Bitcoin Heist Dataset

Random Forests & Boosting

Introduction

In this report I will explore a data set using random forests and boosting ensemble methods. I will compare these methods with baseline methods such as Logistic Regression, Linear Discriminant Analysis, Naive Bayes, and a single Classification Tree method.

(e) About the Data Set

The dataset that will be analyzed is the Bitcoin Heist dataset from the UCI Machine Learning repository. It has 2916697 rows of observations and 10 columns of variables. The last variable, "label" is the independent variable Y and the remaining variables are the dependent variables X.

The dataset can be found at: https://archive.ics.uci.edu/ml/datasets/BitcoinHeistRansomwareAddressDataset

Hypothesis

I believe that the ensemble methods of random forest and boosting will perform better than baseline methods. Random Forest is known to reduce the variance by creating several trees in parallel and taking their average. Boosting is used to improve upon training error, using the weighted average of trees built sequentially.

(e) Exploratory Data Analysis

There are 2916697 rows of observations with 10 columns. The last column 'label' is the independent variable, and the remaining columns are the dependent x variables. The columns of variables are as described below:

- address: String. Bitcoin address.
- year: Integer. Year of transaction.
- day: Integer. Day of the year. 1 is the first day, 365 is the last day.
- length: Integer. Quantifies mixing rounds on Bitcoin.
- weight: Float. Quantifies the merge behavior of transactions by amount.
- count: Integer. Number of merging transactions.
- looped: Integer. Counts how many transactions split their coins.
- neighbors: Integer. Indicates number of neighbors a transaction had.
- income: Integer. Income in terms of Satoshi amount (1 bitcoin = 100 million satoshis).
- label: Category String. Name of the ransomware family (e.g., Cryptxxx, cryptolocker etc) or white (not ransomware).

Read in the Data

```
##    address               year           day            length
##  Length:2916697     Min.   :2011   Min.   :  1.0   Min.   :  0.00
##  Class :character   1st Qu.:2013   1st Qu.: 92.0   1st Qu.:  2.00
##  Mode  :character   Median :2014   Median :181.0   Median :  8.00
##                     Mean   :2014   Mean   :181.5   Mean   : 45.01
##                     3rd Qu.:2016   3rd Qu.:271.0   3rd Qu.:108.00
##                     Max.   :2018   Max.   :365.0   Max.   :144.00
##      weight             count             looped          neighbors
##  Min.   :   0.0000   Min.   :    1.0   Min.   :    0.0   Min.   :    1.000
##  1st Qu.:   0.0215   1st Qu.:    1.0   1st Qu.:    0.0   1st Qu.:    1.000
##  Median :   0.2500   Median :    1.0   Median :    0.0   Median :    2.000
##  Mean   :   0.5455   Mean   :  721.6   Mean   :  238.5   Mean   :    2.207
##  3rd Qu.:   0.8819   3rd Qu.:   56.0   3rd Qu.:    0.0   3rd Qu.:    2.000
##  Max.   :1943.7488   Max.   :14497.0   Max.   :14496.0   Max.   :12920.000
##      income            label
##  Min.   :3.000e+07   Length:2916697
##  1st Qu.:7.429e+07   Class :character
##  Median :2.000e+08   Mode  :character
##  Mean   :4.465e+09
##  3rd Qu.:9.940e+08
##  Max.   :4.996e+13
```

