the context of RFs and trees?
2. How different is prediction variability between RFs and individual trees?
3. How can we get prediction intervals that actually capture the true response at the appropriate frequency?

In addressing the first question, it is not immediately clear what variability even means in a random forest context. If we were in a traditional linear model setting, we could evaluate variability of a model by examining how the slope and intercept coefficients of our model vary. We could also think about $\hat{Y}$ varying However, in random forests there is much less of an intuitive structural representation – there is in fact no model that can be easily written down – so it is important to consider how we are to think about variability in the context of random forests. Furthermore, we also must consider the difference in variability between random forests and individual trees, in terms of the predictions that they produce. One of the central aims of random forests is to stabilize the highly variable nature of individual trees, so we must carefully evaluate to what degree this occurs. Finally, we must evaluate whether our proposed methods of prediction intervals result in intervals that exhibit the error level that we would expect. For example, suppose we are constructing what we expect to be intervals with a 5% error rate. So we would on average expect 5% of our intervals to not contain the true response. If our intervals in fact produce a error rate lower than 5%, then we are doing worse than we could by having intervals that are too wide. Alternatively, if our intervals are actually giving an error rate higher than 5%, then we are worse off because we will be less accurate in inference than we expect. In order to answer these questions, we conduct a number of simulation studies.

6.1.1 Confidence vs. Prediction Intervals

In properly interpreting the results from our simulation studies, it is important to first consider the distinction between confidence and prediction intervals. We consider a hypothetical sample where we are interested in predicting *Age of youngest friend* based on the *Age* of a study participant (Figure 6.1). We fit a simple linear model to the data and construct both confidence intervals (CI) and prediction intervals (PI).

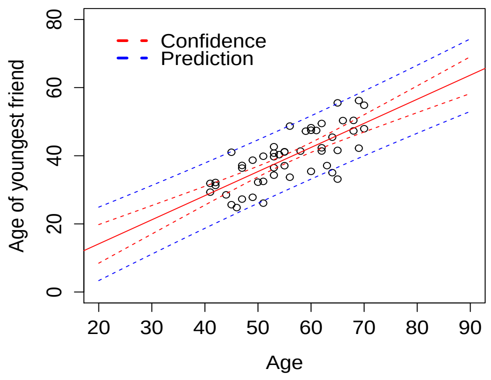


Figure 6.1: Confidence vs. Prediction Intervals

In the figure above, the red-dotted line depicts the *confidence* interval bounds. A confidence interval is intended to capture the *mean response* (at a specific explanatory value) 95% of the time on average. The blue-dotted line depicts the *prediction* interval bounds. This denotes the region where on average 95% of individual *observations* will fall.

Translating Confidence and Prediction Intervals to a Forest and Tree Setting

In order to fully understand the simulations discussed in this thesis, we must first think about how this distinction between confidence and prediction intervals translates to interval construction in the context of random forests and CART trees. If we recall how predictions are generated in random forests and CART trees, predictions are made by averaging the responses of training observations within a given terminal node of a tree. This aspect of forests and trees becomes an important consideration when trying to construct confidence intervals or prediction intervals. We explore this distinction through two simulations.

The first method uses a naive method of constructing intervals that looks at how the mean predicted response varies for a specific observation. Said another way, variability is measure by how the mean response varies across the different nodes that test observations are assigned to in different trees. In thinking about the consequences of intervals being constructed in this manner, it is important to emphasize that intervals in this instance are measuring the predicted response of certain nodes, which is simply an average of the training responses that comprise a certain node. As such, this method ignores any training response variability *within* the actual node. Consequently, intervals constructed in this way are in reality built to capture the *mean* response of the nodes in which an observation falls, rather than the actual response of that individual observation. These intervals can be thought of as being analogous to confidence intervals.

This is where our third and fourth simulations come into play and inform this distinction further. In order to construct intervals that are built to capture the individual responses, the intervals need to account for the training response variability within a node. Consequently, an additional simulation employs a different method of constructing intervals, one which seeks to accurately assess the training response variability within a node, in the hope that the intervals will in fact effectively capture the individual test responses rather than simply the mean response of a certain node. These intervals are to be thought of as being constructed in an analogous manner to prediction intervals.

6.2 Simulation Studies

6.2.1 Naive Approach – General Simulation Structure

We conduct simulation studies that assess our ability to obtain accurate prediction intervals in a naive manner that is similar to the way in which confidence intervals are constructed in the BLB structure. We run these studies using two types of simulated data. In each simulation, we construct both individual CART trees and random forests as a way of comparing the variability of the two models. We are interested in assessing the true error rate of the prediction intervals that we are constructing, in order to see whether our intervals closely match the error rate that we anticipate.

In order to do this, we perform a large number of repetitions of generating naive prediction intervals. Within each repetition, we randomly divide the dataset into a test set (of size $N_{test} = 250$) and a training set (of size $N_{train} = 750$). We then construct S (typically 100) random forests and S individual decision trees. Each random forest and decision tree is generated using a new bootstrap sample of size N_{train} from the training data. Once we have this collection of random forests and decision trees, we use all forests and trees to generate two large collections of predictions for the observations in the test set.

Let $\mathbf{P_{RF}}$ be the resulting predictions from the random forests and let $\mathbf{P_{DT}}$ be the resulting predictions from the single decision trees. So then $\mathbf{P_{RF}}$ and $\mathbf{P_{DT}}$ are both N_{test} by S matrices. With these two large collections of predictions in hand, we construct naive prediction intervals for each test observation. For the i^{th} test observation, the endpoints of our confidence intervals from the random forests and single decision trees become the 2.5 and 97.5 percentiles of the i^{th} rows of $\mathbf{P_{RF}}$ and $\mathbf{P_{DT}}$, respectively. Now that we have the prediction intervals constructed, we check to see whether or not the true response of each observation was actually captured by the interval.

