Università della Svizzera italiana | Faculty of Informatics

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Technical Report - Introduction to Bayesian Computing, Spring 2021

Bayesian Added Regression Tree (BART)

An overview of applications of Bayesian methods in regression

Glejdis Shkëmbi[†], Franciskus Xaverius Erick[†]

[†] Faculty of Engineering, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany

Abstract

Report Info

Published
June 2021

Course
Introduction to Bayesian Computing
Spring 2021

Institutions
Faculty of Informatics
Università della Svizzera italiana
Lugano, Switzerland
&
Faculty of Engineering
Friedrich-Alexander Universität
Erlangen-Nürnberg, Germany

1 Motivation

.

2 Mathematical Background

In its core the FFT is an algorithm that allows us to solve a mathematical problem with less computational effort than needed at first glance. It calculates the discrete Fourier transform (DFT), which can be described as a matrix vector multiplication in only $\mathcal{O}(n \log(n))$ operations[3].

In this section we will provide the most important mathematical basics of the FFT. We will cover the notion of Hilbert spaces, which gives a framework for function spaces and the understanding of functions as bases. Furthermore, we will introduce the main aspects of the Fourier series and transform in their analytical and discrete form.

2.1 Classification and Regression Trees - CART

The Classification and Regression Tree analysis, first introduced by Breiman et al. in 1984, is a well known decision tree method used in constructing predictor variables from the data. It builds a binary decision tree based on some splitting rule on the predictors [1]. Regression trees are used for variables with continuous or ordered discrete values, while classification trees are intended for variables with a finite number of unordered values. Moreover, for regression trees, the prediction error is commonly assessed as the squared difference between the observed and predicted values, while for classification trees the prediction error is evaluated in terms of misclassification cost [2].

In a classification task, we have a training set with n observations, a class variable Y such that Y $\in 1, 2, \cdots, k$ and the predictor variables $X_1, \cdots, X_p$ with fixed dimensionality p and we try to find a model that better predicts the class Y given new values of X [1]. The CART technique for classification problems produces rectangular sets A_j by recursively partitioning the data set a single X variable at a time into disjoint sets $A_1, \cdots, A_j$ and fitting a single prediction model within each partition [2]. The partitioning is repeated until a node is reached for which no split enhances homogeneity, at which point splitting is stopped and this node becomes a terminal node [1]. This partitioning can be visually represented with a decision tree. In figure 1.0 we see a classification tree model with three classes labeled 1, 2 and 3, where the partitions are on the left and the decision tree is on the right. The root node of the tree represents the entire data set, and at each node, if the condition is satisfied, a case is assigned to the left child node. Moreover, the predicted class is beneath each leaf node [2]. In contrast, in a regression task, the Y variable takes ordered values and we are trying to fit to each node a regression model to give predicted values of Y [2].

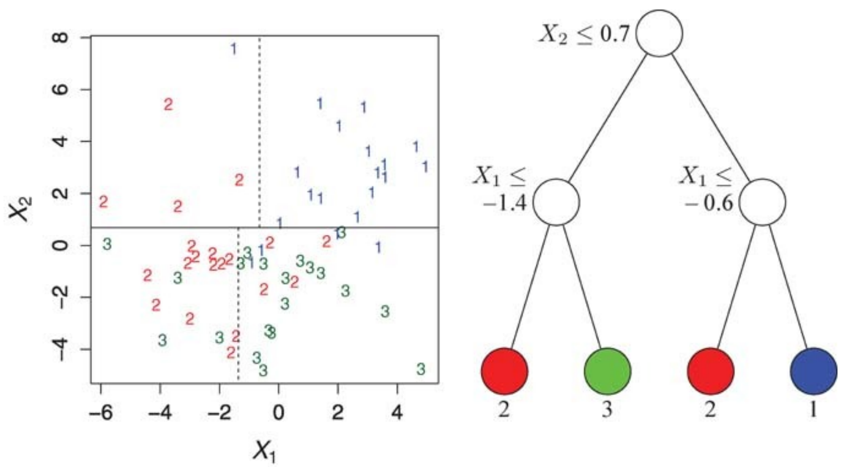

Figure 1: Partitions (left) and decision tree structure (right) for a classification tree model with three classes labeled 1, 2, and 3 [2].

The algorithm starts at the root node of the tree and grows itself as follows [3]:

- Step 1: Explore every possible split on each predictor variable. Binary questions are typically used to produce binary splits.
- Step 2: Select and apply the best split (ie. selects the best feature which helps us better predicting the target class).
- Step 3: Recursively proceed in this way until a stopping criterion is reached.

The stopping criterion is reached when [4]:

- The node's examples all belong to the same class.
- There are no features remaining to distinguish between samples.
- The tree has grown to the predetermined size limit.
- The first problem for a decision tree is determining which feature to split on.

Purity refers to the degree to which a subset of examples contains just a single class, and any subset made of only a single class is pure [4]. There are several purity metrics that may be used to choose the optimal decision tree splitting candidate. C5.0 and CART are two algorithms for classification that follow this approach. They are both very similar, however, C5.0 uses entropy for its impurity measure, while CART uses the Gini index [2]. Entropy is a measure of randomness or disorder within a set of class values. Mathematically, the entropy is formulated as follows:

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i, \tag{1}$$

where S represents the segment of data, c the number of different classes and p the probability of the values being part of class i [4]. When entropy is equal to 0, this indicates that the set is completely homogeneous. The higher the entropy, the more diverse the set is, which means that less information is provided about other data points that may belong in the set. The goal of C5.0 is to find splits that reduce entropy, hence enhancing homogeneity within the groups [4]. Now, using entropy, the Information Gain of a feature F can be calculated as shown below[4]:

$$\text{InfoGain(F) = Entropy(S1) - Entropy(S2).} \tag{2}$$

The higher the information gain is, the more homogeneous the set created after the split on this feature is. On the other hand, Gini index calculates the "impurity" of a feature with respect to the classes [5]. More specifically, it indicates the probability of a specific feature to be classified incorrectly when we do random selection. It is given by the following formula [5]:

$$\sum \sum_{j \neq i} p_i p_j = \left(\sum p_i\right)^2 - \sum p_i^2 = 1 - \sum p_i^2 \tag{3}$$

Breiman et al. (1984) points out that the stopping rules in Step 3 do not work very well in practical applications, as the tree tends to become too large and has too few data points in each terminal node, making its decisions overly specific and increasing the risk of model overfitting. Therefore, to address this issue, there is the need to prune the tree. Pruning a decision tree is the process of reducing its size in order to generalize better to unseen data [4]. There are two types of decision tree pruning: pre-pruning or post-pruning. In pre-pruning, also known as early stopping, we let the tree grow until it reaches a predefined number of decisions or until there is only a small number of examples in the node, and then we stop it [4]. In this way we avoid doing extra work, however, it is difficult to know whether this way we leave out subtle but important patterns. On the other hand, in post-pruning, we intentionally allow a very large tree to grow and then we use pruning criteria to reduce its size to a more appropriate level [4]. This type of pruning ensures us that all of the important information is discovered.

Some of the main advantages of the decision trees include [4]:

1. Are easy to be interpreted even by non-statisticians;
2. Work well with missing variables;
3. Are invariant under transformations in the predictor space;
4. There are extensions: Bayesian version of CART.

Moreover, some of the drawbacks are:

1. A lot of data is needed as the tree-space is big;
2. Might not result in the "best" model;
3. There is selection bias for the splits;
4. May lead to overfitting.

2.2 Markov Chain Monte Carlo - MCMC

In Bayesian statistics, one would normally propose a prior distribution of the samples and these priors are updated according to the likelihood of observations from the samples. In this way, the posterior distribution, which describes the the distribution of the samples given the obtained likelihood, can be determined. The main formula of this update process is given as follows.

$$p(\theta \mid y) = \frac{p(\theta)p(y \mid \theta)}{p(y)} = \frac{p(\theta)p(y \mid \theta)}{\int p(\theta)p(y \mid \theta)d\theta} \tag{4}$$

The integral in the denominator which represents the marginal likelihood of y can be treated as a normalizing constant of the whole expression. For very simple distributions, this normalizing constant can be determined analytically. However, in real life situations, the likelihoods obtained can get highly complex in expression,which makes it impossible to evaluate this integral term. There are various methods that can be used instead to approximate the posterior distribution, without the need to evaluate such complicated integrals. One of the most common method is Markov Chain Monte Carlo, or MCMC.

Markov Chain Monte Carlo, first introduced by physicists at Los Alamos in the 1940's, is a computer-based sampling approach, which, by randomly sampling values out of the distribution, allows one to describe a distribution without knowing in advance all of its mathematical properties [20]. As written in the name, MCMC combines two properties: Monte Carlo and Markov Chain.

Monte Carlo estimates the properties of a distribution by exploring random samples from the distribution, while Markov Chain generates random samples sequentially where each random sample is used as a building block to generate the next random sample [20]. In other words, rather than computing the mean of a normal distribution from its equations, in Monte Carlo approach we draw a large number of random samples from this normal distribution and then compute the sample mean, which is much easier than computing the mean directly from the normal distribution's equations.

Markov Chain is a sequence $X_1, X_2, X_3, \cdots$ of random variables if the conditional distribution of $Xn+1$ given $X_1, X_2, X_3, \cdots, X_n$ depends only on X_n [21].That is to say that the future is independent of the past given the present [21]. This is known under the name of Markov Property. In MCMC, the state of the Markov chain θ^t is a drawn sample that corresponds to a certain distribution at iteration step t. Due to the Markovian property, the states at current iteration only depends on the states of the previous iteration step. The goal of MCMC is to simulate the posterior distribution $p(\theta \mid y)$, which can be treated as the target distribution, by iteratively constructing Markov Chain which stationary distribution converges to the target posterior distribution.

