Final Report

Andrew Ma

May 6, 2020

Contents

1. First Report
   a. Summary Statistics

2. Second Report
   a. Logistic Regression
   b. K Nearest Neighbors

3. Third Report
   a. Splitting the dataset
   b. Ridge Regression
   c. Lasso Regression
   d. Decision Tree

4. Fourth Report
   a. Splitting the dataset
   b. Bagging
   c. Random Forest
   d. Boosting
   e. XGBoost
   f. Neural Network
   g. Support Vector Machine

5. Model Comparisons
6. Conclusion

Introduction
According to a CNBC article from April 2020, credit card debt hit an all time high of $930 billion with young Americans having the highest delinquency rate.* Credit Card use is increasing as well and with that, comes many people applying for a credit cards. These credit card applications are reviewed by algorithms and models that will determine whether an individual gets accepted or denied a credit card.

In this project, I observed the Credit Card datset from the book, "Applied Econometrics with R" (Kleiber-Zielis, New York. ISBN 978-0-387-77316-2). This dataset contained the credit history for a sample of applicants for a type of credit card. The dataset also consisted of 1,319 observations on 12 variables. The response variable was "card", which was whether or not the application for a credit card was accepted. With the data, statistical analysis and machine learning models will be implemented to predicted whether or not a credit card application will be accepted.

To observe the accuracy rate of each model, I can measure the quality of fit of the classification for each model by splitting the dataset into a training and testing set, and check the rate of observations predicted correctly. We look at classification accuracy to see which models are predicting with highest accuracies. I will research which machine learning model will be best at predicting the status of a credit card application.

The first report of this model deals with summary statistics and observations regarding our dataset. In the second report, we are looking at logistic regression and its performance on our dataset. I also build a K nearest neighbors model in my second report too. In the third report, I look at shrinkage methods, Lasso and Ridge, and also a classification tree. In the fourth report, I implement even more tree based models such as Bagging, Random Forest, Boosting, and XGBoost. I also then run a neural network model and support vector machine.

- https://www.cnbc.com/select/us-credit-card-debt-hits-all-time-high/

First Report

```
library(ISLR)
library(AER)
library(ggplot2)
data("CreditCard")
CreditCard = data.frame(CreditCard)
```

1a. Summary Statistics

```
summary(CreditCard)
```

```
##   card        reports             age              income
##  no : 296   Min.   : 0.0000   Min.   : 0.1667   Min.   : 0.210
##  yes:1023   1st Qu.: 0.0000   1st Qu.:25.4167   1st Qu.: 2.244
##             Median : 0.0000   Median :31.2500   Median : 2.900
##             Mean   : 0.4564   Mean   :33.2131   Mean   : 3.365
##             3rd Qu.: 0.0000   3rd Qu.:39.4167   3rd Qu.: 4.000
##             Max.   :14.0000   Max.   :83.5000   Max.   :13.500
##      share              expenditure       owner     selfemp      dependents
##  Min.   :0.0001091   Min.   :   0.000   no :738   no :1228   Min.   :0.0000
##  1st Qu.:0.0023159   1st Qu.:   4.583   yes:581   yes:  91   1st Qu.:0.0000
##  Median :0.0388272   Median : 101.298                        Median :1.0000
##  Mean   :0.0687322   Mean   : 185.057                        Mean   :0.9939
##  3rd Qu.:0.0936168   3rd Qu.: 249.036                        3rd Qu.:2.0000
##  Max.   :0.9063205   Max.   :3099.505                        Max.   :6.0000
##      months         majorcards         active
##  Min.   :  0.00   Min.   :0.0000   Min.   : 0.000
##  1st Qu.: 12.00   1st Qu.:1.0000   1st Qu.: 2.000
##  Median : 30.00   Median :1.0000   Median : 6.000
##  Mean   : 55.27   Mean   :0.8173   Mean   : 6.997
##  3rd Qu.: 72.00   3rd Qu.:1.0000   3rd Qu.:11.000
##  Max.   :540.00   Max.   :1.0000   Max.   :46.000
```

This table of summary statistics provides a brief overview of the data we are presented with. We are provided with the min., 1st quartile, median, mean, 3rd quartile, and max of each variable. There are three variables that hold yes and no values: the output - card, owner(does individual own their home), and selfemp(is the individual self employed). Ialso see that most people who own major cards only have 0 or 1. Another thing that stood out to me when briefly looking at this data overview was that there may be some outliers in the dataset. For the average monthly credit card expenditure, I notice that the mean is 185.057 however the maximum value in that data is 3099.505. This dataset will be interesting to analyze and I will see what conclusions I can draw from it through deeper analysis.

Explanation of variables

The Y value is whether or not an applicant was accepted and it holds two values: yes or no.

A data frame containing 1,319 observations on 12 variables.

card: Factor. Was the application for a credit card accepted?

reports: Number of major derogatory reports.

age: Age in years plus twelfths of a year.

income: Yearly income (in USD 10,000).

share: Ratio of monthly credit card expenditure to yearly income.

expenditure: Average monthly credit card expenditure.

owner: Factor. Does the individual own their home?

selfemp: Factor. Is the individual self-employed?

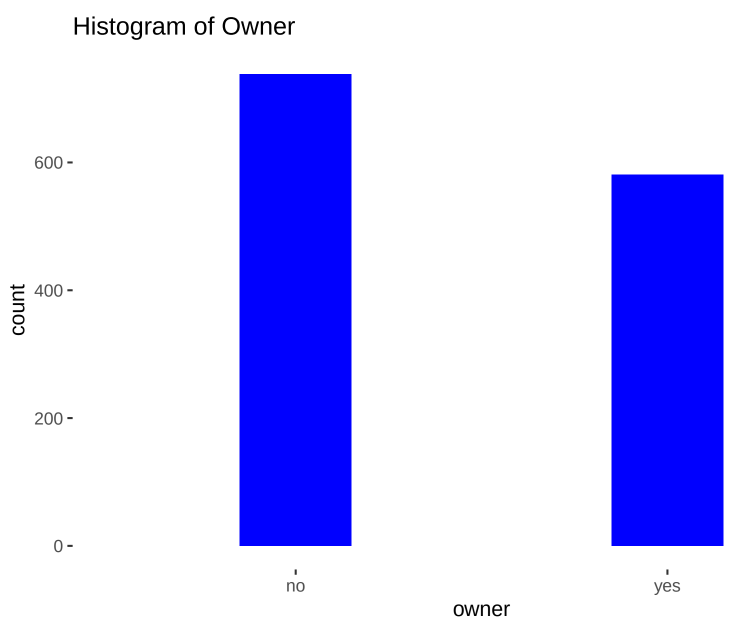
dependents: Number of dependents.

months: Months living at current address.

majorcards: Number of major credit cards held.

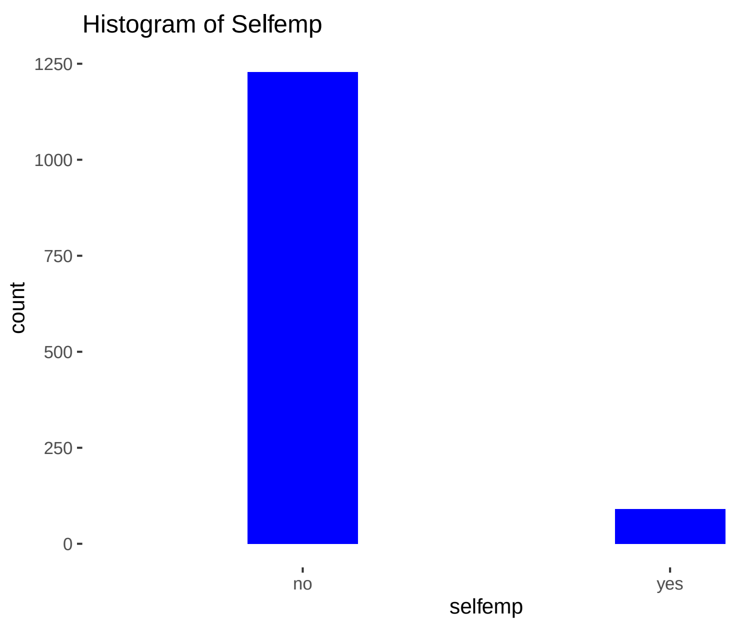
active: Number of active credit accounts.

```
owner_hist<-ggplot(CreditCard, aes(x=owner), space = 50) +
geom_bar(fill="blue", width = 0.3) +
labs(title = "Histogram of Owner") +
theme(panel.background = element_blank())
owner_hist
```