Data Cleaning

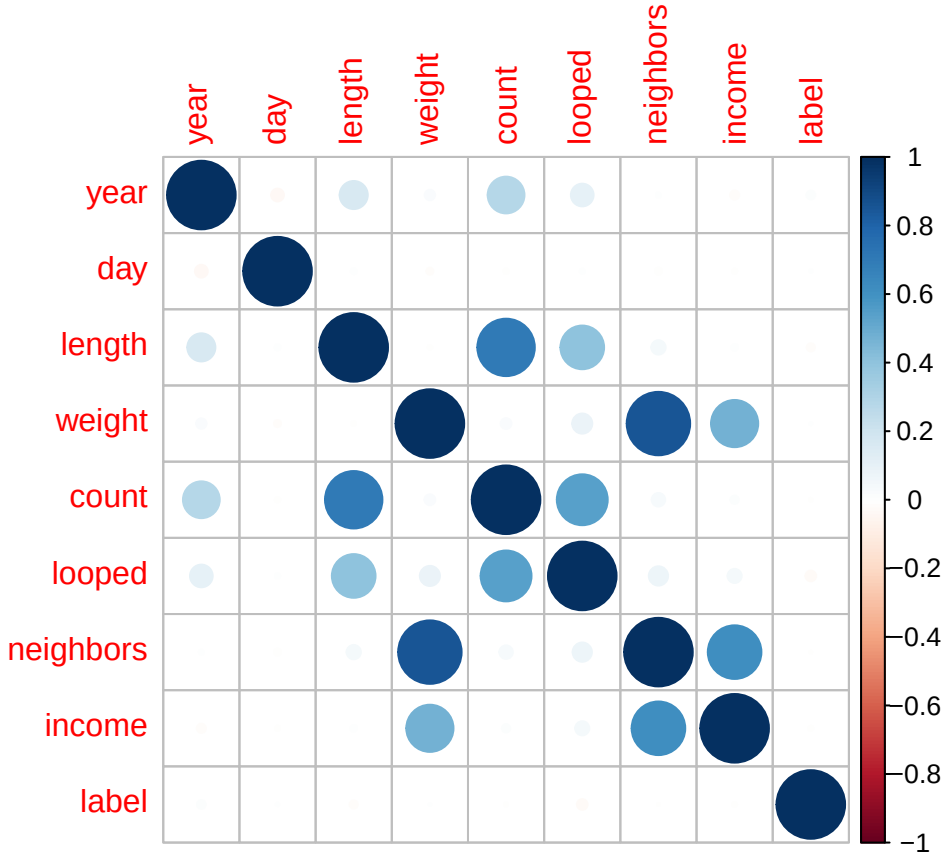
The dataset is very large and contains data all the way from year 2009 to 2018. For speed of analysis and simplicity, I have sampled 14370 rows of the dataset (0.5% of data) and will use this as my data for this assignment. To clean the dataset, the 'address' column was removed and variable types have been set as numeric where appropriate. The y-variable label has also been made into a 0, 1 binary factor. Where label=white (not ransomware), the value is 0 and where label is anything else, meaning that it was ransomware, the value is 1.

