Credit Card Acceptance Model

Introduction

This project will analyze the cross-section data on the credit history for a sample of applicants for a type of credit card. I will see the influence of different variables on whether or not an individual was accepted for a credit card. This data is from cran.r-project.org and was done by William H. Greene. We are evaluating whether or not someone was accepted for a credit card so it only holds two values - yes or no.

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.

Brief Overall Summary Statistics for the data

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

```
library(ggplot2)
```

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.

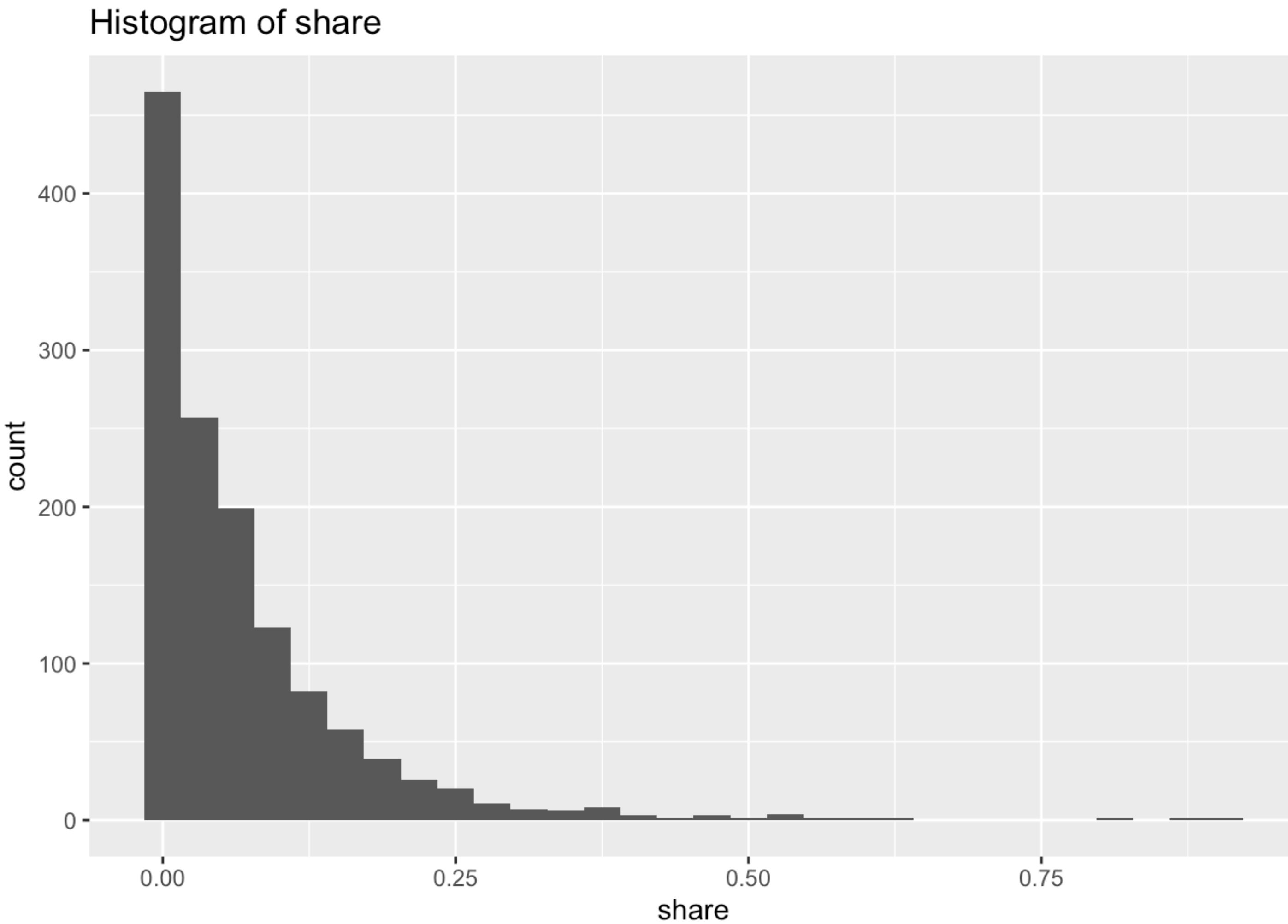
Analysis of four x variables

When looking at the x values provided, I believe the following can better explain the y: share, reports, majorcards, and active. I chose those four x values because intuitively, I assume that a negative impact on those four would negatively impact whether or not an individual gets a credit card so we would be able to see a correlation between them.

Analysis of share

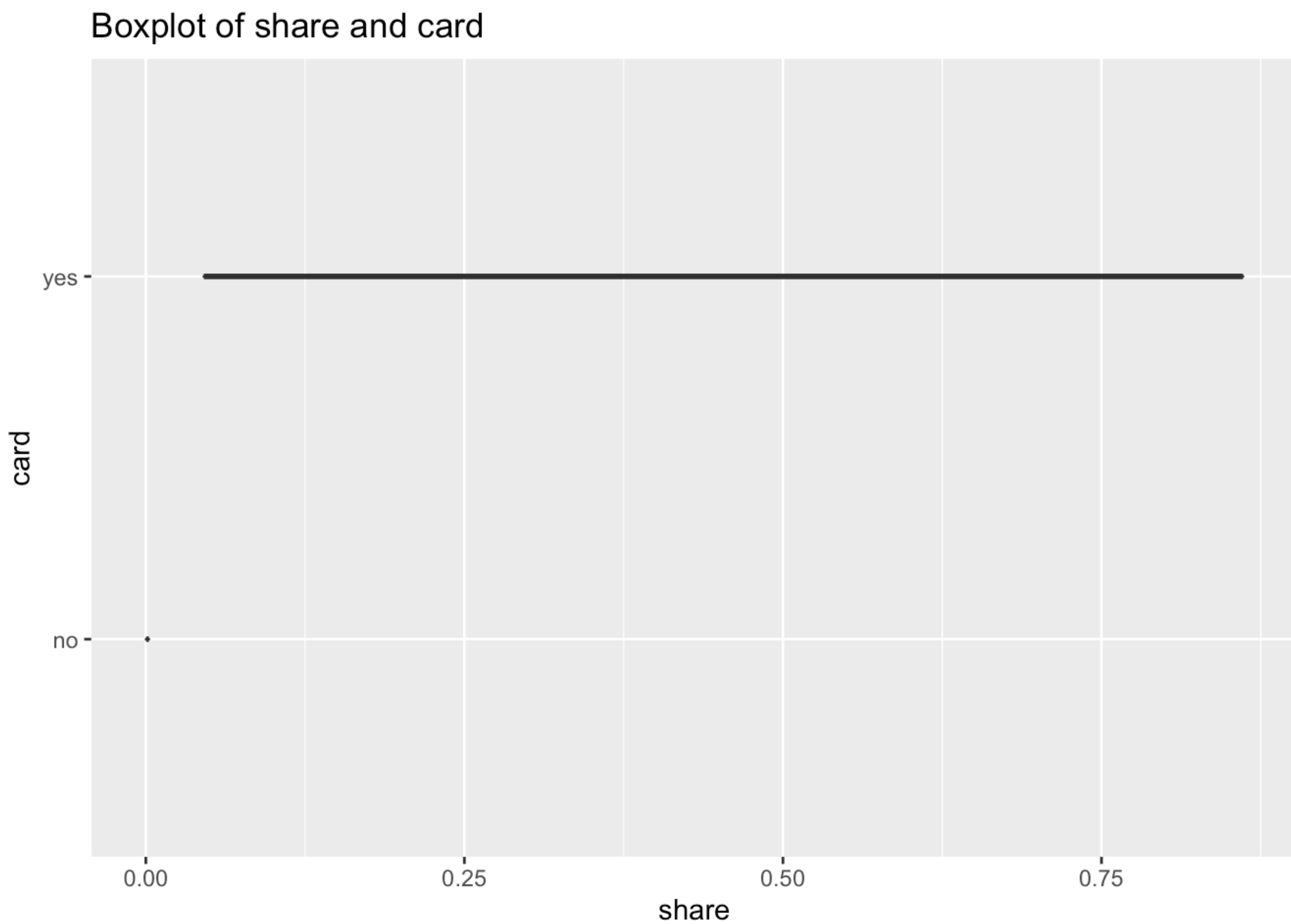
```
ggplot(CreditCard, aes(share))+
  geom_histogram() + ggtitle("Histogram of share")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



In the histogram above, I am analyzing a single variable, share, which is the ratio of monthly credit card expenditure to income. I notice that the histogram is right skewed. This says that people typically do not spend all of their income on credit card purchases. In fact, the data indicates that many people do not spend any money charged to a credit card. However, a good portion of the data indicates that many others do spend part of their income on credit card expenditures.

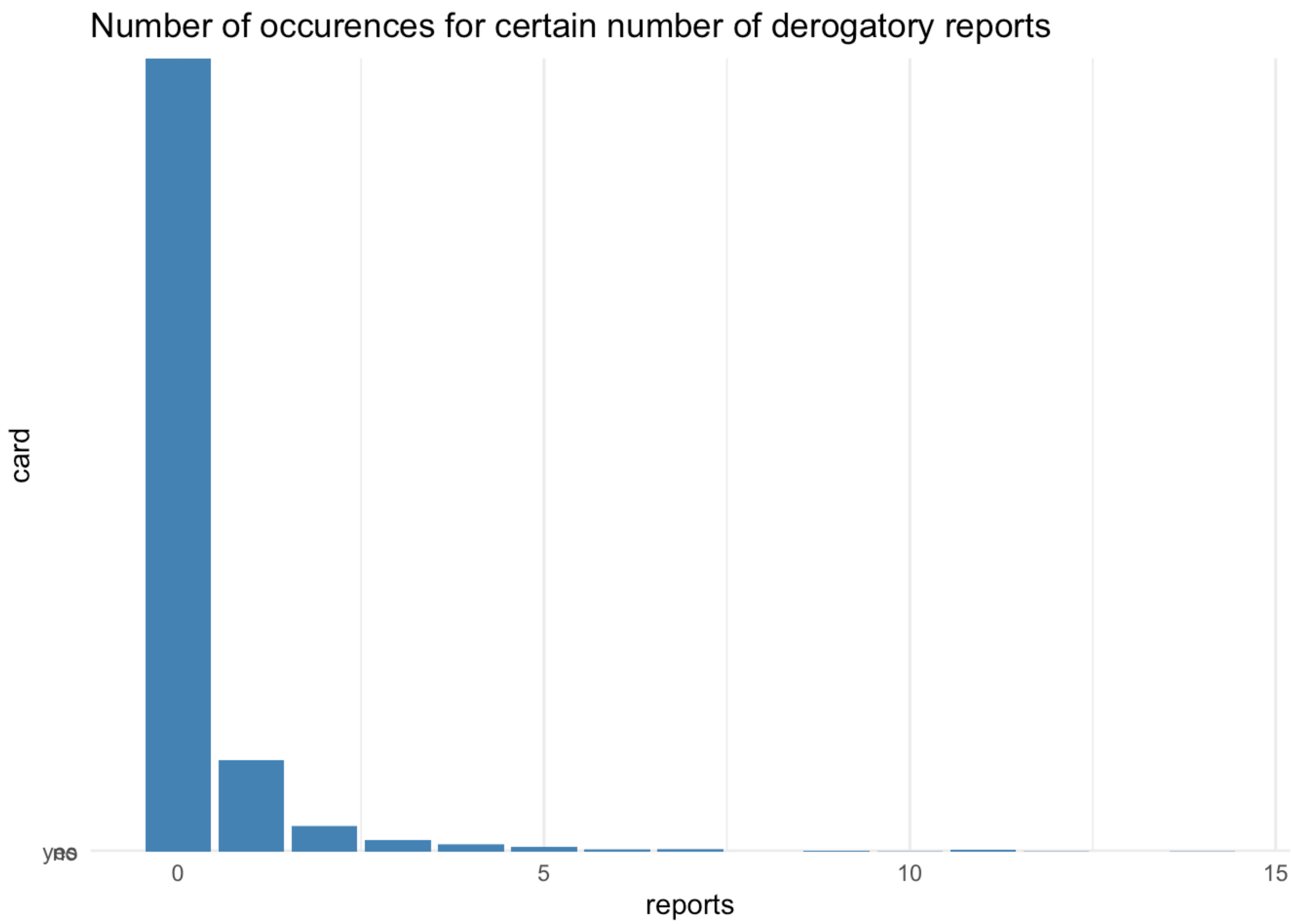
```
ggplot(CreditCard, aes(x=share, y=card)) +
  geom_boxplot() + ggtitle("Boxplot of share and card")
```



When looking at this graph, I notice that there were many credit cards approved even though the share of the monthly card expenditure and yearly income was in a wide range. However, this data also told me that everyone who got rejected for a credit card had a low share which is interesting to see because it seems that a low share is a good thing which means that people are not spending too money on their credit card in relation to their income.

Analysis of reports

