DeVries Writing Project

YOUR TITLE HERE

Ben DeVries

Department of Mathematical Sciences
Montana State University

Date of completion here as Month Day, Year

A writing project submitted in partial fulfillment
of the requirements for the degree

Master of Science in Statistics

APPROVAL

of a writing project submitted by

Ben DeVries

This writing project has been read by the writing project advisor and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the Statistics Faculty.

Date	Andy Hoegh Writing Project Advisor
Date	Ian Laga Writing Projects Coordinator

This is my abstract. Note that I'm using line breaks without trailing white spaces in this .tex file to make version control work better. The XXXX is an important problem for researchers and this writing project was written to provide a solution to this important problem. It contains an exploration of the history of the problem, motivating its importance, an exploration of an application to illustrate its importance, development and/or discussion of statistical software to apply the solutions to the problem, a simulation study to explore the performance of the methods, and a conclusion. It also contains useful references and an appendix containing code used. Or maybe it doesn't do all this - it depends on what you do for your project.

Table of contents

1 Introduction **6**
1.1 Subsection 6

2 Background **6**
2.1 Supervised Learning 6
2.2 Decision Trees 9
2.3 Bayesian Trees 15

3 Methods **20**

4 Conclusion **20**

1 Introduction

Statistical Modelling: the Two Cultures [Breiman, 2001] describes two approaches to data problems; data, and algorithmic models. Data modelling is described as classical statistics where data is assumed to be generated by some stochastic process. For some problems, the true underlying process contains complex dynamics that may or may not be observable. In algorithmic modelling, we don't try to find the true model. Instead, the goal is to find a generalizable approximation. For this project we focus on applications of tree based models.

1.1 Subsection

This is a subsection in my Introduction section.

2 Background

2.1 Supervised Learning

With this project, we chose to focus on tree based methods. The Elements of Statistical Learning [Hastie et al., 2009] was a key source for understanding algorithms and concepts in statistical learning. Our backgroun discussion begins with some notation and concepts in supervised learning. We then discuss the algorithms for several models, using simulation to demonstrate there use. In supervised learning problems we work to map a set of features to a target. When discussing these components, we'll follow the notation used in The Elements of Statistical Learning [Hastie et al., 2009]. X denotes the features while Y and G denote the target for quantitative and categroical responses respectively. If there are N observations of p inputs, then $\mathbf{X}$ is an $N \times p$ matrix. The inputs for the i'th observation are $\mathbf{x}_i$ while $\mathbf{x}_j$ denotes the vector of all observations for the j'th input. A common approach in supervised learning is to assume our featurs and target are observation of random variables with joint distribution $\Pr(Y, X)$. G would replace Y in a classification problem and $\mathbf{X}$ would replace X for multiple covariates. We want to find a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that predicts Y reasonably well. To do this, we define a loss function $L(Y, f(X))$ which penalizes the prediction error on a training set $\;= (x_i, y_i),\; i = 1, 2, ..., N$. The expected prediction error (EPE) is defined as $\mathbb{E}[L(Y, f(X))]$. Our overall goal is to approximate the function f that minimizes the EPE. Using squared errors for loss (L_2), the EPE is minimized by choosing f such that $f(x) = \mathbb{E}[Y|X = x]$ (The Elments of Statistical Learning, pg. 18 [Hastie et al., 2009]). Now we need an algorithm to approximate f using . Choosing an appropriate algorithm will be heavily influenced by the availability and distribution of data. With a quantitative target, we might be able to assume an additive error relationship between Y and $\mathbf{X}$. The additive error model is simply $Y = f(\mathbf{X}) + \epsilon$ where $\mathbb{E}[\epsilon] = 0$ and $\mathbf{X} \perp \epsilon$. Here we are assuming the process that generated Y can be approximated using $\mathbf{X}$ such that the error due to ignored features and other factors has mean zero. For data that was generated under an additive error model, a natural approach to approximate f is to use a sum of basis functions. The additive model is defined as $Y = \beta_0 + \sum_{j=1}^{p} f_j(\mathbf{x}_j) + \epsilon$. In this model, each f_j could be a sum of basis functions. The major limitation to an additive model is we assume no interactions between features. There's no reason we can't add interactions or functions of multiple covariates, but identifying these interactions is difficult. We might be able to fit a model with select select transformations of covariates and every possible interaction, but this would almost certainly result in overfitting. Assuming a generating model with any form of noise implies even the optimal f will not predict every observation perfectly. A model is overfit when global trends are ignored to explain slight irregularities in a sample. Discerning irregularities is highly subjective. Any decision we make based on observed data may lead to overfitting. Remember our overall goal is to minimize the EPE but unfortunately the EPE is not quantifiable as we do not know the true distribution of $Y|X$. Like any other problem in statistics concerning an unknown quantity, the solution is to use estimates based on a sample. To obtain our estimate, we use some form of cross validation. There are a variety of methods to perform cross validation, but the two most common are hold out and K-fold. In hold out, some portion of the data is held out for testing and the rest is used to fit the model. If the model contains hyper parameters, we could split the data into training, validation, and testing. Fitting models to the training set, we can try various values for hyper parameters and compare model performance for predictions on the validation set. After we determine the optimal hyper parameter values, we refit the model on the combined training and validation set before using the testing set to assess the final fit. Using training/testing splits is generally favored for computationally intensive problems with many observations. The hold out method gives us a single sample to estimate the prediction error. In K-fold cross validation we partition the observations into K folds, resulting in K samples of size n/K. A model is fit using all but one fold which is held for testing. The process is then repeated until all folds and thus all observations have been tested. More observations generally implies a better estimate, but refitting models may change estimates. K-fold does not estimate the prediction error of the final estimated model, but still provides some idea of how the model will perform on new data. K-fold is an excellent tool for tuning models prior to testing. So far we have discussed a general approach to problems centered around prediction. In the next subsection we explore the intuition behind two algorithms to approximate f, decision trees and random forests. Later, we turn our attention to tree based models fit with Bayesian methods. Our discussion of Bayesian tree models is primarily centered around Bayesian Additive Regression Trees (BART), but a discussion of predecessors is included.