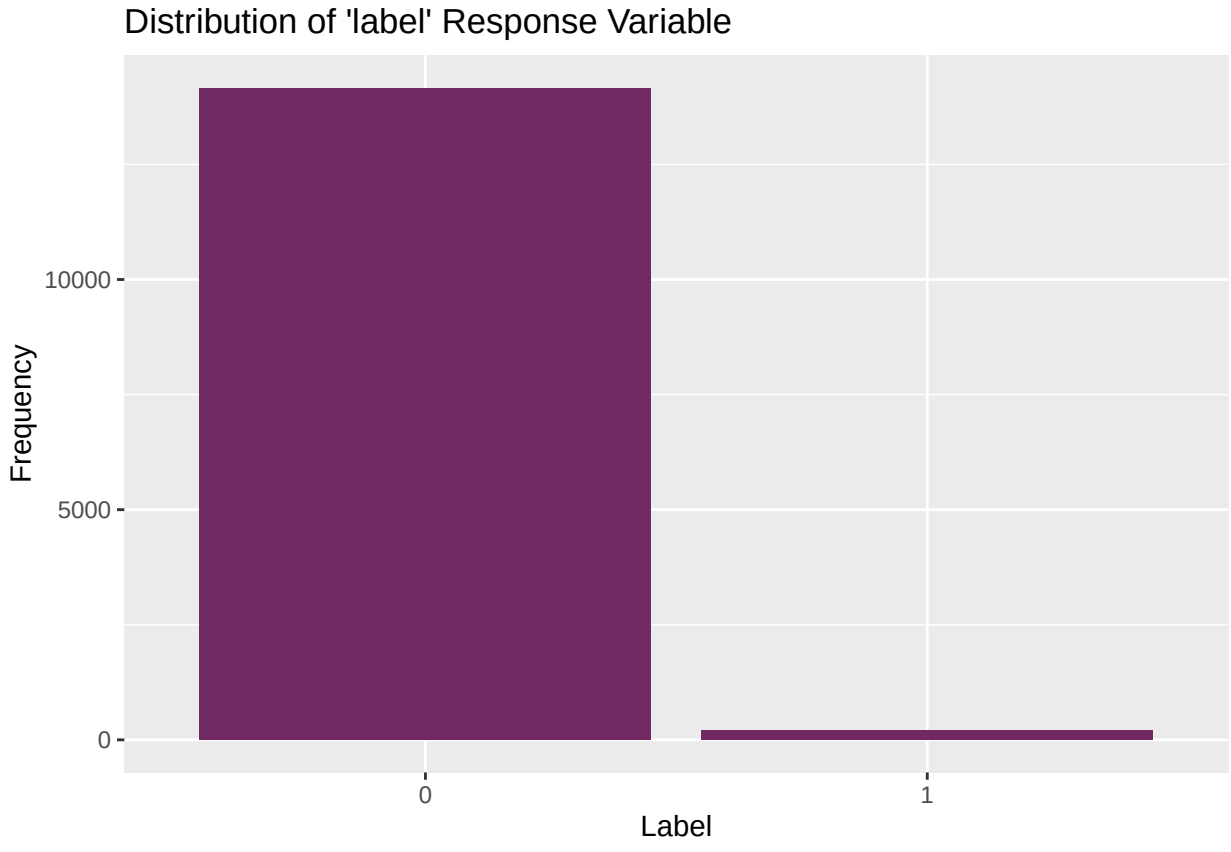
```
##         year day length    weight count looped neighbors     income label
## 193360  2011 162      2 2.0000000     2      0         1  198000000     0
## 816976  2013  56     52 0.0312500     1      0         2  715651058     0
## 2631102 2018  45      2 0.5000000     1      0         1  999900000     0
## 2090820 2016 235      2 0.8333333     1      1         4  391319500     0
## 2541165 2017 320    144 2.3335244  6464      0         2  482043256     0
## 507507  2012 111      0 1.0000000     2      0         2 3140481080     0
```

Split into training and testing sets

One third of the dataset was split into testing data and 2/3 will be used as training data. The response values of 'label' were stored as variables for comparison to calculate testing error.

Visualize dataset





The correlation plot of the variables with one another is shown above. Only about 1.45% of the dataset is ransomware (label=1) as the mean of label is 0.01451. As you can tell, the distribution of the response variable 'label' is highly skewed with most being equal to zero (not ransomware).