2.3 Metropolis-Hastings algorithm

Computationally, the generation of Markov chains in MCMC can be realized through Metropolis-Hastings algorithm. Metropolis-Hastings algorithm employs proposal distribution $q(\theta^t, \theta^{t-1})$, from which each state θ^t at iteration step t is proposed, eventually leading to the desired Markov chain. The proposed state depends on the state from the previous iteration θ^{t+1}. With Metropolis-Hastings algorithm, one would be able to approximate any probability density function $\pi(x)$ using only a proposal distribution function $q(x)$ and a function $f(x)$ which is proportional to the real probability density function $\pi(x)$. This is especially useful in the context of approximating posterior distribution function as of 4. Since the posterior distribution is proportional to the numerator of the right hand side of the equation, one would only need to evaluate the numerator and propose a proposal distribution to approximate the posterior distribution by Metropolis-Hastings algorithm, without the need to analytically compute the complex integral in the denominiator.

In general, the symmetric Metropolis-Hastings ($q(x_t, x_{t-1}) = q(x_{t-1}, x_t)$) algorithm with proposal distribution function $q(x)$ and function $f(x)$ can be described in the following pseudocode.

Algorithm 1 General Metropolis-Hastings algorithm

```
1:  function MH($x_0, n$)
2:      $x_1 = x_0$
3:      for $i \leftarrow 2$ to $n$ do
4:          $y \leftarrow q(x_i, x_{i-1})$
5:          $\alpha \leftarrow \min(1, f(y)/f(x_{i-1}))$
6:          $u \leftarrow$ runif(1)
7:          if $\alpha \geq u$ then
8:              $x_i = y$
9:          else
10:             $x_i = x_{i-1}$
11:     return $x$
```

In the pseudocode, the first state is initialized with a given initial state. The candidate next state is then proposed from the proposal distribution given the previous state. The acceptance ratio α evaluate the probability of moving from the current state to the candidate state. If this acceptance ratio value is lower than a randomly generated number from a uniform distribution ranging from 0 to 1, then the candidate state is taken as the next state. Otherwise, the next state would simply just be the current state (no move from the current state). This is iterated until a maximum number of iteration n is reached. In the case when the proposal distribution is not symmetric, the acceptance ratio would be $\alpha \leftarrow \min\left(1, \frac{f(y)q(y,x_{i-1})}{f(x_{i-1})q(x_{i-1},y}\right)$ instead.

Since the proposal distribution depends on the previous state, this very algorithm is also a case of random walk Metropolis Hasting. The Metropolis-Hasting algorithm is relatively easy to implement and does not suffer from curse of dimensionality, which is a well known issue for importance and rejection sampling. Therefore, the Metropolis-Hastings is currently very widely used for Bayesian inference.

A convergence to the target distribution is also guaranteed. However, care needs to be taken of with regards to the proposal distribution used. If the variance of the proposal distribution is too low, the states proposed would be too close and similar to each other, resulting to a higher degree of autocorrelation and subsequently a slow convergence. If the variance of the proposal is too high, there would also be a higher amount of rejected candidates, which in turn result to slow convergence. To ensure that the final distribution is indeed approximate to the target distribution, a burn-in period is also taken into consideration. This means that the proposed states from the first n (typically 1000 iterations) are ignored as the initial distributions may not reflect the true converged target distribution.

2.4 Gibbs Sampler

Gibbs sampling can be considered to be a special case of Metropolis-Hastings algorithm, whereby the acceptance rate is always taken as 1. It is specially used when sampling multivariate distributions, whereby a "correct" proposal distribution function that ensures reasonable convergence speed may not be easily determined. For Gibbs sampling, the update of each dimension of the state θ_d^t is given as the conditional distributions of each dimension for the dimensions $d = 1, 2, \cdots, D$. In general, Gibbs sampling algorithm is done in the following pseudo-code:

Algorithm 2 General Gibbs sampler algorithm

```
Require: The initial state of $\theta$ has already been initialized,$\theta^1 \neq$ NULL

1: function GIBBS($\theta, n$)
2:     for $i \leftarrow 2$ to $n$ do
3:         $\theta_1^i \leftarrow \pi(\theta_1^i \mid \theta_2^{i-1}, \theta_3^{i-1}, \theta_4^{i-1}, \cdots, \theta_D^{i-1})$
4:         $\theta_2^i \leftarrow \pi(\theta_2^i \mid \theta_1^i, \theta_3^{i-1}, \theta_4^{i-1}, \cdots, \theta_D^{i-1})$
5:         $\theta_3^i \leftarrow \pi(\theta_3^i \mid \theta_1^i, \theta_2^i, \theta_4^{i-1}, \cdots, \theta_D^{i-1})$
6:         $\vdots$
7:         $\theta_D^i \leftarrow \pi(\theta_D^i \mid \theta_1^i, \theta_2^i, \theta_4^i, \cdots, \theta_{D-1}^i)$
8:     return $\theta$
```

Thus for Gibbs sampler, one would only required the conditional distributions in order to approximate the target distribution. As the acceptance rate is always 1, no additional proposal distribution matching is required. However, this method can only be done when the conditional distributions can be sampled from. This is always not the case as for some distributions, the conditional distribution expressions can even be hard to obtain. Furthermore, the acceptance rate of 1 also comes with the cost of the inability to adapt to the cases when the states are highly correlated (high autocorrelation). This would also lead to a slower convergence as the high correlation would makes it harder for the algorithm to cover the whole target distribution. As in the classic Metropolis-Hastings algorithm, burn-periods need to be taken into account to ensure convergence of the Markov Chain.

2.5 Bayesian Additive Regression Tree

As previously mentioned, the classical CART is prone to overfitting. This leads to various research into a more Bayesian approach to the decision trees. By applying Bayesian framework, the data can overcome the assumptions about the depth of trees and the shrinkage needed. The Bayesian framework encodes what is typically an algorithmic approach in a likelihood framework to generate coherent uncertainty intervals, which is uncommon for machine learning methods.

Chipman et al. (2010) proposed a Bayesian "sum-of-trees" model, which is defined by a prior and a likelihood, where each tree is constrained by a regularization prior in order to be a weak learner. Fitting is achieved by a specialized iterative backfitting MCMC algorithm. This "sum-of-trees" model results to be adaptive and very flexible, where each individual tree describes a different part of the underlying mean function [7]. The "sum-of-trees" model differ from other ensemble methods that fit a linear combination of trees, such as boosting [11], random forests [10], bagging [12]. In bagging and random forest models we do random sampling and stochastic search to create a collection of independent trees, and then combine their results by averaging. It is worth noticing that from various literatures, BART performs better compared to LASSO [8], gradient boosting [9], random forests [10] and neural networks with one hidden layer.

2.5.1 Sum of Trees Model

Chipman et al. (2010) consider the problem in which a dependent variable Y needs to be predicted using the p dimensional input vector $\mathbf{x} = (x_1, \cdots, x_p)$:

$$Y = f(x) + \epsilon, \quad \epsilon \sim N\left(0, \sigma^2\right), \tag{5}$$

where function f(x) is unknown. Bayesian statistics aims to approximate the mean of Y given x by the sum of m trees. Each tree is represented by the g_j function:

$$f(x) = E(Y \mid x) \approx h(x) \equiv \sum_{j=1}^{m} g_j(x) \tag{6}$$

We further break down the components of the trees in the g_j function. We define T_j to be the j-th tree with the corresponding terminal nodes $M_j = \{\mu i j, \cdots, \mu b j\}$. b defines the amount of terminal nodes of the corresponding tree. Thus, equation 6 can be rewritten as follows.

$$f(x) = \sum_{j=1}^{m} g(x; T_j, M_j) \tag{7}$$

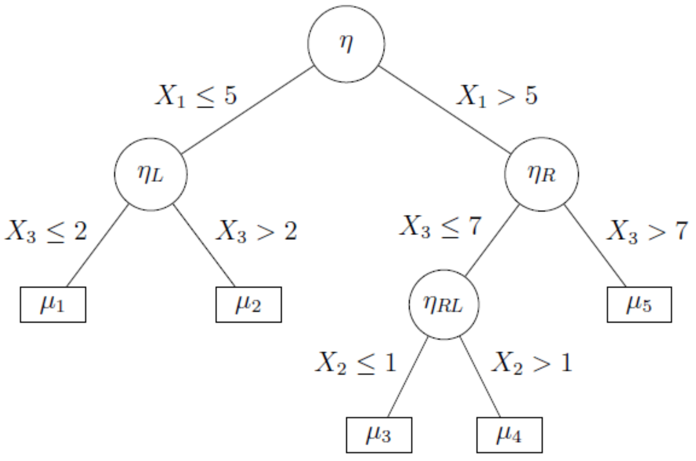

Figure 2: A binary tree example.

2.5.2 Regularization Priors

Bayesian priors are the main features in BART which enforce regularization to the added regression trees, thus ensuring that the trees do not get too significantly big, adapt to the data well while at the same time reduce overfitting. The prior specifications, stated in Chipman et al. (2010), will be described in more detail here.

There are three priors used for regularization, namely:

1. $\pi(T_j)$ the prior of the tree structure itself,
2. $\pi(M_j \mid T_j)$ the prior of the leaf structures given the tree structure,
3. $p(\sigma^2)$ the residual variance.

