Credit Card ML Analysis Report #4

Introduction

This is the third report for my Credit Card Acceptance Project. In this part of the project, I will be incorporating different regression methods such as Bagging, Random Forest, Boosting, and XGBoost Regression. Additionally, for extra credit, I will be tuning parrameters using grid search for my boosting model (4).

```
library(ISLR)
library(AER)
library(ggplot2)
library(dplyr)
library(randomForest)
rm(list=ls())
data("CreditCard")
CreditCard = data.frame(CreditCard)
names(CreditCard)
```

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

```
nrow(CreditCard)
```

```
## [1] 1319
```

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

1. Dividing data into training and testing subset

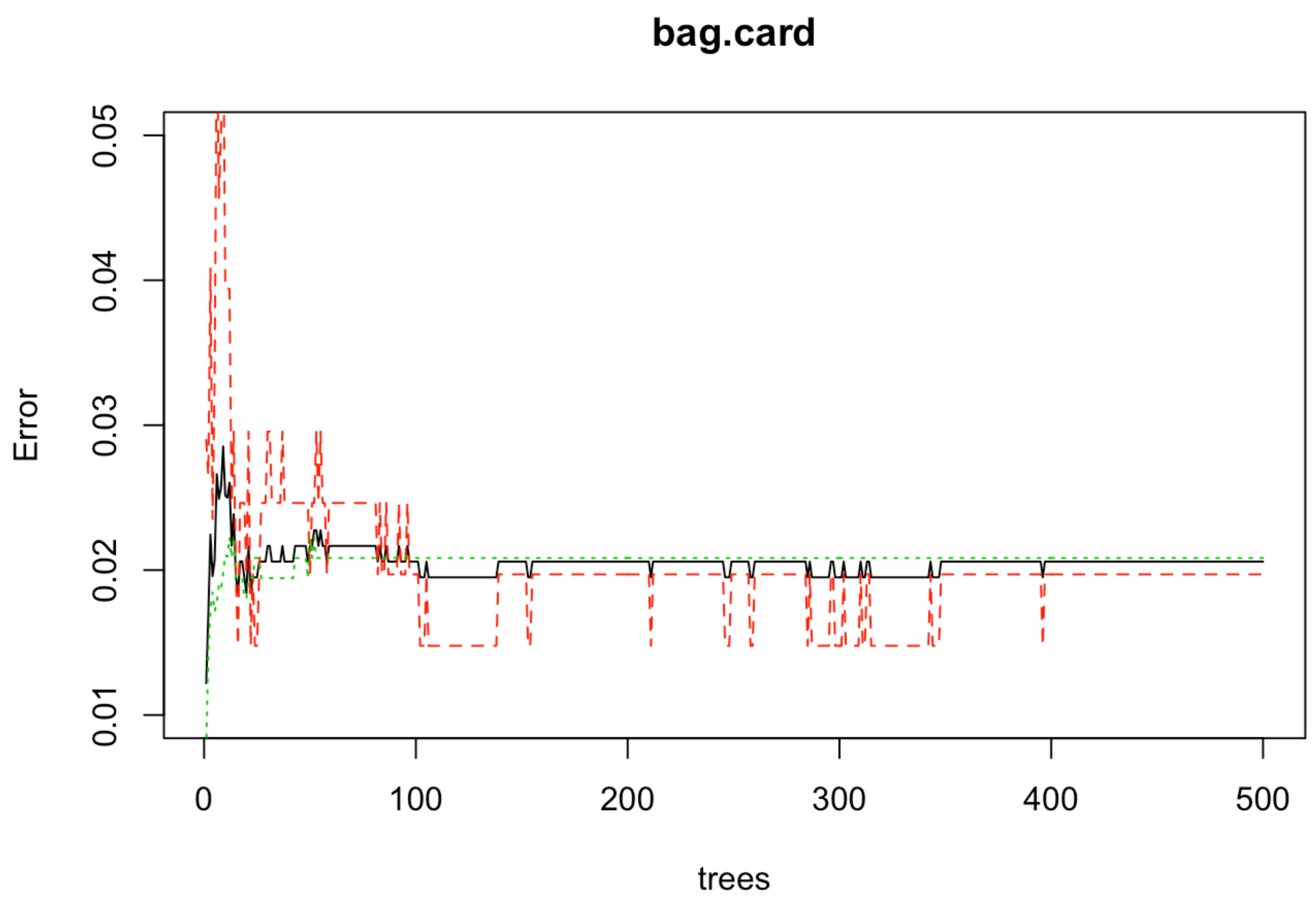
```
set.seed(1)
card_train = CreditCard %>%
  sample_frac(.70)

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

I am dividing my data into half so that I will have inputs to train my model and then use the other half to test my models later. I chose to train on 70% of my data and then test on 30% of my data. 70% is a reasonable number and having 30% to test should be sufficient enough to determine the accuracy of my models below.

2.Fitting a bagging regression

a. Plot the out-of-bag error rate as a function of number of trees

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



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.

b. Error Rate table for Classification