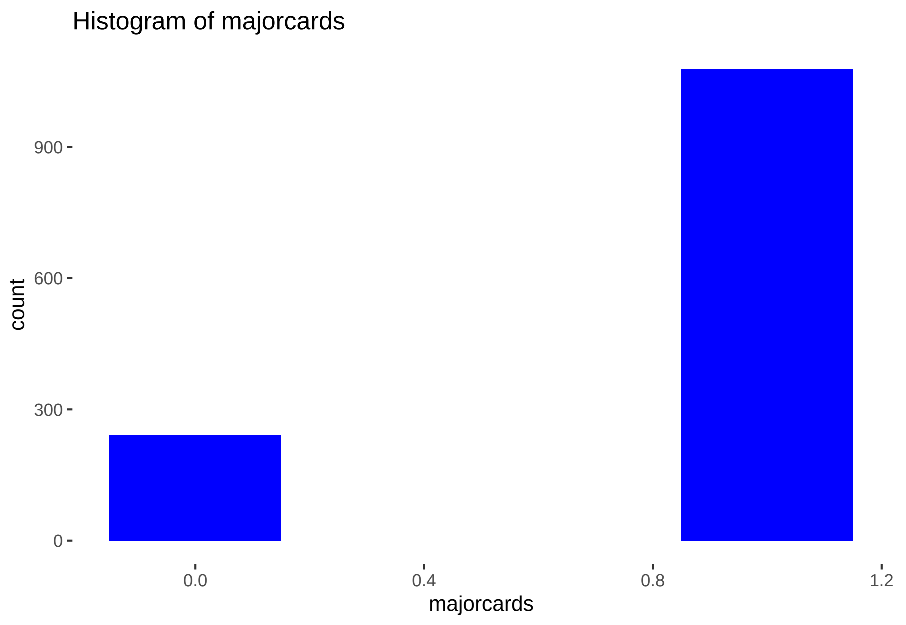
The histogram of the variable owner shows that most applicants that applied for a credit card did not own their own home. So most are likely renting.

```
selfemp_hist<-ggplot(CreditCard, aes(x=selfemp)) +
geom_bar(fill="blue", width = 0.3) +
labs(title = "Histogram of Selfemp") +
theme(panel.background = element_blank())
selfemp_hist
```



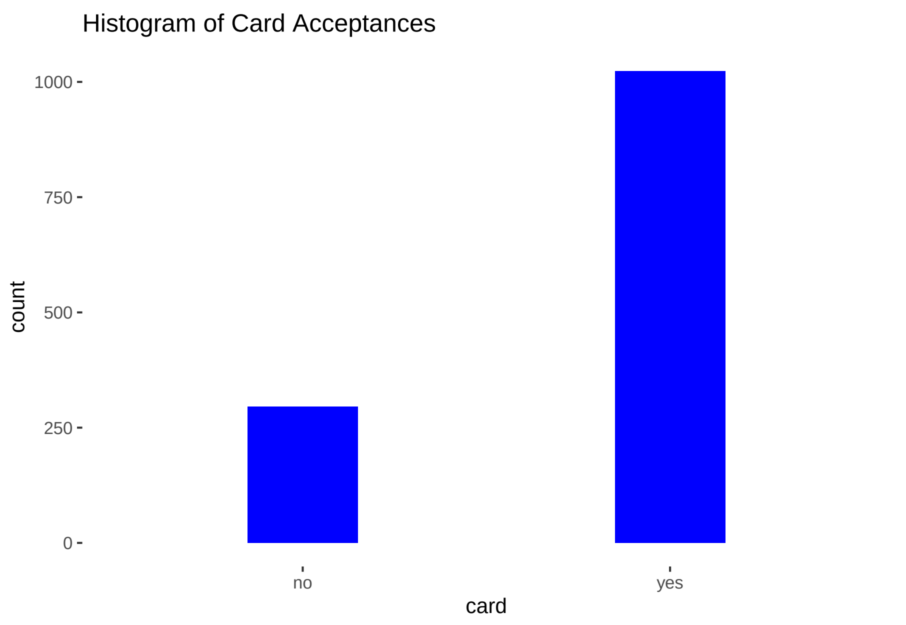
Most applicants are not self employed which means that most will have jobs at companies.

```
majorcards_hist<-ggplot(CreditCard, aes(x=majorcards)) +
geom_bar(fill="blue", width = 0.3) +
labs(title = "Histogram of majorcards") +
theme(panel.background = element_blank())
majorcards_hist
```



The only two values for the number of major cards appliants had were one or zero. This variable was not categorical as well.

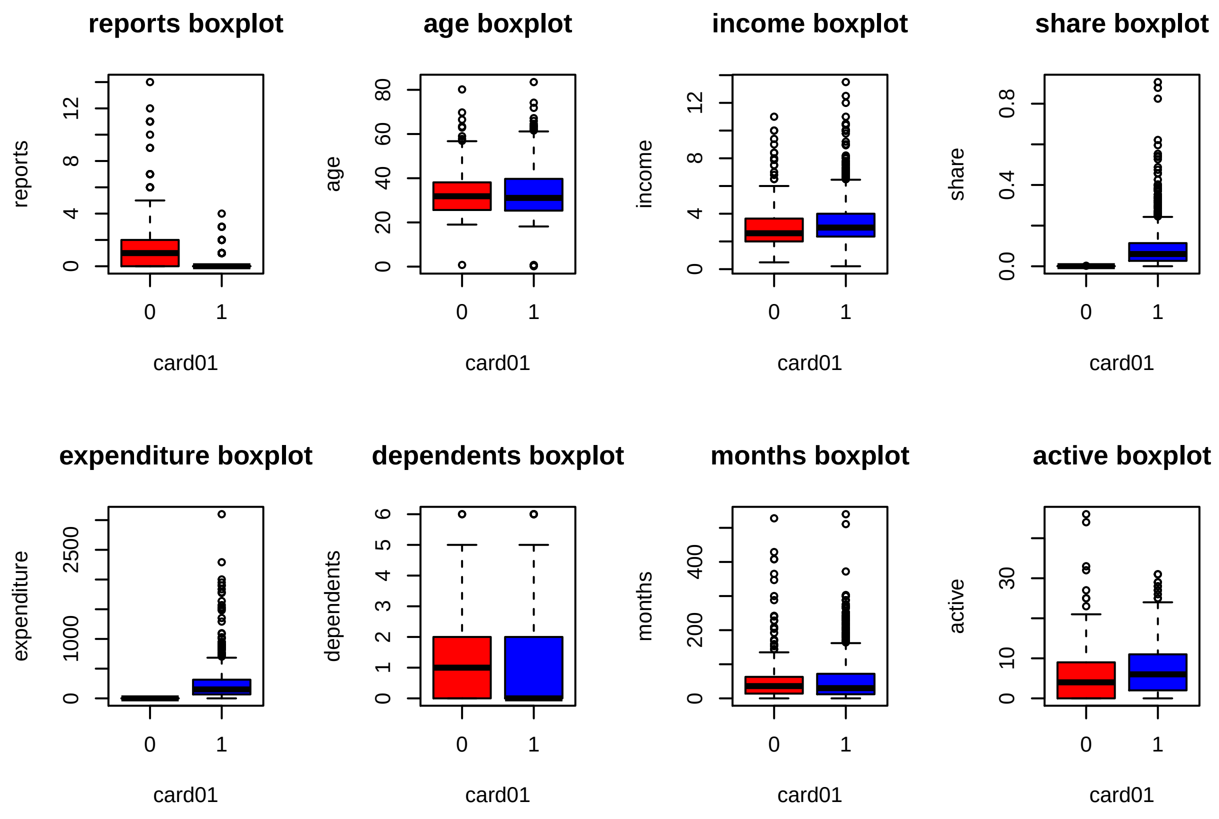
```
cards_hist<-ggplot(CreditCard, aes(x=card)) +
geom_bar(fill="blue", width = 0.3) +
labs(title = "Histogram of Card Acceptances") +
theme(panel.background = element_blank())
cards_hist
```



We also look at the distribution for the response variable, card, and see that most applicants get approved.

```
data(CreditCard)
CreditCard$card01 <- ifelse(CreditCard$card=="yes", 1, 0)
attach(CreditCard)
```

```r
# Boxplots
par(mfrow=c(2,4))
for(i in names(CreditCard)){
  # excluding the card variable and others categorical variables
  if( grepl(i, pattern="^card|owner|selfemp|name|majorcards")){ next}
  boxplot(eval(parse(text=i)) ~ card01, ylab=i, main =paste(i, "boxplot"),
          col=c("red", "blue"))
}
```



The box plots above show the distribution of values for each variable versus the response variable "yes" or "no", which was converted to 0 and 1. We are given the 1st, 3rd, and median. Additionally, the circles represent the outliers.

Summary of Boxplot Observations:

- People who have more derogatory reports are more likely to get rejected.
- The age of applicants between those who get approved and not approved are similar.
- Income between accepted and not accepted are similar.
- For the share boxplot, it looks very close to 0 but with closer look, there are no values of 0 - only values that are very close to 0. This seems to indicate that the credit card company mostly approves people who will use the credit card for more purchases.
- In expenditures, there are values of 0 so it shows that most people who get rejected also do not use a credit card often, supporting our point above.
- Applicants who get denied also have a median of one dependent while the median number among those who get approved is 0 dependents. Most young adults would typically not have any dependents.

- Months living at current address are similar for both.
- Most people

Second Report

2a. Logistic Regression
I have 11 different variables for my y so I will choose 5 variables that I believe have the largest effect - reports, share, selfemp, majorcards,active.