We repeat this procedure a chosen number of *reps*, each time using a different test/training set split. Once we have conducted a large number of repetitions of this procedure, we can then compute a capture rate for each observation by dividing the number of times an observation's response was captured by the number of times that the observation appeared in the test set. The simulation structure is depicted in Algorithm 3.

Algorithm 3 Simulation Structure

```
Inputs: $N, reps, S, test.size, train.size$
Simulate data of size $N$
for $i$ in $1, \ldots, reps$ do
    Split data into test set and training set
    Record indices of test set
    for $j$ in $1, \ldots, S$ do
        Bootstrap training data
        Use bootstrapped data to grow CART tree and RF on training data
        Obtain predictions for test data from CART tree and RF
    for CART and RF do
        Obtain 95% PI for test data using percentile intervals
        Record bounds of each prediction intervals
        Record whether each PI contains corresponding test data response
For each observation in full data set, compute:
Capture rate = $\frac{\text{\# times caught by PI}}{\text{\# times in test set}}$
return Empirical confidence rates, test set counts, CART and RF PIs
```

6.2.2 Linear Data

We first consider generating data ($N = 1000$) from a simple linear model with 10 explanatory variables and a normally distributed noise term. We simulate data of the form

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

To understand the structure of this data, we can consider a visualization of the hypothetical space associated with 2 (of 10) dimensions of the explanatory variables (Figure 6.2).

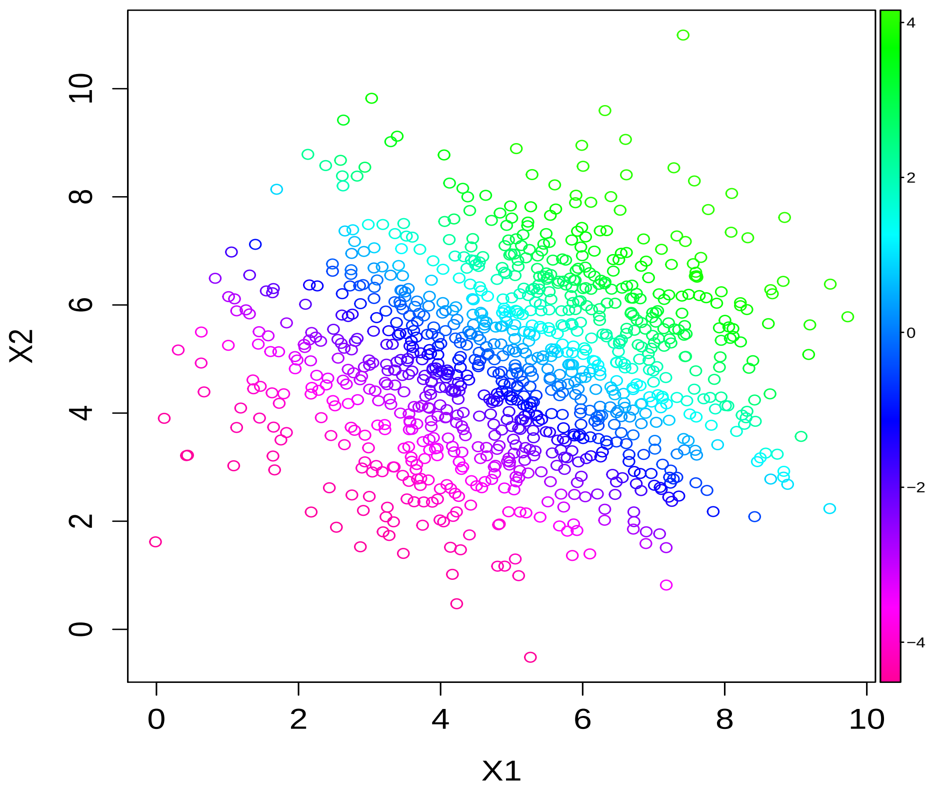
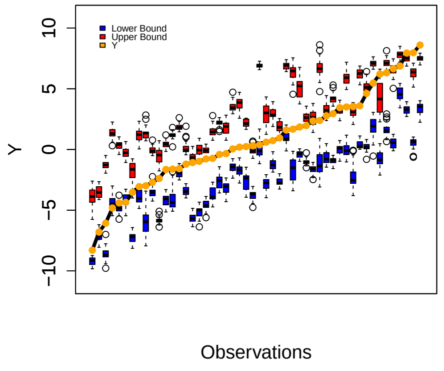
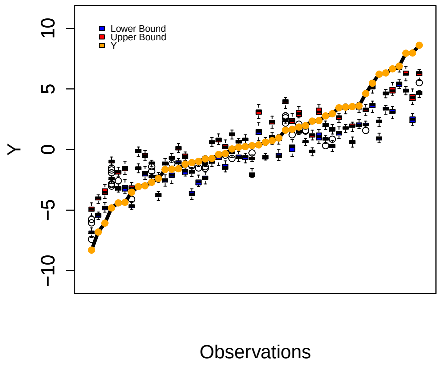


Figure 6.2: Linear Model in 2 Dimensions with Color Representing the Response Variable

In the plot above, the color gradient corresponds to the gradient of the response variable for the full data set. The first thing we notice is that there is a very smooth gradient in the explanatory variables for the response. With this in mind, we see how it would be difficult to draw vertical and horizontal lines so as to create groupings in the explanatory variables that are relatively homogenous in their response (the goal of random forests and decision trees). For this reason, it appears that perhaps data of this structure might not be handled very well by the model that is inherent to trees and random forests. Nevertheless, we conduct a simulation with this data using the standard structure (Algorithm 3). We obtain a number of interesting results (Figure 6.3).



