**ILLINOIS INSTITUTE OF TECHNOLOGY**

**10 W 35th St, Chicago, IL 60616**


# MODEL COMPARISON BETWEEN BART AND CART

Project Members

Saptarshi Maiti (A20447671)

Niti Wattanasirichaigoon (A20406934)

Under the guidance of

**Professor Lulu Kang**

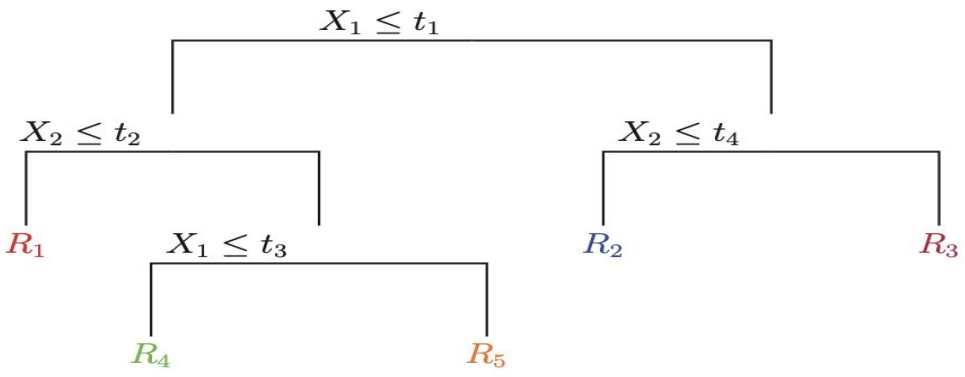
# 1. Introduction

This paper compares the performances between Bayesian Adaptive Regression Models and Classification And Regression Trees Models. In this paper, we will be discussing why BART can outperform CART models in terms of accuracy, and also why BART is less prone to overfitting.

The main difference in the algorithms between the two models is that the BART follows Bayesian statistics and CART follows Frequentist Statistics. In Bayesian, parameters of the model are considered as random and data as fixed whereas in frequentist approach parameters are considered as fixed and data as random. So, in the Bayesian approach, we consider the distribution of parameters in form of prior and posterior; and in the frequentist approach, we only consider distributions in data but not in parameters.

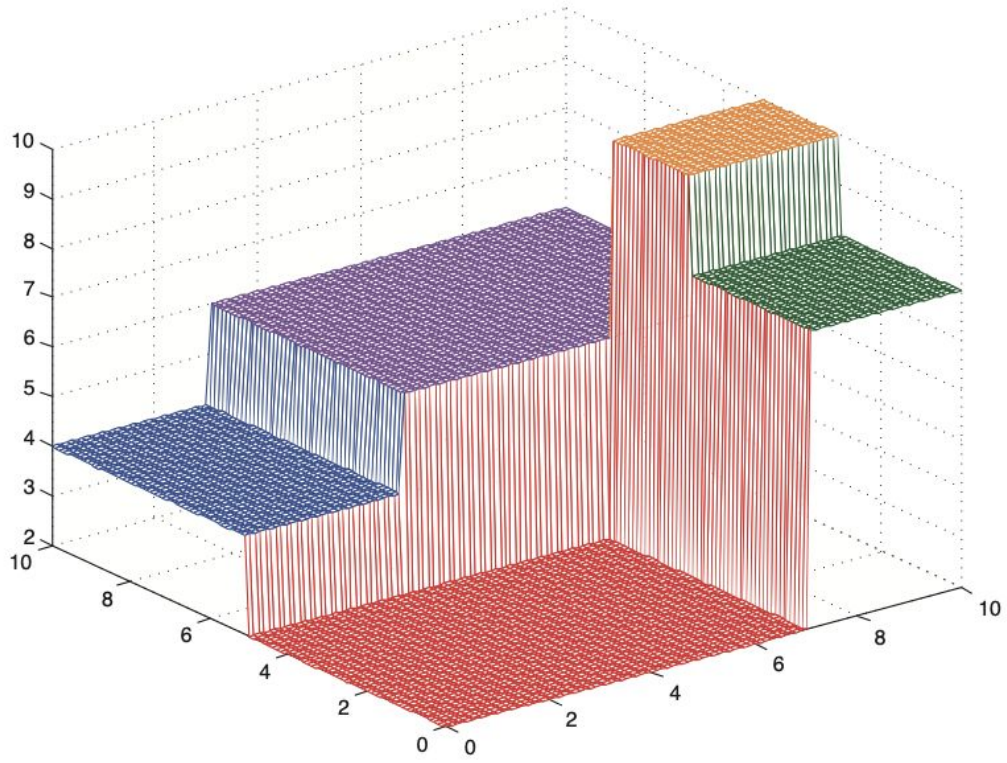
# 2. Classification and Regression Trees Overview