```
bag.card
```

```
##
## Call:
##  randomForest(formula = card ~ ., data = card_train, mtry = ncol(card_train) -
1, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 11
##
##         OOB estimate of  error rate: 2.06%
## Confusion matrix:
##      no yes class.error
## no  199   4  0.01970443
## yes  15 705  0.02083333
```

```
yhat.bag = predict(bag.card, newdata = card_test)
#table(yhat.bag, card_test$card)
#CM = table(yhat.bag, card_test$card)
#accuracy = (sum(diag(CM)))/sum(CM)
#accuracy
#CM
(199+705)/(199+705+15+4)
```

```
## [1] 0.979415
```

The error rate table tells us the accuracy of our model. The error rate for classification was ~97.942%. This means that our bagging model classified our data correctly 97% of the time.

2c.Comparing the error rate to the test error rate in the Ridge and LASSO models from Report 3.

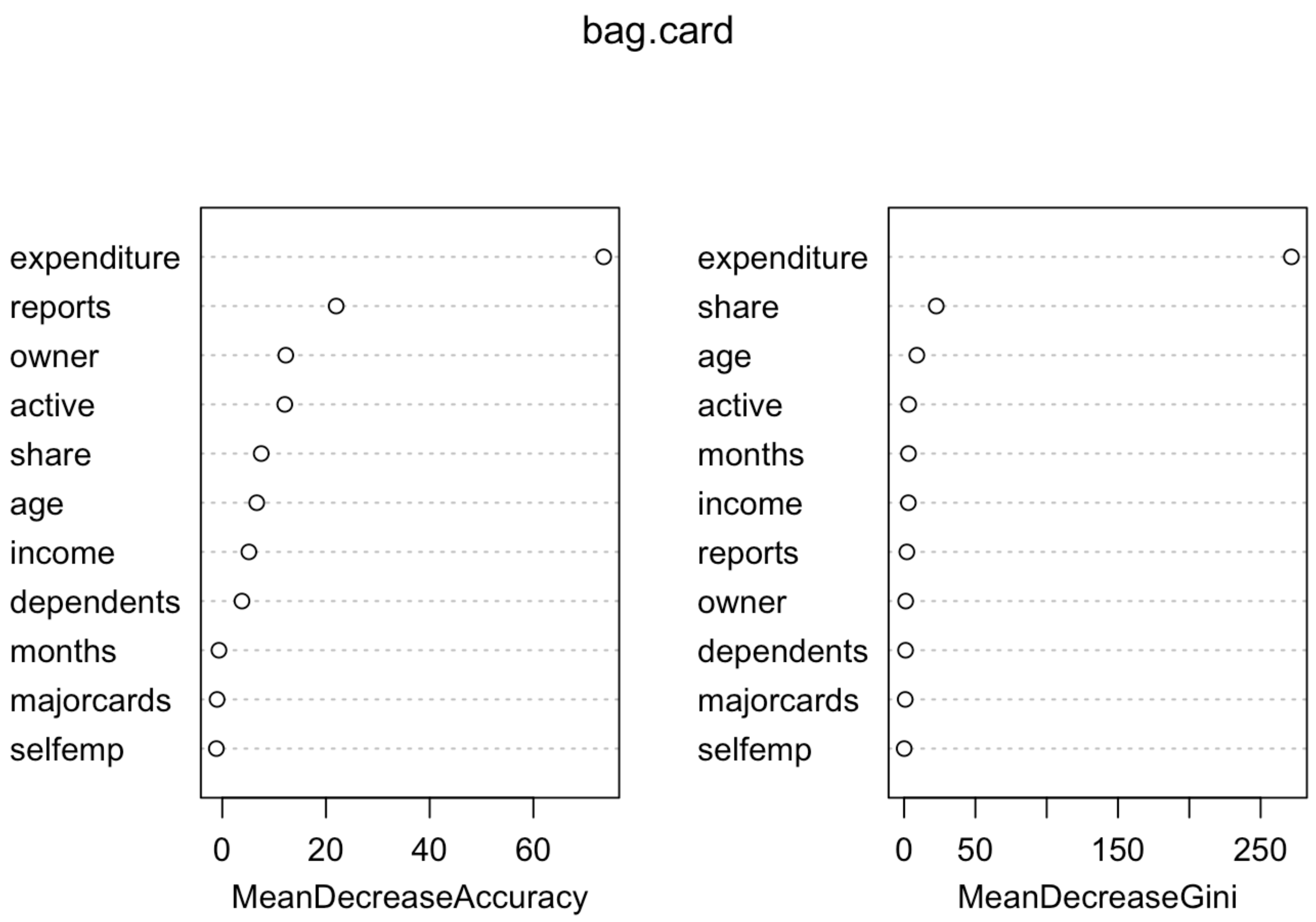
The error rate for bagging was 97.9415%, for LASSO was 97.3%, and for Ridge was 96.9%. Bagging may have been better at classifying my data because my some of my variables and my response variable was categorical. Bagging, which is similar to random forests, can automatically model non-linear data, which is closer to the data I have.

2d.Importance Matrix