(a) CART



(b) Random Forests

Figure 6.3: Collection of Naive CART and RF Prediction Intervals on Linear Data

The above figure depicts 50 selected observations from the linear model dataset. Plotted are boxplots of the lower and upper bounds on the naive prediction intervals for the 50 observations, accompanied by the true response for each observation. We first notice that the bounds for the CART PIs are much more variable than for the random forest PIs. This is demonstrated by the boxplot boxes being wider for CART than for random forests. A second observation comes from noticing that the center of the PIs fluctuate with the data more in the CART case than with the random forest case. Finally, we notice that the lengths of the prediction intervals are larger in the CART case than for random forests.

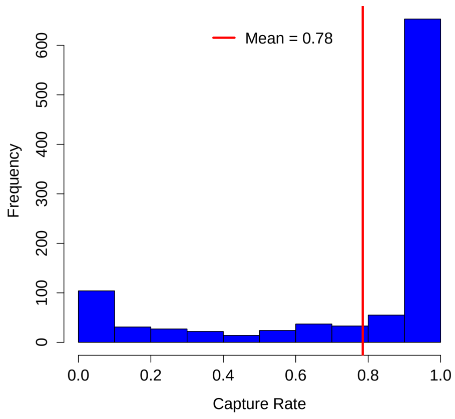
We develop intuition for this observation by revisiting one of the major differences between individual decision trees (CART) and random forests. The prediction used to generate the above intervals are coming from random forests or CART trees. So for a given observation, we obtain one prediction from a CART tree and one prediction from a random forest. As we recall from our previous discussion of the motivation behind random forests, we remember that one of the central aims of random forests make it so that predictions are relatively consistent, much more so than if the predictions were based on single decision trees. This added prediction consistency comes from the fact that random forests are collections of decision trees, each of which is based on a new bootstrap sample and where random variable selection occurs at each node in the tree-building process.

While individual decision trees are known for being highly variable, random forests are in contrast known for being fairly stable in their predictions. These characteristics of random forests and CART provides possible justification for why the naive prediction interval bounds are more variable with CART than with random forests. Since our prediction intervals are based on percentiles from the distribution of predictions that we get for a given observation (found in $\mathbf{P_{RF}}$ and $\mathbf{P_{DT}}$), then more variable predictions would result in more variable percentile intervals. The higher degree of variability that is characteristic of CART also explains the fact that the interval lengths are larger with CART than with random forests, since more variable predictions will result in wider confidence intervals.

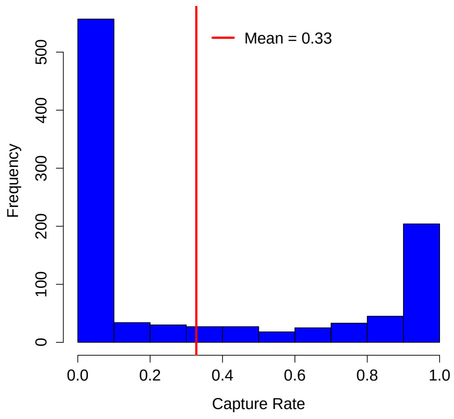
We consider the capture rates that we obtained for the 1000 observations in our simulated data, including a comparison of how these rates differ between CART and random forests (Figure 6.4).

We find that we are much closer to our desired capture rate of 0.95 for CART than for random forests. If we revisit the characteristics of random forests and CART, we can develop some intuition for why this may be happening. As discussed previously, CART trees are known for highly variable predictions, as compared to the stabilization that occurs with random forests. Thus, it's possible that the CART intervals capture our response more frequently because they produce wider intervals that fluctuate more readily. The actual predictions of random forests may be off by a similar amount to any given CART tree, but random forests' additional stability in predicting translates to smaller and less variable confidence bounds, ultimately capturing the response less frequently and consequently lower empirical confidence levels.

The ultimate reason for the lower than expected capture rates is based on the way in which the intervals are constructed. The intervals are built by taking percentiles of the predictions from the different random forests and



(a) CART



(b) Random Forests

Figure 6.4: Response Capture Rates from Naive CART and RF Prediction Intervals on Linear Data

single decision trees. As we recall, each individual prediction for a test observation is chosen to be the mean training response of the node in which the test observation is found. As such, we realize that the intervals constructed in this simulation are not actually built to capture response of an *individual* observation. To capture the individual response, we need to evaluate the variability of the training responses in that same node.

6.2.3 Gaussian Mixture Model Data

We recall that CART trees and random forests are more well-suited to situations in which the data being analyzed is fairly easy to use binary splits in the explanatory variables to separate the data into groupings that are relatively homogenous in their responses. With this understanding, we recognize that CART trees and random forests are more likely to be successful in analyzing highly clustered rather than linear data, and so we consider simulating data from an alternative source. We use a gaussian mixture model from the **MixSim** R package to generate 5 explicit clusters based on 10 explanatory variables [8]. We then simulate the response through drawing from cluster-specific normal distributions that are shifted in their means so as to have no overlap. For each cluster in the explanatory variables, all the responses come from the same normal distribution. The different normal distributions are randomly assigned to each cluster. These normal distributions are of the form

$$N(\mu_i, 1)$$

$$\boldsymbol{\mu} = (1, 5, 9, 13, 17)$$

To understand the structure of this data, we can consider a visualization of the hypothetical space associated with 2 (of 10) dimensions of the explanatory variables (Figure 6.5).

