Framingham Heart Study Classification

Hannah Pavlovich

March 24, 2024

1 Introduction

This project looks at the Framingham Heart Study data set as found on kaggle (https://www.kaggle.com/datasets/aasheesh200 heart-study-dataset).

The purpose of this project is to build a model that most accurately predict whether a person will have heart disease in Ten Years. I will use classification methods, described further below, with a special focus on Decision Trees, Random Forests, and Gradient Boosting.

2 Exploratory Data Analysis

2.1 About the Data

The Framingham data set has 1 response variable, 15 predictor variables, and 4,238 data points. This makes a dataset of size: (4328, 16). The variables are as follows:

Variable	Description	Type
male	male or female("M" or "F")	integer
age	Age of the patient	continuous, integer
education	years of post-high school education	discrete
currentSmoker	whether or not current smoker	binary, 1=Yes, 0=No
cigsPerDay	number of cigarettes that the person smoked on average in one day	continuous
BPMeds	whether or not the patient was on blood pressure medication	binary, 1=Yes, 0=No
prevalentStroke	whether or not the patient had previously had a stroke	binary, 1=Yes, 0=No
prevalentHyp	whether or not the patient was hypertensive	binary, 1=Yes, 0=No
diabetes	whether or not the patient was hypertensive	binary, 1=Yes, 0=No
totChol	total cholesterol level	continuous
sysBP	systolic blood pressure	continuous
diaBP	diastolic blood pressure	continuous
BMI	Body Mass Index	continuous
heartRate	heart rate	continuous
glucose	glucose level	continuous
TenYearCHD	10-year risk of coronary heart disease CHD	binary, 1=Yes, 0=No

The response variable is **TenYearCHD**, which is denotes if the person has Heart Disease in 10 years. In this dataset, 644 responses are 1, or there are 644 instances of Heart Disease in 10 years. This is 15.19% of the dataset, a fairly small amount.

2.2 Missing Data

This dataset has missing data in the following locations:

variable	no. missing data
male	0
age	0
education	105
currentSmoker	0
cigsPerDay	29
BPMeds	53
prevalentStroke	0
prevalentHyp	0
diabetes	0
totChol	50
sysBP	0
diaBP	0
BMI	19
heartRate	1
glucose	388
TenYearCHD	0

Here we can see there are 645 missing values, with over half in glucose at 388, and second most in education with 105 missing values. The NA's are in 582 data points. Because they are in so many data points, removing the rows would drastically change our dataset. Instead, the NA's are imputed with the means from each variable.