```
importance(bag.card)
```

```
##                    no        yes MeanDecreaseAccuracy MeanDecreaseGini
## reports     22.0236641  6.7675908           21.9637194       1.93802065
## age          6.0995818  2.6280046            6.6453400       8.93093572
## income       5.7631407 -1.8918871            5.1341352       2.97805246
## share        8.2720378  5.7183983            7.5114532      22.56172625
## expenditure 79.5179643 60.4343225           73.5513648     271.58990072
## owner       12.8820021 -0.7036757           12.2486822       1.11033071
## selfemp     -0.9798094 -1.4169902           -1.1416214       0.06195347
## dependents   3.3209148  2.4694371            3.7923617       1.01939738
## months       1.0557747 -5.6309809           -0.6411475       3.07775314
## majorcards  -0.9223070 -0.2818893           -1.0024696       0.77509842
## active      14.5703481 -7.5903292           12.0563938       3.23151796
```

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



This importance matrix displays the importance of each attribute using the fitted classifier. I have the matrix and also the graph of the matrix by using the function varImpPlot. MeanDecreaseAccuracy gives us how much the accuracy decreases by ommitting the respective variable. The MeanDecreaseGini tells us the decrease of the Gini impurity when a variable is chosen to split a node. One thing we notice is that expenditure is a variable that does affect our bagging model because our accuracy decreases the greatest when we omit that variable.

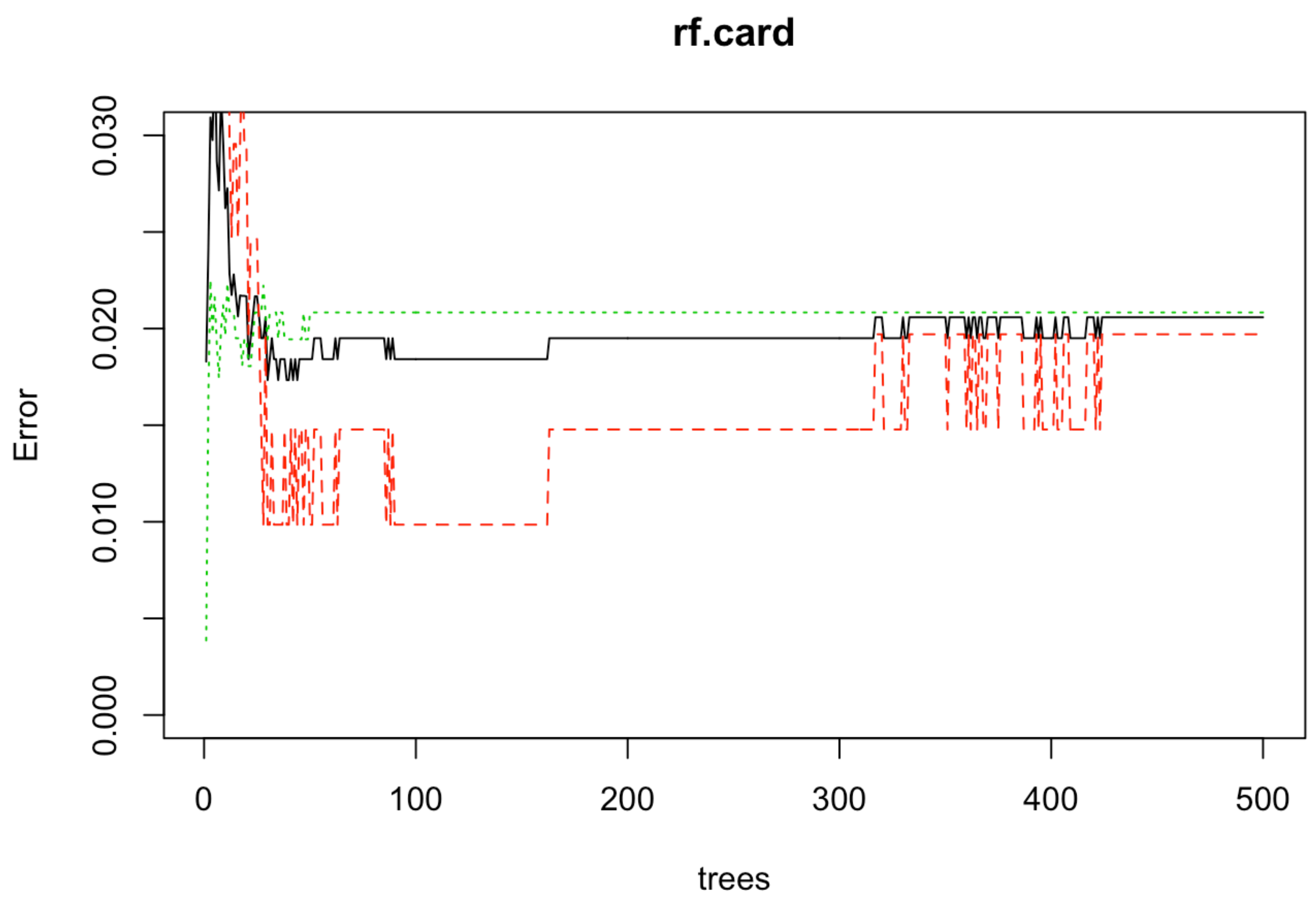
3.Fitting a Random Forest Regression

a. Ploting the out-of-bag error rate as a function of number of predictors considered in each split

```
set.seed(1)
rf.card = randomForest(card~.,
                         data = card_train,
                         mtry = 5,
                         importance = TRUE,
                         do.trace = 100) #do.trace gives you the OOB MSE for every 10
0 trees
```

```
## ntree      OOB      1      2
##   100:   1.84%  0.99%  2.08%
##   200:   1.95%  1.48%  2.08%
##   300:   1.95%  1.48%  2.08%
##   400:   1.95%  1.48%  2.08%
##   500:   2.06%  1.97%  2.08%
```

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



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.

b.Using out of bag error to tune the number of predictors in each split (mtry)