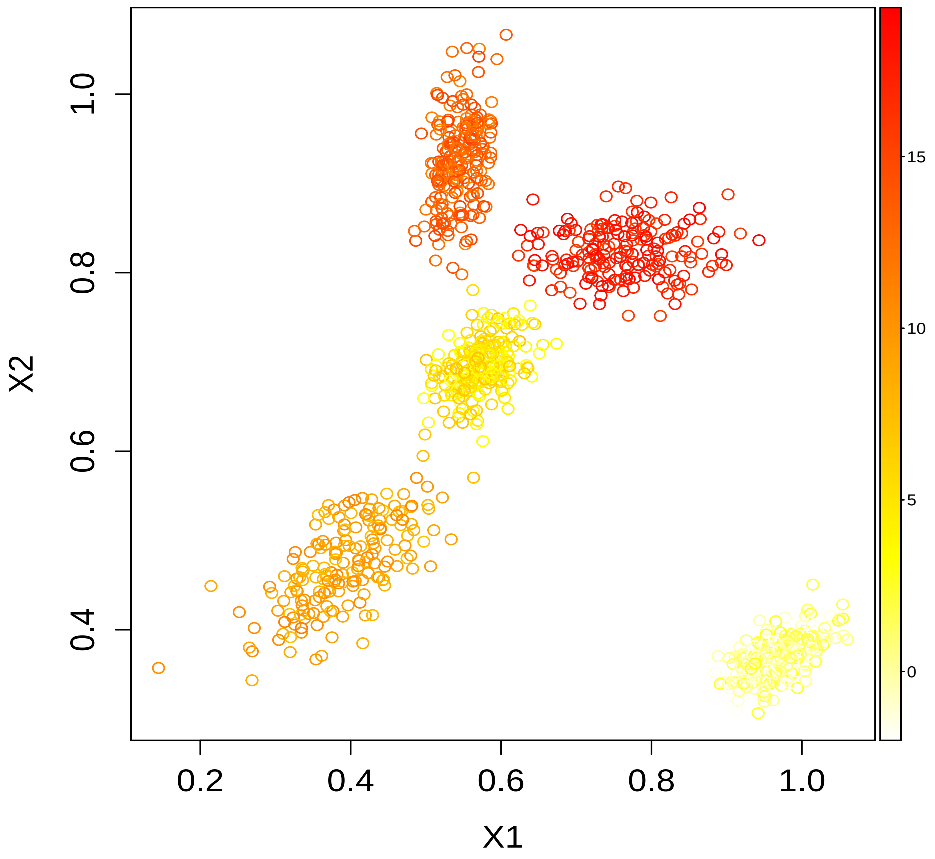
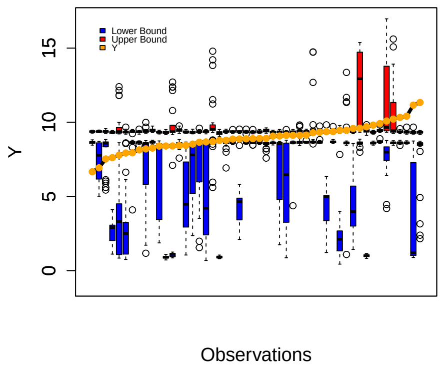
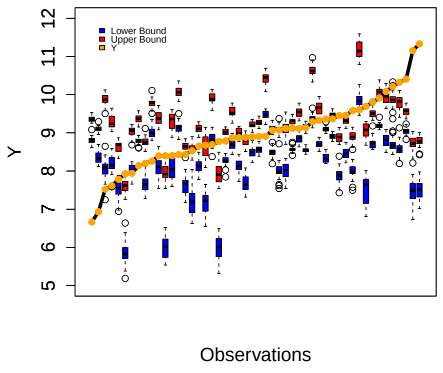


Figure 6.5: Clustered Model in 2 Dimensions with Color Representing the Response Variable

In the plot above, the color gradient corresponds to the gradient of the response variable for the full data set. The first thing we notice is that there is a very clear distinction within the explanatory variables for the different clusters. In contrast to the data from the linear model, we can see how it would be fairly straightforward to isolate each cluster by drawing vertical and horizontal partitions within the explanatory variables. For this reason, it appears that perhaps data of this structure might be handled much better by the model inherent to trees and random forests than was the linear data. We conduct a simulation with this data using the standard structure (Algorithm 3). We obtain a number of interesting results (Figure 6.6).



(a) CART



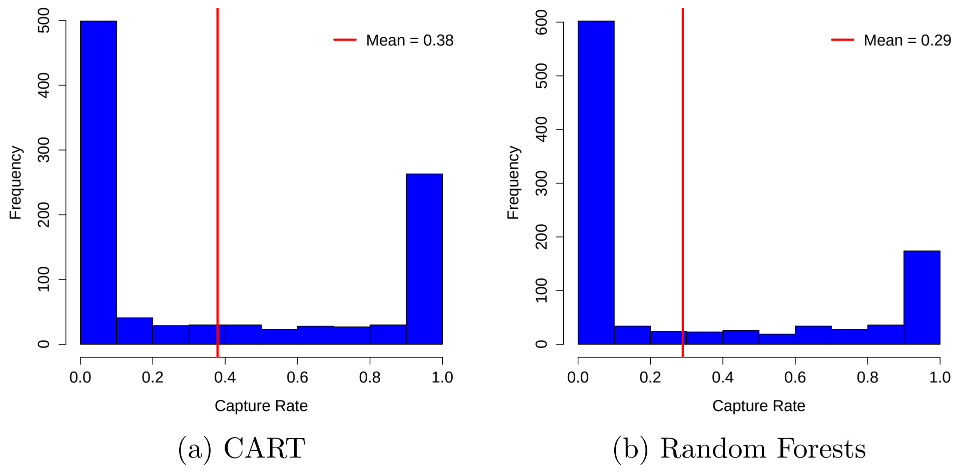
(b) Random Forests

Figure 6.6: Collection of Naive CART and RF Prediction Intervals on Clustered Data

The above figure depicts 50 selected observations from the clustered data set. Plotted are boxplots of the lower and upper bounds on the confidence intervals for the 50 observations, accompanied by the true response for each observation. We first notice that the bounds for both the CART and random forest PIs are fairly disorderly.