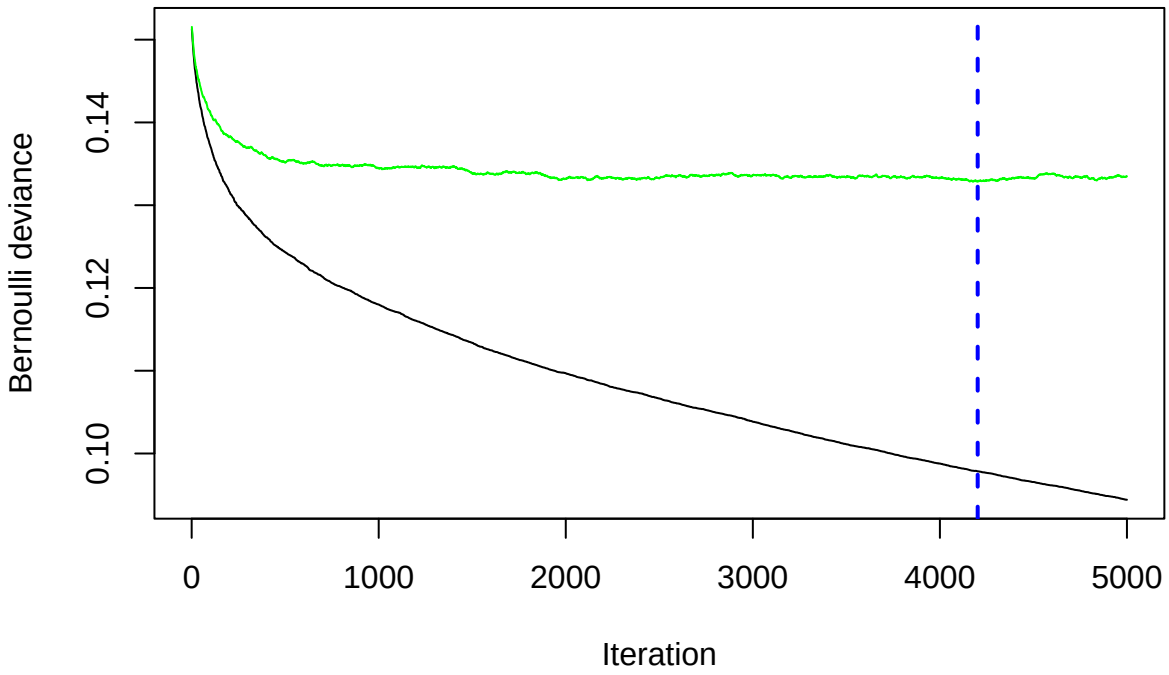
Methods

(a) (d) Boosting Algorithm

The generalized boosting machine library will be used for AdaBoost and the Gradient boosting machine. The distribution is Bernoulli for classification. I will tune the parameters of n.trees, shrinkage, and interaction depth. I will perform a grid search and for loop starting with various values of shrinkage from 0.005 to 0.3. Shrinkage refers to learning rate. See appendix for code.

```
##   learning_rate      RMSE trees   Time
## 1         0.050 0.3619613  4939 20.611
## 2         0.100 0.3623120  1992 20.368
## 3         0.300 0.3641267   465 20.690
## 4         0.010 0.3654719  4960 20.551
## 5         0.005 0.3675712  4959 20.554
```

The learning rate value with the lowest RMSE is 0.05. This is the tuned value for the shrinkage parameter.

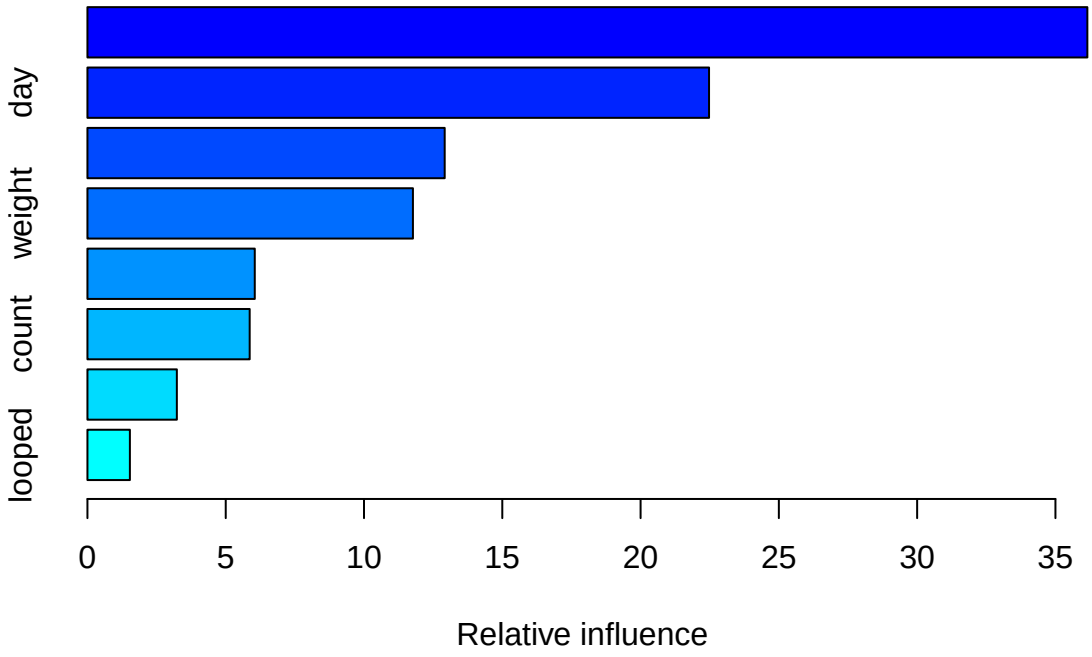



```
## The optimal n.trees iterations is: 4202
```