Each of the prior are discussed in more details as follows.

1. $\pi(T_j)$ the prior of the tree structure itself. This prior itself has 3 separate components
   (a) The probability that the node depth $d = 1, 2, \cdots, n$ is not terminal. The node depth is defined as the distance from the root node. This is expressed as
   $$\frac{\alpha}{(1+d)^{\beta}}, \quad \alpha \in (0,1), \beta \geq 0. \tag{8}$$
   Where α and β are hyperparameters to be tuned. This component enforces regularization on the depth of the trees itself so that they do not get too large. Chipman et al. (2010) recommends default values of $\alpha = 0.95$ and $\beta = 2$.
   (b) The probability that the i-th feature out of p features of data is taken to be the decisive splitting variable for a node. Chipman et al. (2010) uses uniform distribution over all the possible features p.
   (c) The probability that for the selected i-th feature , that a certain value of cut off point is used. Chipman et al. (2010) uses uniform distribution over all the discrete possible splitting values of that certain feature.
2. The prior $\pi(M_j \mid T_j)$ can be expressed as the product of all probabilities of the observed terminal nodes given the tree.
   $$\pi(M_j \mid T_j) = \prod_i p(\mu_{ij} \mid T_j). \tag{9}$$
   Chipman et al. (2010) recommends a normal distribution for the prior $p(\mu_{ij} \mid T_j) \sim \mathcal{N}(0, \sigma_\mu^2)$, whereby $\sigma_\mu = \frac{0.5}{k\sqrt{m}}$ and k is a hyperparameter to be tuned. The default value of k is taken as k = 2. Thus, this term would ensure that the leaves output get closer to 0 (the center of the normal distribution) when the variance term σ_μ gets smaller.
3. $p(\sigma^2)$, which is the residual variance. The residual variance helps to impose regularization effect on the whole term such that the model will not overfit to the training data. This prior is modelled with the conjugate prior inverse chi square distribution, which is none other than a special case of inverse Gamma distribution, as follows.
   $$p(\sigma^2) \sim \frac{\nu\lambda}{\chi_\nu^2}. \tag{10}$$
   With λ and ν as the hyperparameters. More frequently, another hyperparameter is q is chosen in order to determine the hyperparameter λ from looking at the prior distribution. While there are more detailed approaches to choosing the hyperparameters, for convenience, Chipman et al. (2010) additionally suggests to use the default values of $\nu = 3$ and $q = 0.90$ to avoid overfitting and yield good results.

Taking these prior components into account, the resulting regularization prior distribution is none other than:

$$\begin{aligned} p\left((T_1, M_1), \ldots, (T_m, M_m), \sigma\right) &= \left[\prod_j p\left(T_j, M_j\right)\right] p(\sigma) \\ &= \left[\prod_j p\left(M_j \mid T_j\right) p\left(T_j\right)\right] p(\sigma) \quad = \left[\prod_j \prod_i p\left(\mu_{ij} \mid T_j\right) p\left(T_j\right)\right] p(\sigma) \end{aligned} \tag{11}$$

The likelihood is taken to be the likelihood of outputting certain values of Y given the tree models with its specifications. This likelihood is modelled with a normal distribution, with the mean being the best guess possible for the specific configurations of the added regression trees.

$$y_l \sim \mathcal{N}(\mu_l, \sigma^2) \tag{12}$$

2.5.3 Posterior distributions

With the priors and likelihood specified in the previous sections, the posterior distribution is now left to be approximated. Direct analytical evaluation of the posterior distribution is not possible as it is not feasible to analytically compute the marginal likelihood from the non-trivial prior distribution functions. Chipman et al. (2010) proposed a specialized backfitting MCMC algorithm to approximate the posterior distribution. The specialized backfitting MCMC algorithm was developed off a similar backfitting MCMC algorithm proposed by Tibshirani et al. (2000) for approximating posterior distributions of generalized additive models. The pseudo-code for the specialized backfitting MCMC algorithm algorithm basing on the version summarized by Lakshminarayanan et al. (2015) is as follows[ref!].

Algorithm 3 Specialized backfitting MCMC algorithm

```
1: function SPECIALBACKFITMCMC($X$, $Y$, various BART hyperparameters)
2:     for $j \leftarrow 1$ to $m$ do
3:         $T_j^{(0)} \leftarrow$ single node initialization
4:         Sample $M_j^{(0)} \mid T_j^{(0)}$
5:     for $i \leftarrow 1$ to maxiter do
6:         Sample $\sigma^(i) \mid (T_1^{(i-1)}, M_1^{(i-1)}), (T_2^{(i-1)}, M_2^{(i-1)}), \cdots, (T_m^{(i-1)}, M_m^{(i-1)}), \boldsymbol{\varepsilon}$
7:         for $j \leftarrow 1$ to $m$ do
8:             Compute Residual $R_{-j}^i$
9:             Sample $T_j^{(i)} \mid R_{-j}^i, \sigma^2$
10:            Sample $M_j^{(i)} \mid T_j^{(i)}, R_{-j}^i, \sigma^2$
```

The specialized backfitting MCMC algorithm is fundamentally based on Gibbs sampling algorithm. The first Gibbs iteration is done as initialization of the tree and leaves parameters for the subsequent Gibbs sampling iterations. Note the residual term R_{-j} fitted to all trees except for the tree to be sampled from. The residual term is given by the following expression.

$$\boldsymbol{R}_{-j} := \boldsymbol{y} - \sum_{t \neq j} \mathcal{T}_t^{\mathcal{M}}(\boldsymbol{X}) \tag{13}$$

There are three main conditionals that are sampled within each Gibbs iteration.

1. $T_j^{(i)} \mid R_{-j}^i, \sigma^2$. This is the conditional of the tree structure conditioned on the special residual term from the remaining trees and the residual variance. This component is sampled by an additional Metropolis-within-Gibbs sampler that decides one of the four possible steps (GROW, PRUNE, CHANGE, SWAP) to be taken against the j-th tree structure of the current Gibbs iteration. For each of the decision, the acceptance ratio of each move decision is evaluated similarly to how acceptance ratios are calculated in the classic Metropolic-Hastings algorithm.

2. $M_j^{(i)} \mid T_j^{(i)}, R_{-j}^i, \sigma^2$. This is the conditional of the leaves parameters conditioned on the j-th tree structure of the current Gibbs iteration, special residual term from the remaining trees and the residual variance. Since the prior and likelihood are both of normal distributions, the conditional can be directly sampled from a conjugate normal distribution with parameters calculated from the parameters of the priors and the likelihood.

3. $\sigma^(i) \mid (T_1^{(i-1)}, M_1^{(i-1)}), (T_2^{(i-1)}, M_2^{(i-1)}), \cdots, (T_m^{(i-1)}, M_m^{(i-1)}), \boldsymbol{\varepsilon}$. This is the conditional of the residual variance conditioned on all the tree structures, the corresponding leaves parameters and the whole residual term. The whole residual term for the entire tree structures can be calculated as follows.

$$\boldsymbol{\varepsilon} := \boldsymbol{y} - \sum_{t=1}^{m} \mathcal{T}_t^{\mathcal{M}}(\boldsymbol{X}) \tag{14}$$

Since the prior is taken as the conjugate prior inverse chi-squared distribution, sampling from the posterior can be directly done from a conjugate inverse chi-squared distribution.

2.5.4 Recommended Hyperparameters

In summary, the recommended hyperparameters for the modelling of the priors are $\alpha = 0.95, \beta = 2, \nu = 3, q = 0.90, and k = 2$. The recommended number of trees is taken as m = 200. To ensure that the MCMC converges, a sufficient number of burn-in iterations needs to be taken into account.

3 Experiment Methods

The experiments will be done in two parts. First experiment is to run a BART model in R and evaluate the results and its performance. This is done by utilizing a prewritten BART library in R called "bartMachine". The second part of the experiment is to compare the performance of the BART model itself with another BART model from another R library called "BART" and, a classic regression tree model, and a random forest model.

3.1 Dataset

In this study, the diabetes data which is free and publicly available on Hastie, Tibshirani, Wainright website, is being used [24]. The dataset contains observations on 442 patients. The response of interest y is a quantitative measure of disease progression one year after baseline. The 442 diabetes patients were measured on 10 baseline variables: age, sex, body-mass index, average blood pressure, and six blood serum measurements [24]. Moreover, before the data is being used, they are first standardized to have zero mean and unit L2-norm [24]. The complete data used in our experiments contains 64 feature variables. The first column is the response y, the next 10 columns are the age, sex, BMI, BP and the serum measurements, and the rest of the columns are the squares and products of the original 10. The statistical feature extraction is already performed in our dataset, therefore no further preprocessing is applied to the diabetes dataset. For all the tests, 75% of the data is taken as the training data with the remaining 25% as the test data.