```
ggplot(CreditCard, aes(x=reports, y=card)) +
  geom_bar(stat="identity", fill="steelblue")+
  theme_minimal()+ ggtitle("Number of occurences for certain number of derogatory rep
orts")
```

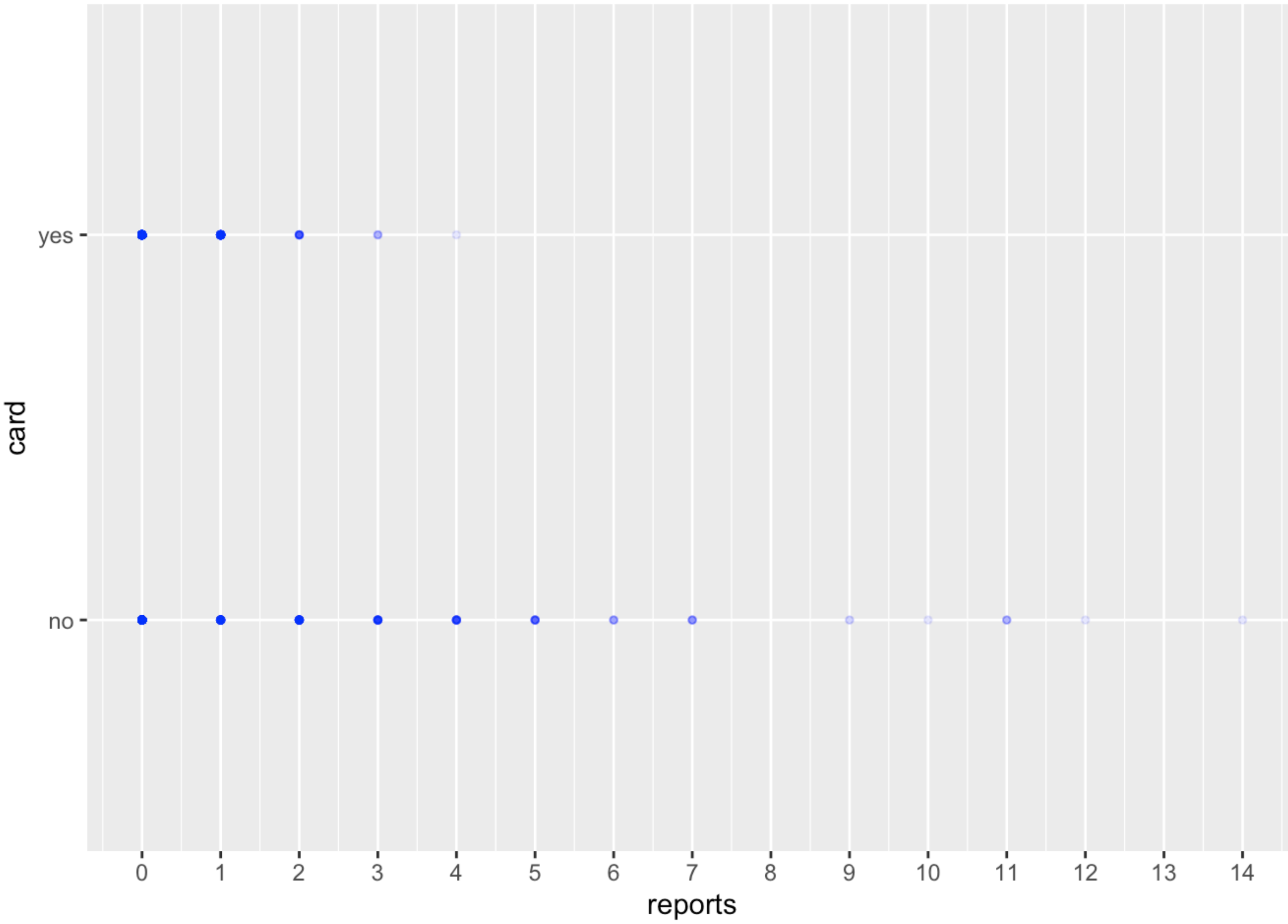



```
labs(y="count")
```

```
## $y
## [1] "count"
##
## attr(,"class")
## [1] "labels"
```

This barplot tells us that most people that applied for credit cards did not have any derogatory reports. The y-axis count tells us the number of observations of the derogatory reports that we see. This tells us that most people applying for a credit card did not commit crimes or do anything illegal since those that did may be deterred from opening a credit card if they committed a crime.

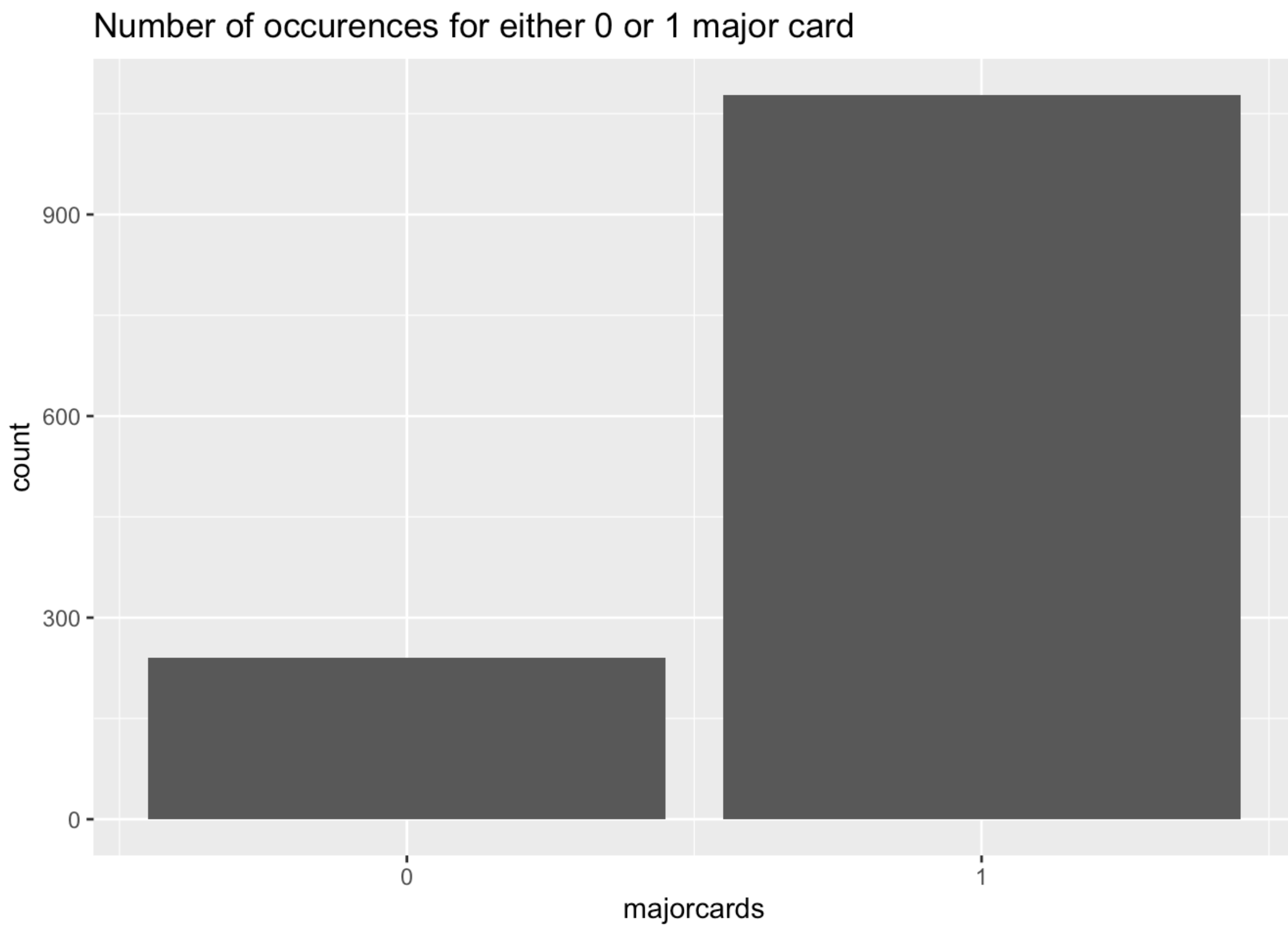
```
ggplot(CreditCard, aes(x=reports, y=card))+
  geom_point(color='blue', size = 1, alpha = 0.1) + scale_x_continuous(breaks = seq(0
, 14, by = 1)) +
  labs(y="card", x="reports")
```



When looking at the plot I see that most people who get approved for a card typically do not have any derogatory reports. The shade of the points indicate the density so for example, if a point was more blue, that means it had many more points at that spot than another that was lighter. Also, I see that people who got rejected for a credit card had low reports as well. However, people with a high number of derogatory reports got rejected for a credit card as well. Most people who got approved for a credit card had less than 3 derogatory reports.

Analysis of majorcards