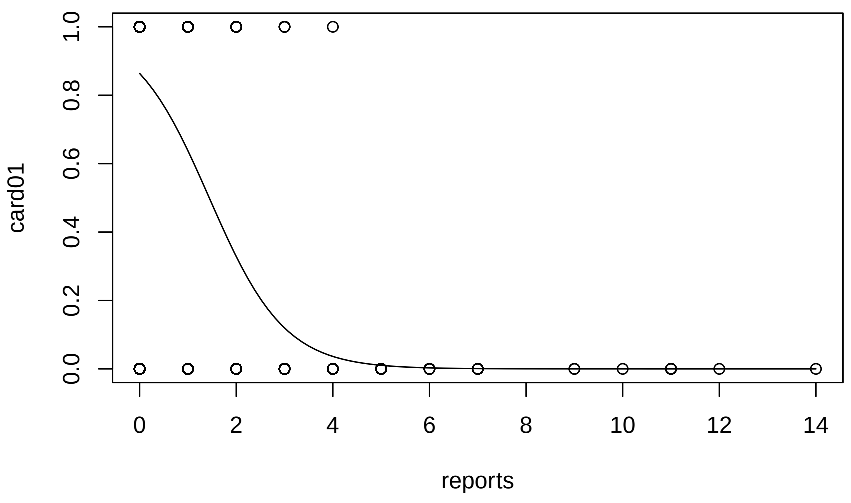
```
glm.fits1 <- glm(card~reports+share+selfemp+majorcards+active,
                 data = CreditCard, family = binomial)
glm.probs1 = predict(glm.fits1,CreditCard,type="response")
glm.pred1 = rep(0,length(glm.probs1))
glm.pred1[glm.probs1>.99]=1
table1 = table(glm.pred1,CreditCard$card)
logperf = (sum(diag(table1)))/sum(table1)
table1
```

```
##
## glm.pred1  no yes
##         0 296  46
##         1   0 977
```

```
logperf
```

```
## [1] 0.9651251
```

```
plot(reports,card01,xlab="reports",ylab="card01")
g=glm(card~reports,family=binomial,data = CreditCard)
curve(predict(g,data.frame(reports=x),type="resp"),add=TRUE)
```



The logistic regression graph above shows how the number of derogatory reports looks like when plotted against card.

The logistic regression used with the variables reports, share, +selfemp, majorcards, active had a 97.4981% classification rate. The five variables that I think most correctly predicted my model with logistic regressions are reports, share, selfemp, majorcards, active, data. I think those most logically predict my data because of the fact that negative affects of the X would negatively affect the Y. For example, having a high number of derogatory reports would cause someone to not be accepted for a credit card. While observing the correct predictions, true negatives and true positives, I see that the error rate is .974981 which is extremely high.

I will see the performance of logistic regression using all variables.

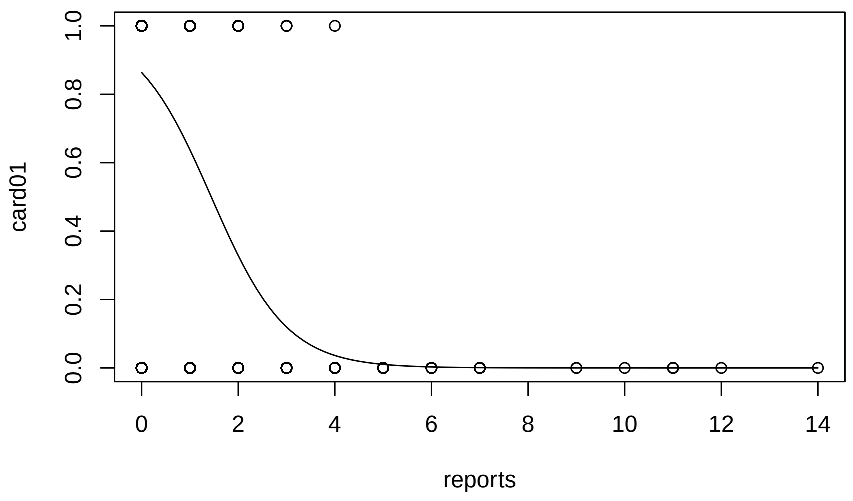
```
data("CreditCard")
library(class)
library(dplyr)
data("CreditCard")
CreditCard$card <- ifelse(CreditCard$card=="yes", 1, 0)
CreditCard$owner <- ifelse(CreditCard$owner=="yes", 1, 0)
CreditCard$selfemp <- ifelse(CreditCard$selfemp=="yes", 1, 0)
set.seed(123)
train = CreditCard %>% sample_frac(.7)
test = CreditCard %>% setdiff(train)

glm.fits1 <- glm(card~.,
                 data = train, family = "binomial")
logitmodel = predict(glm.fits1,test,type="response")
logpred = rep(0,length(logitmodel))
logpred[logitmodel>.1]=1
logit_matrix = table(logpred,test$card)
logperf = (sum(diag(logit_matrix)))/sum(logit_matrix)
logit_matrix
```

```
##
## logpred   0   1
##       0  67   3
##       1  17 309
```

```
logperf
```

```
## [1] 0.9494949
```

```
plot(reports,card01,xlab="reports",ylab="card01")
g=glm(card~reports,family=binomial,data = CreditCard)
curve(predict(g,data.frame(reports=x),type="resp"),add=TRUE)
```



The logit matrix above represents how the classification changes as the number of reports increases. This is a logical graph because as the number of derogatory reports a person has increases, they should start getting denied for a credit card application.

```
logit_matrix = table(logpred,test$card)
logperf = (sum(diag(logit_matrix)))/sum(logit_matrix)
logit_matrix
```

```
##
## logpred   0   1
##       0  67   3
##       1  17 309
```

```
logperf
```

```
## [1] 0.9494949
```

When running logistic regression on all the available variables, I see that I get an accuracy of 94.94%.
2b. K Nearest Neighbors

```
library(class)
library(dplyr)
data("CreditCard")
CreditCard$card <- ifelse(CreditCard$card=="yes", 1, 0)
CreditCard$owner <- ifelse(CreditCard$owner=="yes", 1, 0)
CreditCard$selfemp <- ifelse(CreditCard$selfemp=="yes", 1, 0)
set.seed(123)
train = CreditCard %>% sample_frac(.7)
test = CreditCard %>% setdiff(train)

X_card_trn = train[, -1]
Y_card_trn = train$card

# testing data
X_card_tst = test[, -1]
```

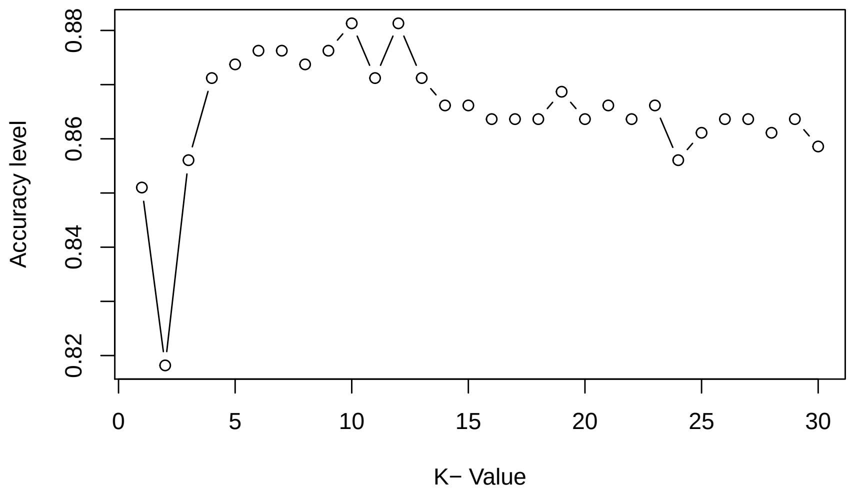
```r
Y_card_tst = test$card
set.seed(123)
card_pred = knn(train = scale(X_card_trn),
                test  = scale(X_card_tst),
                cl    = Y_card_trn,
                k     = 12,
                prob  = TRUE)

set.seed(123)
i=1
k.optm=1
bestk = 0;
besti = 0;
for (i in 1:30){
  set.seed(123)
  knn.mod <- knn(train=scale(X_card_trn), test=scale(X_card_tst), cl=Y_card_trn,
                k=i, prob = TRUE)
  k.optm[i] <- sum(Y_card_tst == knn.mod)/NROW(Y_card_tst)
  k=i
   if(k.optm[i]>bestk) {
     bestk = k.optm[i]
     besti = k
   }
}
bestk
```

```
## [1] 0.8813131
```

```r
besti
```

```
## [1] 10
```

```r
plot(k.optm, type="b", xlab="K- Value",ylab="Accuracy level")
```



```
knn_matrix = table(card_pred, Y_card_tst)
knnperf = (sum(diag(knn_matrix)))/sum(knn_matrix)
knn_matrix
```

```
##           Y_card_tst
## card_pred   0   1
##          0  46   9
##          1  38 303
```

```
knnperf
```

```
## [1] 0.8813131
```

This K nearest neighbors output correctly classifies 88.1% of my testing data. Also we see that the optimal value of K is 10.

```
mlperformance<-matrix(c(logperf, knnperf),ncol=1,byrow=TRUE)
rownames(mlperformance)<-c("Logistic Regression", "K Nearest Neighbors")
colnames(mlperformance)<-c("Performance")
mlperformance <- as.table(mlperformance)
mlperformance
```

```
##                       Performance
## Logistic Regression   0.9494949
## K Nearest Neighbors   0.8813131
```

Third Report

Libraries and code used to set up the third report

```
library(ISLR)
library(AER)
library(ggplot2)
library(dplyr)
library(glmnet)
data("CreditCard")
CreditCard = data.frame(CreditCard)
CreditCard$card <- ifelse(CreditCard$card=="yes", 1, 0)
CreditCard$owner <- ifelse(CreditCard$owner=="yes", 1, 0)
CreditCard$selfemp <- ifelse(CreditCard$selfemp=="yes", 1, 0)
```

3a. Splitting dataset

```
library(caret)
library(dplyr)
library(glmnet)
data("CreditCard")
CreditCard = data.frame(CreditCard)
CreditCard$card <- ifelse(CreditCard$card=="yes", 1, 0)
CreditCard$owner <- ifelse(CreditCard$owner=="yes", 1, 0)
CreditCard$selfemp <- ifelse(CreditCard$selfemp=="yes", 1, 0)

set.seed(123)
train = CreditCard %>% sample_frac(.7)
```

```r
test = CreditCard %>% setdiff(train)
x_train = model.matrix(card~., train)[,-1]
x_test = model.matrix(card~., test)[,-1]
y_train = train$card
y_test = test$card

x = model.matrix(card~., CreditCard)[,-1]
y = CreditCard$card
```

With this part, I divided 70 percent of my data into a training dataset and 30 percent to testing subset.