```
rf.card = randomForest(card~.,
                         data = card_train,
                         mtry = 5,
                         importance = TRUE,
                         do.trace = 100) #do.trace gives you the OOB MSE for every 10
0 trees
```

```
## ntree      OOB      1      2
##   100:   2.28%  2.46%  2.22%
##   200:   2.06%  1.97%  2.08%
##   300:   1.95%  1.48%  2.08%
##   400:   1.95%  1.48%  2.08%
##   500:   1.95%  1.48%  2.08%
```

We need to use the out of bag error to tune the number of predictors in each split. Typically, the value for mtry should be number of variables divided by 3. However, it is beneficial to look at the out of bag error to truly determine which mtry is better. I chose 5 for mtry. When using 6, all the out of bag error rates were 2.06% when ntrees were 100 to 500. When using 4 for mtry, the out of bag error rates were 2.06% for 100 and then were 1.95 percent for the rest of ntrees. mtry=5 gives us the lowest out of bag rate so it should be the one we use.

c. Error rate table for classification

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

```
##
## Call:
##  randomForest(formula = card ~ ., data = card_train, mtry = 5,      importance = T
RUE, do.trace = 100)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 1.95%
## Confusion matrix:
##      no yes class.error
## no  200   3  0.01477833
## yes  15 705  0.02083333
```