One can estimate the optimal number of iterations using the gbm.perf function. This is determined through cross validation. The number above shows the optimal number of iterations or trees M shown by the dashed blue line. The black and green lines show the deviance of training and testing datasets respectively.

```
##   n.trees shrinkage interaction.depth      rmse
## 1    4202      0.05                 5 0.3516602
## 2    4202      0.05                 7 0.3530398
## 3    4202      0.05                 3 0.3552706
```

Next, the interaction.depth parameter was tuned through a grid search of values from 5 to 7. The optimal value was 5 because it had the lowest RMSE.




```
##                 var   rel.inf
## income       income 36.154300
## day             day 22.474902
## year           year 12.917259
## weight       weight 11.769268
## length       length  6.050337
## count         count  5.866267
## neighbors neighbors  3.232995
## looped       looped  1.534670
```

The final boosting model was fitted using the tuned parameter values. A summary of the relative influence of each predictor variable is shown above. Income seems to be the variable that is most influential.

Training and Testing Errors of Boosting Method

```
## Predicted classification probabilities of first ten rows:

##  [1] 8.794537e-05 1.675972e-03 6.528085e-05 1.025453e-03 1.063782e-04
##  [6] 8.645405e-06 3.894623e-05 6.711007e-05 2.435774e-04 2.487775e-06

## Predicted label values of first ten rows:

##  [1] 0 0 0 0 0 0 0 0 0 0

## The training error is: 0

## The testing error is: 0.0148225469728601
```