3b. Ridge Regression

```r
grid = 10^seq(10, -2, length = 100)
ridge_mod = glmnet(x_train, y_train, alpha=0, lambda = grid, family = "binomial")
ridge_pred = predict(ridge_mod, s = 4, newx = x_test)
y_pred = ifelse(ridge_pred>0.2,1,0)
ridgeperf_matrix = table(y_pred, y_test)
ridgeperf = (sum(diag(ridgeperf_matrix)))/sum(ridgeperf_matrix)
ridgeperf
```

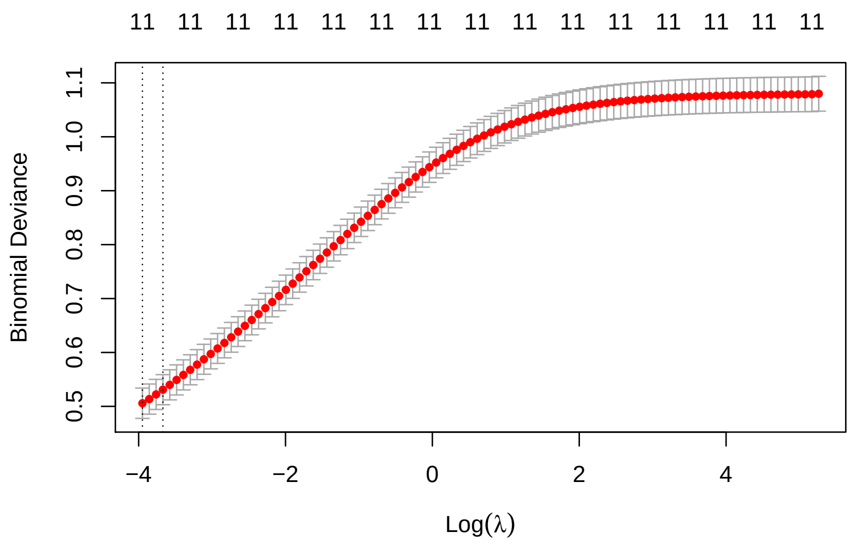
```
## [1] 0.2121212
```

Before tuning my Ridge regression and using s = 4, I get a classification rate of 21% which is very bad.

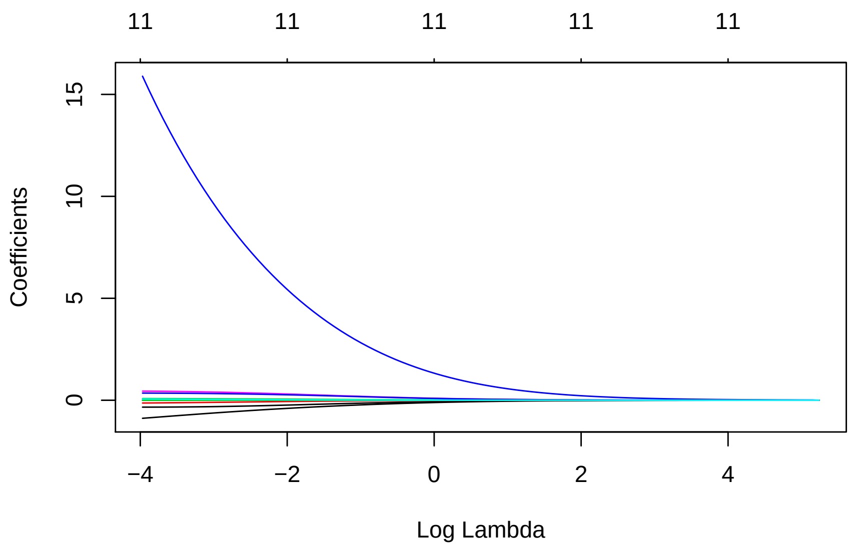
```r
set.seed(123)
ridge_cv.out = cv.glmnet(x_train, y_train, alpha = 0, family = "binomial")
bestlam = ridge_cv.out$lambda.min
bestlam
```

```
## [1] 0.0192841
```

```r
plot(ridge_cv.out)
```



```
ridge_pred = predict(ridge_mod, s = bestlam, newx = x_test)
y_pred = ifelse(ridge_pred>0.2,1,0)
out = glmnet(x, y, alpha = 0, family = "binomial")
ridge_coef = predict(out, type = "coefficients", s = bestlam)[1:12,]
plot(out, xvar = "lambda")
```



```
ridge_coef
```

```
##   (Intercept)       reports           age        income         share
## -0.2106042865 -0.8794117412 -0.0011188634  0.0816910611 15.7349005394
##   expenditure         owner       selfemp    dependents        months
##  0.0047603656  0.4453774135 -0.3375251444 -0.1351700870 -0.0003018424
##    majorcards        active
##  0.3512370822  0.0607617769
```

```
ridge_matrix = table(y_pred, y_test)
ridgeperf = (sum(diag(ridge_matrix)))/sum(ridge_matrix)
ridgeperf
```

```
## [1] 0.9292929
```

```
bestlam
```

```
## [1] 0.0192841
```

```
ridge_matrix
```

```
##       y_test
## y_pred   0   1
##      0  61   5
##      1  23 307
```

The λ within 1 standard error is 0.01384886 in my Ridge Regression. The graph shows the relationship between the cross-validation error and log of λ which is selected. The dash is the minimum lambda. This is a plot of the coefficients of the variables. We see how with Ridge Regression, we are not decreasing the number of variables which is why they are all 11 on top. While most of these coefficients are small, the largest coefficients are dependents and income. Furthermore, we see the improvement tuning does on our model when we originally had 21% of our testing data correctly classified as opposed to 92% after tuning.

3c. Lasso Regression

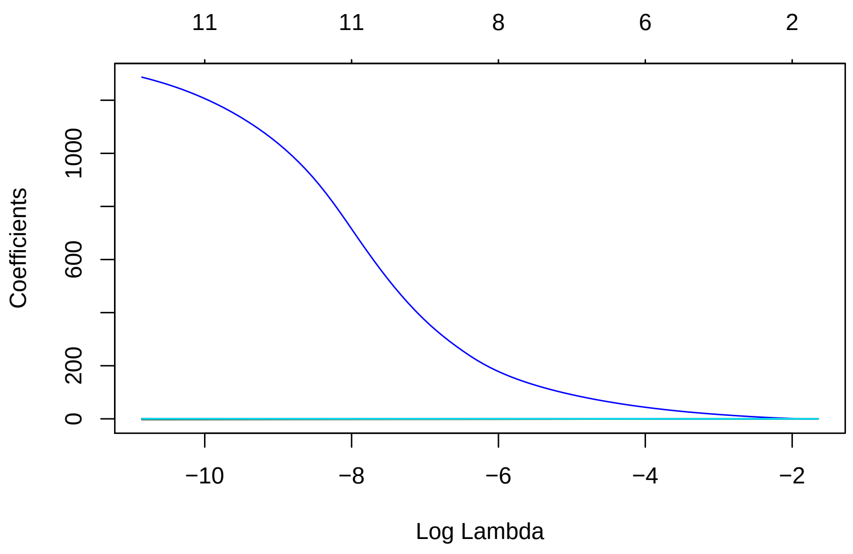
```
grid = 10^seq(10, -2, length = 100)
lasso_mod = glmnet(x_train, y_train, alpha=1, lambda = grid, family = "binomial")
lasso_pred = predict(lasso_mod, s = 4, newx = x_test)
y_pred = ifelse(lasso_pred>0.2,1,0)
lassoperf_matrix = table(y_pred, y_test)
lassoperf = (sum(diag(lassoperf_matrix)))/sum(lassoperf_matrix)
lassoperf
```

```
## [1] 0.2121212
```