2.2 Decision Trees

Decision trees are a type of hierarchical model where the the feature space is partitioned into hyperrectangular regions, and a model is fit for each region. For regression tasks, we typically use mean only models. In this case the decision tree model is a step function. Unlike our mean only models for linear regression, we do not make any distributional assumptions. Without a distribution, we can't quantify the uncertainty in our predictions or the decision boundaries. Additionally, the binary structure of trees makes the effects of individual predictors less interpretable than a linear model. Small changes in a feature $x_{i,j}$ may not effect the associated prediction $\hat{y}_i$. On the other hand, trees greatly outperform linear models with more complex data. The hierarchical structure naturally handles multicollinearity, and can help us find interactions between covariates. Even though we don't believe $\mathbf{X}$ is related to $\mathbf{y}$ by a step function, we can gain insight to the true relationship by a step function approximation. An example of a tree model is shown in Figure 1. Every decision after the top root node is conditioned on the previous decisions, allowing us to display the hierarchy graphically.

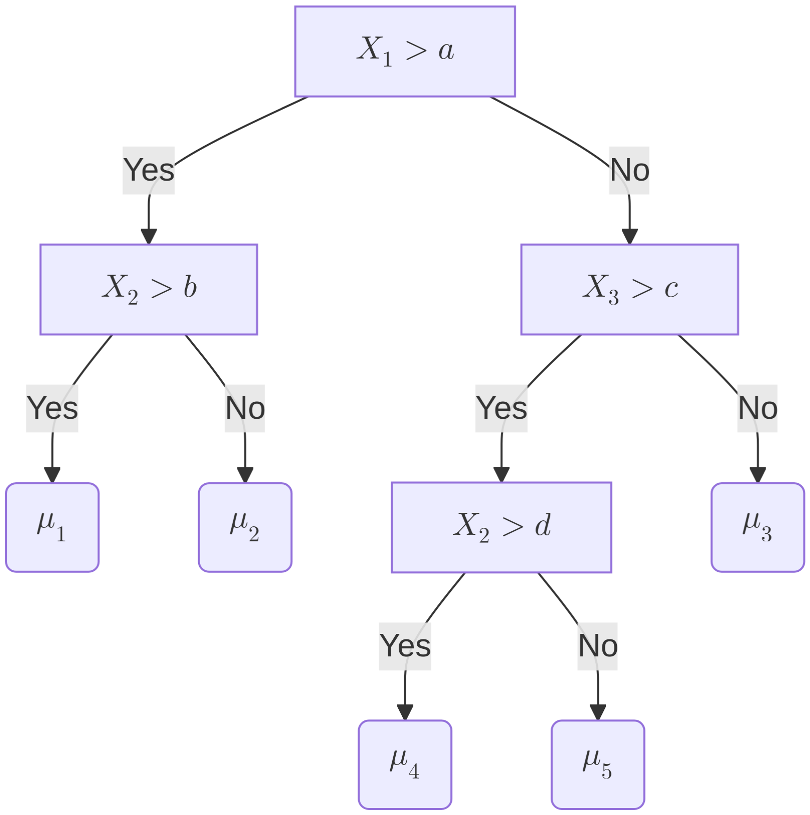


Figure 1: Visualization of binary decision tree

Step functions are very flexible and for any training set $\mathbf{X}$ with target Y where