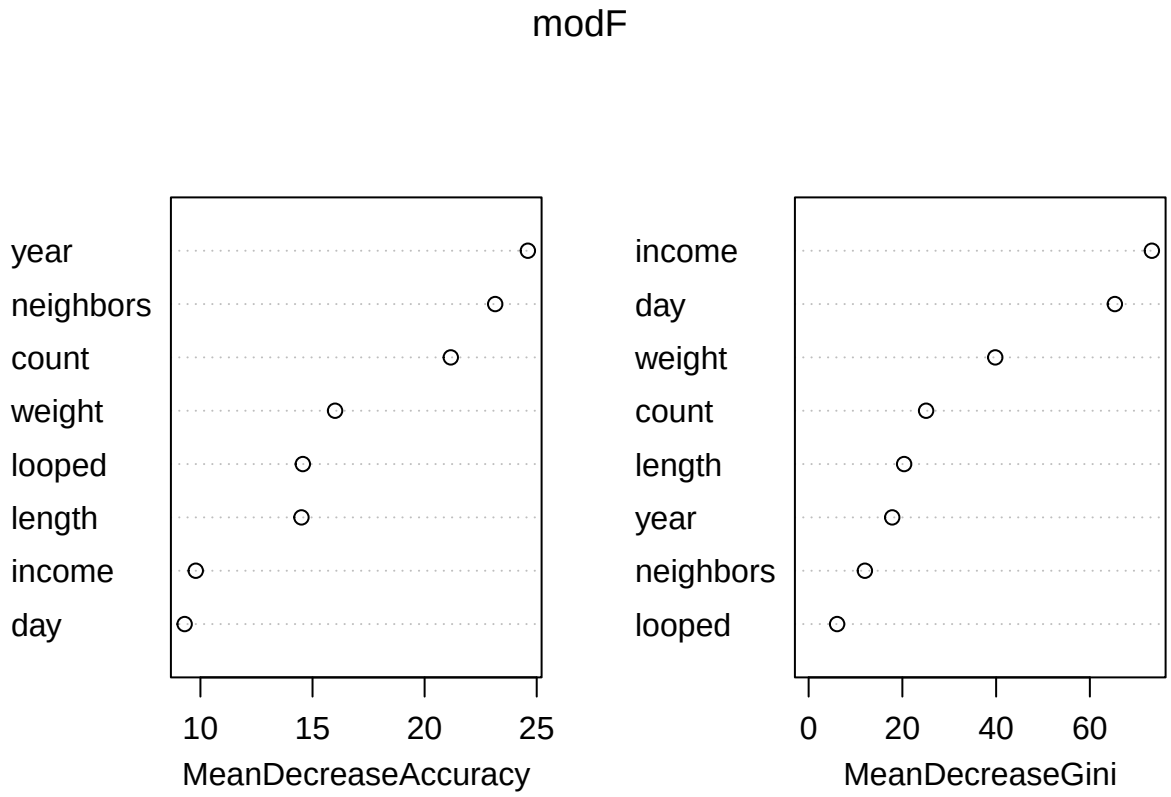
(a) (d) Random Forest

The randomForest library will be used to fit a random forest model. The parameter of ntree will be tuned using the caret package. The values of mtry and nodesize parameters will be determined using the given guidelines in lecture. For classification, mtry is equal to the square root of the number of predictor variables and nodesize is equal to one.

```
## The tuned mtry value is: 3

## The tuned nodesize value is: 1

## The tuned ntree value is: 500
```

Using the above tuned values, the final random forest model will be built.

```
##           MeanDecreaseAccuracy
## year                 24.605340
## day                   9.295847
## length               14.506009
## weight               16.005739
## count                21.166882
## looped               14.567266
## neighbors            23.148471
## income                9.791995

##           MeanDecreaseGini
## year             17.832644
## day              65.328194
## length           20.372167
## weight           39.816107
## count            25.063455
## looped            6.075421
## neighbors        11.997689
## income           73.227611
```



Above are the importance measures of each variable. Importance is determined by the mean decrease in accuracy (MeanDecreaseAccuracy) or the mean decrease in node impurity (MeanDecreaseGini). The plots visually show that income is the most important variable based on decrease in node impurity and neighbors is the most important based on decrease in accuracy.

```
## The predicted testing error is: 0.0144050104384134
```

(b) (d) Comparison with Other Methods

Stepwise Logistic Regression

```
##
## Call:
## glm(formula = label ~ count + looped + income, family = "binomial",
##     data = btctrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.3469  -0.1860  -0.1829  -0.1533   3.6164
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.037e+00  1.028e-01 -39.269  < 2e-16 ***
## count        9.347e-05  4.451e-05   2.100  0.03573 *
## looped      -5.975e-04  2.497e-04  -2.393  0.01672 *
```

```
## income      -2.186e-10  7.921e-11  -2.759  0.00579 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1452.7  on 9579  degrees of freedom
## Residual deviance: 1420.2  on 9576  degrees of freedom
## AIC: 1428.2
##
## Number of Fisher Scoring iterations: 12

## The testing error is: 0.0144050104384134
```

Stepwise logistic regression selects for predictor variable to be included in the final logistic regression model. It iterates through several steps with different variables included.