A second observation comes from examining the very large box sizes of the CART PIs for certain observations. This indicates that the predictions for these observation are extremely variable with single CART trees. This makes intuitive sense based on the highly variable nature of CART trees, but we can develop further intuition by considering the structure of the data for this simulation. As seen in the 2-dimensional visualization of clustered data, there is much larger separation within the explanatory variables. Additionally, the clusters are drastically different in their response. If a CART tree incorrectly groups an observation with observations from a different cluster, then the prediction that it produces will be drastically deviated from the true response (Figure 6.5). While this is not happening every time, we hypothesize that it happens frequently enough to produce such highly variable confidence bounds. We don't see this same pattern with the random forest PIs, since random forests have the benefit of stabilizing the highly variable trees. As such, even if a few trees in the forest misallocate an observation to an incorrect cluster, there are enough trees in the forest that this does not completely disrupt the eventual confidence bounds. Finally, it's interesting to compare the performance between the linear and the clustered data. While the performance of the CART PIs was dramatically reduced with the clustered data as compared to the linear data, this occurred only slightly for the random forest PIs.

We can also consider the capture rates that we obtained for the 1000 observations in our simulated data, as well as how these rates differ between CART and random forests (Figure 6.7).



(a) CART (b) Random Forests

Figure 6.7: Response Capture Rates from Naive CART and RF Prediction Intervals on Clustered Data

We find that we are nowhere near our desired capture rate of 0.95 for CART or random forests. However, as discussed previously the performance decreased less for random forests than for CART.

6.2.4 Alternative Methods of Obtaining Prediction Intervals

If we are to develop intervals that will capture the responses of individual observations at the rate that we would anticipate, then from these previous simulations it is clear that we must consider a different method for obtaining intervals. Recall that the prediction for test observations at some node t is simply the average of the responses for the training data that comprise that node. As discussed previously, this means that predictions obtained in this way do not account for whatever variance may exist in the responses for the training data at a specific node. Furthermore, generating intervals in this manner would not be expected to capture the individual test responses, since this would require that the intervals compensate for the response variability of the training observations for the node. As such, we need to develop a way to construct intervals that account for the variability in training response for a node.

We propose a new method for obtaining prediction intervals, and we test this method using a single random forest (as compared to the hundreds of forests used in previous simulations). This method for establishing prediction intervals is depicted in Algorithm 4 for a single test observation $(\mathbf{X}_{\text{test}}, \mathbf{Y}_{\text{test}})$, and we will refer to prediction intervals from this method as Within-Node Variability (WNV) prediction intervals. This procedure can be easily repeated for all observations in the test set.

Algorithm 4 Within-Node Variability (WNV) Prediction Intervals

```
**Inputs:** $(\mathbf{X}_{\text{test}}, \mathbf{Y}_{\text{test}}), nodeSize, nFriends, numTrees$
**Outputs:** One prediction interval for $\mathbf{Y}_{\text{test}}$, based on $\mathbf{X}_{\text{test}}$
Grow random forest using training data, specifying the parameters $numTrees$ and $nodeSize$
**Compute** $s_{\text{all}} = ((\sum_{i=1}^{M} |t_i| \cdot s_i^2)/(\sum_{i=1}^{M} |t_i|))^{\frac{1}{2}}$
$t_i$ is the $i^{\text{th}}$ terminal node that is of a size greater than $nFriends$
$s_i^2$ is the squared standard deviation of the training response in the $i^{\text{th}}$ terminal node that is of a size greater than $nFriends$
$M$ is the number of nodes that are of size greater than $nFriends$

**for** $t$ in $1, \ldots, numTrees$ (terminal nodes that contain $(\mathbf{X}_{\text{test}}, \mathbf{Y}_{\text{test}})$) **do**
    **if** $|t| < nFriends$ **then**
        $s_t = s_{\text{all}}$
    **else**
        $s_t = \sqrt{\frac{1}{|t|} \sum_{j=1}^{|t|} (y_{t,i,\text{train}} - \bar{y}_{t,\text{train}})^2}$
        $y_{t,i,\text{train}}$ is the response of the $i^{\text{th}}$ training observation at node $t$
        $\bar{y}_{t,\text{train}}$ is the mean response of the training observations at node $t$
**Compute** $s_{\text{pred}} = \sqrt{(1/\sum_{t=1}^{\text{numTrees}} N_t) \cdot (\sum_{t=1}^{\text{numTrees}} N_t \cdot s_t^2)}$
Obtain final prediction interval bounds as
$\bar{\text{pred}} \pm 2 \cdot \sqrt{s_{\text{pred}}^2 + \frac{s_{\text{pred}}^2}{numTrees}}$

$\bar{\text{pred}}$ is the mean prediction for $\mathbf{Y}_{\text{test}}$ across all the trees within the random forest.
**return** Final prediction interval bounds.
```

In order to evaluate the performance of this algorithm for obtaining accurate prediction intervals, we conduct another simulation that is similar to Algorithm 3. However, in this simulation we only use a single random forest, rather than multiple random forests and multiple CART trees. An additional difference is that in this simulation we are constructing 95% prediction intervals using Algorithm 4, rather than the naive prediction intervals using the percentile interval procedure from the first two simulations. In these simulations we again consider two types of data, one set from a simple linear model and the other using the **MixSim** R package, in an analogous manner to the data generated for the first simulation. We use the parameters

- $N = 1000$ ($testSize = 250$, $trainSize = 750$)
- nodeSize = 10
- nFriends = 5
- numTrees = 5
- reps = 1000

Through other simulations using this method of interval construction, we found that substantially increasing the *numTrees* parameter did relatively little to improve the capturing performance of the prediction intervals that were generated. So in order to keep computational time low, we conducted the simulation using $numTrees = 5$. We first consider a number of interesting results from the simulation using the linear model data (Figure 6.11)