$y_i|\mathbf{X}_i = y_j|\mathbf{X}_i$ for all i, j, we can fit a tree that will have no error on the training set. With this flexibility, over fitting is a major concern. We can control the overall size and shape of trees with three hyper-parameters. First is the complexity parameter α. α scales the penalty which is based on the number of terminal nodes (mean only models). Next we have the minimum split and minimum bucket sizes. The minimum bucket is the fewest number of observations allowed in a terminal node. Similarly, the minimum split is the least number of observations allowed in one outcome of a binary decision. We generally choose the complexity parameter by growing the largest possible tree, then removing terminal nodes with fewer observations. Next we compare the trees via cross validation and choose the tree with minimal cross validated error. This process of choosing an appropriate tree size is commonly referred to as pruning. While pruning helps reduce the risk of over fitting, large regression trees are inherently unstable. We'll demonstrate this with data simulated under the model described below. A simulated testing data set displaying the true function along with the fit from a tree chosen by cross validation on training is displayed in Figure 2. Figure 3 displays the number of splits for the optimal complexity parameter chosen by cross validation.

$$y_i = \cos(108t_i \sin(108t_i + w_i)) \cdot \exp(\sin(108t_i \cos(108t_i + x_i))) + z_i$$

$$W_i, X_i, Z_i \sim N(0, \sigma^2)$$

$$t_i = -\frac{\pi}{18}, ..., \frac{\pi}{18}$$

$$i = 1, 2, ..., 350$$

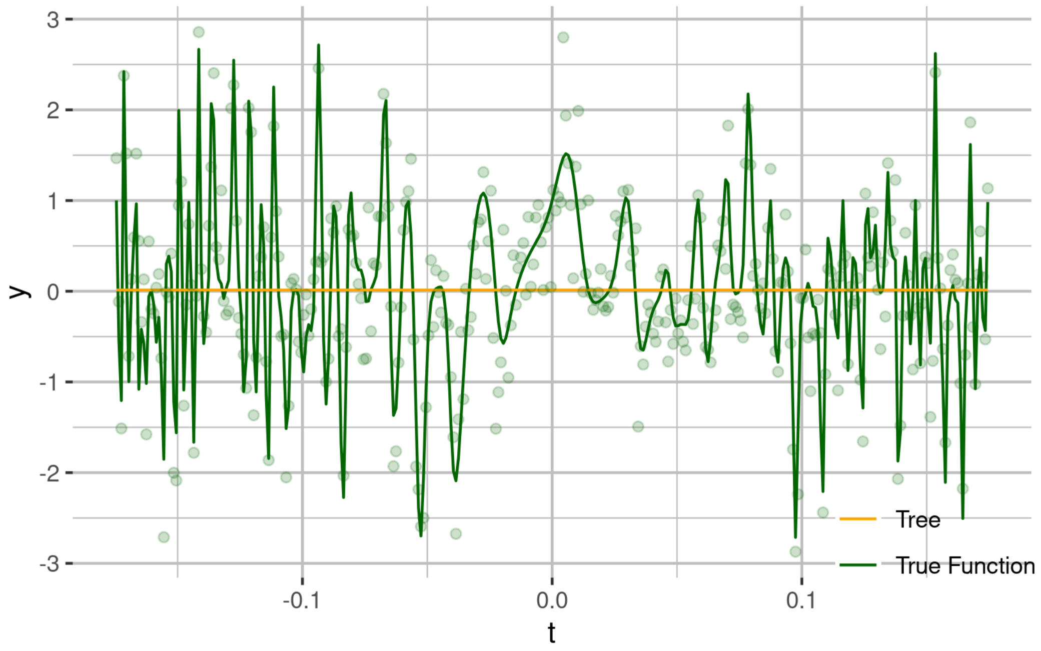


Figure 2: Visualization of simulated data and predictions

Decision trees are very unstable for this data. Across all three noise levels, the number of splits chosen by cross validation ranges from zero to over 300. The variability in the size/structure of trees suggest we can't have much faith in predictions from an individual tree. This led to the development of the random forest algorithm by Leo Breiman and Adele Cutler. Random forests are an adaptation of bagging. Bagging is simply averaging the predictions of models fit to bootstrapped samples. This reduces the variability of predictions at the cost of bias. The random forest algorithm randomly selects subsets of all the available predictors and fits decision tree to bootstrapped samples. A fitted value $y_i|\mathbf{x}_i$ is obtained by averaging predictions from all trees where $\mathbf{x}_i$ was not in the bootstrap sample. Combining many trees, our estimated model is still a step function, but appears

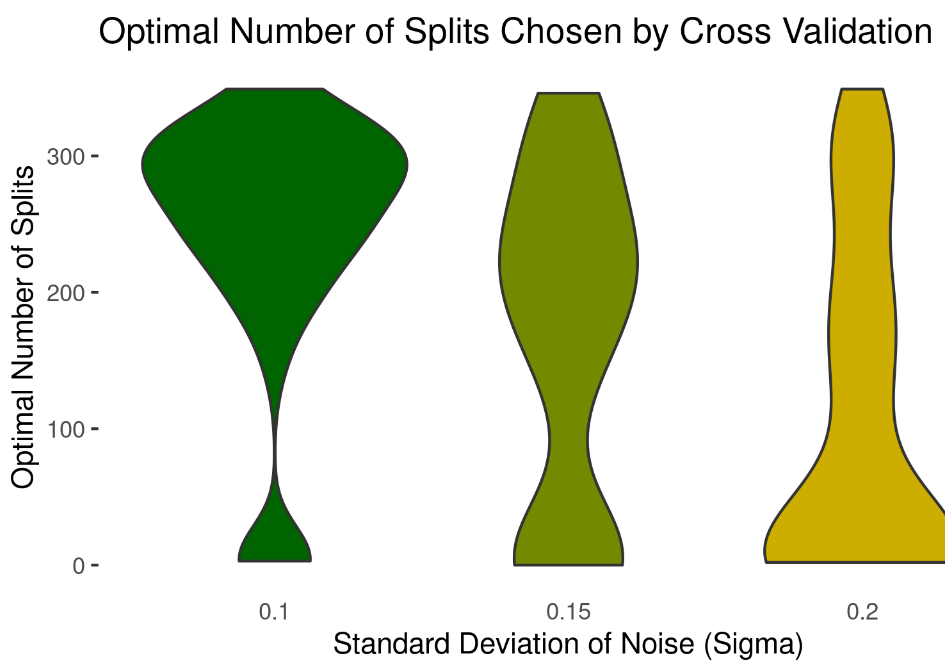


Figure 3: Violin plots of optimal number of splits for simulated data

much smoother than trees. This leads to more stable predictions as seen in Figure 5.

We see considerably less variation in the mean squared error of random forests fit to the same data as individual decision trees, but single tree models generally outperformed a random forest. Random forests are designed for data with several covariates. With a single predictor, the random forest algorithm degenerates to bagging with out of bag predictions. Bagging reduces the variance of predictions at the cost of bias, and thus the "random forest" predictions were often worse than predictions from individual trees. Pages 600, 601 of [Hastie et al., 2009] display a proof showing that the bias of a random forest is the same as the bias of any individual tree in the ensemble. The randomization in the random forest algorithm implies trees within the ensemble are almost certainly smaller than a single tree fit