```
ggplot(CreditCard, aes(majorcards))+ scale_x_continuous(breaks=c(0,1)) +
  geom_bar() + ggtitle("Number of occurences for either 0 or 1 major card")
```




```
# labs(y="count")
```

We see that most people who applied for a credit card only owned 1 major credit card already. More than a third of the people who applied with already 1 credit card, applied without having any major credit card at all. We will need to do further research to see whether or not it affected whether or not someone was approved for a credit card.

```
ggplot(CreditCard, aes(x=majorcards, y=card))+
  geom_point(color='blue', size = 1, alpha = 0.1) + scale_x_continuous(breaks=c(0,1))
```

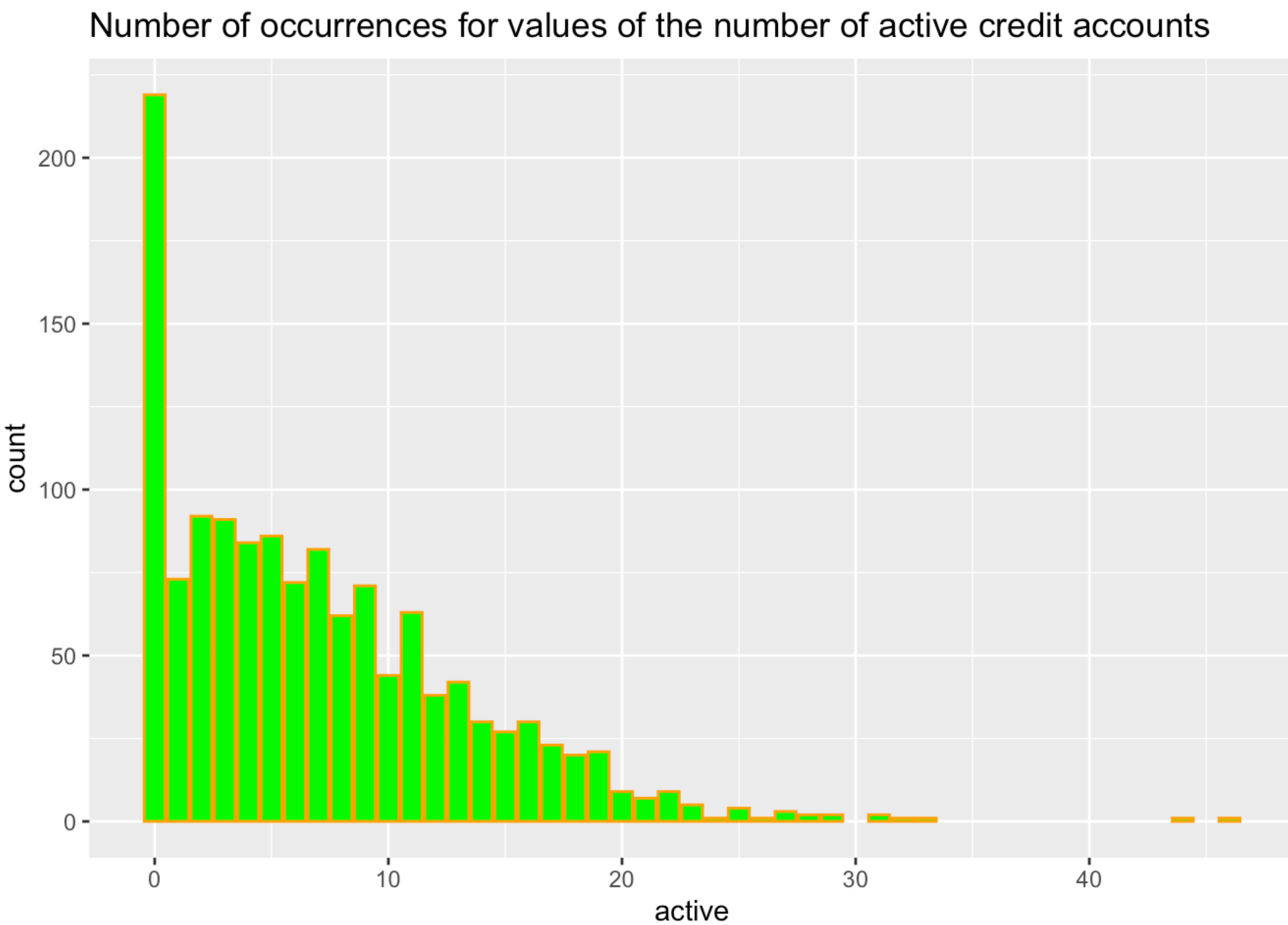

```
labs(y="card", x="majorcards")
```

```
## $y
## [1] "card"
##
## $x
## [1] "majorcards"
##
## attr(,"class")
## [1] "labels"
```

```
This graph shows me that there is typically an equal distribution between people who
get approved for a credit card with the number of majorcards they have. I deduce that
probably having either 0 or 1 major cards will not affect the acceptance of a credit
card. Based off the density of those points, it seems that all four are almost the sa
me color so they all have a similar distribution. Thus, having 0 or 1 major credit ca
rd will not be a large influence on the value of "card" which is the acceptance of th
e credit card.
```

Analysis of Active

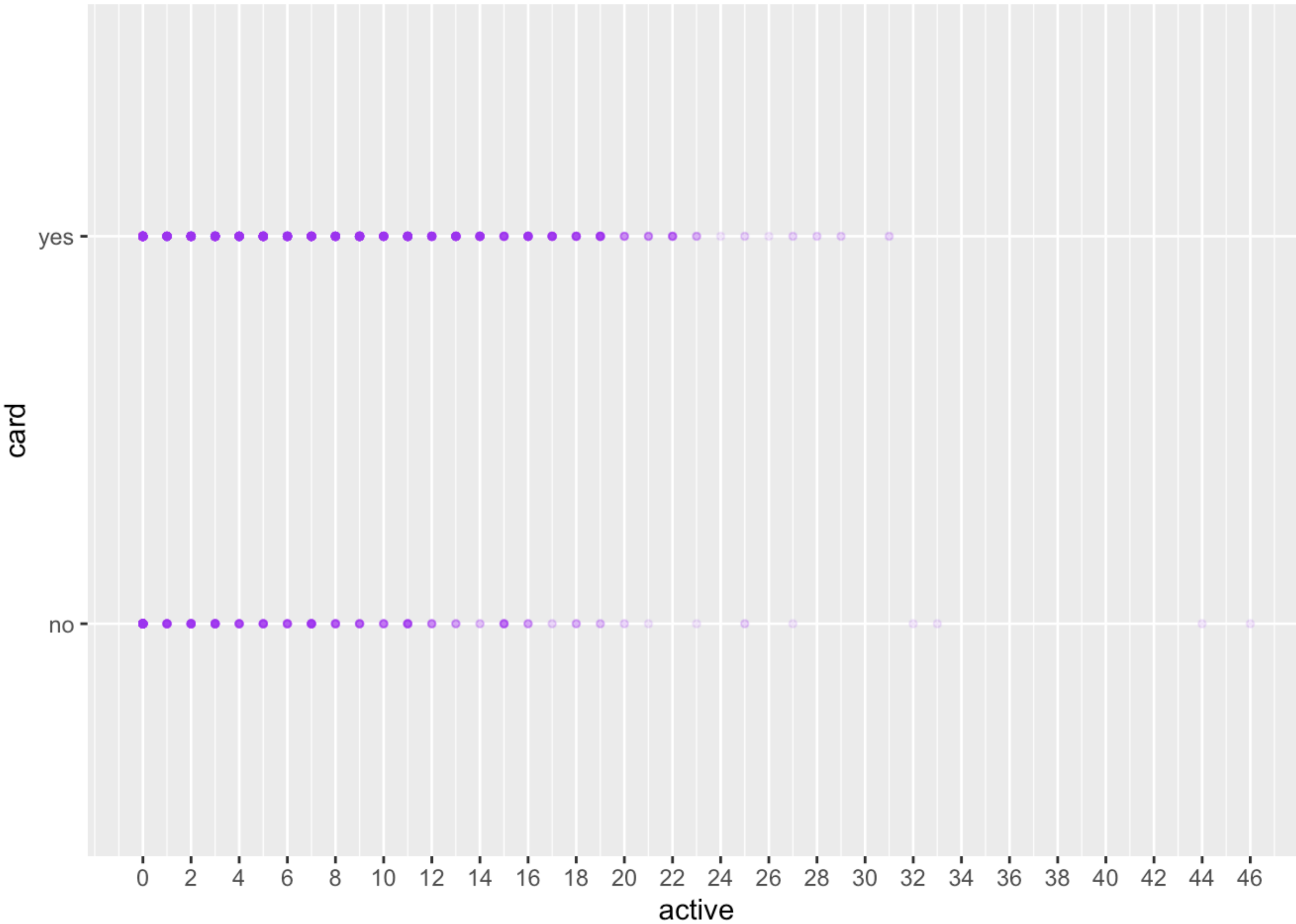
```
ggplot(CreditCard, aes(active))+
  geom_bar(color = "orange", fill = "green") + ggtitle("Number of occurrences for val
ues of the number of active credit accounts")
```