Classification and Regression Trees are also known as decision trees, defined by recursively partitioning the input space, and defining a local model in each resulting region. To explain the CART approach, consider the tree in below figure:




The first node asks if $x_1$ is less than some threshold $t_1$. If yes, we then ask if $x_2$ is less than some other threshold $t_2$. If yes, we are in the bottom left quadrant of space, $R_1$. If no, we ask if $x_1$ is less

than $t_3$ and so on. The result of these axis parallel splits partitions the 2-dimensional space into 5 regions, as shown in figure below:




We can now associate a mean response with each of these regions, resulting in the piecewise constant surface. We can write the model in the following form

$$f(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}] = \sum_{m=1}^{M} w_m \mathbb{I}(\mathbf{x} \in R_m) = \sum_{m=1}^{M} w_m \phi(\mathbf{x}; \mathbf{v}_m)$$

where $\varphi(x;v_m)$ is the basis function, $R_m$ is the m'th region, $w_m$ is the mean response in this region, and $v_m$ encodes the choice of variable to split on, and the threshold value, on the path from the root to the m'th leaf. This makes it clear that a CART model is just like an adaptive basis-function model as shown below:

$$f(\mathbf{x}) = w_0 + \sum_{m=1}^{M} w_m \phi_m(\mathbf{x})$$

where these basis functions define the regions, and the weights specify the response value in each region. We can generalize this to the classification setting by storing the distribution over class labels in each leaf, instead of the mean response.

**Growing a tree:** Finding the optimal partitioning of the data is NP-complete (Hyafil and Rivest 1976), so it is common to use the greedy algorithm to compute a locally optimal MLE. This method is used by CART, (Breiman et al. 1984) C4.5(Quinlan 1993), and ID3 (Quinlan 1986), which are three popular implementations of the method.

The split function chooses the best feature, and the best value for that feature, as follows:

$$(j^*, t^*) = \arg \min_{j \in \{1,\ldots,D\}} \min_{t \in \mathcal{T}_j} \text{cost}(\{\mathbf{x}_i, y_i : x_{ij} \leq t\}) + \text{cost}(\{\mathbf{x}_i, y_i : x_{ij} > t\})$$

where the cost function for a given dataset will be defined below. For notational simplicity, we have assumed all inputs are real-valued or ordinal, so it makes sense to compare a feature $x_{ij}$ to a numeric value t. The set of possible thresholds $T_j$ for feature j can be obtained by sorting the unique values of $x_{ij}$ . For example, if feature 1 has the values {4.5, −12, 72, −12}, then we set $T_1$ = {−12, 4.5, 72}. In the case of categorical inputs, the most common approach is to consider splits of the form $x_{ij} = c_k$ and $x_{ij} \neq c_k$, for each possible class label $c_k$. Although we could allow for multi-way splits (resulting in non-binary trees), this would result in data fragmentation, meaning too little data might "fall" into each subtree, resulting in overfitting.

**Regression Cost:**

$$\text{cost}(\mathcal{D}) = \sum_{i \in \mathcal{D}} (y_i - \overline{y})^2$$

where $\bar{y} = \frac{1}{|D|} \sum_{i \varepsilon D} y_i$ is the mean of the response variable in the specified set of data.

Alternatively, we can fit a linear regression model for each leaf, using as inputs the features that were chosen on the path from the root, and then measure the residual error.