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

```
##
## card_pred  no yes
##       no   92   7
##       yes   1 296
```

```
#CM = table(yhat.rf, card_test$card)
#accuracy = (sum(diag(CM)))/sum(CM)
#accuracy
#CM
(296+92)/(296+92+7+1)
```

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

The accuracy rate for random forest is ~97.9798. We look at whether or not it tested it properly by looking at the no-no and yes-yes.

d.Random Forest Error rate compared to LASSO/Ridge/Bagging

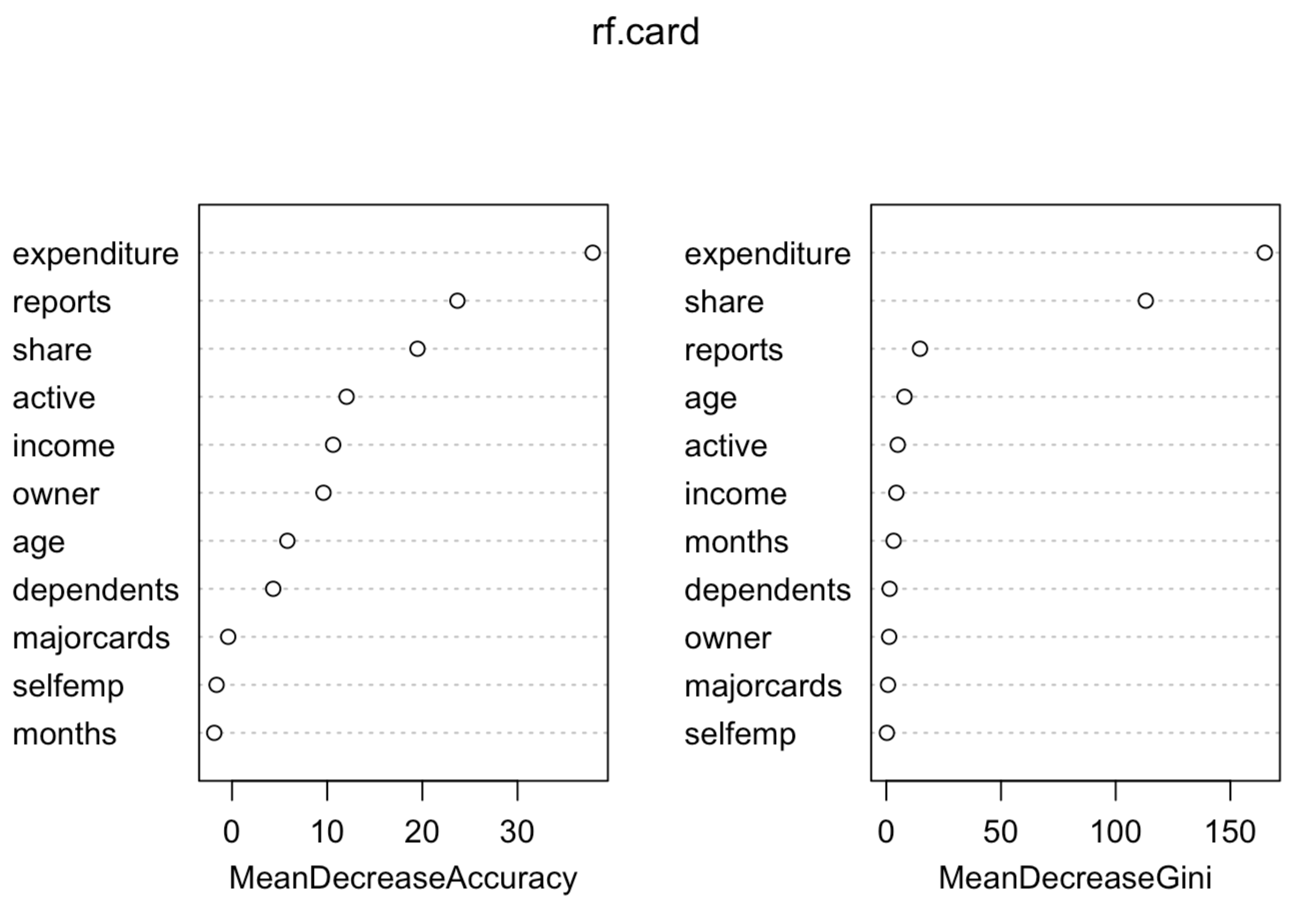
The Random Forest Error rate was 97.9798% and the error rate for bagging was 97.9415%, for LASSO was 97.3%, and for Ridge was 96.9%. Random Forest may have been better at classifying my data because my some of my variables and my response variable was categorical. Random Forest can automatically model non-linear data, which is closer to the data I have which may be why it performed better than LASSO and Ridge. Additionally, I would expect random forest and bagging to have similar error rates because they are similar to each other. The difference is that in random forest, only a subset of features are selected at random out of the total and the best split feature from the subset is used to split each node in a tree, unlike in bagging where all features are considered for splitting a node.

e. Importance Matrix