2.2.1 Variable Distribution

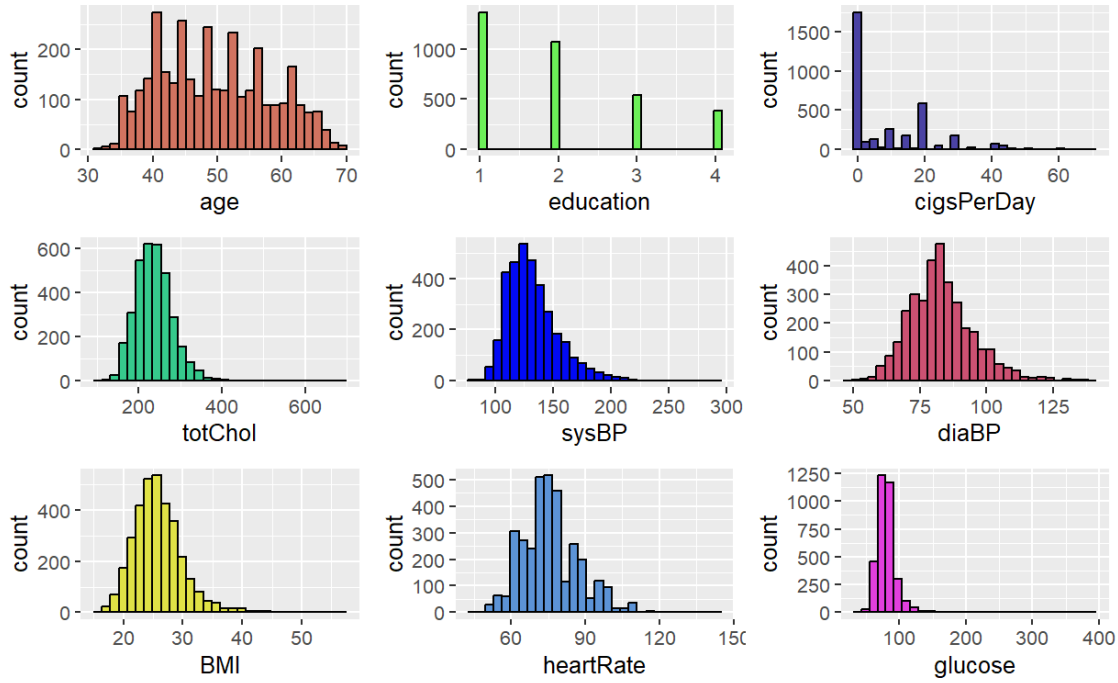


Figure 1: Variable Distribution

Above I have plotted the distributions of the continuous variables in the dataset. Most data is skewed to

the left, from which I can infer that the data is normally distributed with some outliers on the higher ends of glucose and total cholesterol. Moving forward, I decide not to remove the outliers, as I find they do not effect my analysis to a large enough degree.

2.2.2 Multi-Collinearity

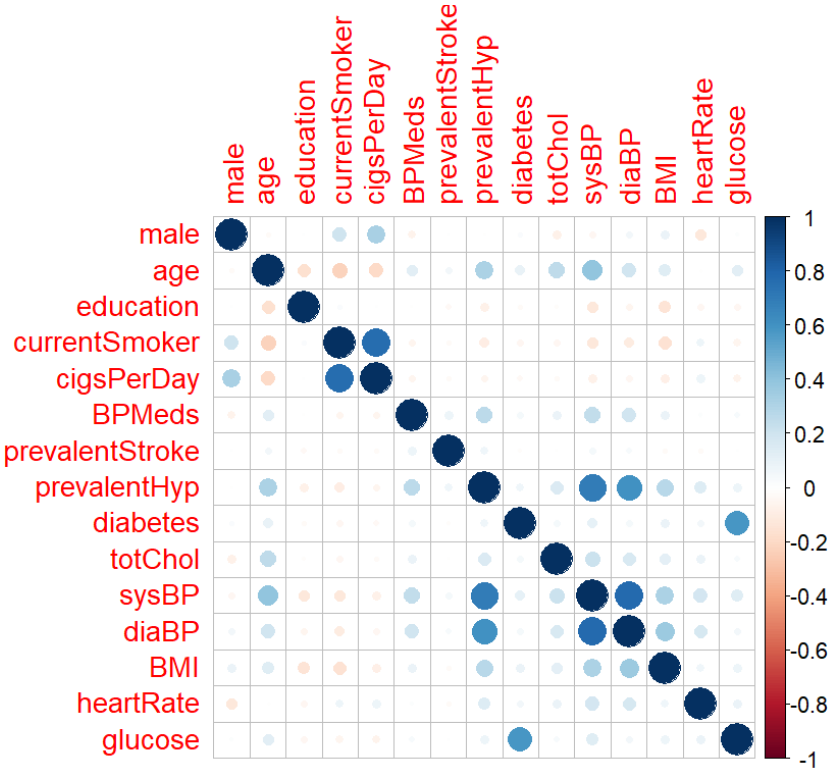
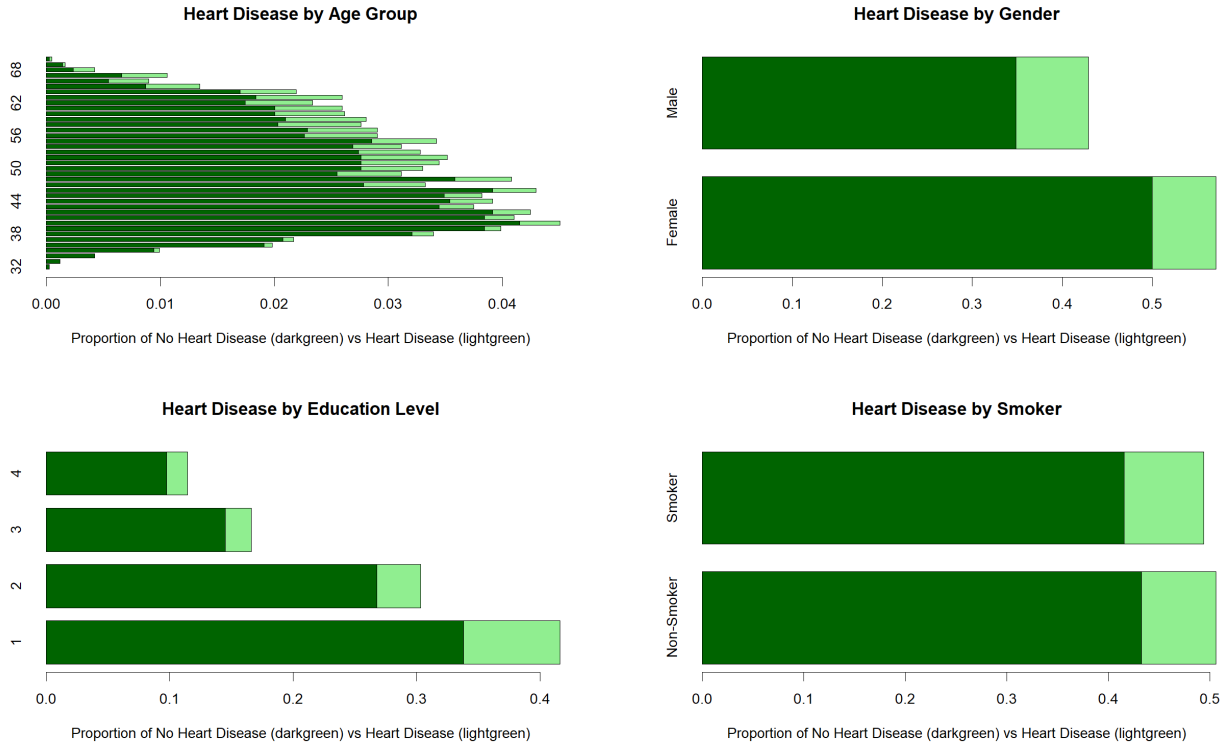