**Classification Cost:** In the classification setting, there are several ways to measure the quality of a split. First, we fit a multinoulli model to the data in the leaf satisfying the test Xj < t by estimating the class-conditional probabilities as follows:

$$\hat{\pi}_c = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}(y_i = c)$$

where D is the data in the leaf. Given this, there are several common error measures for evaluating a proposed partition:

- **Misclassification rate**. We define the most probable class label as $y_c = argmax\ \pi_c$. The corresponding error rate is then

$$\frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}(y_i \neq \hat{y}) = 1 - \hat{\pi}_{\hat{y}}$$

- **Entropy,** or **deviance**:

$$\mathbb{H}(\hat{\boldsymbol{\pi}}) = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c$$

  Note that minimizing the entropy is equivalent to maximizing the information gain (Quinlan 1986) between test Xj < t and the class label Y , defined by

$$\begin{aligned}\text{infoGain}(X_j < t, Y) &\triangleq \mathbb{H}(Y) - \mathbb{H}(Y|X_j < t) \\ &= \left(-\sum_c p(y=c)\log p(y=c)\right) \\ &\quad + \left(\sum_c p(y=c|X_j < t)\log p(c|X_j < t)\right)\end{aligned}$$

since $\hat{\pi}_c$ is an MLE for the distribution $p(c|X_j < t)$.

- **Gini index**

$$\sum_{c=1}^{C} \hat{\pi}_c(1-\hat{\pi}_c) = \sum_c \hat{\pi}_c - \sum_c \hat{\pi}_c^2 = 1 - \sum_c \hat{\pi}_c^2$$

This is the expected error rate. To see this, note that $\pi_c$ is the probability a random entry in the leaf belongs to class c, and $(1 - \pi_c)$ is the probability it would be misclassified.

**Pruning a tree:** To prevent overfitting, we can stop growing the tree if the decrease in the error is not sufficient to justify the extra complexity of adding an extra subtree. However, this tends to be too myopic.

The standard approach is therefore to grow a "full" tree, and then to perform pruning. This can be done using a scheme that prunes the branches giving the least increase in the error.

To determine how far to prune back, we can evaluate the cross-validated error on each such subtree, and then pick the tree whose CV error is within 1 standard error of the minimum.

**Pros and cons of trees:** CART models are popular for several reasons: they are easy to interpret, they can easily handle mixed discrete and continuous inputs, they are insensitive to monotone transformations of the inputs (because the split points are based on ranking the data points), they perform automatic variable selection, they are relatively robust to outliers, they scale well to large data sets, and they can be modified to handle missing inputs.

However, CART models also have some disadvantages. The primary one is that they do not predict very accurately compared to other kinds of models. This is in part due to the greedy nature of the tree construction algorithm. A related problem is that trees are unstable: small changes to the input data can have large effects on the structure of the tree, due to the hierarchical nature of the tree-growing process, causing errors at the top to affect the rest of the tree. In frequentist terminology, we say that trees are high variance estimators. And, the solution to this problem is **Random Forest** and **Boosting**.

## 2.1 Random Forests

In random forests, we build a number of decision trees from bootstrapped training samples. But when building these decision trees, only a random subset of the predictors are used during consideration at each split. A fresh sample of m predictors is taken at each split, and typically we choose $m \approx \text{sqrt}(p)$ that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors. This allows weaker predictors to have more chance of being considered during a split, and making the random forest model able to generalize better. The final prediction will be the average or the majority vote (depending on if the problem is regression or classification) of all the trees.

## 2.2 Boosting

Boosting on the other hand, grows multiple trees sequentially - each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set. The series of slow learners allow the model to perform very well. Boosting has three tuning parameters: the number of trees $B$, the shrinkage parameter $\lambda$ (small positive number), and the number $d$ of splits in each tree. The tuning parameters can be selected via cross-validation to prevent over fitting.