```
importance(rf.card)
```

```
##                     no         yes MeanDecreaseAccuracy MeanDecreaseGini
## reports     23.9952864  9.52004261           23.6819264       14.6859772
## age          4.6958231  4.30762101            5.8152224        7.9382769
## income      10.6425161 -1.69747681           10.6243706        4.3711568
## share       21.4706172 11.10979709           19.4872937      113.1319013
## expenditure 45.4494778 24.39526897           37.8987483      164.9972169
## owner        9.7709563 -0.08635893            9.6096033        1.2455610
## selfemp     -0.9756488 -1.83134246           -1.6236548        0.2464662
## dependents   3.7638116  2.59108849            4.3272358        1.4291735
## months      -0.9175269 -2.93885818           -1.8680799        3.1990622
## majorcards  -0.6731560  0.64507284           -0.4041174        0.7518260
## active      13.7759389 -3.50782320           12.0384882        4.9928955
```

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



In

the graphs above, we see the effects of the variables. It is similar to bagging for reasons stated above and MeanDecreaseAccuracy and MeanDecreaseGini also tell us the effect of the variables. MeanDecreaseAccuracy gives us how much the accuracy decreases by ommitting the respective variable. The MeanDecreaseGini tells us the decrease of the Gini impurity when a variable is chosen to split a node. One thing we notice is that expenditure is a variable that does affect our bagging model because our accuracy decreases the greatest when we omit that variable.

f.Plot the test error and out-of-bag error in a same graph vs mtry and show that they follow a similar pattern

```
oob.err<-double(11)
test.err<-double(11)

#mtry is no of Variables randomly chosen at each split
if(FALSE){
for(mtry in 1:11)
{
  rf=randomForest(card ~ . , data = card_train, mtry=mtry, ntree=400)