```
  # labs(y="count")
```

```
This graph tells us the distribution of the number of active credit accounts for the
dataset we have. Most people who apply for a credit card have 0 active credit account
s. This is logical because if someone does not have a credit account, it seems that t
hey may want to go apply for a credit card. The data shows that the distribution star
ts to lower as we increase the number of active credit accounts. This is logical as w
ell because since they already have active credit accounts, they are less likely to g
o want to get another credit card.
```

```
ggplot(CreditCard, aes(x=active, y=card))+
  geom_point(color='purple', size = 1, alpha = 0.1) +scale_x_continuous(breaks = seq(
0, 46, by = 2)) +
  labs(y="card", x="active")
```



This graph indicates to me that the number of active credit accounts may not greatly influence the acceptance of the credit card. The distribution for the range from 0 to 6 of active credit accounts look very similar for yes and no in cards. Furthermore, we see from the graph that a good number of individuals with a higher number of active credit accounts, from 10 to 20, even get approved for the credit card. In fact, if we look at a high number like 16 active credit accounts, we see that more people got accepted, rather than rejected, for the credit card.

Credit Card ML Analysis Report #2

Introduction

This is the second report for my Credit Card Acceptance Project. In this part of the project, I will be incorporating the comments I received on the first project while also running logistic regressions on my y-variable with several x-variables. I can observe the correct predictions by having a table of true positives, false positives, true negatives, and false negatives. This is because my Y, or cards, is a categorical/classification. I will try to see the effect of different combinations of X's and also observing the effect of taking some variables away. I will also be trying to run a K-nearest neighbors classification on the data as well.

2.1

```
library(ISLR)
library(AER)
library(ggplot2)
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
```

I have 11 different variables for my y so I will choose 5 variables that I believe have the largest effect.

```
glm.fits1 <- glm(card~reports+share+selfemp+majorcards+active,data = CreditCard, fami
ly = binomial())
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
glm.probs1 = predict(glm.fits1,CreditCard,type="response")
glm.pred1 = rep(0,length(glm.probs1))
glm.pred1[glm.probs1>.5]=1
table1 = table(glm.pred1,CreditCard$card)
table1
```

```
##
## glm.pred1   no  yes
##         0  294   23
##         1    2 1000
```

```
prob1 = (1000+294)/1319
prob1
```

```
## [1] 0.9810462
```

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 .9810462 which is extremely high.

```
glm.fits2 <- glm(card~reports+share+selfemp+dependents+months,data = CreditCard, fami
ly = binomial())
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
glm.probs2 = predict(glm.fits2,CreditCard,type="response")
glm.pred2 = rep(0,length(glm.probs2))
glm.pred2[glm.probs2>.5]=1
table2 = table(glm.pred2,CreditCard$card)
table2
```

```
##
## glm.pred2  no yes
##         0 295  25
##         1   1 998
```

```
prob2 = (998+295)/1319
prob2
```

```
## [1] 0.9802881
```

Next, I decided to replace majorcards and active from Model 1 with dependents and months. I decided to run a logistic regression and it had an error rate of .9802881. This was lower than the first one so it might say that having majorcards and active were better at helping the model predict than dependents and months. This makes sense because I chose the variables from Model 1 based on my own intuition on having great influence on those variables.

```
glm.fits3 <- glm(card~reports+share+selfemp,data = CreditCard, family = binomial())
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
glm.probs3 = predict(glm.fits3,CreditCard,type="response")
glm.pred3 = rep(0,length(glm.probs3))
glm.pred3[glm.probs3>.5]=1
table3 = table(glm.pred3,CreditCard$card)
table3
```

```
##
## glm.pred3  no yes
##         0 295  25
##         1   1 998
```

```
prob3 = (998+295)/1319
prob3
```

```
## [1] 0.9802881
```

For Model 3, I wanted to check how taking away variables would affect my logistic regression. With this, I decided to see the effect of my first three variables that I chose(which were the same for Model1 and Model2). These variables were reports, share, and selfempl. When looking at this, I saw that the error rate is.9802881 which means that it correctly predicted it about 98% of the time, which is also the same as when I had the two extra variables in Model 2. This does not make sense since having more variables should help our model predict better, but it is less in this case.

```
glm.fits4 <- glm(card~reports,data = CreditCard, family = binomial())
glm.probs4 = predict(glm.fits4,CreditCard,type="response")
glm.pred4 = rep(0,length(glm.probs4))
glm.pred4[glm.probs4>.5]=1
table4 = table(glm.pred4,CreditCard$card)
table4
```

```
##
## glm.pred4   no  yes
##         0  104   18
##         1  192 1005
```

```
prob4 = (1005+104)/1319
prob4
```

```
## [1] 0.8407885
```

In this model, I decided to see the effect of one variable on the model. I decided to do a logistic regression based on the model with the X, reports. I see that the it predicted it correctly .8407885, or ~84%. This makes sense because if we decrease the X to one variable, it would be harder to correctly predict the Y.

```
glm.fits5 <- glm(card~reports+income+owner+share+selfemp+dependents+majorcards+active
,data = CreditCard, family = binomial())
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
glm.probs5= predict(glm.fits5,CreditCard,type="response")
glm.pred5 = rep(0,length(glm.probs5))
glm.pred5[glm.probs5>.5]=1
table5 = table(glm.pred5,CreditCard$card)
table5
```

```
##
## glm.pred5   no  yes
##         0  293   23
##         1    3 1000
```

```
prob5 = (1000+293)/1319
prob5
```

```
## [1] 0.9802881
```

In this fifth model, I decided to increase the number of variables to 8 variables. The error rate became .9802881, which is less than my first model. This does not make sense because my first model's X were present in this model and this model included more variables. Since we had more variables to predict, it should have been a higher number but it was not. This means that the variables I added dependents, majorcards, and active, do not help us predict the model too well.

2.2