Table 1: Original diabetes data [25]

	AGE	SEX	BMI	BP	Serum measurements					Response	
Patient	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	y
1	59	2	32.1	101	157	93.2	38	4	4.9	87	151
2	48	1	21.6	87	183	103.2	70	3	3.9	69	75
3	72	2	30.5	93	156	93.6	41	4	4.7	85	141
4	24	1	25.3	84	198	131.4	40	5	4.9	89	206
5	50	1	23.0	101	192	125.4	52	4	4.3	80	135
6	23	1	22.6	89	139	64.8	61	2	4.2	68	97
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
441	36	1	30.0	95	201	125.2	42	5	5.1	85	220
442	36	1	19.6	71	250	133.2	97	3	4.6	92	57

3.2 Evaluation Metrics

In our regression setting, to measure the performance of our models, R squared and root mean squared error (RMSE) are used. R squared is the proportion of variance explained by the model:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(y_i - \tilde{y}\right)^2}{\sum_{i=1}^{n} \left(y_i - \bar{y}\right)^2} \tag{15}$$

The RMSE, which is the square root of the variance of the residuals, is given by the formula:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}\right)^2} \tag{16}$$

It indicates how close the observed data points are to the model's predicted values. Lower values or RMSE indicate better fit, as RMSE measures how accurately our model predicts the response y.

3.3 BART R Libraries

There are currently multiple implemented BART libraries in R. The "bayesTree" library was developed by Chipman et al. (2010), sticking to the original implementation as specified in the literature. However, the "bayesTree" library does not support parallelization and detailed model diagnostics. In 2013, Kapelner and Bleich published their implementation of BART in their "bartMachine" library. The "bartMachine" library allows one to easily model a BART model, evaluate in detail statistics of the model fitting against datasets, use multiple computing cores, optimize the hyperparameters used for prior modelling, evaluate convergence of the MCMC and many other statistical tests. This allows us to evaluate the performance and statistics of our BART model in a very detailed manner. Another more recent BART library is "BART", developed by McCulloh and Sparapani in 2017. This "BART" library provides various statistical testings and has higher compability with other libraries in R, which makes it easier for one to implement functionalities and tests for the created BART model. However, this "BART" library only supports multi-threading if MPI is installed in the user's computer. In addition, the "BART" library is still relatively in development and the features and functionalities of this library are still updated frequently up to this day. For this reason, we will use "bartMachine" as the main R library used to model and evaluate our BART model. The BART model implemented in "BART" library will be used solely for comparison. Aside from the functionalities and statistical tests available, each BART libraries also differ slightly in their implementation and interpretation of some prior parameters. As such, it is also worth investigating to see if the default implementations of BART in "bartMachine" and "BART" would return similar performance when trained with the same dataset. The following table summarize the features and differences of each of the three libraries as summarized by Sparapani in a BART Bootcamp lecture slide.

Table 2: Comparison of the different BART libraries in R [].

	BayesTree	BART	bartMachine
Author(s)	Chipman	McCulloch	Kapelner
	McCulloch	Sparapani	Bleich
Computer language	C++	C++	java
Dependencies	None	Rcpp	rJava
Multi-threaded	No	Yes	Yes
predict function	No	Yes	Yes
Missing data handling	No	No	Yes
Variable selection	No	No	Yes
Variable importance	Yes	Yes	Yes
Tree transition proposals	3	3	3
Partial dependence plots	Yes	No	Yes
Continuous & binary	Yes	Yes	Yes
Convergence diagnostics	Continuous	All	Continuous
Cross-validation	No	No	Yes

3.4 Random Forest

Random forest, championed by Leo Breiman and Adele Cutler, is an ensemble method combining decision trees and bagging [22]. Using the concepts of random feature selection and bagging, first, it generates a collection of trees, and then each of the trees votes for the most popular class [4] (see Figure 1.2). To improve stability and accuracy of the regression models, bagging is being used. Bagging does random sampling with replacement. That is to say that many different "copies" of the training data are created, then weak classifiers are applied to these "copies", and their result is combined. This will result in reduced variance and low overfitting [23]. Building the ensembles, for the k-th decision tree, a random vector k with identical distribution for all the trees in the forest is generated [22]. These vectors are independent of all the previous random vectors i where $i = 1, \cdots, k-1$. Next a tree is grown using the training set and the generated random vector [22]. This results in a classifier $h(x, k)$ where x denotes the input vector and k denotes the generated random vector [22]. Moreover, the trees are allowed to grow without pruning, which results in low bias. Some of the main advantages of RF include[4]:

- Can work with categorical and continuous features;
- Can handle missing and noisy data;
- Can perform well even with a large number of features as it selects only a subset of them.

4 Results and Discussion

4.1 First experiment results : bartMachine

After the dataset is split according to 3:1 train:test ratio as specified in 3.1, the model is constructed with the default bartMachine model constructor. The default values of prior hyperparameters used in bartMachinde models are equal to the values specified in 2.5.4, with the exception of number of trees which is taken as m = 50 instead of m = 200. As of the official documentation of bartMachine [], the burn-in Gibbs iterations are 250 and the default post burn-in Gibbs iterations are 1000.

4.1.1 Results

By using the default parameters, we obtain an in-sample RSME value of 41.6 and in-sample R-squared value of 0.708 as provided by the output of the bartMachine model diagnostics. This essentially means that the model explains 70.8% of the variation in the data. The in-sample fitted values against actual values plot is plotted with the command *plot_y_vs_yhat(bart_machine, credible_intervals = TRUE)* and the resulting plot is shown in Figure 3.

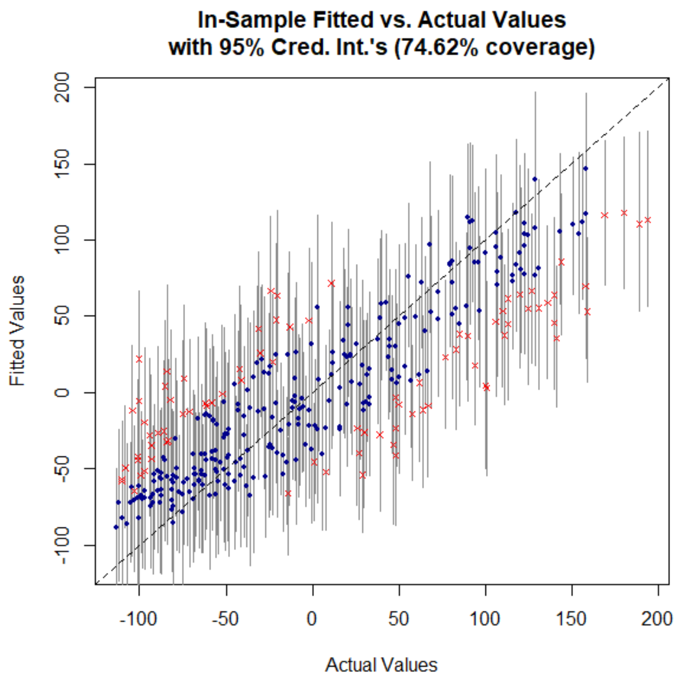

Figure 3: The plot of in-sample fitted values against actual values show that 74.62% of the dataset actually have their ground truths lie within the 95% credible interval .

The plot shows that in overall, most of the predicted values lie around the diagonal of the plots. While the credible interval is set to 95%, only 74.62% of the dataset actually have their ground truths lie within the 95% credible interval of the posterior distribution. This is due to the fact that the credible interval does not take into account of the regularization uncertainty term as discussed in the section 2.5.2. The author of bartMachine additionally implemented a prediction interval term which takes the regularization uncertainty term into account, giving larger intervals for each dataset. This is performed with the command *plot_y_vs_yhat(bart_machine, prediction_intervals = TRUE)* and the resulting plot is shown in Figure 4.

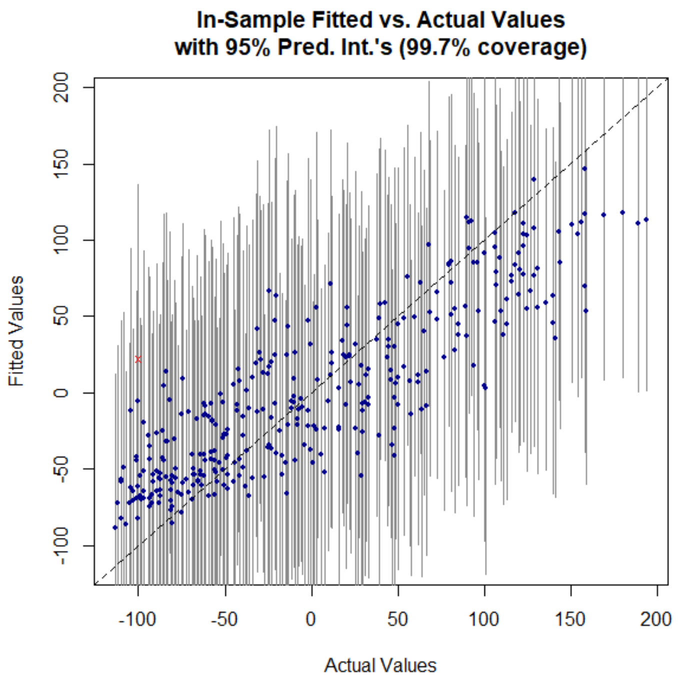

Figure 4: The plot of in-sample fitted values against actual values show that 99.7% of the dataset actually have their ground truths lie within the 95% prediction interval .

From the figure, we can see that the with the regularization error considered, 99.7% of the dataset actually have their ground truths lie within the 95% prediction interval of the posterior distribution. Thus, we can see that the model performs reasonably well with the training dataset. Additionally, the 10-fold cross validation RSME of the model is observed to be 56.9, while the 10-fold cross validation pseudo R-squared of the model is observed to be 0.45. These are obtained by the command *k_fold_cv(x_train, y_train, k_folds = 10)*. The out-of-sample performance of our BART model is also tested against the test dataset with the command *oos_perf = bart_predict_for_test_data(bart_machine, x_test, y_test)*. We obtain an out-of-sample test RMSE value of 56.1. This is about 33% higher than the in-sample-RMSE values, which also mean that the bartMachine model overfits slightly to the train dataset with the current default hyperparameter values. We will investigate if altering the prior hyperparameters and specifications might result to less overfitted predictions in the hyperparameter optimization section.