Figure 2: Predictor Correlation

The above plot displays correlations between the predictor variables. There is a strong correlation between currentSmoker and cigsPerDay, as currentSmoker is assumed per cigsPerDay. There is also strong correlation between sysBP and diaBP, as they are both blood pressure.From this information, I can infer there is not strong enough multicollinearity to harm the analysis.

2.3 Heart Disease by Variables



Above are four variables plotted against heart disease. We can see that those with 1 year of education have a higher proportion of heart disease than those with more years, men have more heart disease than women, and heart disease is most prevalent between ages 50-60.

This information can be helpful as we build our models in understanding our outputs.

3 Methods

3.1 Data Preparation

As discussed in EDA, data must be imputed by using the mean of the variable. The response variable: TenYearCHD, will also be made binary.
The data is next split to training and testing sets at 80/20. The data can be split because there are many data points.

3.2 Single Tree

The first method I apply is the Single Decision tree, using rpart package in R and the 'gini' split. I start with the T0 tree, the simplest tree.
After forming the T0 model, I ran a loop to find the best split in the model. When training and testing a classification model, the response may be continuous instead of binary. For this reason, we split the response at a certain point, usually at 0.5, and assign the greater values to 1 and the lesser to 0. This does not always lead to the most accurate results, so my loop tests the split at values between 0.3 and 0.7
Once the optimal split is found, I look to see if the branches can be pruned and apply any pruning techniques.

3.3 Random Forest

I next look at Random Forest using the randomForest Package from R. Here, I can also extract the "Important Variable" and see if they align with my EDA.
The Random Forest model will be tuned by testing different numbers of trees, node size, and number of variables sampled. In selecting the best paramters, I need to be aware of very low training errors, as this could be the result of overfitting instead of a good model.

3.4 Boosted Tree

I xgboost package from R to find the Boosted Tree. The package has robust cross validation methods built in, so I will not perform the same tuning methods that I did in the previous tree methods.

3.5 Comparisons with other Classifications

The purpose of this assignment is to find the best model for predicting heart disease in patients in Framingham. To do this, I introduce models previously studied in this course.
To find the best model, I run a Monte Carlo simulation using the previous tree methods and to create LDA, QDA, Naive Bayes, Logistic Regression, and KNN models. From here, I collect the average of all simulations and compare the testing and training errors for each model.