```
summary(glm.fits1)
```

```
##
## Call:
## glm(formula = card ~ reports + share + selfemp + majorcards +
##     active, family = binomial(), data = CreditCard)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.406   0.000   0.000   0.000   2.900
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -4.07873    0.60417  -6.751 1.47e-11 ***
## reports       -2.57145    0.96748  -2.658  0.00786 **
## share       2610.28045  482.96805   5.405 6.49e-08 ***
## selfempyes     0.46171    0.65609   0.704  0.48160
## majorcards     0.52302    0.53461   0.978  0.32791
## active         0.10532    0.03125   3.371  0.00075 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1404.6  on 1318  degrees of freedom
## Residual deviance:  146.5  on 1313  degrees of freedom
## AIC: 158.5
##
## Number of Fisher Scoring iterations: 16
```

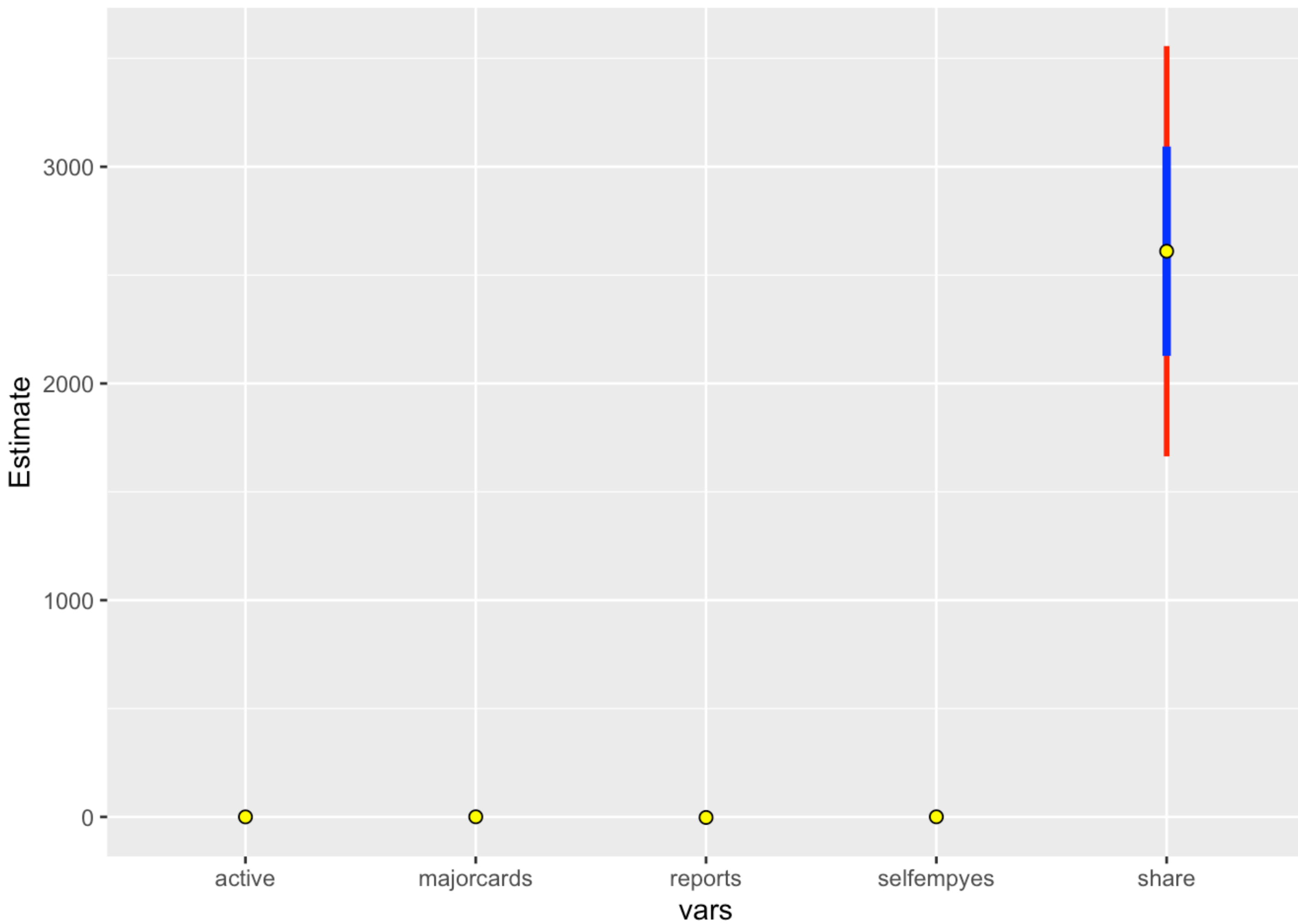
Model 1 is my best regression with the following X: reports, share, selfempl, majorcards, and active. The effect, or the values of their estimates in summary, is -4.07873, -2.57145, 2610, 0.46171, .52302, and .10532, respectively. The negative coefficient, reports, for this predictor suggests that if the reports suggest that the reports were higher for one person, they were more likely to be rejected for a credit card. This definitely makes sense because as I stated before, someone who has a higher number of derogatory reports should not be accepted for a credit card as much as someone who does not. Additionally, when looking at the P-values, I see that the p-value for reports is .00786 which indicates that there is some association between reports and credit card acceptances. Additionally, the other coefficients indicate the effect on that variable on the output, y. The coefficients are different from 0 however, selfempyes, majorcards, and active are not too significantly different from 0.

2.3

Model 1 is my best model with a correct prediction 98.10462% of the time. The true positive rate is 1000/1023 yes' and for the false positives, it is 23/1000. It tells me that the model is very reliable when predicting the classifications for y.

2.4

```
glm.fits1summary=summary(glm.fits1)
coefs=as.data.frame(glm.fits1summary$coefficients[-1,1:2])
names(coefs)[2]="se"
coefs$vars=rownames(coefs)
ggplot(coefs, aes(vars,Estimate)) +
  geom_errorbar(aes(ymin=Estimate-1.96*se,ymax=Estimate+1.96*se),lwd=1, colour="red",
width=0)+
  geom_errorbar(aes(ymin=Estimate - se,ymax=Estimate+se),lwd=1.5,colour="blue",width=
0)+
  geom_point(size=2,pch=21,fill="yellow")
```



If the graphs intersect 0 they’re not significant and if they do then that variable is significant. We see that for share,

PROBIT REGRESSION

```
glm.fits=glm(card~reports+share+selfemp+majorcards+active,data=CreditCard,family = bi
nomial(link = "probit"))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
glm.probs=predict(glm.fits,CreditCard,type="response")
glm.pred=rep("0",length(glm.probs))
glm.pred[glm.probs>.5]="1"
table(glm.pred,CreditCard$card)
```

```
##
## glm.pred  no yes
##        0 294  24
##        1   2 999
```

```
mean(glm.pred==CreditCard$card)
```

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

```
logitProb = (999+294)/1319
logitProb
```

```
## [1] 0.9802881
```

When running the probit regression, I see that the standard error is .9802881. Compared to my Logistic regression, which had .981. The logistic regression predicted my model better than my probit regression. The true positives and true negatives were 999 and 294 respectively.

CreditCardReport3

Introduction

In the third report, I will be using data from the Credit Card Acceptance Report. In this project, I will be incorporating results from my first and second reports while rulling LASSO and Ridge regressions. LASSO, or Least Absolute Selection and Shrinkage Operator, and Ridge are shrinkage methods. I will be conducting Ridge first in my report. Then, I will be running a LASSO regression. This would allow me to then compare both methods to how my logit method in report 2 performed. I will then run a regression or classification tree. Also, for extra credit, I will use boot-strap and fit 100 different trees to my boot-strap subsamples.

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

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

```
dim(CreditCard)
```

```
## [1] 1319   12
```

1)Divide credit card data into training and testing subsets and setting up for ridge and lasso regressions

```
set.seed(1)
train = CreditCard %>% sample_n(654)
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
```

RIDGE REGRESSION

2a.Tuning the model: Cross-Validation

```
grid = 10^seq(10, -2, length = 100)
ridge_mod = glmnet(x, y, alpha = 0, lambda = grid, family = "binomial")
cv.out = cv.glmnet(x_train, y_train, alpha = 0, family = "binomial") # Fit ridge regression model
on training data
```

2b. Choosing lambda within one standard error of minimum lambda

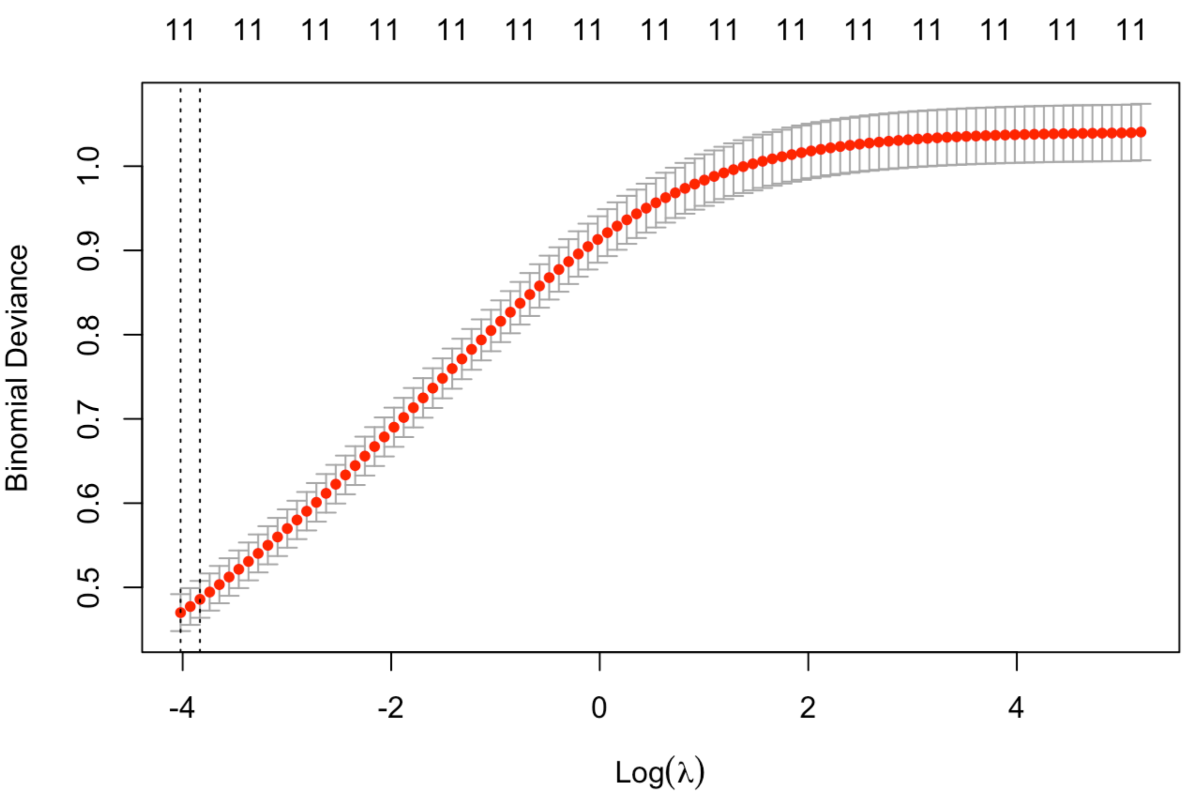
```
bestlam = cv.out$lambda.min  # Select lamda that minimizes training MSE
bestlam
```

```
## [1] 0.0179514
```

The best model is gonna be close to an OLS model since my lambda is 0, but I will choose 1.

2c. Show the cross-validation error and the chosen lambda in a graph