# 3. Bayesian Adaptive Regression Trees Overview

Bayesian additive regression trees (BART; Chipman, George, and McCulloch 2010), which depart from predecessors in that they rely on an underlying Bayesian probability model rather than a pure algorithm. BART has demonstrated substantial promise in a wide variety of simulations and real world applications such as predicting avalanches on mountain roads (Blattenberger and Fowles 2014), predicting how transcription factors interact with DNA (Zhou and Liu 2008) and predicting movie box office revenues (Eliashberg 2010).

BART is a Bayesian approach to nonparametric function estimation using regression trees. BART can be considered a sum-of-trees ensemble, with a novel estimation approach relying on a fully Bayesian probability model. Specifically, the BART model can be expressed as:

$$\boldsymbol{Y} = f(\boldsymbol{X}) + \boldsymbol{\mathcal{E}} \approx \mathcal{T}_1^{\mathcal{M}}(\boldsymbol{X}) + \mathcal{T}_2^{\mathcal{M}}(\boldsymbol{X}) + \ldots + \mathcal{T}_m^{\mathcal{M}}(\boldsymbol{X}) + \boldsymbol{\mathcal{E}}, \qquad \boldsymbol{\mathcal{E}} \sim \mathcal{N}_n\left(\mathbf{0}, \sigma^2 \boldsymbol{I}_n\right)$$

where Y is the n×1 vector of responses, X is the n×p design matrix (the predictors column-joined) and E is the n × 1 vector of noise. Here we have m distinct regression trees, each composed of a tree structure, denoted by T, and the parameters at the terminal nodes (also called leaves), denoted by M. The two together, denoted as TM represents an entire tree with both its structure and set of leaf parameters.

The structure of a given tree $T_t$ includes information on how any observation recurses down the tree. For each nonterminal (internal) node of the tree, there is a "splitting rule" taking the form $x_j < c$, which consists of the "splitting variable" $x_j$ and the "splitting value" c. An observation moves to the left child node if the condition set by the splitting rule is satisfied (and it moves to the right child node otherwise). The process continues until a terminal node is reached. Then, the observation receives the leaf value of the terminal node. BART can be distinguished from other ensemble-of-trees models due to its underlying probability model. As a Bayesian model, BART consists of a set of priors for the structure and the leaf parameters and a likelihood for data in the terminal nodes. The aim of the priors is to provide regularization, preventing any single regression tree from dominating the total fit.

The prior for the BART model has three components: (1) the tree structure itself, (2) the leaf parameters given the tree structure, and (3) the error variance $\sigma^2$ which is independent of the tree structure and leaf parameters.

$$\begin{aligned}\mathbb{P}\left(\mathcal{T}_1^{\mathcal{M}},\ldots,\mathcal{T}_m^{\mathcal{M}},\sigma^2\right) &= \left[\prod_t \mathbb{P}\left(\mathcal{T}_t^{\mathcal{M}}\right)\right]\mathbb{P}\left(\sigma^2\right) = \left[\prod_t \mathbb{P}\left(\mathcal{M}_t \mid \mathcal{T}_t\right)\mathbb{P}\left(\mathcal{T}_t\right)\right]\mathbb{P}\left(\sigma^2\right) \\ &= \left[\prod_t \prod_\ell \mathbb{P}\left(\mu_{t,\ell} \mid \mathcal{T}_t\right)\mathbb{P}\left(\mathcal{T}_t\right)\right]\mathbb{P}\left(\sigma^2\right),\end{aligned}$$

We first describe P ($T_t$), the component of the prior which affects the locations of nodes within the tree. Node depth is defined as distance from the root. Thus, the root itself has depth 0, its first child node has depth 1, etc. Nodes at depth d are nonterminal with prior probability $\alpha(1+d)^{-\beta}$ where $\alpha \in (0, 1)$ and $\beta \in [0, \infty]$. This component of the tree structure prior has the ability to enforce shallow tree structures, thereby limiting complexity of any single tree and resulting in more model regularization. Default values for these hyperparameters of $\alpha = 0.95$ and $\beta = 2$ are recommended by Chipman et al. (2010). For nonterminal nodes, splitting rules occur in two parts. First, the predictor is randomly selected to serve as the splitting variable. In the original formulation, each available predictor is equally likely to be chosen from a discrete uniform distribution with probability that each variable is selected with probability 1/p. This is relaxed in our implementation to allow for a generalized Bernoulli distribution where the user specifies $p_1, p_2,..., p_p$ (such that $\sum p_j=1$), where each denotes the probability of the jth variable being selected a priori. And, the prior component $P(M_t \mid T_t)$ which controls the leaf parameters. Finally, the Metropolis-within-Gibbs sampler (Geman and Geman 1984; Hastings 1970) is employed to generate draws from the posterior distribution of $P(T_1^M,...,T_m^M, \sigma^2 \mid y)$.