4 Results

4.1 Single Tree

4.1.1 Grow Tree

To train the model, first grow the tree using the RPart package. Below are the written and visual results for T0:
n= 3390

node), split, n, loss, yval, (yprob)
    * denotes terminal node

1) root 3390 526 0 (0.84483776 0.15516224)
    2) age < 46.5 1416 103 0 (0.92725989 0.07274011) *
    3) age>=46.5 1974 423 0 (0.78571429 0.21428571)
        6) sysBP < 176.75 1835 357 0 (0.80544959 0.19455041) *
        7) sysBP >= 176.75 139 66 0 (0.52517986 0.47482014)
            14) glucose < 81.98338 69 23 0 (0.66666667 0.33333333) *
            15) glucose >= 81.98338 70 27 1 (0.38571429 0.61428571) *

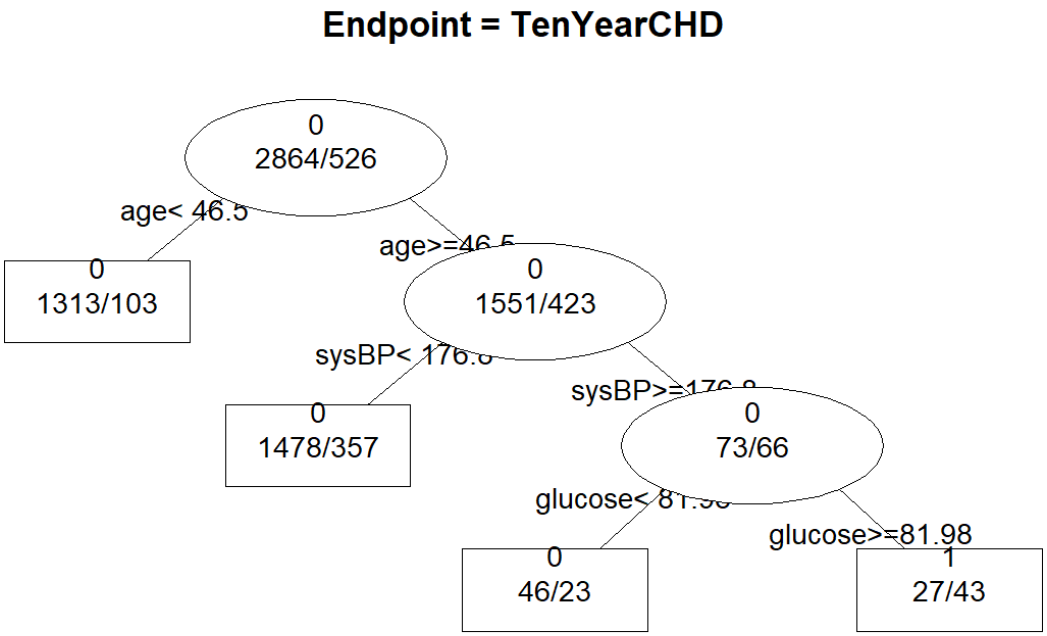


Figure 3: T0 Tree

The first split we see here is age at 46.5, which is in line with what we saw in EDA. The bar plot indicated that age was a good indicator of heart disease. Looking at the matrix, the split at age is the largest first jump, followed by sysBP and glucose. Each step minimizes the loss function, and at glucose the nodes have reached their minimum.

4.1.2 Split Tuning

Above, we split the response at 0.5, with continuous values over 0.5 designated as Yes Heart Disease, and under 0.5 as No Heart Disease. After running a loop testing this split the following data is found:

	split	test.error	training.error
1	0.30	0.157227	0.150943
2	0.35	0.150442	0.150943
3	0.40	0.150442	0.150943
4	0.45	0.150442	0.150943
5	0.50	0.150442	0.150943
6	0.55	0.150442	0.150943
7	0.60	0.150442	0.150943
8	0.65	0.155162	0.150943
9	0.70	0.155162	0.150943

Each split has the same training error, and the testing error is also very similar across all splits. For this reason, I maintain the split at 0.5 moving forward.