```
plot(cv.out)
```



In this graph, it is showing me the values of lambda that results in the smallest cross validation for 0. This plot will allow me to plot the MSE as a function of lambda.

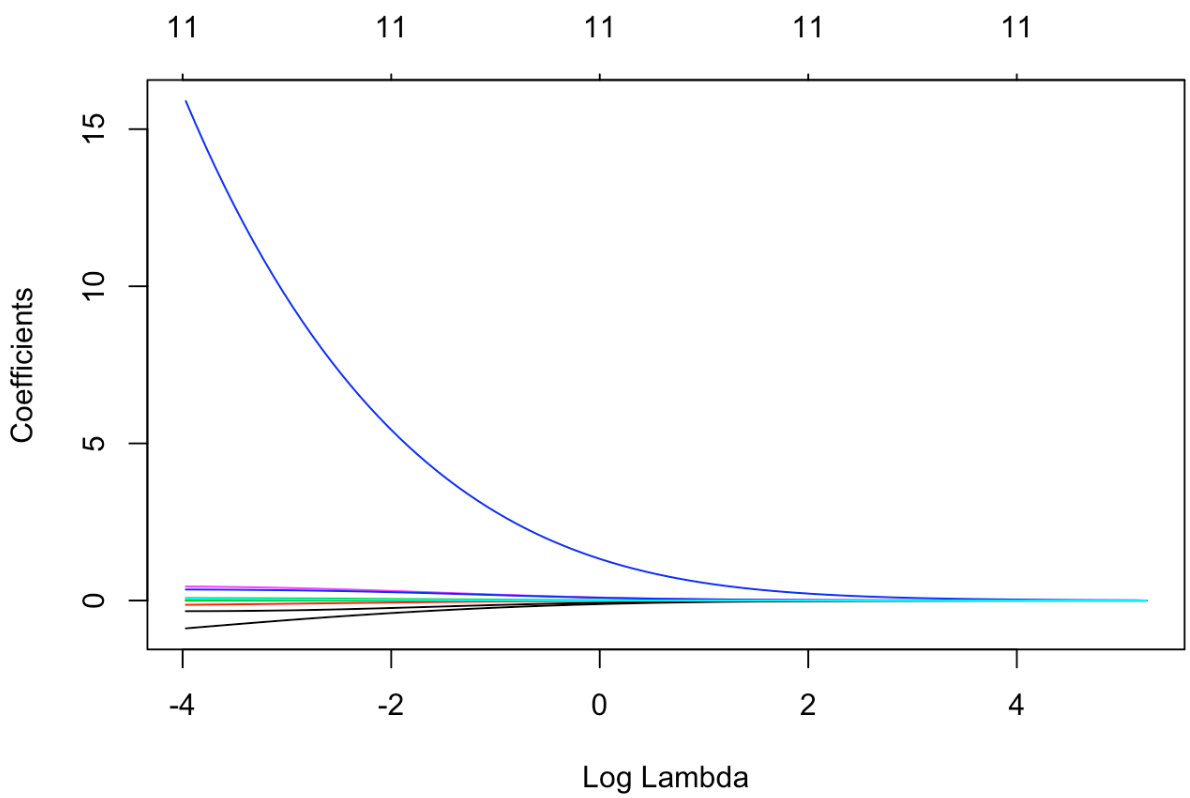
2d. Show how the coefficients vary with lambda in a graph

```
ridge_pred = predict(ridge_mod, s = 1, newx = x_test) # Use best lambda to predict test data
mean((ridge_pred - y_test)^2) # Calculate test MSE
```

```
## Warning in Ops.factor(ridge_pred, y_test): '-' not meaningful for factors
```

```
## [1] NA
```

```
out = glmnet(x, y, alpha = 0, family = "binomial") # Fit ridge regression model on the FULL datase
t (train and test)
plot(out, xvar = "lambda")
```



In this graph, we see that none of the coefficients are exactly zero, which is correct as shown below. We are plotting the coefficients for the different values of lambda.

2e. Report the coefficients correspondent with the chosen l

```
predict(out, type = "coefficients", s = bestlam)[1:12,] # Display coefficients using lambda chosen
by CV
```

```
##   (Intercept)       reports           age        income         share 
## -0.2172574440 -0.8850259823 -0.0011027159  0.0818480789 15.8922215919 
##   expenditure      owneryes    selfempyes    dependents        months 
##  0.0048072023  0.4458577211 -0.3376107430 -0.1357754099 -0.0003058545 
##    majorcards        active 
##  0.3515142631  0.0611774102
```

2f. Report the MSE (Error Rate for classification) in the test subset

```
ridge_pred = predict(ridge_mod, s = bestlam, newx = x_test) # Use best lambda to predict test data
mean((ridge_pred - y_test)^2) # Calculate test MSE
```

```
## Warning in Ops.factor(ridge_pred, y_test): '-' not meaningful for factors
```

```
## [1] NA
```

THE MSE is close to 101 when compared to the test data. #start of lasso regression

LASSO REGRESSION

2a.Tuning the model: Cross-Validation

```
lasso_mod = glmnet(x_train, 
                   y_train, 
                   alpha = 1, 
                   family="binomial") # Fit lasso model on training data
```

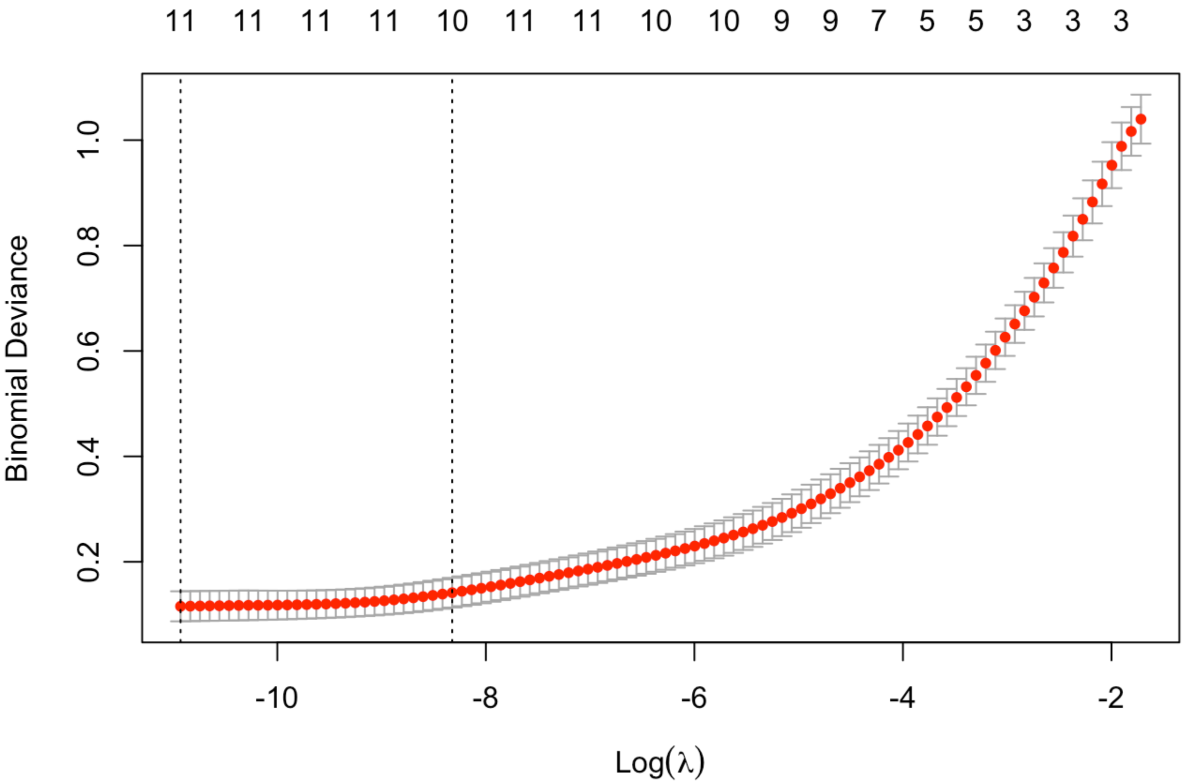
2b. Choosing lambda within one standard error of minimum lambda

```
set.seed(1)
cv.out = cv.glmnet(x_train, y_train, alpha = 1, family="binomial") # Fit lasso model on training d
ata
```

```
## Warning: from glmnet Fortran code (error code -99); Convergence for 99th lambda
## value not reached after maxit=100000 iterations; solutions for larger lambdas
## returned
```

```
## Warning: from glmnet Fortran code (error code -98); Convergence for 98th lambda
## value not reached after maxit=100000 iterations; solutions for larger lambdas
## returned
```

```
plot(cv.out) # Draw plot of training MSE as a function of lambda
```

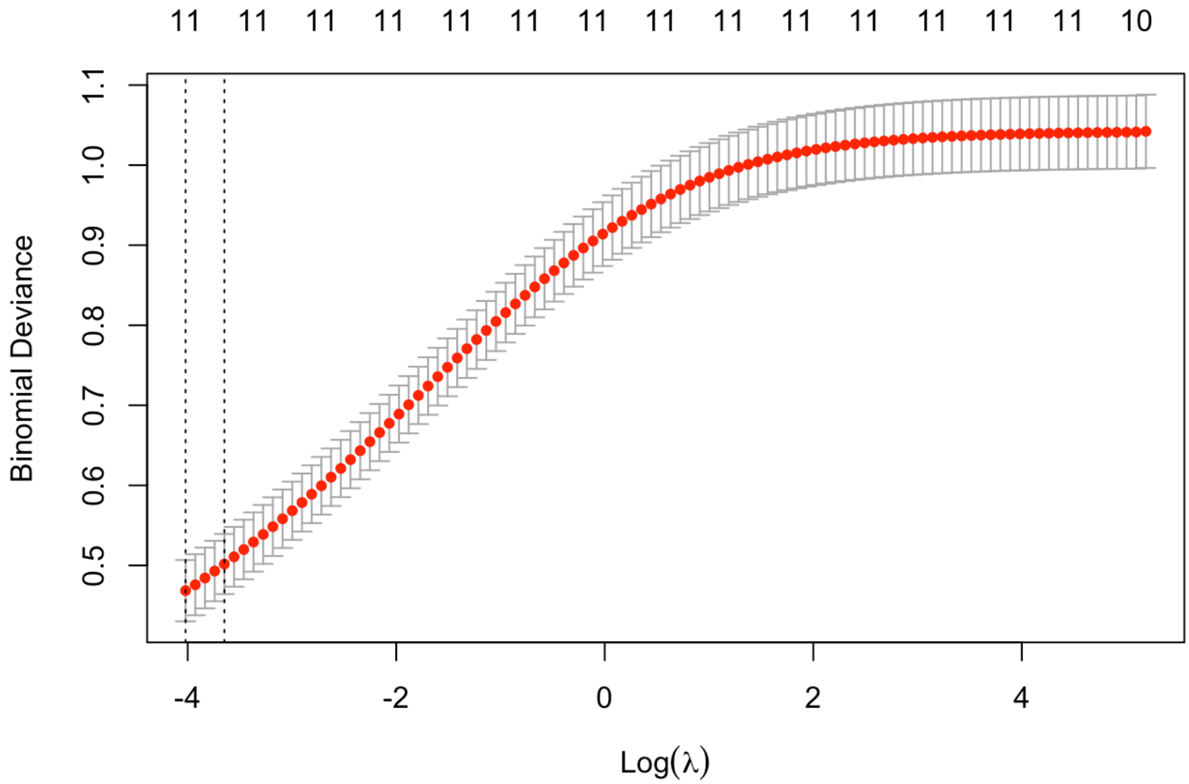