Linear Discriminant Analysis

```
## Call:
## lda(btctrain[, 1:8], btctrain[, 9])
##
## Prior probabilities of groups:
##          0          1
## 0.98549061 0.01450939
##
## Group means:
##       year      day   length    weight    count    looped neighbors     income
## 0 2014.467 180.1087 44.79695 0.5367324 720.6433 233.67408  2.301133 4943241126
## 1 2014.856 175.5036 44.54676 0.5528921 850.9496  54.01439  1.913669  505192471
##
## Coefficients of linear discriminants:
##                       LD1
## year       1.850645e-01
## day       -1.050401e-03
## length    -4.265288e-03
## weight     1.388447e-01
## count      4.808447e-04
## looped    -1.005242e-03
## neighbors -1.352020e-02
## income    -8.173186e-13

## The testing error is: 0.0144050104384134
```

Naive Bayes

```
##           Length Class  Mode
## apriori   2      table  numeric
## tables    8      -none- list
## levels    2      -none- character
## isnumeric 8      -none- logical
## call      4      -none- call

## The testing error is: 0.818371607515658
```

Single Tree Model

A single tree method will be a baseline comparison model. The optimal complexity parameter (cp) parameter was tuned and used to create the model.

```
## Call:
## rpart(formula = label ~ ., data = btctrain, method = "class",
##     parms = list(split = "gini"))
##   n= 9580
##
##   CP nsplit rel error xerror xstd
## 1  0      0         1      0    0
##
## Node number 1: 9580 observations
##   predicted class=0  expected loss=0.01450939  P(node) =1
##     class counts:  9441   139
##    probabilities: 0.985 0.015

## The testing error is: 0.0144050104384134
```

Results

(c) Comparison of Models' Testing Errors

Below are the results of all the models. We are interested in comparing the testing errors.

```
## Testing Error of Each Model:

##                           model testing_error
## 1                      Boosting    0.01482255
## 2                 Random Forest    0.01440501
## 3             Stepwise Logistic    0.01440501
## 4 Linear Discriminant Analysis    0.01440501
## 5                   Naive Bayes    0.81837161
## 6                   Single Tree    0.01440501
```

Findings

Based off testing errors, boosting and random forest ensemble methods were similar to baseline methods. My hypothesis was that they would have an advantage because Random Forest reduces the variance by creating several trees in parallel and taking their average. Boosting improves error by using the weighted average of trees built sequentially. But any difference is rather small as boosting had only marginally higher testing error compared to stepwise logistic regression, linear discriminant analysis, and single tree. In this dataset, most of the values for the label variable are 0, as seen in the exploratory data analysis. This imbalance may be why the results are so similar across the board, because most predictions by all the different methods would result in 0 and be similarly accurate. All methods had similar testing errors, with the exception of Naive Bayes. Naive Bayes had a much higher testing error because many predictions were 1 when in fact they were 0. Naive Bayes requires the assumption of independence amongst predictors, which is not fully the case here, as shown in the correlation plot in the EDA section. Also this method works best with high dimensional data which is not the case here, as there are far less features than rows of observations.

The dataset that was used is the Bitcoin Heist dataset from the UCI Machine Learning repository. It is a very large dataset of 2916697 observations spanning many years. As such, a random subset of this data was used for the purposes of this assignment. The data was cleaned to remove the 'address' column of the alpha-numeric bitcoin addresses and the 'label' variable was changed to a binary factor of 0 or 1 where 0 is not ransomware and 1 is ransomware. One third of the data was divided as the testing data and the remaining 2/3 was training data. The goal was to create models to classify whether or not the observation is ransomware based on the predictor variables of year, day, length (mixing rounds), weight (merge behavior), count (number of transactions), looped (coin splitting), neighbors, and income. Most of the label values are 0, indicating not ransomware. Ransomware is a rare event, with only 1.5% of data being ransomware. This makes tbe data highly skewed, but makes sense in the real world context of where this data.

In the boosting method, the parameters of n.trees, shrinkage, and interaction depth were tuned using grid search and for loops of the various combinations, starting with shrinkage which refers to the learning rate. The parameter n.trees is the number of iterations or base trees. Interaction depth refers to the interactions between X's. For the random forest method, grid search and a for loop was also used to tune the parameter ntree while the guidelines for classification were used to determine mtry and nodesize. Income was found to be the most influential variable in boosting and the most important by mean decrease in node impurity in random forest.

Appendix

Code has been attached as a separate document.