4.1.3 Pruning

Next I perform pruning on the tree. The tree is already small with only 4 branches, which is also seen in the image below:

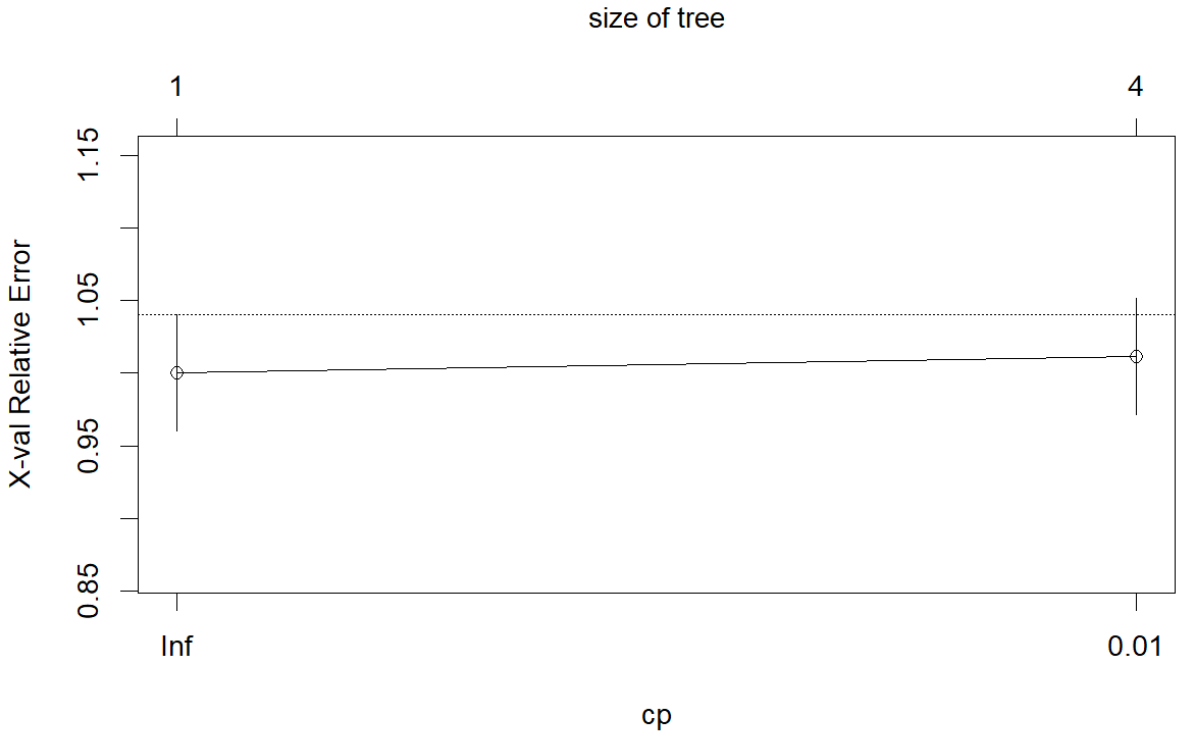


Figure 4: Complexity Parameter

From the image, I can infer that there is not much change per size of tree or cp (complexity parameter).

Next, I extract the optimal CP: **0.01013942**. From here, I retrain and test the model. The result: **testing error = 0.13915**, is better than the results we saw without the tuned complexity parameter. Thus, the tree does need to pruned to achieve a better model.

4.2 Random Forest

Next I look at Random Forests, which, while harder to interpret, provide more accurate results by using hundreds of decisions trees.

Using the Gini Index again, I built a RandomForest model using the standard parameters. Plotting the variable importance, the most important variables by Gini and sysBP, BMI, and totChol. Using "Accuracy" the most important variables are SysBP, diaBP, and age.