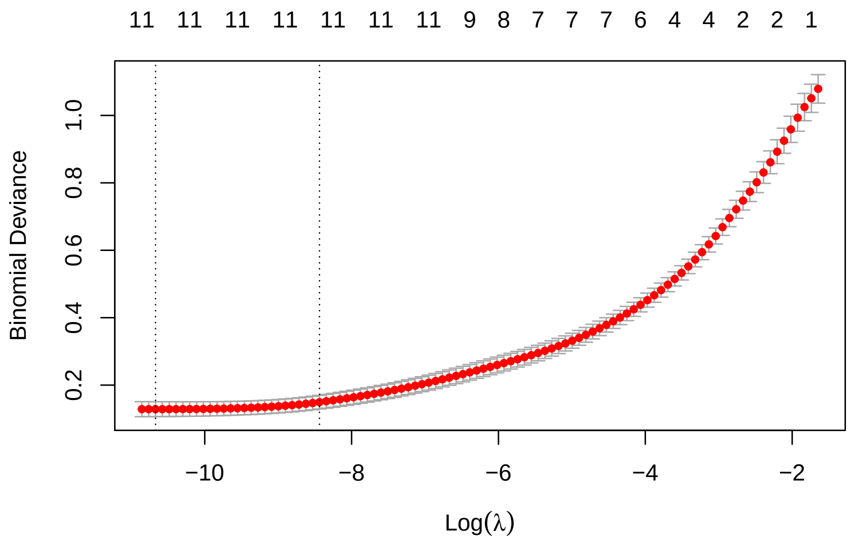
```
set.seed(123)
lasso_mod = glmnet(x_train,
                   y_train,
                   alpha = 1,
                   family = "binomial")
ls(lasso_mod)
```

```
##  [1] "a0"         "beta"       "call"       "classnames" "dev.ratio"
##  [6] "df"         "dim"        "jerr"       "lambda"     "nobs"
## [11] "npasses"    "nulldev"    "offset"
```

```
plot(lasso_mod,  xvar = "lambda")
```



```
set.seed(1)
lasso_cv.out = cv.glmnet(x_train, y_train, alpha = 1, family = "binomial")
plot(lasso_cv.out)
```



```
bestlam = lasso_cv.out$lambda.1se
lasso_pred = predict(lasso_mod, s = bestlam, newx = x_test)
y_pred = ifelse(lasso_pred>0.2,1,0)
lassoperf_matrix = table(y_pred, y_test)
lassoperf = (sum(diag(lassoperf_matrix)))/sum(lassoperf_matrix)
lassoperf
```

```
## [1] 0.979798
```

```
bestlam
```

```
## [1] 0.0002166229
```

```
out = glmnet(x, y, alpha = 1, lambda = grid, family = "binomial")
lasso_coef = predict(out, type = "coefficients", s = bestlam)[1:12,]
lasso_coef
```

```
## (Intercept)     reports         age      income       share expenditure
## -0.73851677 -1.05336272  0.00000000  0.07419872 70.23998753  0.00000000
##       owner     selfemp  dependents      months  majorcards      active
##  0.22440783  0.00000000 -0.01879572  0.00000000  0.17443795  0.06705179
```

```
lasso_coef[lasso_coef != 0]
```

```
## (Intercept)     reports      income       share       owner  dependents
## -0.73851677 -1.05336272  0.07419872 70.23998753  0.22440783 -0.01879572
##  majorcards      active
##  0.17443795  0.06705179
```

```
lasso_matrix = table(y_pred, y_test)
lassoperf = (sum(diag(lasso_matrix)))/sum(lasso_matrix)
lassoperf
```

```
## [1] 0.979798
```

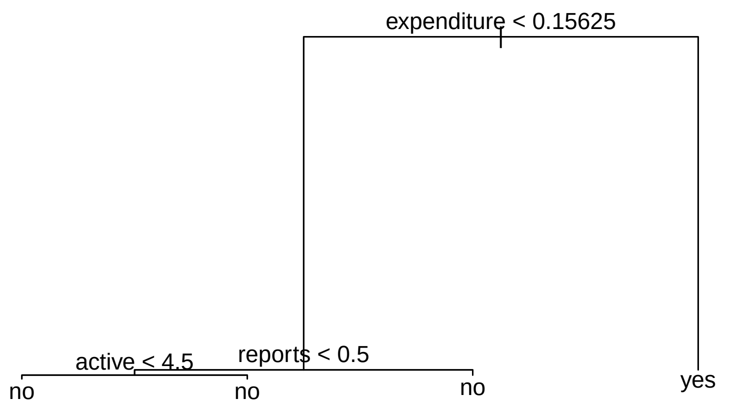
```
bestlam
```

```
## [1] 0.0002166229
```

Lasso Regression has a 97.9% classification rate. Furthermore, we see how many of the coefficients in the graph are at zero, which is what Lasso regression does.

3d. Decision Tree

```
data("CreditCard")
CreditCard = data.frame(CreditCard)
CreditCard$owner <- ifelse(CreditCard$owner=="yes", 1, 0)
CreditCard$selfemp <- ifelse(CreditCard$selfemp=="yes", 1, 0)
CreditCard$card = as.factor(CreditCard$card)

train = CreditCard %>% sample_frac(.7)
test = CreditCard %>% setdiff(train)

library(tree)
tree_card=tree(card~., data =train)
plot(tree_card)
text(tree_card, pretty = 0)
```

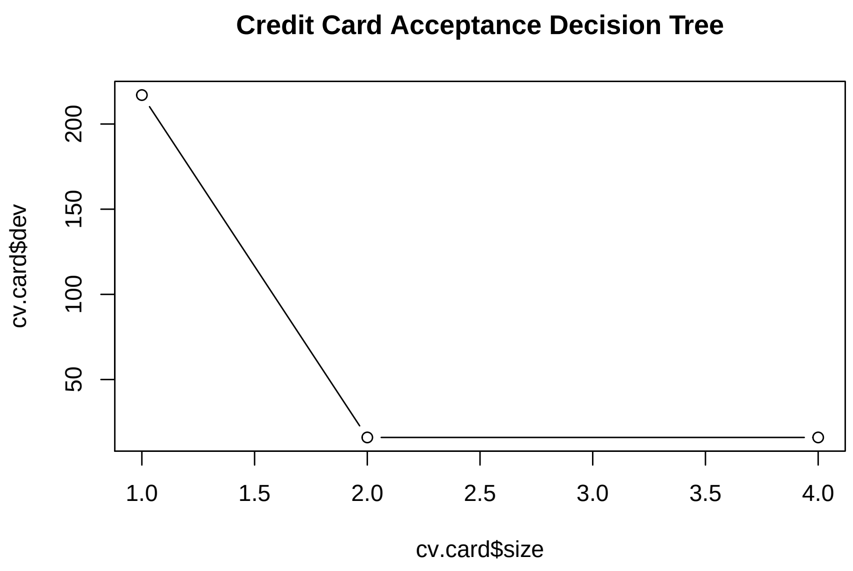


This tree has a depth of 3 and 4 leaves.

```
tree_pred = predict(tree_card, test, type = "class")
tree_table = table(tree_pred, test$card)
treeperf = (sum(diag(tree_table)))/sum(tree_table)
treeperf
```

```
## [1] 0.9848485
```

```
cv.card = cv.tree(tree_card, FUN = prune.misclass)
plot(cv.card$size, cv.card$dev, type = "b")
title("Credit Card Acceptance Decision Tree")
```



The decision tree for the data we have is very simple and it looks like it does not need to be pruned any more to achieve accurate results.

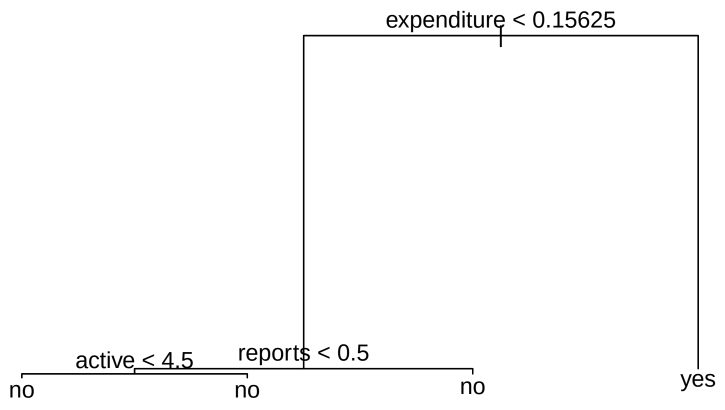
```
tree_table = table(tree_pred, test$card)
treeperf = (sum(diag(tree_table)))/sum(tree_table)
treeperf
```

```
## [1] 0.9848485
```

```
tree_table
```

```
##
## tree_pred  no yes
##       no   79   6
##       yes   0 311
```

```
prune_card = prune.misclass(tree_card, best = 5)
plot(prune_card)
text(prune_card, pretty = 0)
```



```
tree_pred = predict(prune_card, test, type = "class")
prune_table = table(tree_pred, test$card)
treeperf = (sum(diag(prune_table)))/sum(prune_table)
treeperf
```

```
## [1] 0.9848485
```

When trying to prune our decision tree, we get the same tree which shows that the tree will not need any more splits and already performs as well as it can.

```
mlperformance<-matrix(c(logperf, knnperf, ridgeperf, lassoperf, treeperf),ncol=1,
                      byrow=TRUE)
rownames(mlperformance)<-c("Logistic Regression", "K Nearest Neighbors", "Ridge",
                           "Lasso", "Decision Tree")
colnames(mlperformance)<-c("Performance")
mlperformance <- as.table(mlperformance)
mlperformance
```

```
##                       Performance
## Logistic Regression     0.9494949
## K Nearest Neighbors     0.8813131
## Ridge                   0.9292929
## Lasso                   0.9797980
## Decision Tree           0.9848485
```