The command *rmse_by_num_trees(bart_machine, num_replicates = 20)* allows us to plot the RSME of the models against the number of trees used, whereby the average RSME of the 20 replicates of the models are used for the evaluation. The plot is shown in Figure 5.

While the out-of-sample RMSE generally decreases with increasing number of trees, we can observe that the RMSE actually increases after reaching a local minimal value at m = 40 trees, reaching a maximum RMSE value at m = 100 trees and decreasing again up to a minimum value at m = 200 trees. Thus, for our bartMachine model with our current dataset, larger amount of trees do not result to a significantly more generalizable BART model. The overfitting problem is not resolved as well as the out-of-sample RMSE values do not decrease towards the value of our in-sample RMSE value no matter how many trees are used for the modelling. One may prefer to have a smaller amount of trees for similar performance with smaller model sizes and less complexities. The number of trees is one of the hyperparameters that can be tuned to

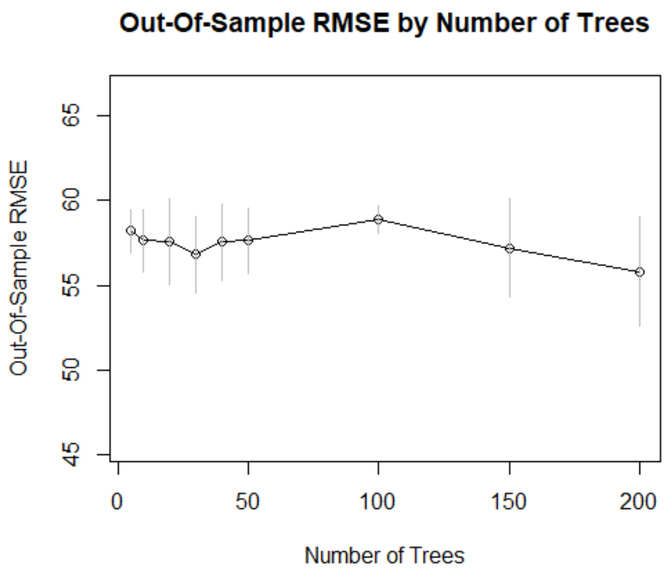

Figure 5: The plot of out-of-sample RMSE by the number of trees for our bartMachine model. We can then deduce which number of trees results to the best out-of-sample RMSE.

obtain the most optimal performing BART model, which will be discussed in a later section.

4.1.2 Assumption checking

The bartMachine provides two tests to check the assumptions that one normally use for regression, namely the test of normality of the residuals (Shapiro-Wilks p-value test) and the residual variance test (Heteroskedasticity test). This is performed by running the command *check_bart_error_assumptions(bart_machine)*. The resulting plans are shown in figure 6.

From the p-value of 0.167 , the null-hypothesis that the residual follows a normal distribution is not rejeceted, thus we can conclude that the residual indeed follows a normal distribution. This is to be expected as the training sample size n = 331. This is also confirmed visually from the QQ plot, whereby the points roughly lie on the diagonal of the plot. The heteroskedaticity plot also confirms that the residuals are roughly equal across the range of dataset values. Thus we can also observe that the assumptions for our BART model is fulfilled.

4.1.3 MCMC convergence analysis

The bartMachine also allows us to monitor the MCMC convergence diagnostics through both the trace of the residual value σ^2 and the trees parameters across iterations. This is performed by the command *plot_convergence_diagnostics(bart_machine)*. The result of the MCMC convergence diagnostics is shown in Figure 7.

There are 4 plots produced by the convergence diagnostic command. The first plot on the top left is the plot of residual term σ^2 over post burn-in iterations. While the σ^2 values fluctuated slightly, the fluctuations mostly lie within the 95% credible interval of the variance of the residual term. Thus, from this plot we can observe that the chain explores the different parts of the sample space of residuals multiple times. The second plot on the top right shows the acceptance rate of the the Metropolis-Hastings part in each Gibbs iteration. The grey part on the left half of the figure indicates the burn-in period, while the right half indicates the post burn-in period. Interestingly, the acceptance rate does not differ significantly during burn-in period and after burn-in period. The third plot in the bottom left is the plot of the trace of the number of nodes per tree over post burn-in iterations. The blue line indicates the average number of nodes and leaves per tree, while the upper bound and lower bound values are indicated with black lines. The number of nodes and leaves does not differ significantly over the post burn-in iterations. Lastly, the plot in the bot-

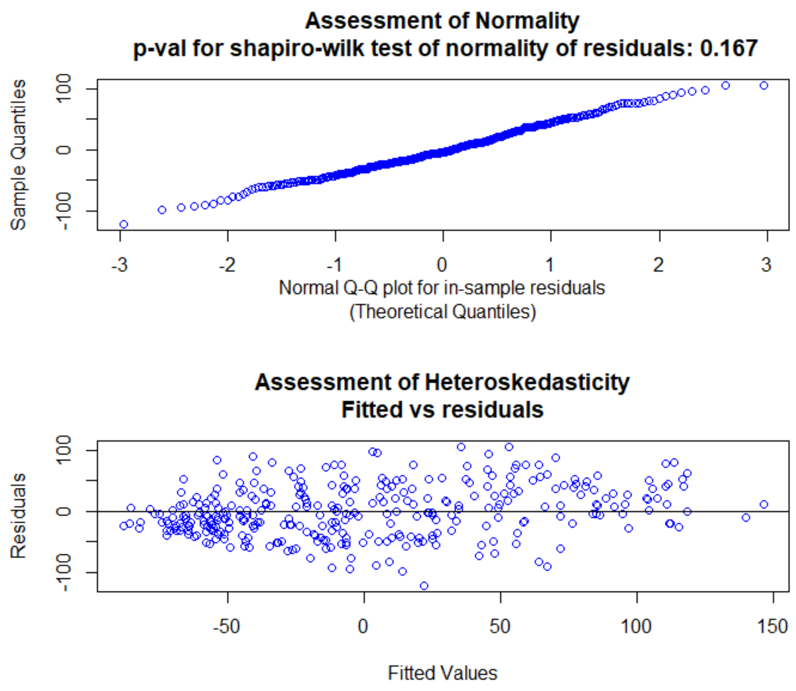

Figure 6: The Q-Q plot and heteroskedasticity plot of the residuals. Both plots show that the residuals follow a normal distribution

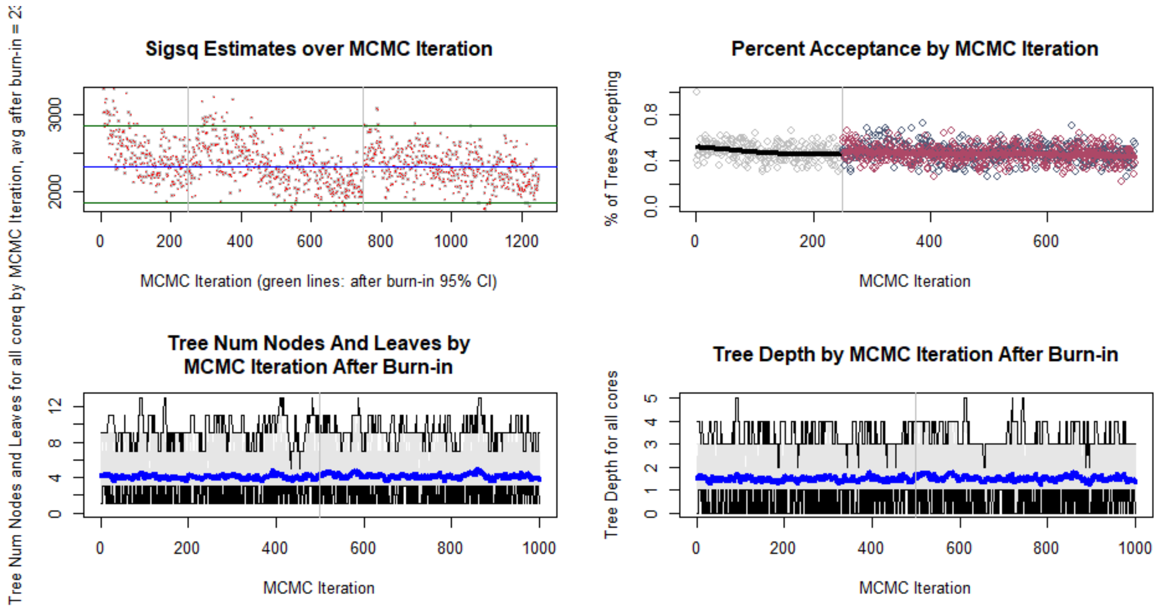

Figure 7: The MCMC convergence diagnostic plots of (top left) the residual term σ^2 over iterations, (top right) the Metropolis-Hastings tree acceptance rate over iterations, (bottom left) the number of leaves per trees over iterations, and (bottom right) the depth of trees over iterations.

tom right is the plot of the depth of the trees over post burn-in iterations. Similar to the previous plot, the blue line indicates the average depth of the trees with the upper bound and lower bond values indicated with black lines. Likewise, the depth of the trees do not vary significantly over post burn-in iterations. Since these four plots show the relative stability of the parameters of the model, we can conclude that the chain has indeed converged to the posterior distributions after burn-in given the prior models proposed.