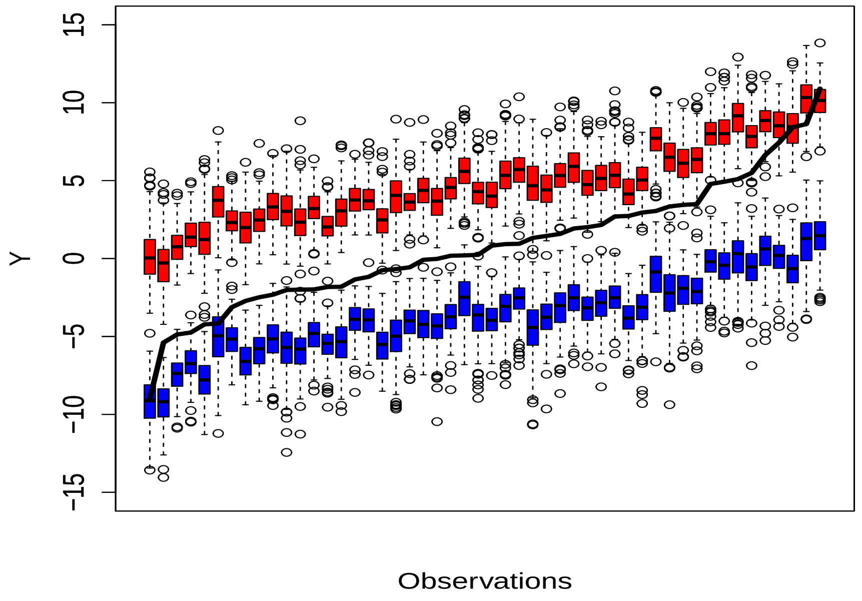


Figure 6.8: Collection of WNV RF Prediction Intervals on Linear Data

The above figure depicts 50 selected observations from the linear model dataset used in this simulation. Plotted are boxplots of the lower and upper bounds on the prediction intervals for the 50 observations, accompanied by the true response for each observation. We first notice that the prediction intervals do a fairly good job at capturing the true response. The intervals properly adjust to changes in the response variable. However, the intervals become less reliable for capturing the more extreme responses of the depicted observations.

We can also consider the overall response capture rate that we obtain for the 1000 observations in our simulated data (Figure 6.9). We find that many observations have a capture rate fairly close to the expected capture rate of 0.95, since we are trying to construct 95% prediction intervals.

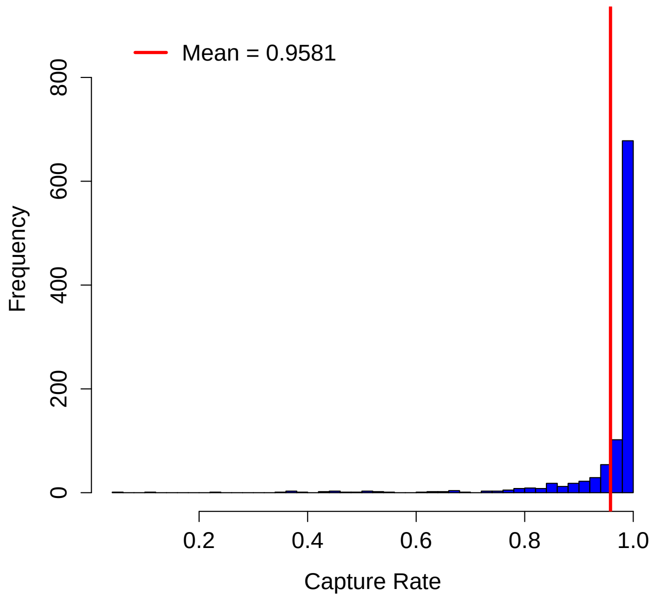


Figure 6.9: Capture Rates for WNV RF Intervals on Linear Data

We also find that 353 of the 1000 observations were captured every time. Since we conducted 1000 repetitions of the simulation and the test data set in each repetition was of size 250, on average we would expect a given observation to be in the test set for ≈ 250 of those repetitions. Prediction intervals are only constructed when an observation is randomly selected to be in the test set, so this means that on average, the capture rates in Figure are based on ≈ 250 prediction intervals. Our finding that 353 of the 1000 observations were captured every time is not surprising, as these observations can be thought of as being analogous to those which are in the center of the prediction interval in Figure 6.1.

We also consider the proportion of observations captured by the prediction intervals in each iteration of the simulation (Figure 6.10). We find that in most of the repetitions in the simulation, close to 95% of the test responses were captured, which is what we would expect. The deviations from a 95% capture rate were relatively small.

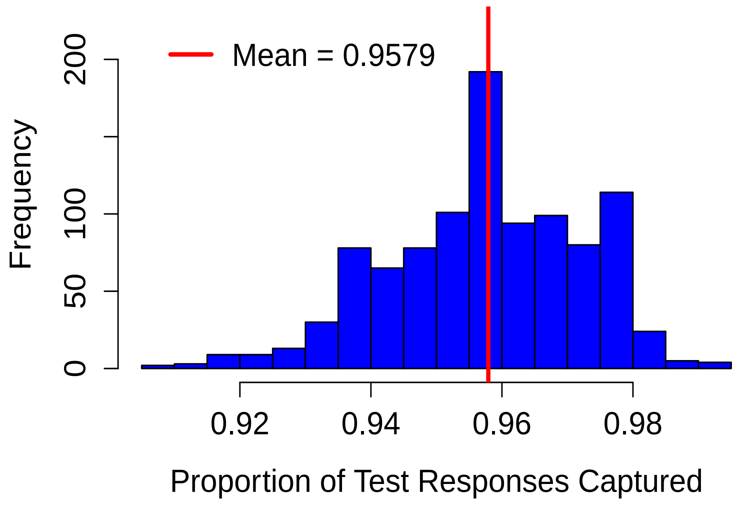


Figure 6.10: Overall Capture Rates (for Each Reptition of Simulation) for WNV RF Prediction Intervals on Clustered Data

In order to better elucidate the characteristics of our newly proposed method of prediction interval construction, we also consider the results from the simulation using the clustered data (Figure 6.11)