Fourth Report

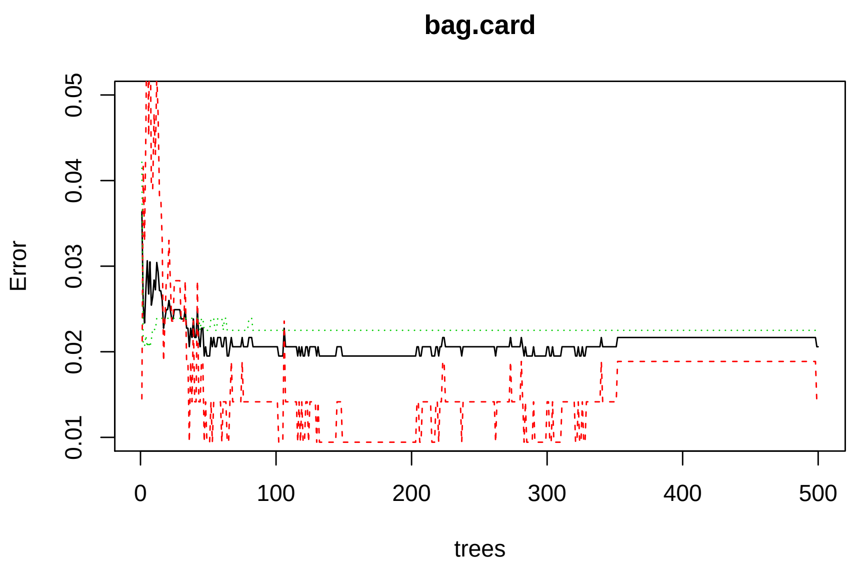
4a. Splitting dataset

```
data("CreditCard")
CreditCard = data.frame(CreditCard)
library(randomForest)
set.seed(123)
card_train = CreditCard %>%
  sample_frac(.70)
```

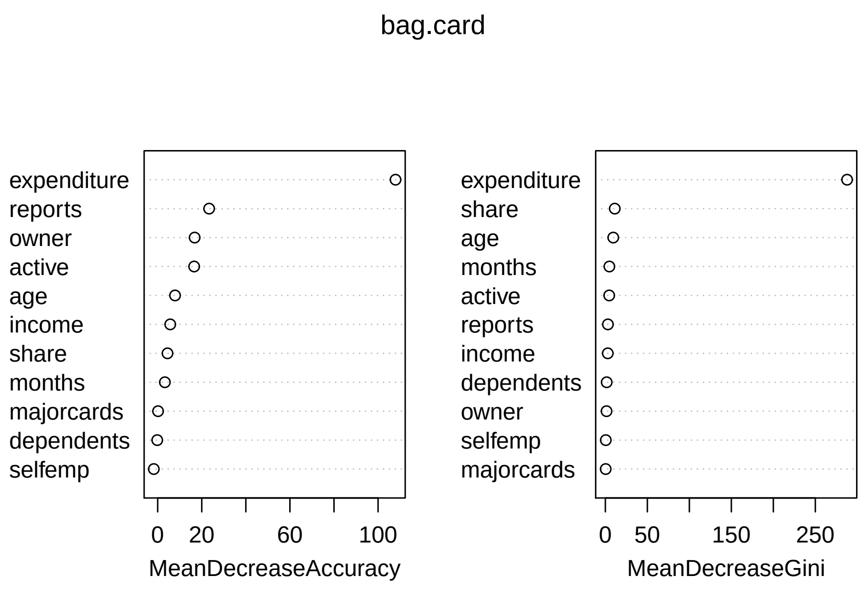
```r
card_test = CreditCard %>%
  setdiff(card_train)
```

4b. Bagging

```r
set.seed(123)
bag.card = randomForest(card ~., data = card_train,
                        mtry=ncol(card_train) - 1,
                        importance=TRUE)
plot(bag.card, ylim=c(0.01, .05))
```



```r
varImpPlot(bag.card)
```



This plot displays the out of bag error rate as a function of number of trees. It tells us the misclassification rate of the overall training data which is the line in black. The red line indicates the misclassification rate for the yes'. The green line tells us the misclassification rate for the no's. The x-axis is the trees and the y-axis is the error rate. Our argument mtry tells us that all 11 predictors are going to be considered for each split of the tree. Also, We see that the error is decreasing as we keep splitting the trees which is correct.

```
set.seed(123)
yhat.bag = predict(bag.card, newdata = card_test)
table(yhat.bag, card_test$card)
```

```
##
## yhat.bag  no yes
##      no   80   4
##      yes   4 308
```

```
CM = table(yhat.bag, card_test$card)
baggingperf = (sum(diag(CM)))/sum(CM)
baggingperf
```

```
## [1] 0.979798
```

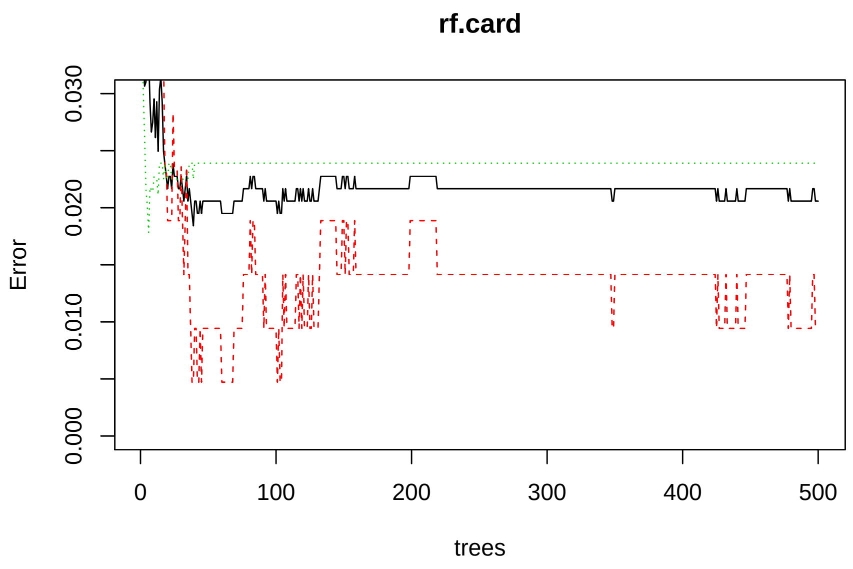
Bagging regression predicted our testing data to 97.98% accuracy.

4c. Random Forest

```
set.seed(123)
rf.card = randomForest(card~.,
                        data = card_train,
                        mtry = 5,
                        importance = TRUE,
                        do.trace = 100)
```

```
## ntree      OOB      1      2
##   100:   2.06%  0.94%  2.39%
##   200:   2.28%  1.89%  2.39%
##   300:   2.17%  1.42%  2.39%
##   400:   2.17%  1.42%  2.39%
##   500:   2.06%  0.94%  2.39%
```

```
plot(rf.card, ylim = c(0, 0.03))
```



```r
yhat.rf = predict(rf.card, newdata = card_test)
table(yhat.rf, card_test$card)
```

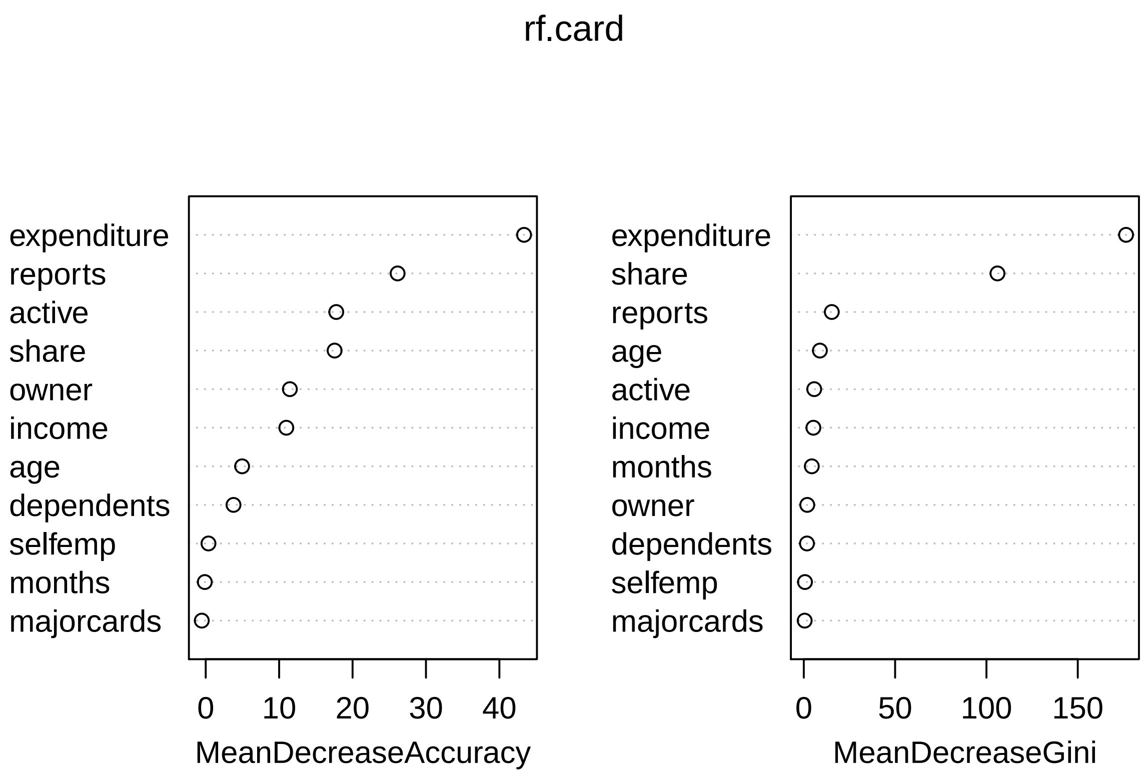
```
##
## yhat.rf  no yes
##      no   81   4
##      yes   3 308
```

```r
CM = table(yhat.rf, card_test$card)
rfperf = (sum(diag(CM)))/sum(CM)
rfperf
```

```
## [1] 0.9823232
```