4.1.4 Hyperparameter optimization

The bartMachine library also has a built-in hyperparameter optimization functionality, which allows one to easily optimize the hyperparemeters used for the prior modelling via cross-validations. This is done with the command *bart_machine_cv <- bartMachineCV(x_train, y_train)*, whereby different combinations of the parameters k, ν, q, and m are applied and the cross-validation errors are evaluated. The winner sets of hyperparameters which result to the lowest cross-validation error values are then applied to build a new optimized bartMachine model. For our current BART model, the winning hyperparameter combination is $k = 2$, $\nu = 0.9, q = 3$, and $m = 50$, which is thus the same hyperparameter combination used in our default bartMachine model. For our dataset, it is thus not necessary to optimize the hyperparameters further. The in-sample, test and cross validation RSME and pseudo R-square values of the optimized model are summarized in the following table.

Table 3: Table of RMSE and Pseudo R^2 values of the bartMachine BART model.

	In-sample	10-fold CV	Test
RMSE	41.6	56.9	56.1
Pseudo R^2	0.708	0.45	-

4.1.5 Variable Importance and Partial Dependence

Since there are a total of 64 covariates in the dataset, we would like to investigate which variables are considered more important for the prediction of our BART model. This can be performed by the command *investigate_var_importance(bart_machine, num_replicates_for_avg = 20)*. The plot is shown in Figure **??**.

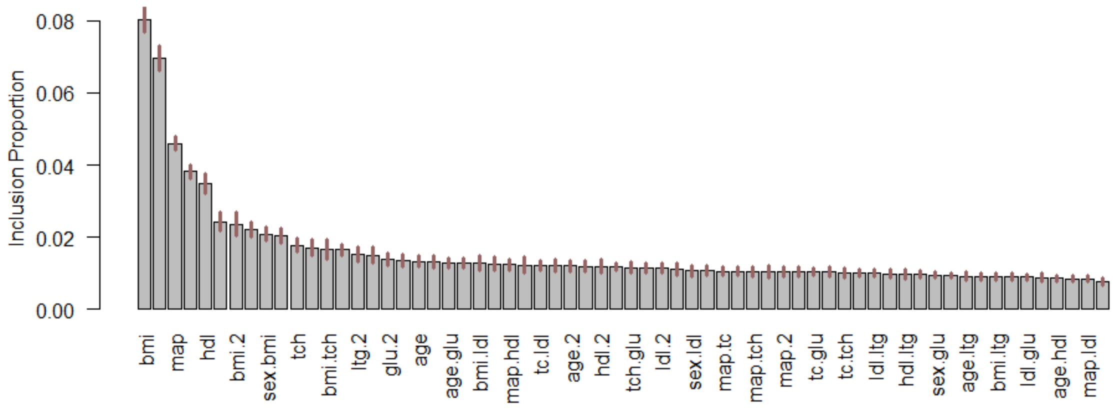

Figure 8: Variable importance plot of the bartMachine model.

According to the variable importance plot, the covariates BMI and map (mean arterial blood pressure) are the most significant covariates for our bartMachine model. The variable importance plot also shows that out of the 64 covariates ,only the five most important covariates are of significantly higher importances. We may want to investigate the partial dependence plots of the covariates in order to investiage the effect of each covariates to the BART model output in more detail. This can also be done with the bartMachine library through the command *pd_plot(bart_machine_cv, j = "bmi")* for example. The partial dependence plots of the two covariates with the highest importance values are shown in the following figures.

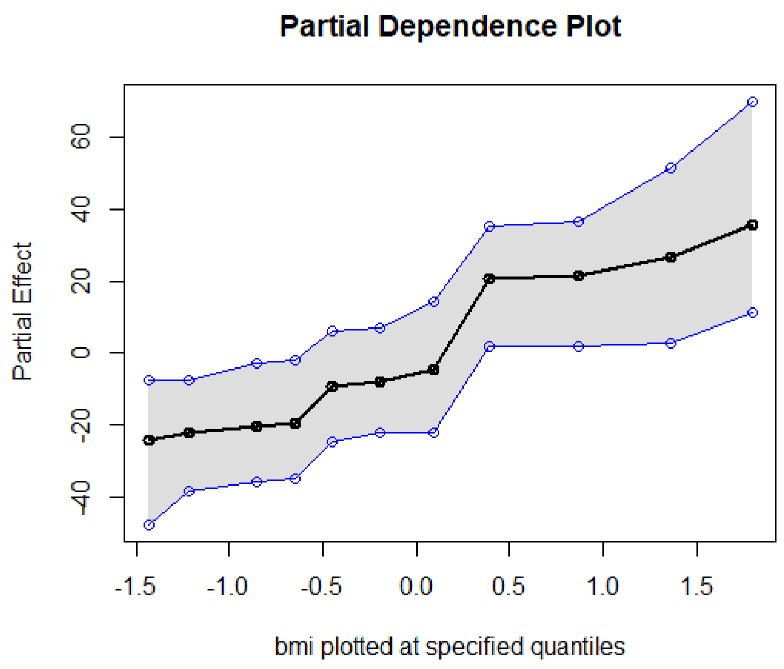

Figure 9: Partial dependence plot of the BMI covariate.

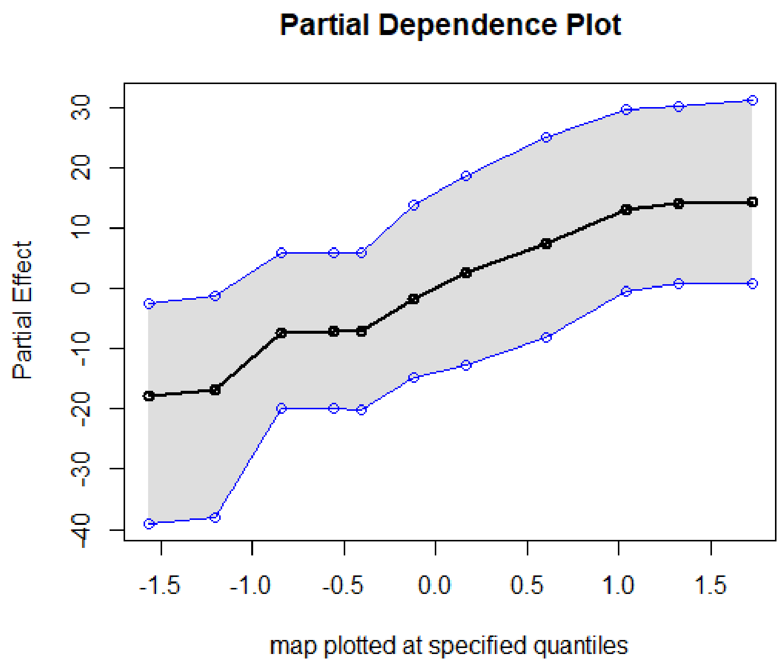

Figure 10: Partial dependence plot of the map covariate.

4.2 Second experiment results : Performance Comparison

4.2.1 Comparison with BART library

4.2.2 Classic CART Results

Regression trees are generated through the rpart package in R. We fit the model on the training data by setting method="anova" as we are working in a regression setting. The regression tree is shown in Figure 11.

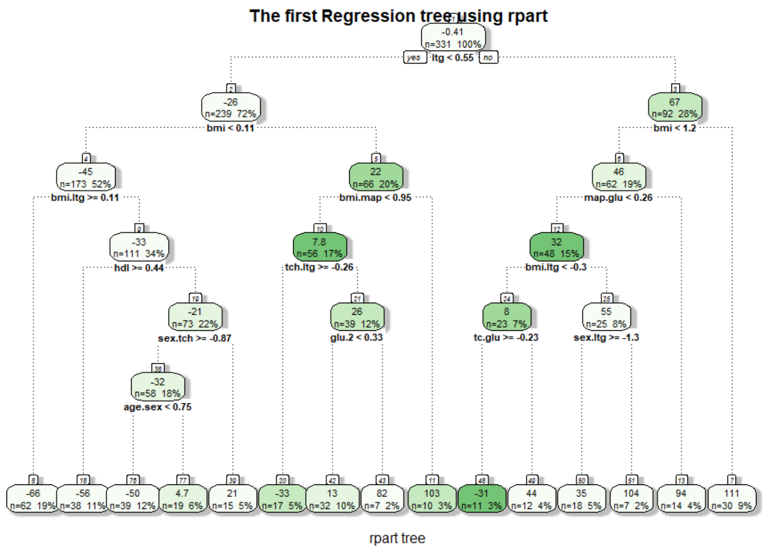

Figure 11: The first regression tree using rpart.

To compare the error for each cp value, cross-validation is performed such that the error related with a given cp value is computed on the hold-out validation data. In figure 12 such a plot is shown, where the y-axis is the cross-validation error, the lower x-axis is the cp value and the upper x-axis is the number of terminal nodes. From the plot we see that the best value of cp, resulting in the lowest cross-validation error, is for cp=0.0426. This value of cp is used in the next section to perform pruning of the tree.