  #oob.err[mtry] = rf$mtry[400] #Error of all Trees fitted on training

  pred=predict(rf,card_test) #Predictions on Test Set for each Tree
  CM = table(rf, card_test$card)
  accuracy = (sum(diag(CM)))/sum(CM)
  test.err[mtry]= with(card_test, accuracy) # "Test" Mean Squared Error
 #print(mtry)
  card_pred = predict(rf.card, newdata=card_test)
table(card_pred, card_test$card)

}
}
#round(test.err ,2) #what `mtry` do you use based on test error?
#round(oob.err,2) #does training error give you the same best `mtry`?
#matplot(1:mtry , cbind(oob.err,test.err), pch=20 , col=c("red","blue"),type="b",ylab
="Mean Squared Error",xlab="Number of Predictors Considered at each Split")
#legend("topright",legend=c("Out of Bag Error","Test Error"),pch=19, col=c("red","blu
e"))
```

4.Fit a Boosting Regression

a. Error Rate Table for classification

```
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```r
set.seed(2)
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(.5)

card_test = CreditCard %>%
  setdiff(card_train)

boost.card = gbm(card~.,
                  data = card_train,
                  distribution = "bernoulli", #"bernoulli" for logitic regression
                  n.trees = 500,
                  interaction.depth = 4)
#summary(boost.card)
yhat.boost = predict(boost.card,
                        newdata = card_test,
                        n.trees = 5000)
```

```
## Warning in predict.gbm(boost.card, newdata = card_test, n.trees = 5000): Number
## of trees not specified or exceeded number fit so far. Using 500.
```

```r
#summary(boost.card)
#yhat.boost
#card_pred = predict(yhat.boost, newdata=card_test)
CM = table(yhat.boost, card_test$card)
accuracy = (sum(diag(CM)))/sum(CM)
accuracy
```

```
## [1] 0.001517451
```

```r
#CM
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.
```

```r
yhat.boost = predict(boost.card,
                        newdata = card_test,
                        n.trees = 500)
#yhat.boost
```

b.Error Rate compared to LASSO/Ridge/Bagging/Random Forest

c.Importance Matrix

5.XGBoost regression

```r
library(xgboost)
```

Preparing data for XGBoost Model

```r
#CreditCard$card = as.numeric(as.factor(CreditCard$card))
#CreditCard$owner = as.numeric(as.factor(CreditCard$owner))
#CreditCard$selfemp = as.numeric(as.factor(CreditCard$selfemp))
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)
#dtrain <- xgb.DMatrix(as.matrix(sapply(X_train, as.numeric)), label=Y_train)
X_test <- as.matrix(card_test[!names(card_train) %in% c("card")])
```

a. Error rate table

```
if(FALSE){
set.seed(1)
card.xgb = xgboost(data=dtrain,
                     max_depth=2,
                     eta = 0.1,
                     nrounds=40, # max number of boosting iterations (trees)
                     lambda=0,
                     print_every_n = 10,
                     objective="binary:logistic") # for classification: objective = "
binary:logistic"

yhat.xgb <- predict(card.xgb,X_test)
CM = table(yhat.xgb, card_test$card)
accuracy = (sum(diag(CM)))/sum(CM)
accuracy
CM
set.seed(2)
param <- list("max_depth" = 2, "eta" = 0.1, "objective" = "reg:linear", "lambda" = 0)
cv.nround <- 500
cv.nfold <- 5
card.xgb.cv <- xgb.cv(param=param, data = dtrain,
                        nfold = cv.nfold,
                        nrounds=cv.nround,
                        early_stopping_rounds = 20 # training will stop if performanc
e doesn't improve for 20 rounds from the last best iteration
       )
}
```

b. Comparing error rate Lasso/Ridge/Bagging/RandomForest/Boosting models

c.Importance matrix

dtrain <- xgb.DMatrix(as.matrix(sapply(X_train, as.numeric)), label=Y_train)

Extra Credit: Tuning parameters using grid search for Boosting model in part 4