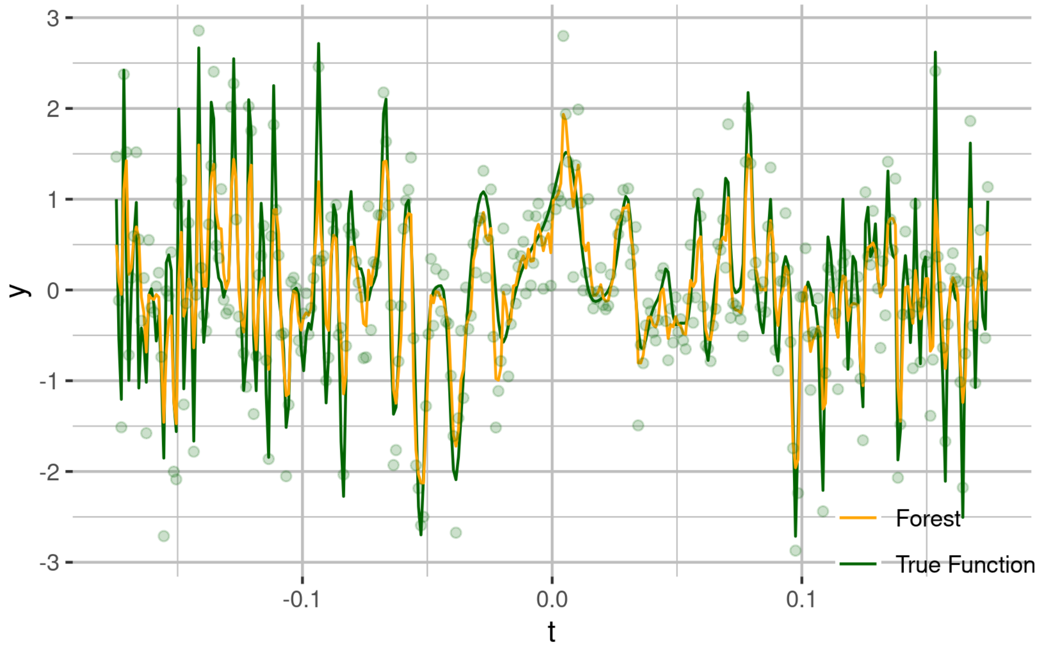


Figure 4: Violin plots of MSE for tree and forest models on simulated data

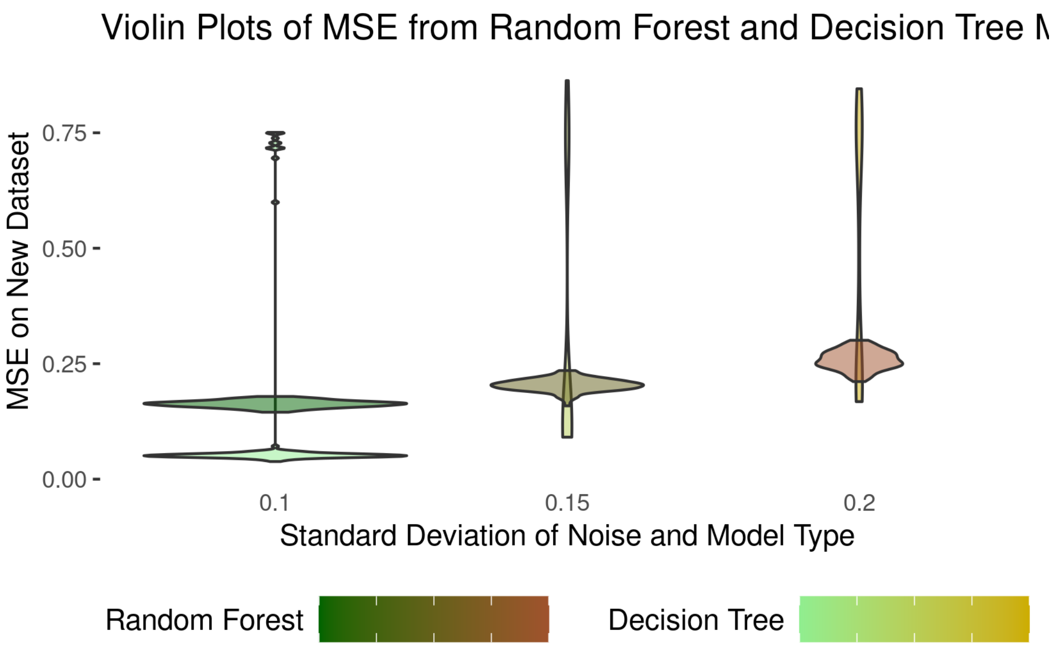


Figure 5: Violin plots of MSE for tree and forest models on simulated data

to the data. In cases where we have access to several predictors, large trees will create complex hierarchies that are unlikely to exist in the true model. The following simulation study shows the random forest algorithm consistently outperform a single tree for data generated under a linear model with correlated covariates. Violin plots of MSE on testing data are displayed in Figure 6.