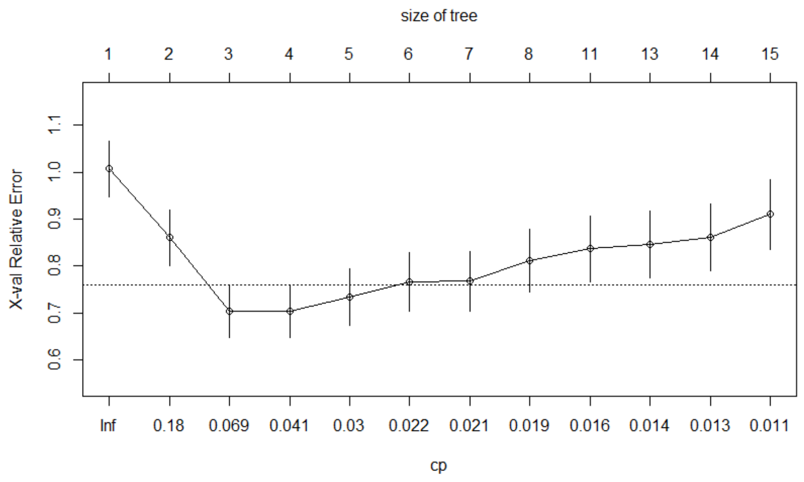

Figure 12: Plot of the cross validation results for the tree across different cp values.

The tree is pruned to avoid overfitting, by minimizing the cross-validation error - xerror. This is performed through automatic selection of the complexity parameters associated with the smallest cross-validation error. The pruned tree is displayed in figure 13. In addition, the 10-fold cross validation repeated 10 times is used to help us find the best combinations of algorithm parameters for our classification problem by tuning the model parameters. In order to achieve such tuning, the CARET package of R, which stands for Classification and Regression Training, is being used. The 10-fold cross validation is repeated 10 times by specifying the method parameter and the repeats parameter of the trainControl function to "repeatedcv" and 10 respectively. In this way, each subset is used at least once for training the classifier, and the average performance on all the 10 folds is outputted. The model is tuned through the complexity parameter (cp). Moreover, the tuneLength parameter , which makes the model try different default values for the main parameter cp, is set to 100. This means that the total number of combinations evaluated is 100. Figure 14 displays how the change in cp value affects the RMSE value. RMSE was used to select the optimal model using the smallest value. As it can be seen in the plot, the final value used for the model was cp = 0.033 with RMSE=62.8 and R^2=0.36 in the training set, and with RMSE = 63.9 and R^2= 0.33 in the test set. An R squared of 0.33 tells us that our model explains only 33% of the variation in the data.

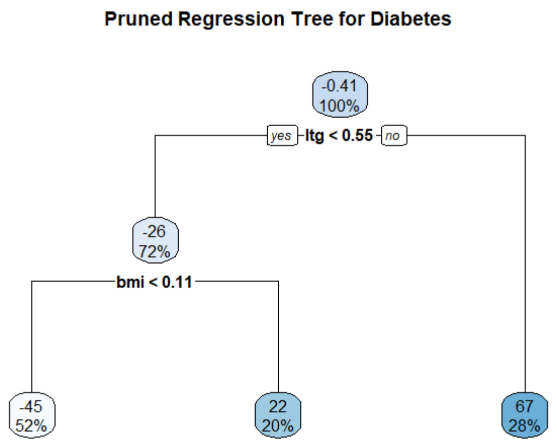

Figure 13: The pruned regression tree.

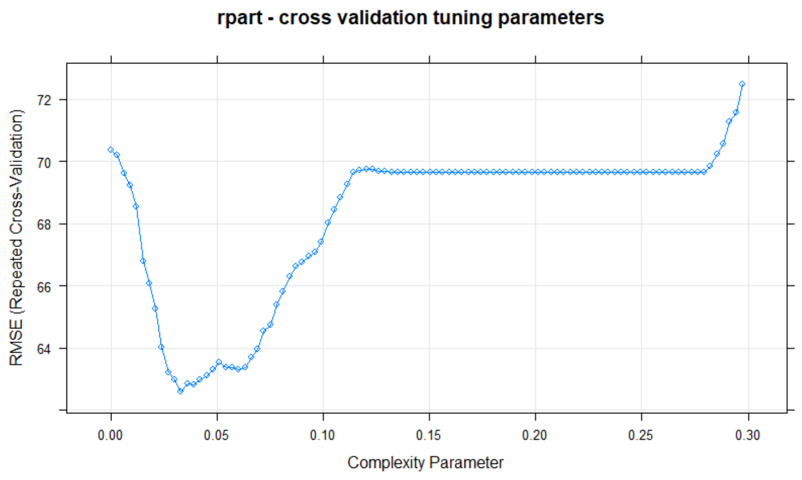

Figure 14: Plot of cross validation RSME against tuning complexity parameters.

In addition to tuning the cp value, we perform further tuning of minsplit and maxdepth. minsplit represents the minimum number of data points required to split a node before making it terminal, while maxdepth represents the maximum number of internal nodes. To do so, grid search is used to automatically search across a range of differently tuned models to recognise the most optimal parameter setting with the highest performance. The search range of minsplit is set from 5-20 and the search range of maxdepth is set from 1-15, which results in 240 different combinations. It results that the lowest error is achieved with minsplit =

Table 4: Regression tree models comparison.

	Initial tree with default parameters	Pruned tree	10-fold cross validation tree	Tuned tree
RMSE on test set	65.78	68.02	63.9	63.9
R Squared on test set	0.33	0.25	0.33	0.33

13, maxdepth = 3 and cp = 0.023. However, comparing such a model with the previous model, where we did cross-validation and cp tuning, we observe that the both models perform quite the same in terms of RMSE and R-squared (see Table 4). However, we see a much better performance of the model after performing 10-fold cross-validation and identifying the most optimal cp value, compared to the initial tree with default parameter setting or the pruned tree. The final regression tree after performing parameter tuning is shown in Figure 15.

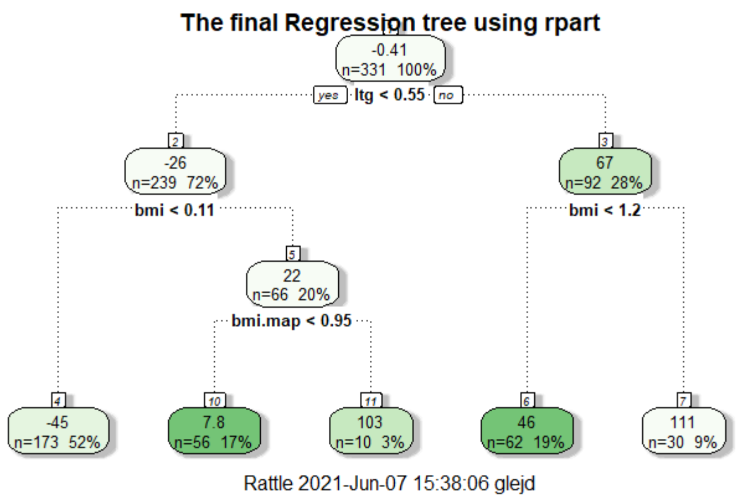

Figure 15: Final regression tree after parameter tuning.

Lastly, the barplot of the variables that have the most largest weight in the prediction are plotted in figure 16. It is clear that ltg, bmi, bmi.ltg, tc, and bmi.2 are the top 5 most important variables in our regression model.

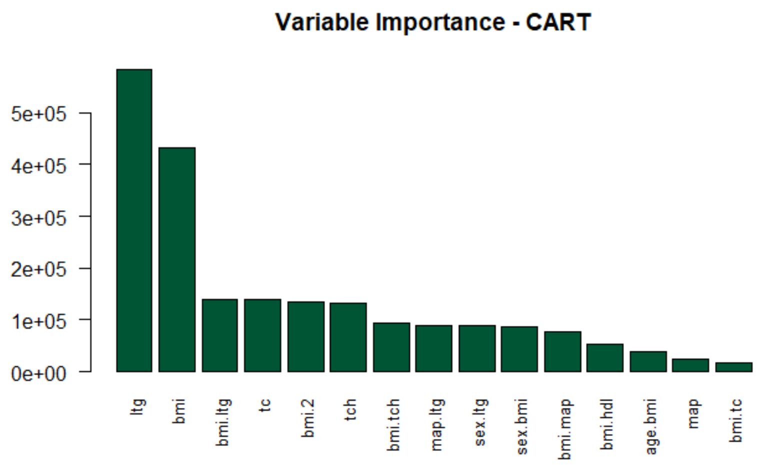

Figure 16: Final regression tree after parameter tuning.

4.2.3 Random Forest Results

As mentioned before, in RF each tree is allowed to grow to a bootstrap resampled data set, and an advantage of it is the fact the RF has an out-of-bag (OOB) sample which gives a good and efficient approximation of the test error. The plot in figure shows the OOB error versus the validation error for the random forest. As expected, there are some differences between them. The out-of-bag error seems to be constantly higher than the test error.

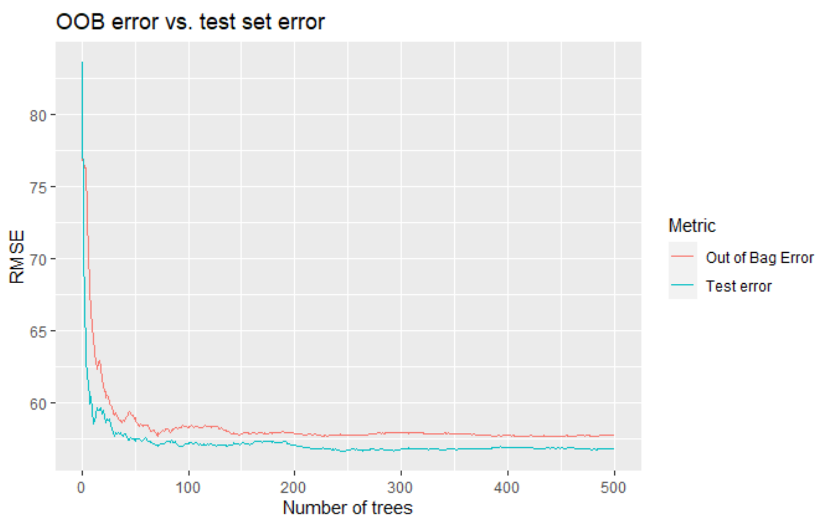

Figure 17: Random forest out-of-bag error versus validation error.