```
bestlam = cv.out$lambda.min # Select lamda that minimizes training MSE
bestlam
```

```
## [1] 1.79514e-05
```

2c. Show the cross-validation error and the chosen l in a graph

```
set.seed(1)
cv.out = cv.glmnet(x_train, y_train, alpha = bestlam, family="binomial") # Fit lasso model on trai
ning data
plot(cv.out) # Draw plot of training MSE as a function of lambda
```

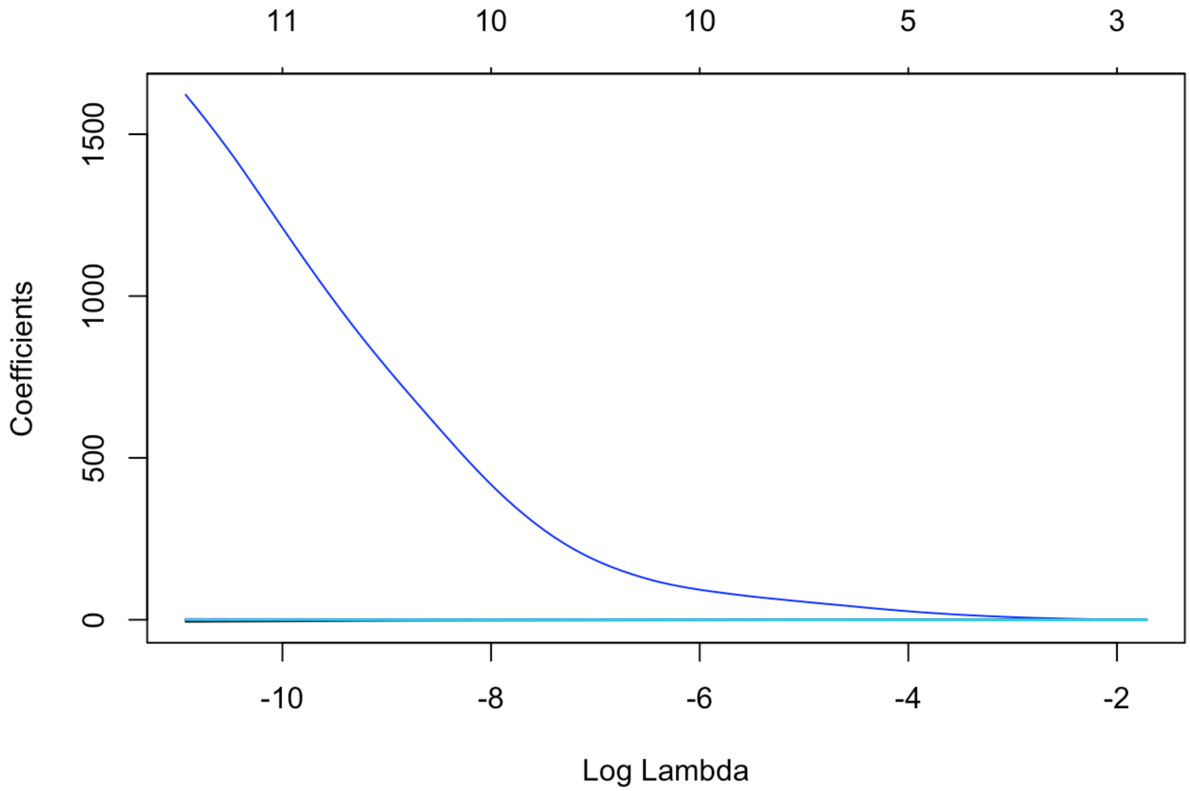


2d. Show how the coefficients vary with l in a graph

```
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")
```



In this graph, we see that there is a lot of It looks like a lot of the coefficients are overlapping each other.

2e. Report the coefficients correspondent with the chosen l

```
out = glmnet(x, y, alpha = 1, lambda = grid, family = "binomial") # Fit lasso model on full datase
t
lasso_coef = predict(out, type = "coefficients", s = bestlam)[1:12,] # Display coefficients using
lambda chosen by CV
lasso_coef
```

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

```
lasso_coef[lasso_coef != 0] # Display only non-zero coefficients
```

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

```
bestlam
```

```
## [1] 1.79514e-05
```

2f. Report the MSE (Error Rate for classification) in the test subset

```
lasso_pred = predict(lasso_mod, s = bestlam, newx = x_test) # Use best lambda to predict test data
mean((lasso_pred - y_test)^2) # Calculate test MSE
```

```
## Warning in Ops.factor(lasso_pred, y_test): '-' not meaningful for factors
```

```
## [1] NA
```

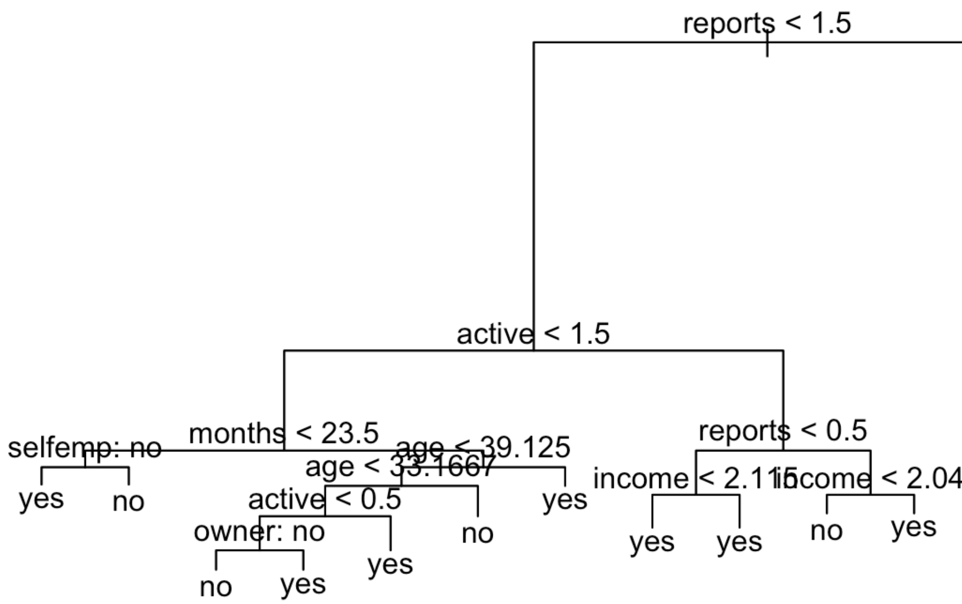
Running Classification Tree

3a. Fit and plotting a tree

```
library(tree)
tree_creditcard = tree(card ~ (active+majorcards+months+dependents+selfemp+owner+income+age+report
s), train)
summary(tree_creditcard)
```

```
##
## Classification tree:
## tree(formula = card ~ (active + majorcards + months + dependents +
##     selfemp + owner + income + age + reports), data = train)
## Variables actually used in tree construction:
## [1] "reports" "active"  "months"  "selfemp" "age"     "owner"   "income"
## Number of terminal nodes:  12
## Residual mean deviance:  0.6367 = 408.8 / 642
## Misclassification error rate: 0.1101 = 72 / 654
```

```
plot(tree_creditcard)
text(tree_creditcard, pretty = 0)
```



3b. Error rate and mse on tree

```
tree_pred = predict(tree_creditcard, test, type = "class")
table(tree_pred, test$card)
```

```
##
## tree_pred  no yes
##       no   72  27
##       yes  84 482
```

```
mean((tree_pred - CreditCard$card)^2)
```

```
## Warning in Ops.factor(tree_pred, CreditCard$card): '-' not meaningful for
## factors
```

```
## [1] NA
```

```
errorrate = (72+482)/(72+27+84+482)
errorrate
```

```
## [1] 0.8330827
```

The error rate is about 83% so we see that the pruned tree is classifying it correctly 83% of the time which is around the same as our shrinkage methods.

Use cross validation to prune your tree

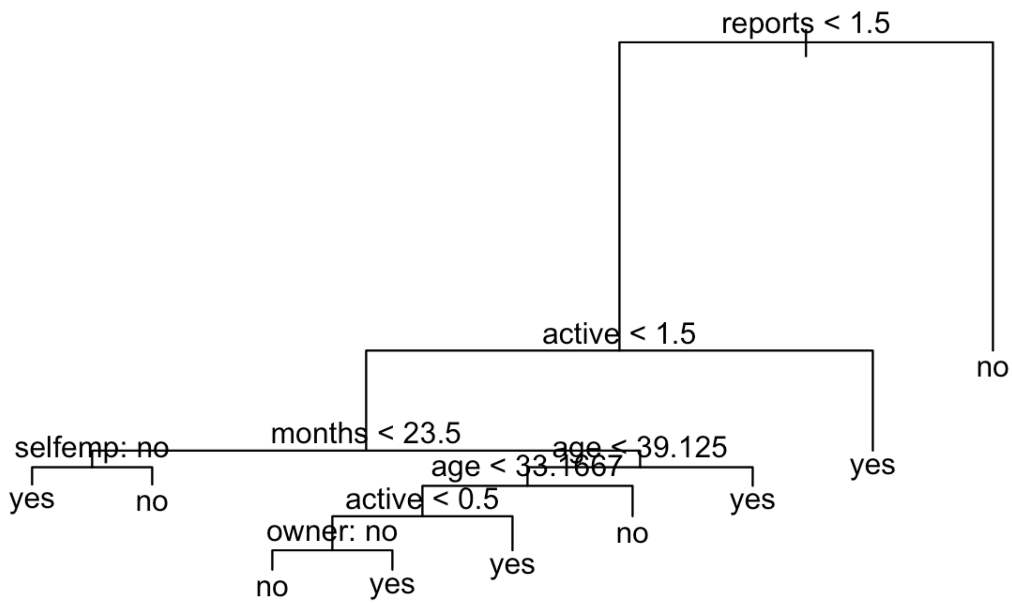
```
set.seed(5)
cv.creditcard = cv.tree(tree_creditcard, FUN = prune.misclass)
cv.creditcard
```