This graph shows us the error of our random forest as we increase the number of trees. The green and red lines are errors of application accepted and application denied.

```r
varImpPlot(rf.card)
```



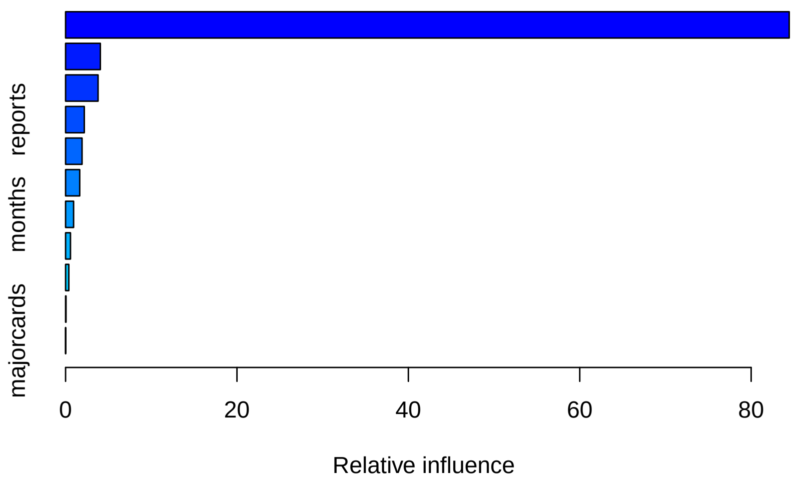
The hyperparameters for random forest are: mtry: which is the number of variables used at each split, ntree: which is the total number of trees, nodesize: which is the number of observations that we want in the terminal nodes (closely related to the depth of each tree). Also, looking at the plot, as we can see, this plot displays the out of random forest error rate as a function of number of trees. It tells us the misclassification rate of the overall training data which is the line in black. The red line indicates the misclassification rate for the yes'. The green line tells us the misclassification rate for the no's. The x-axis is the trees and the y-axis is the error rate. Our argument mtry tells us that all 11 predictors are going to be considered for each split of the tree. Also, We see that the error is decreasing as we keep splitting the trees which is correct.

4d. Boosting

```
library(gbm)
library(randomForest)
data("CreditCard")
ls(CreditCard)
```

```
##  [1] "active"      "age"         "card"        "dependents"  "expenditure"
##  [6] "income"      "majorcards"  "months"      "owner"       "reports"
## [11] "selfemp"     "share"
```

```
CreditCard$card <- ifelse(CreditCard$card=="yes", 1, 0)

set.seed(123)
card_train = CreditCard %>%
  sample_frac(.70)

card_test = CreditCard %>%
  setdiff(card_train)

set.seed(123)
boost.card = gbm(card~.,
                   data = card_train,
                   distribution = "bernoulli",
                   n.trees = 500,
                   interaction.depth = 4)

summary(boost.card)
```



```
##                     var       rel.inf
## expenditure expenditure 84.442363473
## share             share  4.057486424
## age                 age  3.793389031
## reports         reports  2.185358785
## active           active  1.923705119
## income           income  1.649576005
## months           months  0.937511760
## dependents   dependents  0.572464156
## owner             owner  0.380654947
## selfemp         selfemp  0.050880616
## majorcards   majorcards  0.006609683
```

```r
yhat.boost = predict(boost.card,
                        newdata = card_test,
                        n.trees = 5000)

boost_pred = predict(boost.card, card_test, n.trees=500, type = "response")
y_pred = ifelse(boost_pred>0.2,1,0)
boost_matrix = table(card_test$card, y_pred)
boostingperf = (sum(diag(boost_matrix)))/sum(boost_matrix)
boostingperf
```

```
## [1] 0.9722222
```

```r
boost.card
```

```
## gbm(formula = card ~ ., distribution = "bernoulli", data = card_train,
##     n.trees = 500, interaction.depth = 4)
## A gradient boosted model with bernoulli loss function.
## 500 iterations were performed.
## There were 11 predictors of which 11 had non-zero influence.
```

The relative influence of Boosting shows us that the most important variables in determine the status of an application is mainly expenditure, but then share and age.

```r
boost_matrix
```

```
##    y_pred
##       0   1
##   0  76   8
##   1   3 309
```

```r
boostingperf
```

```
## [1] 0.9722222
```

Boosting has a classification rate of 97.2%.
4e. XGBoost

```r
library(xgboost)
data("CreditCard")
CreditCard = data.frame(CreditCard)

CreditCard$card <- ifelse(CreditCard$card=="yes", 1, 0)
CreditCard$owner <- ifelse(CreditCard$owner=="yes", 1, 0)
CreditCard$selfemp <- ifelse(CreditCard$selfemp=="yes", 1, 0)

card_train = CreditCard %>%
  sample_frac(.70)

card_test = CreditCard %>%
  setdiff(card_train)
```

```
Y_train <- as.matrix(card_train[,"card"])
X_train <- as.matrix(card_train[!names(card_train) %in% c("card")])
dtrain <- xgb.DMatrix(data = X_train, label = Y_train)

X_test <- as.matrix(card_test[!names(card_train) %in% c("card")])
set.seed(123)
set.seed(2)
card.xgb = xgboost(data=dtrain,
                     max_depth=2,
                     eta = 0.1,
                     nrounds=40,
                     lambda=0,
                     print_every_n = 10,
                     objective="binary:logistic")
```

```
## [1]  train-error:0.011918
## [11] train-error:0.011918
## [21] train-error:0.011918
## [31] train-error:0.011918
## [40] train-error:0.011918
```

```
set.seed(123)
yhat.xgb <- predict(card.xgb,X_test)
y_pred = ifelse(yhat.xgb>0.2,1,0)
xgboost_matrix = table(y_pred, card_test$card)
xgboostperf = (sum(diag(xgboost_matrix)))/sum(xgboost_matrix)
xgboost_matrix
```

```
##
## y_pred   0   1
##      0  86  10
##      1   1 299
```

```
xgboostperf
```

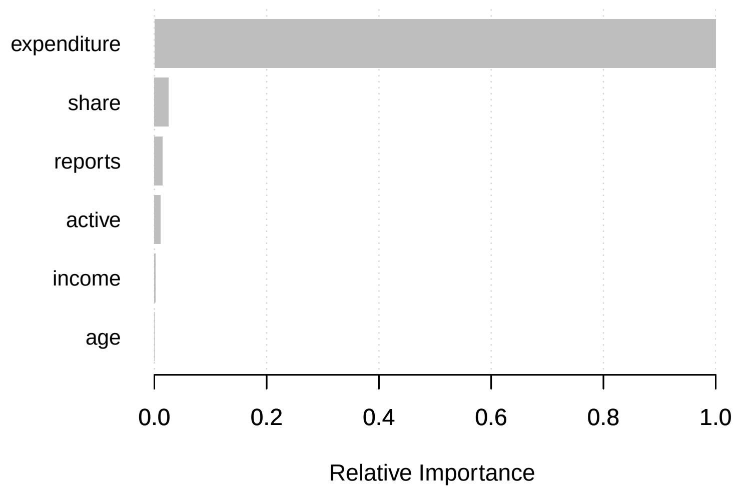
```
## [1] 0.9722222
```

XGBoost had the same performance as Boosting.

```
importance <- xgb.importance(colnames(X_train),model=card.xgb)
importance
```

```
##        Feature         Gain      Cover  Frequency
## 1: expenditure 9.486217e-01 0.56324246 0.30275229
## 2:       share 2.423926e-02 0.03679621 0.06422018
## 3:     reports 1.417048e-02 0.18102553 0.26605505
## 4:      active 1.077693e-02 0.04322104 0.13761468
## 5:      income 2.191567e-03 0.03445663 0.10091743
## 6:         age 3.022896e-08 0.14125814 0.12844037
```

```
xgb.plot.importance(importance, rel_to_first=TRUE, xlab="Relative Importance")
```



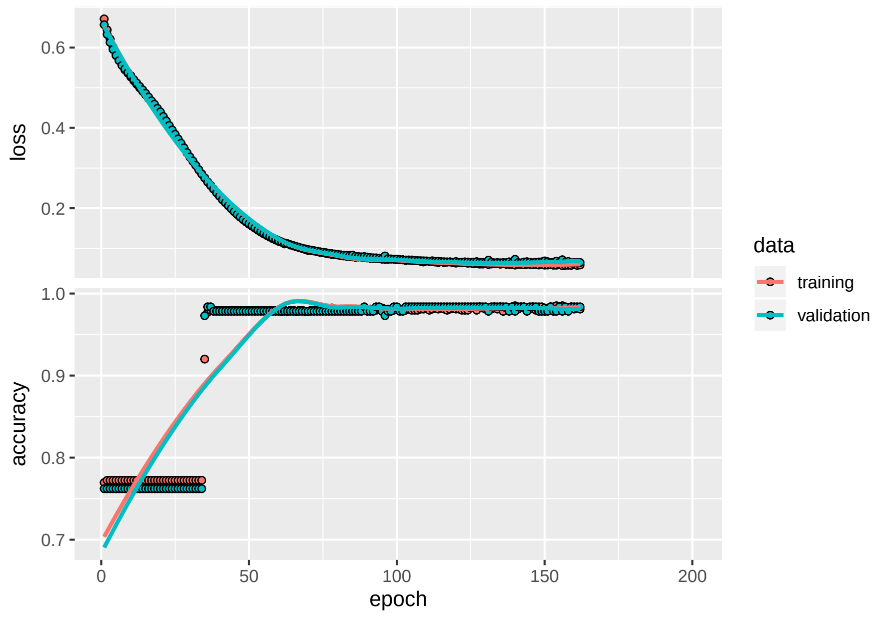
XGBoost also found expenditure, share, and reports as the most important variables.