BART can easily be modified to handle classification problems for categorical response vari- ables. In Chipman et al. (2010), only binary outcomes were explored but recent work has extended BART to the multiclass problem (Kindo, Wang, and Pe 2013). Our implementa- tion handles binary classification and we plan to implement multiclass outcomes in a future release.

For the binary classification problem (coded with outcomes “0” and “1”), we assume a probit model,

$$\mathbb{P}\left(Y = 1 \mid \boldsymbol{X}\right) = \Phi\left(\mathcal{T}_1^{\mathcal{M}}(\boldsymbol{X}) + \mathcal{T}_2^{\mathcal{M}}(\boldsymbol{X}) + \ldots + \mathcal{T}_m^{\mathcal{M}}(\boldsymbol{X})\right)$$

where $\Phi$ denotes the cumulative density function of the standard normal distribution. In this formulation, the sum-of-trees model serves as an estimate of the conditional probit at x which can be easily transformed into a conditional probability estimate of Y = 1.

In the classification setting, the prior on $\sigma^2$ is not needed as the model assumes $\sigma^2 = 1$. The prior on the tree structure remains the same as in the regression setting and a few minor modifications are required for the prior on the leaf parameters. Sampling from the posterior distribution is again obtained via Gibbs sampling in combination with a Metropolis-Hastings step.

# 4. Methodology

We applied different decision trees from CART and BART over the same dataset and compared their performances. The models include 1) Decision Tree, 2) Random Forest, 3) Xgboost, and 4) BART. The pruned tree was not included, as it yielded the exact same tree as the Decision Tree.

## 4.1 Hotel Booking Demand Dataset

The dataset that we chose to use in this study is the Hotel Booking Demand Dataset. It consists of two datasets H1(resort hotels) and H2 (city hotels) with the same structure, 31 variables describing the 40,060 observations of H1 and 79,330 observations of H2. Each observation represents a hotel booking. Both datasets comprehend bookings due to arrive between the 1st of July of 2015 and the 31st of August 2017, including bookings that effectively arrived and bookings that were cancelled.

## 4.2 Model Fitting

We used the H1 dataset as our training set and H2 as our testing set to see how the trees perform on large unseen datasets. We split the training data again into the training and validation set with a 80:20 ratio and used the training set to train our models. The 7 fields used as predictors include: Lead Time, Country, Market Segment, Deposit Type, Customer Type, Required Car Parking Spaces, and Arrival Date Week Number. And the output label ‘IsCanceled’, a binary indicator of whether the hotel booking has been cancelled or not. We then evaluated the performance of each model on the validation and the test set and compared their error rates.

# 5. Results

The training, validation, and testing error rates of each model can be found in the table below.

| Models | Training Error | Validation Error | Test Error |
|---|---|---|---|
| Decision Tree | 0.1807545 | 0.1815004 | 0.2680953 |
| Random Forest | 0.1516606 | 0.1579545 | 0.2561471 |
| XGBoost | 0.136260 | 0.1465485 | 0.3245304 |
| BART | 0.157 | 0.1607789 | 0.2692424 |

The validation and test confusion matrices are shown below. For more details, please refer to out codes files.