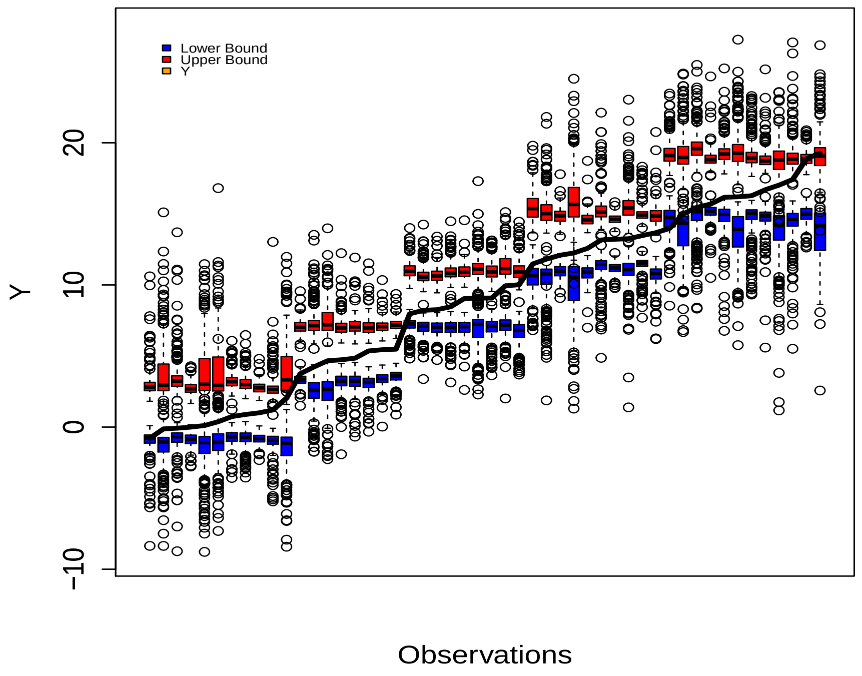


Figure 6.11: Collection of WNV RF Prediction Intervals on Clustered Data

The above figure depicts 50 selected observations from the clustered dataset used in this simulation. Plotted are boxplots of the lower and upper bounds on the prediction intervals for the 50 observations, accompanied by the true response for each observation. We first notice that the prediction intervals do a fairly good job at capturing the true response. The intervals properly adjust to the different clusters. However, we also notice that the intervals do not exhibit substantial adjustment *within* a given cluster. So for one observation that has a response near the low end of responses for a given cluster, and another observation that has a response which is near the high end for that cluster, the prediction intervals will still be relatively the same. However, across all observations in each cluster, it appears as though the intervals depicted in the plot are capturing the true responses at approximately the rate that we would expect.

We can also consider the overall response capture rate that we obtain for the 1000 observations in our simulated data (Figure 6.12). We find that many observations have a capture rate fairly close to the expected capture rate of 0.95, since we are trying to construct 95% prediction intervals.

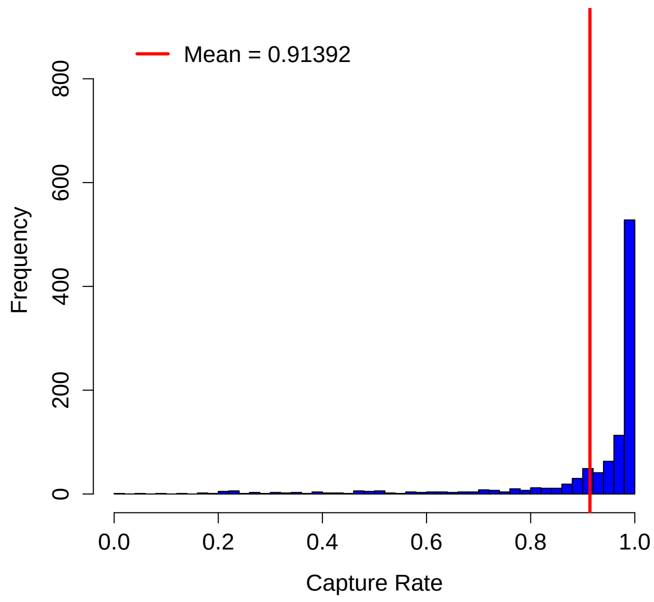


Figure 6.12: Capture Rates for WNV RF Intervals on Clustered Data

We also find that 240 of the 1000 observations were captured every time. Since we conducted 1000 repetitions of the simulation and the test data set in each repetition was of size 250, on average we would expect a given observation to be in the test set for ≈ 250 of those repetitions. Prediction intervals are only constructed when an observation is randomly selected to be in the test set, so this means that on average, the capture rates in Figure are based on ≈ 250 prediction intervals. Our finding that 353 of the 1000 observations were captured every time is not surprising, as these observations can be thought of as being analogous to those which are in the center of the prediction interval in Figure 6.1. We would expect that these 353 observations have responses which are relatively close to their cluster-specified mean (μ).

We also consider the proportion of observations captured by the prediction intervals in each iteration of the simulation (Figure 6.10). We find that in most of the repetitions in the simulation, close to 91% of the test responses were captured, which is a smaller proportion than what we would expect. The deviation from a 91% capture rate was relatively small. The cause of this lower-than-expected capture rate could be due to the reasons discussed above, where perhaps the prediction intervals frequently fail to capture the edge response of the different clusters.