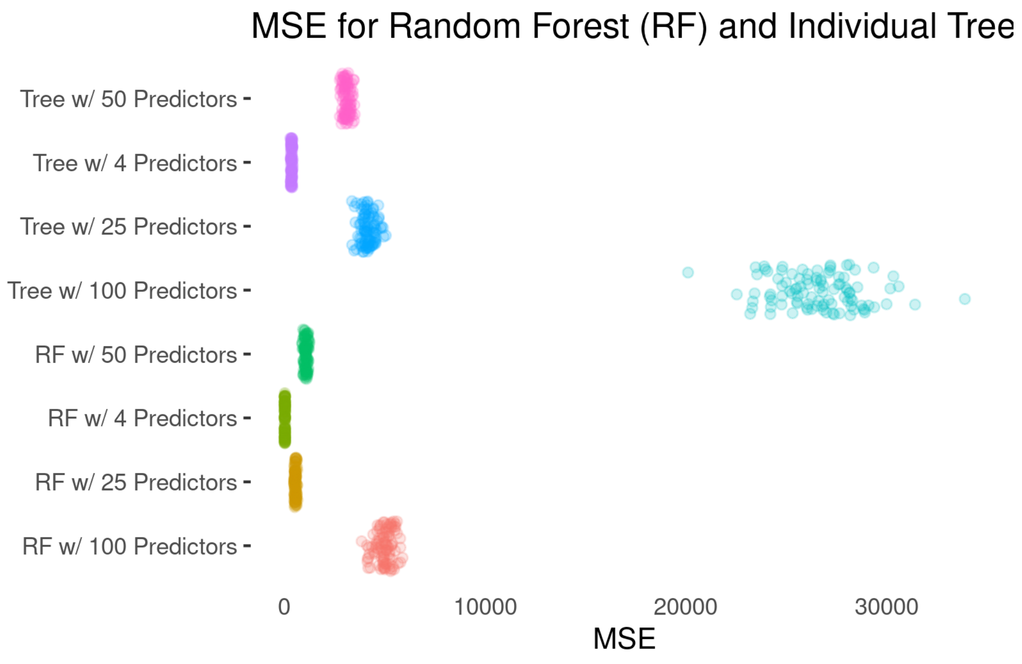


Figure 6: Violin plots of MSE for tree and forest models on linear model with several predictors

2.3 Bayesian Trees

In the current and following section we discuss Bayesian methods for tree algorithms, beginning with Bayesian Classification And Regression Trees. Here, we briefly discuss the Bayesian approach to a regression tree. There are many ways to specify a Bayesian tree, but we'll focus on models developed by doctors Hugh

Chipman, Edward George, and Robert Mcculoch. Chipman et al.s ## Bayesian Additive Regression Trees (BART) The other learning algorithm we explored was BART, or Bayesian Additive Regression Trees. Introduced by Chipman et al [Chipman et al., 2010], BART is a flexible nonparametric regression method that combines the strengths of decision trees and Bayesian modeling. For a continuous predictor, BART assumes $y_i \sim N(\sum_{b=1}^{m} g(\mathbf{x}_i; T_b, M_b), \sigma^2)$ where T_b is the b'th tree, M_b are the associated terminal nodes, and $g(\mathbf{x}_i; T_b, M_b)$ is the function that assigns $\mathbf{x}_i$ to μ_{lb}, $l \in \{1, 2, ..., |M_b|\}$. Unlike the random forest algorithm, the trees are not bagged and each tree has access to each feature. The algorithm starts by growing M trees with a single terminal node (stumps). Gibbs sampling is used to iteratively update each tree conditioned on all other trees. Highly informative priors prevent any single tree from dominating the model. BART assumes independent priors for the trees and standard deviation. The joint prior is given by

$$p((T_1, M_1), (T_2, M_2), ..., (T_m, M_m), \sigma^2) = p(\sigma^2) \prod_{b=1}^{m} p(M_b|T_b) p(T_b)$$