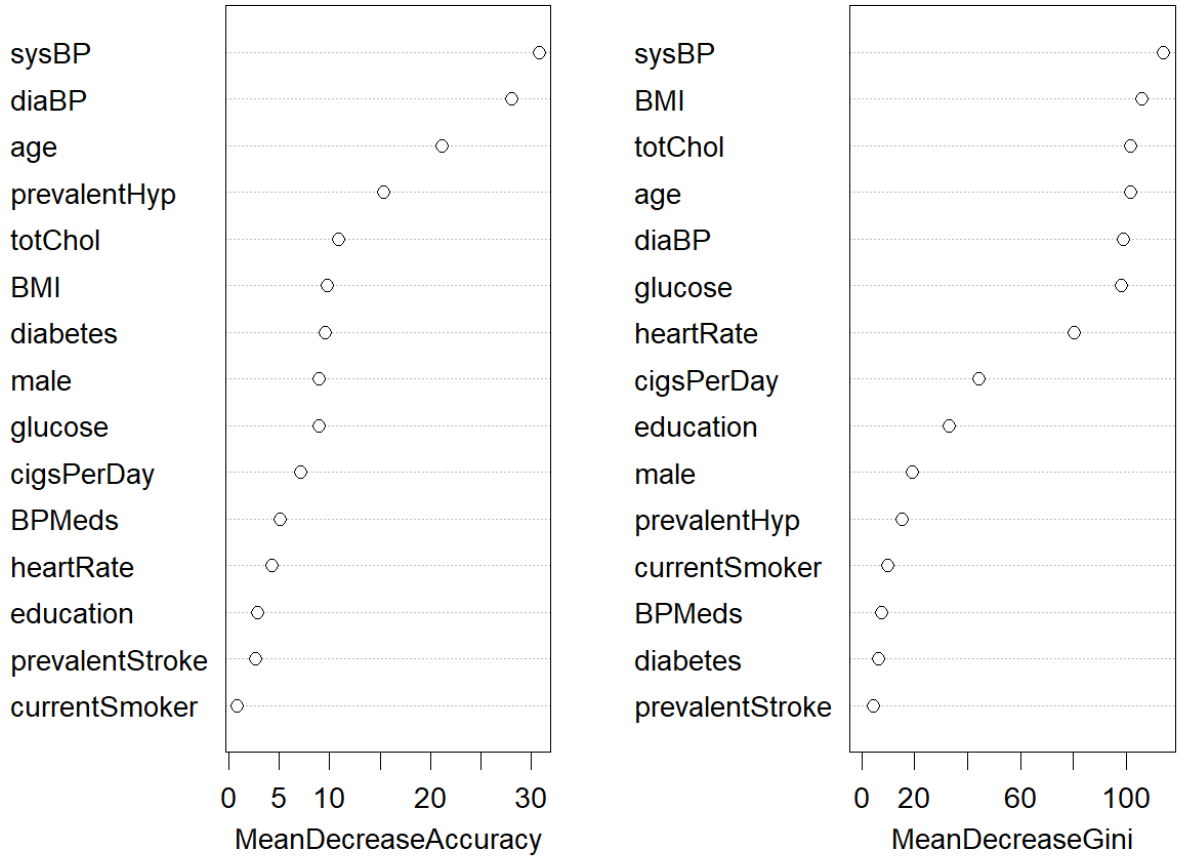


Figure 5: Variable Importance

The decision tree had age as its first split, and then SysBP and Glucose, so there is some overlap with "important variables" between the two methods. I could infer then that sysBP and age are the two most important factors, so I may adjust my study to have better measurements around these methods, or try to find different variables that may produce an even better model.
The results for this model are as follows:

	training.error	test.error
1	0.000295	0.142689

The very low training error suggests that this model may be overfit, and so further investigation should be done before accepting the model and its testing error. The testing error, meanwhile, is slightly better than the decision tree training error at **0.142689 vs 0.150943**

4.2.1 Tuning Random Forest

The Random Forest created is based on the default parameters:

ntree = number of tress to grow, and the default is 500.

mtry = number of variables randomly sampled as candidates at each split. The default is $\sqrt{p}$=3 for classification

nodesize = minimum size of terminal nodes. The default value is 1 for classification

To find the most accurate model, I then tuned the different parameters, looping through multiple values of all three to find the lowest testing error. The parameters I used were:

ntree = seq(300,700,50)
mtry = seq(1,10,1)
nodesize = seq(1,6,1)

In my testing, I noticed that a large number of combinations returned a training error of 0, which could mean that the model is overfit and not an accurate model. Because of this, to extract the top values, I only looked at values with a training error between 4-10% to avoid this problem.

	ntree	mtry	nodesize	training.error	test.error
8	300	2	2	0.089381	0.143868
24	300	4	6	0.051327	0.143868
30	300	5	6	0.045133	0.143868
63	350	1	3	0.154277	0.143868
64	350	1	4	0.153392	0.143868
66	350	1	6	0.153982	0.143868
68	350	2	2	0.089971	0.143868
69	350	2	3	0.098525	0.143868
70	350	2	4	0.104130	0.143868
71	350	2	5	0.105015	0.143868
77	350	3	5	0.053687	0.143868
125	400	1	5	0.153392	0.143868
127	400	2	1	0.073746	0.143868
128	400	2	2	0.090855	0.143868
130	400	2	4	0.104425	0.143868

As seen above, the lowest testing errors with the tuned parameters are not as low as the testing error with the default parameter. I will not use the default parameters, however, because the default parameters result in an overfit model. Moving forward, I will use the first row of the table: **ntree=300, mtry=2, nodesize=2**

Choosing the proper parameters results in a better model because it reduces cahnces of overfitting.