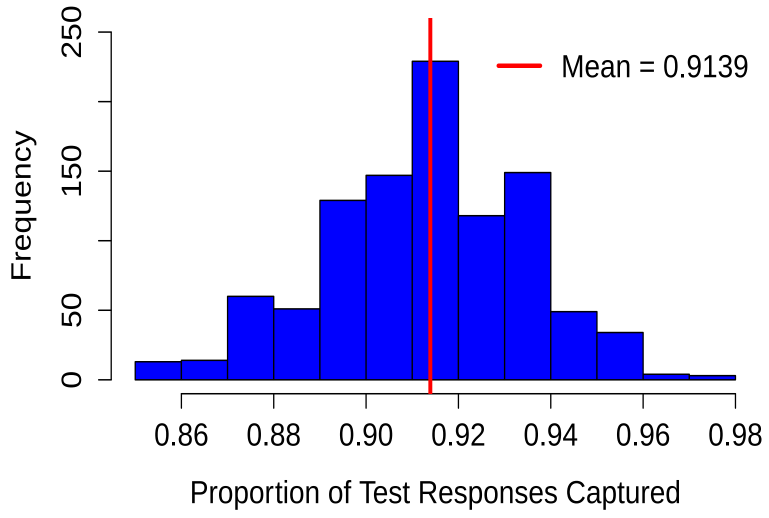


Figure 6.13: Overall Capture Rates (for Each Reptition of Simulation) for WNV RF Prediction Intervals on Clustered Data

From the simulations of node-based intervals using linear and clustered data, we observed that the capture rates for many of the observations were fairly close to the 95% rate that we would expect, with this being more the case for the linear data than the clustered data. As discussed, the subpar performance on the clustered data could be due to the structure of the clustered data. It is possible that these lower-than-expected capture rates are due to the individual trees misallocating a test response to the wrong cluster. In our simulations using WNV intervals, we used a value of $numTrees = 5$, which is extremely small for random forests (the standard value in the **R** package implementation is $numTrees = 500$). Using a small value for $numTrees$ diminishes the stabilization that random forests provide as an advantage over single CART trees. Increasing the value of $numTrees$ could perhaps improve this issue.

Chapter 7

Conclusion

In this thesis, we explored various ways to develop inference frameworks for random forests, with the goal of finding a way to produce obtain accurate prediction intervals from random forests. We first investigated the feasibility of using ideas from the Bag of Little Bootstraps to develop a procedure for obtaining accurate prediction intervals on predictions that are produced by random forests. We first discussed some of the most fundamental concepts behind BLB, namely sampling distributions and bootstrapping. We then contextualized these concepts within the larger picture of BLB, also introducing the specifics of the BLB algorithm and why it is such a powerful methodology.

Following this presentation of BLB and its characteristics, we provided a discussion on decision trees and various methods that employ them, namely bagging and random forests. We conveyed the specifics of the CART algorithm, which is the foundation for bagging and random forests. Following these discussions, we explored the specifics of implementing a BLB-like structure for random forests. It became clear that one of the more difficult aspects of this endeavor is to effectively implement a multinomial random variable into the tree construction component of the random forest, so as to mirror BLB's use of a multinomial random variable. We proposed a method for doing so.

Following this discussion of a potential BLB-like structure for random forests, we explored different ways to produce prediction intervals for two varieties of data, one generated from a linear model and the other generated using a Gaussian mixture model. We considered two approaches to obtaining prediction intervals for these data within simulation studies. The first method for prediction intervals was a naive approach of simply obtaining quantiles for the predictions produced from a large collection of random forests and CART trees. This method ignores any variability in training response at the nodes from which predictions are generated. Through simulation studies, we verified that this naive approach of constructing percentile intervals based on the random forest and CART predictions resulted in test response capture rates that were lower than would be expected for accurate prediction intervals. We proposed a second method for interval construction which aimed to resolve this issue, as it obtained prediction interval bounds (for random forests only) by examining the variability in training response for different nodes. We assessed the performance of these Within-Node Variability (WNV) intervals using additional simulations, and found that this new method produced intervals with far superior response capture performance compared to the original naive percentile approach.

The capture rates and appearance of the intervals (adjustment to the different test responses and the centers of the intervals) were vastly improved by moving from the naive method to our WNV intervals, but there was a distinct performance discrepancy between the intervals on the linear data and those on the clustered data. The clustered data exhibited lower-than-expected capture rates, which could possibly be due to our use of a low value for $numTrees$ in our simulation.

Future work could delve deeper into the parameter space of our simulations and WNV interval procedure in order to better elucidate the performance and suitability of such intervals. More specific investigations could entail examining the *per group* capture rate for the clustered data. Through considering different amounts of overlap in the clusters, such investigations could clarify if in fact certain characteristics specific to the clustered data are directly contributing to the lower-than-expected capture rates. Finally, in this thesis we merely proposed a BLB-like random forest implementation. Future investigations could be aimed at implementing our WNV intervals into our proposed BLB-like framework for random forests.

Bibliography

[1] Breiman, L. (2001). Random Forests. *Machine Learning.* 45(1), 5-32.

[2] Breiman, L., Friedman, J., Olshen, R., Stone, C. (1984).“Classification and Regression Trees”, Wadsworth.

[3] Efron, B., Tibshirani, R. (1994). “An Introduction to the Bootstrap”, Chapman & Hall.

[4] Hesterberg, Tim C. (2015) What Teachers Should Know About the Bootstrap: Resampling in the Undergraduate Statistics Curriculum, The American Statistician, 69(4), 371-386.

[5] James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani (2013). “An Introduction to Statistical Learning: With Applications in R”, Springer.

[6] Kleiner, A., Talwalkar, A., Sarkar, P., Jordan, M.I. (2014). A Scalable Bootstrap for Massive Data. *Journal of the Royal Statistical Society: Series B.* 76(4), 795-816.

[7] Mennitt, Daniel, Kirk Sherrill, and Kurt Fristrup (2014). A Geospatial Model of Ambient Sound Pressure Levels in the Contiguous United States. *The Journal of the Acoustical Society of America J. Acoust. Soc. Am.* 135(5), 2746-764.

[8] Volodymyr Melnykov, Wei-Chen Chen, Ranjan Maitra (2012). MixSim: An R Package for Simulating Data to Study Performance of Clustering Algorithms. *Journal of Statistical Software*, 51(12), 1-25. http://www.jstatsoft.org/v51/i12/.