where $p(M_b|T_b) = \prod_{l=1}^{|M_b|} p(\mu_{lb}|T_b)$, and $p(\sigma^2) \sim \nu\lambda/\chi^2_\nu$ with hyperparameters ν, λ. The hyper parameters are selected by first choosing a point estimate $\hat{\sigma}$. By default, this is the residual standard deviation for a multiple linear regression model. ν is then fixed ($\nu \in [3, 10]$ reccomended) and λ solved for by imposing the constraint $\Pr(\sigma < \hat{\sigma}) = q$, where q is an additional hyper parameter. In the prior for a tree $p(T_b)$, the probability that a node at depth $d \in \mathbb{N}$ is non-terminal is given by $\alpha(1 + d)^{-\beta}$, $\alpha \in (0, 1)$, $\beta \in [0, \infty)$. A discrete uniform prior imposes the initial belief that each feature equally likely to be selected for a nodes splitting rule. Similarly, a discrete uniform across the observed values of the selected predictor serves as the prior for the cut point in the binary decision. A $N(|M|\mu_\mu, |M|\sigma_\mu^2)$ prior is assumed for each $\mu_{lb}|T_b$ where $|M|$ is the total number of terminal nodes across all trees. The hyperparameters μ_μ and σ_μ are chosen based on the data such that $\min(Y) = |M|\mu_\mu - k\sqrt{|M|}\sigma_m u$ and $\max(Y) = |M|\mu_\mu - k\sqrt{|M|}\sigma_\mu$. In [Chipman et al., 2010], they reccomend choosing $k \in [1, 3]$. The BART R package rescales Y such that $Y \in [-0.5, 0.5]$ and chooses $\mu_\mu = 0 \implies \sigma_\mu = \frac{0.5}{k\sqrt{|M|}}$. One downside to this is we are no longer utilizing out of bag predictions so we need a testing set to quantify model performance. We also note that BART is not a fully bayesian model as the number of trees m is fixed and data is used to inform priors. Even if the model isn't fully Bayesian, we can leverage the MCMC samples to obtain a posterior predictive distribution. Hyper parameters can then be tuned using a testing set with the goal of bringing prediction intervals to the nominal level. Another major advantage to BART is the trees are smaller than in random forests due to the regularizing prior. While BART is still a black box, we may be able to draw potential associations between the features and target. Small trees are easy to interpret and the posterior distribution of μ's provides probabilities from which we can infer the importance of features. An example of BART fit to the same simulated data used in Figure 2 is displayed in Figure 7. Choosing $m = 250, k = 0.6, \alpha = 0.99, \beta = 0.5, \hat{\sigma} = 2\sqrt{\mathbb{V}[Y]}$ resulted in a 95% prediction interval within 1% of nominal coverage on the testing data for which parameters were tuned.

```
[1] 0.3415504
```

```
[1] 0.3848066
```

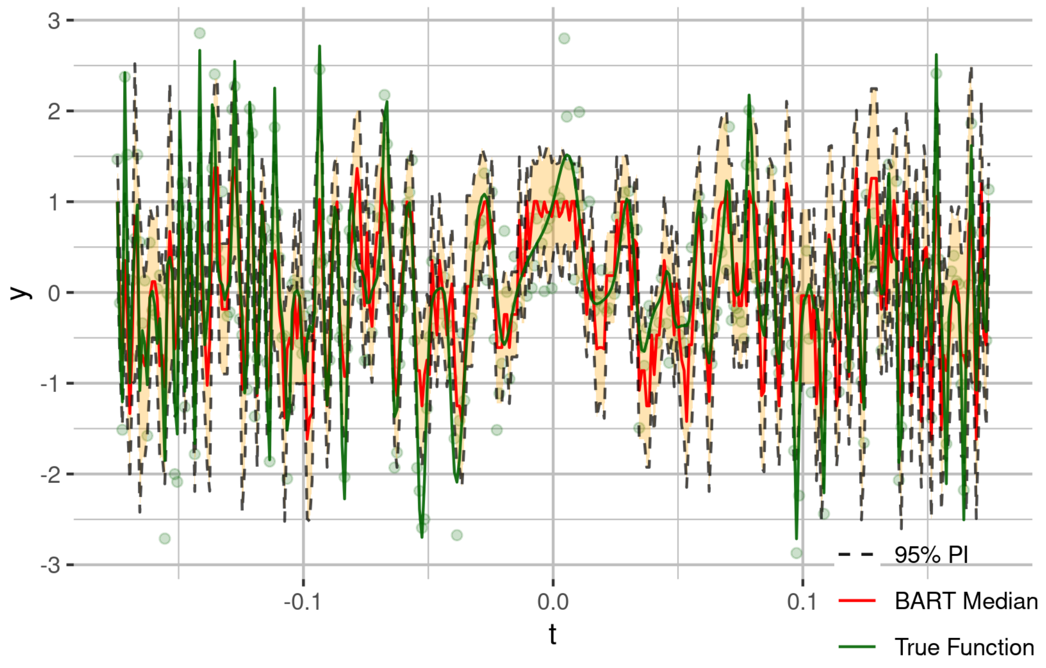
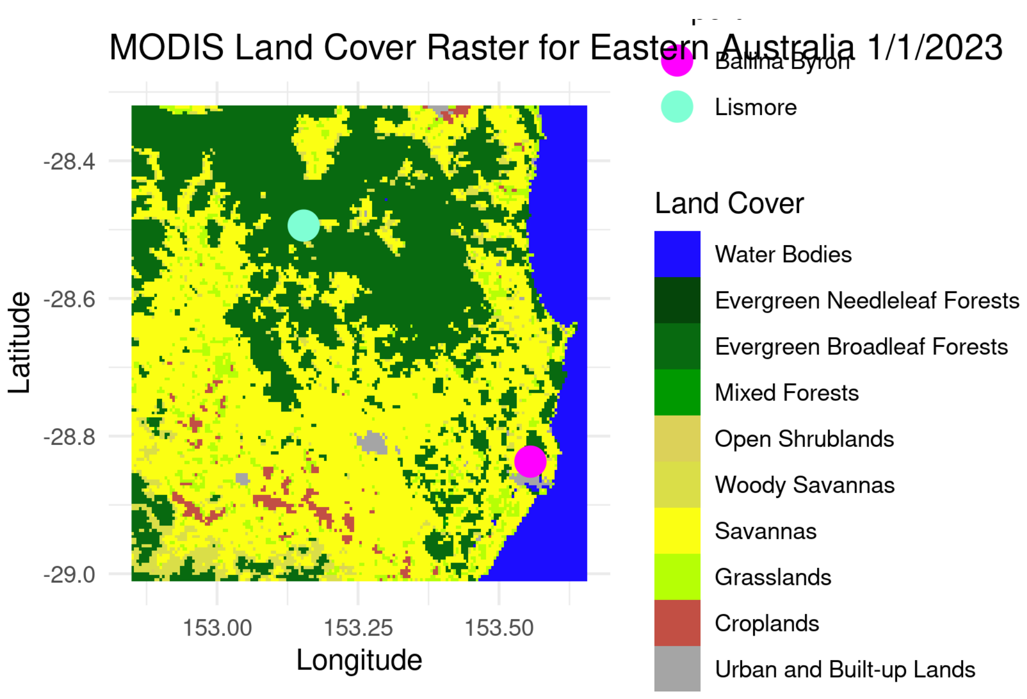