## Validation Confusion Matrices:

Decision Tree

| Prediction | Reference 0 | 1 |
|---|---|---|
| 0 | 5304 | 971 |
| 1 | 483 | 1253 |

Random Forest

| Prediction | Reference 0 | 1 |
|---|---|---|
| 0 | 5339 | 886 |
| 1 | 366 | 1329 |

XGBoost

| Prediction | Reference 0 | 1 |
|---|---|---|
| 0 | 5332 | 719 |
| 1 | 455 | 1505 |

BART

| Prediction | Reference 0 | 1 |
|---|---|---|
| 0 | 5302 | 803 |
| 1 | 485 | 1421 |

Test Confusion Matrices:

Decision Tree

| Prediction | Reference 0 | 1 |
|---|---|---|
| 0 | 46214 | 21254 |
| 1 | 14 | 11848 |

Random Forest

| Prediction | Reference 0 | 1 |
|---|---|---|
| 0 | 46214 | 21254 |
| 1 | 14 | 11848 |

XGBoost

| Prediction | Reference 0 | 1 |
|---|---|---|
| 0 | 35458 | 14975 |
| 1 | 10770 | 18127 |

BART

| Prediction | Reference 0 | 1 |
|---|---|---|
| 0 | 43216 | 18347 |
| 1 | 3012 | 14755 |

# 6. Discussion

Here, we are using 2 datasets, i.e. one for Training and Validation, and another for Testing. Training and Validation is done on Resort Dataset, and Testing is done on City Hotel Dataset. Thus, the validation error should be naturally lower than the test error because the validation set comes from the same dataset as the training set. In our study, the validation error will measure how well the model performs on unseen resort hotel data, and the test error will represent how poor the model will be able to generalize to city hotel data.

Looking closely at the error rate, we observe that all the model's training error is less than its validation error. This is a good sign as it tells us that the model is learning on the training data. XGBoost provides us with the least validation error (0.1465485), meaning that it performs the best out of all models on resort hotel data. We expect XGBoost to outperform Decision Trees and Random Forests when we have a substantial amount of data. We can see that XGBoost is well trained to the training set and is poor at generalizing to the test set (city hotels).

We see that in training and validation Decision Tree, Random Forest and XGBoost outperforms BART in terms of testing misclassification rate. This is because of a computational issue in implementing BART in R (out of memory issue). We are able to build a model, with the most basic hyperparameter setting i.e num_trees = 50, num_burn_in = 100 and num_iterations_after_burn_in = 500 and remaining parameters as default, with this setting we see that in terms of misclassification rate, BART is very close to XGBoost, Decision Tree and Random Forest in training and validation dataset. But in the testing dataset, we see that BART outperforms XGBoost and is very close to that of Decision Tree and Random Forest.

Based on the above experiment on the test dataset, we see that BART is less sensitive to overfitting compared to XGboost. This is because in BART, we are using priors which acts as a regularizer which prevents this model to overfit the data.

# 7. Conclusion

In this study, we compared the performances of different decision tree models of CART and BART. The XGBoost model performed the best on the validation set but is also most prone to overfitting as we see it generalizes poorly on the city hotels dataset. Meanwhile, Random Forest and BART perform better than the Decision Tree on the validation set, and all except XGBoost perform about the same on the test set. It is also important to note that we are not using the ideal hyperparameter values for the BART model since it is too computational expensive for our machines. If we are able to use the recommended hyperparameter values for BART, however, we would expect BART to perform better in both the validation and test sets.

This project allowed us to explore another family of classification algorithms, the decision trees, that were not thoroughly covered in our course. We were able to test and learn how each of these tree methods differ from each other, the pros and cons of each, and draw logical conclusions from our experiments. Although Decision Trees and BART might not be as popular as Random Forests and XGBoost in the industry, knowing the perks and situations of when to use each model will definitely be useful for our career.

# 8. References


- Kevin P Murphy Machine Learning A Probabilistic Perspective: http://noiselab.ucsd.edu/ECE228/Murphy_Machine_Learning.pdf
- bartMachine: Machine Learning with Bayesian Additive Regression Trees: https://cran.r-project.org/web/packages/bartMachine/vignettes/bartMachine.pdf
- Hotel booking demand datasets: https://www.sciencedirect.com/science/article/pii/S2352340918315191#s0005