In figure 18, the plot of the random forest model with default parameter setting is shown. The plot illustrates the error rate as we increase the number of trees in the model. The error rate seems to stabilize with 100 trees, but it still continues to decrease even with a bigger number of trees at a much slower rate. According to this plot, the number of trees with the lowest MSE of 57.7 is 479 trees. However, to define the best parameter setting, tuning was performed in the model, which is described in more detail in the next section.

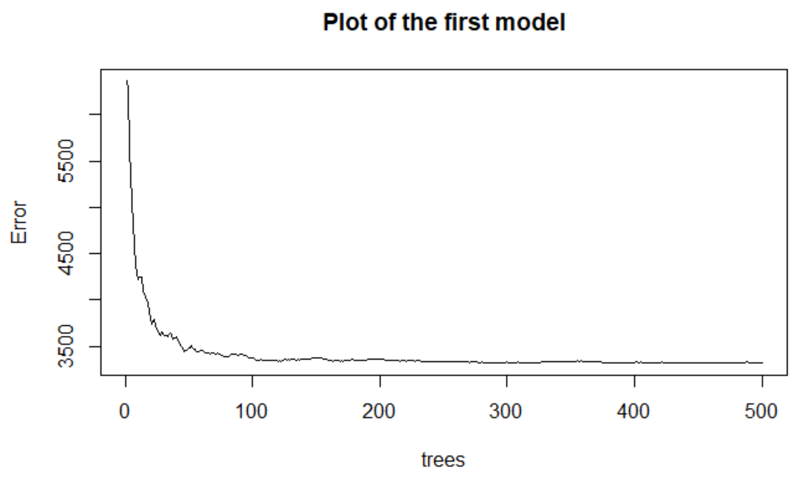

Figure 18: Plot of the error of the first RF model with default parameter setting against number of trees.

RF is mainly controlled by four parameters: mtry stands for the number of variables randomly sampled as candidates at each split, maxnodes stands for the depth of the tree, nodesize stands for the maximal number of observations in each cell, and ntree stands for the number of trees allowed to grow [26]. Hence, 10-fold cross validation is used to identify the optimal parameter setting. To tune the model, a grid of 1-50

Table 5: Random forest model comparison before and after parameter tuning.

	Initial RF with default parameters	RF with tuned parameters
RMSE on test set	56.8	56.8
R Squared on test set	0.465	0.469

is defined by setting the search parameter of TrainControl function to "grid", where each point in the grid represents a specific combination of parameters. In order to select the most optimal model, RMSE was used for selection using the smallest value, aiming for the lowest error, and so the best performance. First, a search for the best mtry is conducted. Figure 19 shows how the RMSE value changes with respect to the number of variables used in the model, mtry score. As a result, the lowest RMSE score is obtained with a value of mtry equals to 10. As it can be clearly seen from the plot, there is a great drop in the cross-validation error from 0-10 randomly selected predictors, and after 10 there is a slow increase in the error.

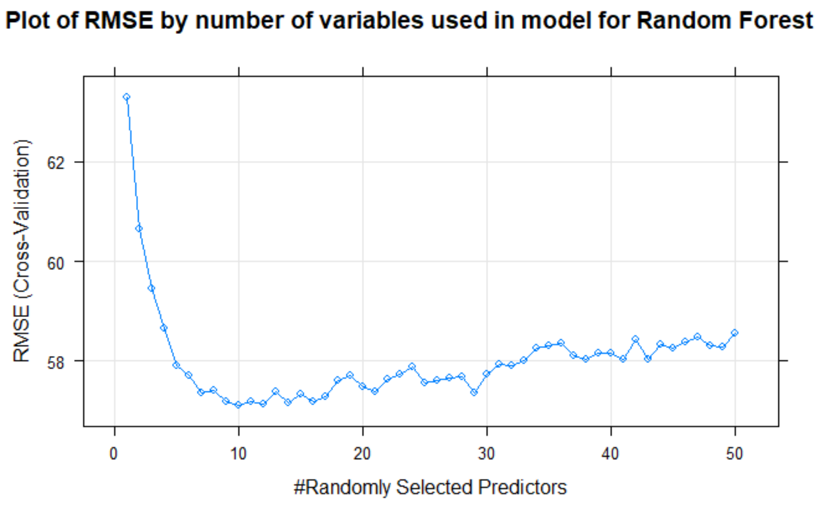

Figure 19: Plot of RMSE by number of variables used in the model for RF.

Next, we search for the best maxnodes parameter value. This is done in iterations, with the range being enlarged by ten in each iteration to attain the lowest RMSE. The lowest RMSE score of approximately 46.8 and the better R-squared value of 0.232 is obtained with a value of maxnode equals to 11. The final parameter tuning is done to find the best ntrees score. This is performed by taking a vector [10, 50, 100, 250, 300, 350, 400, 450, 500, 550, 600, 800, 1000]. As a result, ntree equal to 50 seems to be the most optimal choice, both in terms of RMSE and R-squared values. Finally, the RF model is trained with the following parameter setting: maxnodes = 11, mtry = 10, ntree = 50. As a result, the RMSE on the test set is 56.8, while the R-squared = 0.47. Moreover, as it can be seen from the table below, after tuning we did not receive better results, as expected otherwise.

The 5 most important features for the random forest based on node purity are ltg, bmi, map, bmi.ltg, bmi.2 as shown in Figure 20, while based on MSE value ltg, bmi, map, tch, mpa.glu. Comparing these with the results of the CART, we see that the features with the highest weight in prediction are almost the same.

5 Limitations and Outlook

6 Conclusion

References

[1] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control.* Cambridge University Press, 2019.

[2] Kartik Chaudhary. Understanding audio data, fourier transform, fft and spectrogram features for a speech recognition system. https://towardsdatascience.com/

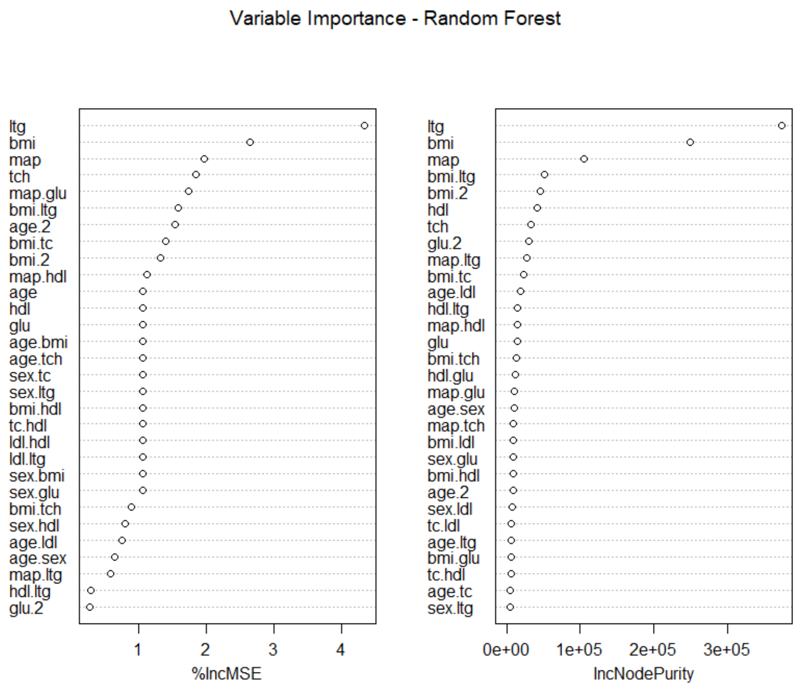

Figure 20: Variable importance for random forest.

understanding-audio-data-fourier-transform-fft-spectrogram-and-speech-recognition-a4072d228520, January 2021.

[3] William T Cochran, James W Cooley, David L Favin, Howard D Helms, Reginald A Kaenel, William W Lang, George C Maling, David E Nelson, Charles M Rader, and Peter D Welch. What is the fast fourier transform? *Proceedings of the IEEE*, 55(10):1664–1674, 1967.

[4] I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Transactions on Information Theory*, 36(5):961–1005, 1990.

[5] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[6] Steven G. Johnson. Fft benchmark results. http://www.fftw.org/speed/, January 2021.

[7] Steven G. Johnson. Juliacon 2019: Keynote: Professor steven g. johnson. https://www.youtube.com/watch?t=2588&v=mSgXWpvQEHE&feature=youtu.be, January 2021.

[8] Steven G. Johnson and Matteo Frigo. Implementing FFTs in practice. In C. Sidney Burrus, editor, *Fast Fourier Transforms*, chapter 11. Connexions, Rice University, Houston TX, September 2008.

[9] Semyon Khokhriakov, Ravi Reddy, and Alexey Lastovetsky. Novel model-based methods for performance optimization of multithreaded 2d discrete fourier transform on multicore processors. *arXiv preprint arXiv:1808.05405*, 2018.

[10] D.N. Rockmore. The fft: An algorithm the whole family can use. *Computing in Science and Engineering*, 2:60 – 64, 02 2000.