Figure 7: Visualization of bart

```r
modis <- raster("~/Downloads/modis_landcover2023.tif")

modis_df <- as.data.frame(modis, xy = T) |>

  mutate(land_cover = factor(LC_Type2, labels = c(

    "Water Bodies", "Evergreen Needleleaf Forests", "Evergreen Broadleaf Forests",

    "Mixed Forests", "Open Shrublands", "Woody Savannas", "Savannas", "Grasslands",

    "Croplands", "Urban and Built-up Lands"



  )))

modis_pal <- c(
  "Water Bodies" = "#1c0dff",
  "Evergreen Needleleaf Forests" = "#05450a",
  "Evergreen Broadleaf Forests" = "#086a10",
  "Deciduous Broadleaf Forests" = "#78d203",
  "Mixed Forests" = "#009900",
  "Closed Shrublands" = "#c6b044",
  "Open Shrublands" = "#dcd159",
  "Woody Savannas" = "#dade48",
  "Savannas" = "#fbff13",
  "Grasslands" = "#b6ff05",
  "Croplands" = "#c24f44",
  "Urban and Built-up Lands" = "#a5a5a5",
  "Non-Vegetated Lands" = "#f9ffa4"
)
ggplot() +
  geom_raster(aes(x = x, y = y, fill = land_cover),
              data = modis_df) +
  scale_fill_manual(name = "Land Cover", values = modis_pal) +
  geom_point(aes(x = 153.5557, y = -28.8371, color = "Ballina Byron"), size = 5) +
  geom_point(aes(x = 153.1536, y = -28.4941, color = "Lismore"), size = 5) +
  scale_color_manual(name = "Airport", values = c("magenta", "aquamarine")) +
  labs(title = "MODIS Land Cover Raster for Eastern Australia 1/1/2023",
      x = "Longitude", y = "Latitude") +

  theme_minimal()
```



3 Methods

4 Conclusion

Amazing conclusions will be described in this section.

References

Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3), Aug 2001. doi:

10.1214/ss/1009213726.

Hugh A. Chipman, Edward I. George, and Robert E. McCulloch. Bart: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1), Mar 2010. doi: 10.1214/09-aoas285.

Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The elements of Statistical Learning: Data Mining, Inference, and prediction.* Springer, 2 edition, 2009.