4.3 Gradient Boosting

Next I looked at gradient boosting, which looks at the predictions of multiple weak models to produce one strong model.
The parameters I chose in this model were:

booster = "gbtree"
eta = 0.01
max-depth = 8
gamma = 4
subsample = 0.75
colsample-bytree = 1
objective = "multi:softprob"
eval-metric = "mlogloss"
num-class = length(levels(data2-train$TenYearCHD))+1

The results of the model are as following:

	training.error	test.error
1	0.1551622	0.1391509

The training error for this model is markedly better than for the random forest or single decision tree.

4.4 Other Methods

I next chose to test the tree methods against the methods we have learned in this course. I ran a Monte Carlo simulation 100 times, and the results are below:

	mean	median	var	CI
Decision Tree Training	0.150442	0.150442	0.000000	Inf
Decision Tree Testing	0.150943	0.150943	0.000000	Inf
Random Forest Training	0.044142	0.043953	0.000002	59.427432
Random Forest Testing	0.144858	0.145047	0.000005	126.780093
Gradient Boosting Training	0.155162			
Gradient Boosting Testing	0.139151			
LDA Training	0.148059	0.148083	0.000011	86.041108
LDA Testing	0.149882	0.148585	0.000142	24.640290
QDA Training	0.169752	0.170059	0.000013	92.825521
QDA Testing	0.172665	0.171580	0.000196	24.180283
Naive Bayes Training	0.175286	0.175516	0.000011	103.484474
Naive Bayes Testing	0.176108	0.175708	0.000136	29.613420
Logistic Regression Training	0.144360	0.143953	0.000011	86.524258
Logistic Regression Testing	0.146344	0.146226	0.000141	24.121385
KNN Training	0.147003	0.146903	0.000012	83.584780
KNN Testing	0.152476	0.154481	0.000124	26.834102

Because the xgboost package already provides extensive cross validation, it was not included in the Monte Carlo simulation.

In order of training error, the most accurate models are: Gradient Boosting, Random Forest, Logistic Regression, LDA Testing, Decision Tree, KNN, QDA Testing, and Naive Bayes. This will be discussed more in the next section.

5 Findings

Gradient Booting is the most accurate model with a Testing Error of 13.92%. Gradient Boosting is the most robust to overfitting, which can be a problem with Random Forest models. This model was the most difficult to train and took the longest to produce results, even outside of the Monte Carlo simulation. This

data set has only 16 variable, including the binary response. For a larger dataset, this model may not be appropriate unless extensive variable selection were first performed.

The **Random Forest** model is the second most accurate with a Testing Error of 14.49%. As mentioned above, Random Forest is prone to overfitting, which is why tuning the parameters was important in building this model. The ensemble method does work well with this dataset, although it is computationally expensive. An advantage of running this algorithm is the output of the most important variables. The information that systolic blood pressure and BMI are important variables may help a doctor explain to her patient their risks most succinctly.

The **Logistic Regression** model performs well at 14.63% testing error. Logistic regression models are in general good classification methods, especially when the dataset is less complicated. Even though there are many datapoints, 16 variables is relatively small. The low accuracy on the modle also indicates that the data suggests a linear boundary. That this model performs better than KNN and Naive Bayes also indicates that the response separation is linear, as the advantage of the malleable boundary is not taken. Because this method is also less computationally expensive, I would use Logistic Regression again for a similar dataset.

Linear Discriminant Analysis (LDA) performs well on this dataset with a testing error of 15.00%. In EDA, I saw that the continuous variables followed the gaussian distribution with some outliers, which indicates that an LDA would fit well. LDA is sensitive to outliers, and the EDA did show what could possibly be outliers, which may be why this model did not perform the best. A good solution would be to remove outliers and run this model again. This model may perform better on datasets with more normal data, and I would not choose this model with the histograms presented in EDA.

Logistic Regression and LDA performed better than other non-ensemble classification methods, from which I can infer that there is a linear decision boundary. These are two good methods for a similar sized data set because they are not as computationally expensive as the ensemble method and the results do not have a large loss of accuracy.