4e. Neural Net

Setting up the data for the Neural Net

```
library(keras)
library(ISLR)
library(dplyr)
library(AER)
library(tensorflow)
data("CreditCard")
CreditCard = data.frame(CreditCard)
CreditCard$owner <- ifelse(CreditCard$owner=="yes", 1, 0)
CreditCard$selfemp <- ifelse(CreditCard$selfemp=="yes", 1, 0)
```

```
CreditCard$card_yes <- ifelse(CreditCard$card == "yes",1,0)
set.seed(123)
card_train = CreditCard %>%
  sample_frac(.7)

card_test = CreditCard %>%
  setdiff(card_train)

train_labels <- to_categorical(card_train[,"card_yes"],2)
train_data <- as.matrix(card_train[!names(card_train) %in% c("card","card_yes")])
test_data <- as.matrix(card_test[!names(card_train) %in% c("card","card_yes")])
test_labels <- to_categorical(card_test[,"card_yes"])

model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "softmax",
              input_shape = dim(train_data)[2]) %>%
  layer_dense(units = 64, activation = "softmax") %>%
  layer_dense(units = 2, activation= "softmax")

model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
early_stop <- callback_early_stopping(monitor = "val_loss", patience = 20)

epochs=200
history_class <- model %>% fit(
  train_data,
  train_labels,
  epochs = epochs,
  validation_split = 0.2,
  callbacks = list(early_stop)
)
plot(history_class)
```



```
test_predictions <- model %>% predict(test_data)
test_class <- model %>% predict_classes(test_data)
error_table = table(test_labels[,2], test_class)
neuralnetperf = (sum(diag(error_table)))/sum(error_table)
neuralnetperf
```

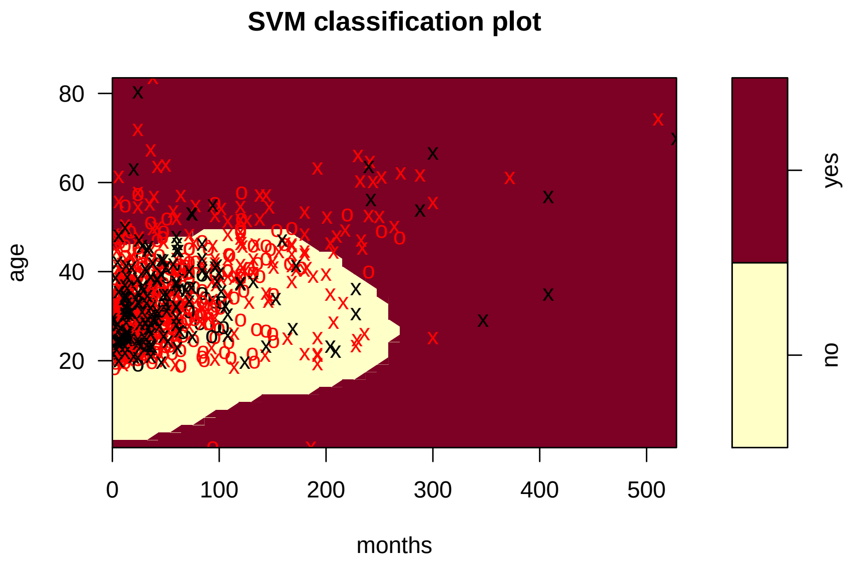
```
## [1] 0.9873737
```

The Neural Net had a classification rate of 98.7%. The graph also shows us the accuracy and loss of our training and validation sets as our epochs increase. Our epochs is the number of back propogations we want to do for the model and we set it to stop after it doesnt change much in a certain number of iterations. As the epoch starts to approach 75, it does not change too much.

4g. Support Vector Machines

```
data("CreditCard")
library(dplyr)
library(ggplot2)
library(e1071)
set.seed(123)
CreditCard$card = as.factor(CreditCard$card)
```

```
CreditCard$owner <- ifelse(CreditCard$owner=="yes", 1, 0)
CreditCard$selfemp <- ifelse(CreditCard$selfemp=="yes", 1, 0)
training_set = CreditCard %>%
  sample_frac(.7)

testing_set = CreditCard %>%
  setdiff(training_set)

dat = data.frame(x = X_train, y = as.factor(Y_train))

CreditCard$card = factor(CreditCard$card, levels = c(0, 1))

set.seed(123)
tune.out = tune(svm, card~., data = training_set, kernel = "radial",
                ranges = list(cost = c(0.1,1,10,100,1000), gamma = c(0.5,1,2,3,4)))
bestmod = tune.out$best.model
plot(bestmod, training_set, age~months)
```



```
svm_table = table(true = testing_set$card, pred = predict(tune.out$best.model,
                                                 newdata = testing_set))
svmperf = (sum(diag(svm_table)))/sum(svm_table)
svmperf
```

```
## [1] 0.8737374
```

The SVM only had an accuracy rate of 87.37 percent. Also with the graph above, it is observed that an SVM can still be applied to data that is not linearly separable. It makes the assumption that similar data points will be close together on a graph. We can see that to an extent re low values and low values of age would typically get rejected for a credit card.

{5. Model Comparisons}

```
mlperformance<-matrix(c(logperf, knnperf, ridgeperf, lassoperf, treeperf, baggingperf,
                        rfperf, boostingperf, xgboostperf, neuralnetperf, svmperf),
                        ncol=1,byrow=TRUE)
rownames(mlperformance)<-c("Logistic Regression", "K Nearest Neighbors", "Ridge",
                           "Lasso", "Decision Tree", "Bagging","Random Forest",
                           "Boosting", "XGBoost", "Neural Network",
                           "Support Vector Machine")
colnames(mlperformance)<-c("Performance")
mlperformance
```

```
##                          Performance
## Logistic Regression        0.9494949
## K Nearest Neighbors        0.8813131
## Ridge                      0.9292929
## Lasso                      0.9797980
## Decision Tree              0.9848485
## Bagging                    0.9797980
## Random Forest              0.9823232
## Boosting                   0.9722222
## XGBoost                    0.9722222
## Neural Network             0.9873737
## Support Vector Machine     0.8737374
```

Among the machine learning models, Neural Networks perform the best. Between the Shrinkage Models, Lasso is better than Ridge and this may be because the model is simpler and smaller. Between the Tree Models, the decision tree was the most accurate and Boosting/XGBoost was least accurate. After comparing all models, I see that neural networks performed the best on the testing data by correctly classifying 98.7% of it. The order of the models is as followed from greatest to least: neural network, decision tree, random forest, Bagging and Lasso, Logistic Regression, Ridge, Boosting and XGBoost, K nearest neighbors, and support vector machines.

Conclusion

From the rankings, it seems as if logistic regression, a simple model does perform well. In the first report, I also took a subset of variables and when I applied a logistic model to them, it predicted it better as opposed to using all the variables. This indicates that there are certainly variables that hold much higher weights than others. The weights I used for the first model were reports, share, selfemp, majorcards, and active. This had a classification rate of 96.5%, which was better than using all of the variables. This may have been because using all the variables may cause it to overfit on the testing data.

Furthermore, Lasso performed better than Ridge Regression by a difference of about 5%. This is significant because 5% of our dataset is a large amount of values. This shows that setting some of the coefficients of the variables that are not important to 0 will make an impact on our model. Lasso performs subset selection which not only sets coefficients to 0, but gives smaller and simpler models that are easy to interpret as opposed to Ridge. Subset selection is preferable in our dataset.

In our third report, I analyzed tree based models and again, the simpler model seemed to work the best and that was a decision tree. Bagging, Random forest, Boosting, and XGBoost also performed decently and similarly to each other as they were all around the 98% range. Because our dataset might be considering only a limited number of variables (only 11), more complex models may not be needed. If we had a larger dataset, models such as Bagging and XGBoost may perform better than Decision Trees.

Neural Networks, a powerful machine learning model, performed the best on my dataset. Neural nets used epochs and tested the model using backpropogation in order to determine the different weights of my network. However, while running the code, it is clear that the accuracy provided by neural networks also causes there to be a long run time when running the model. But, the neural network also classified my data correctly 98.73% of the time. The next highest was decision trees at 98.4%. This is interesting to observe because a complex model barely performed better than a single tree model.