```
## $size
## [1] 12 11  9  7  2  1
##
## $dev
## [1] 112 115 106  98  97 140
##
## $k
## [1] -Inf  0.0  1.5  3.0  3.2 43.0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
#plot(cv.creditcard$card, cv.creditcard$dev, type = "b")
```

3d.Plotting the pruned tree

```
prune_creditcard = prune.misclass(tree_creditcard, best = 8)
plot(prune_creditcard)
text(prune_creditcard, pretty = 0)
```




```
#plot(cv.creditcard$card, cv.creditcard$dev, type = "b")
```

`# {r} #tree_pred = predict(prune_creditcard, test, type = "class") #table(tree_pred, CreditCard$card) #`

Extra Credit: using the boot strap

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

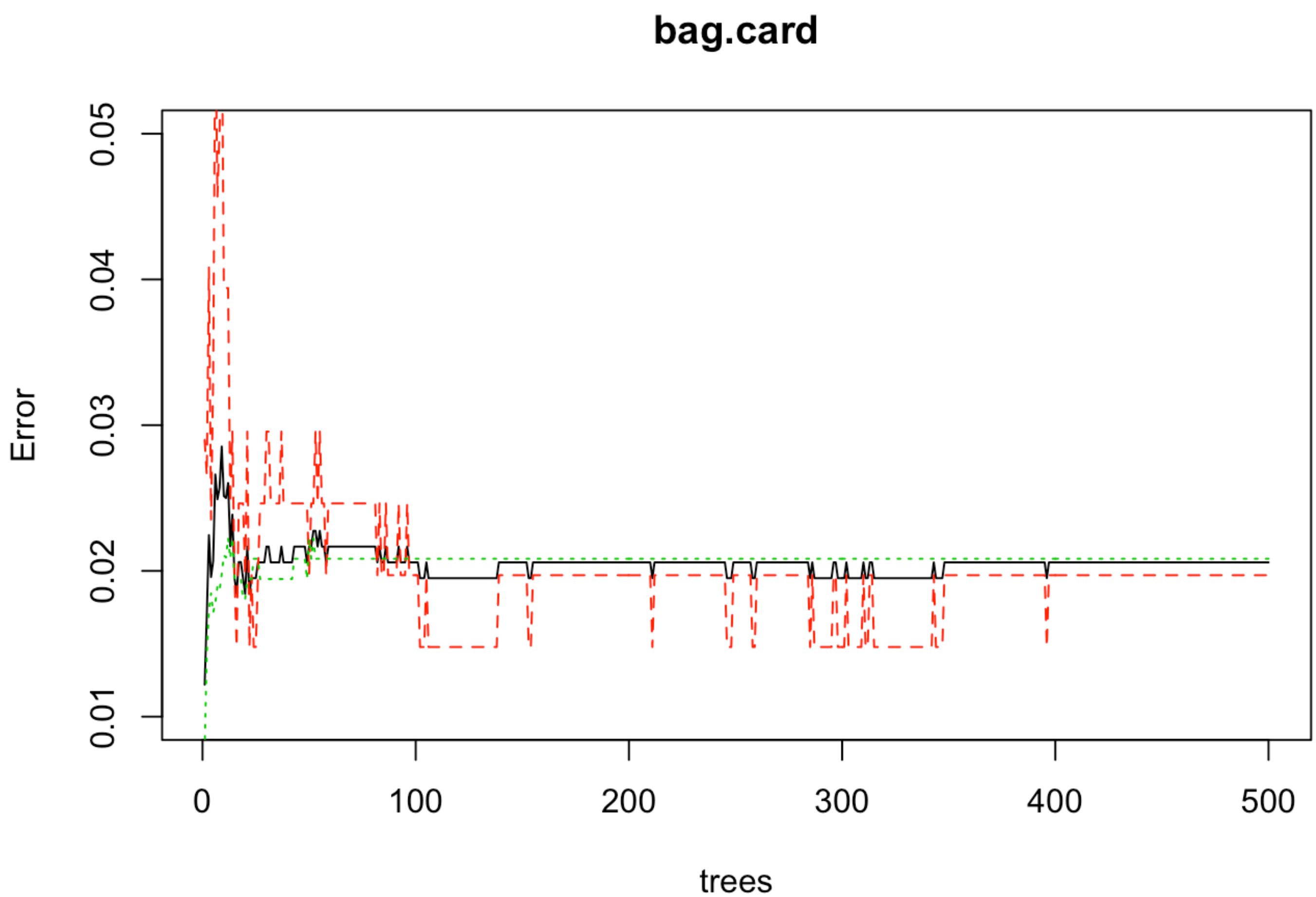
```
set.seed(1)
card_train = CreditCard %>%
  sample_frac(.70)

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.

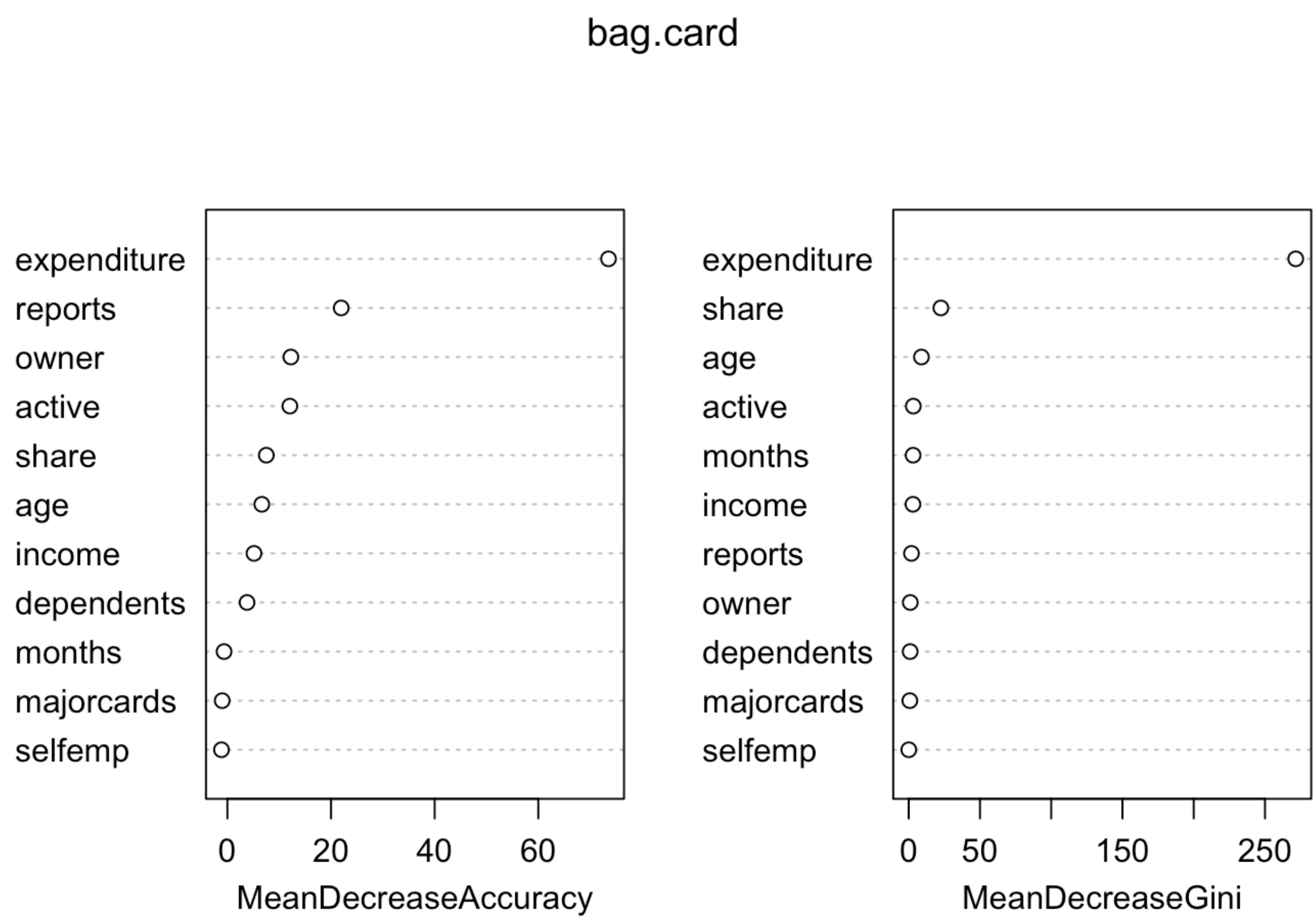
The error rate for bagging was 97.9415%, for LASSO was 97.3%, and for Ridge was 96.9%. Bagging may have been better at classifying my data because my some of my variables and my response variable was categorical. Bagging, which is similar to random forests, can automatically model non-linear data, which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

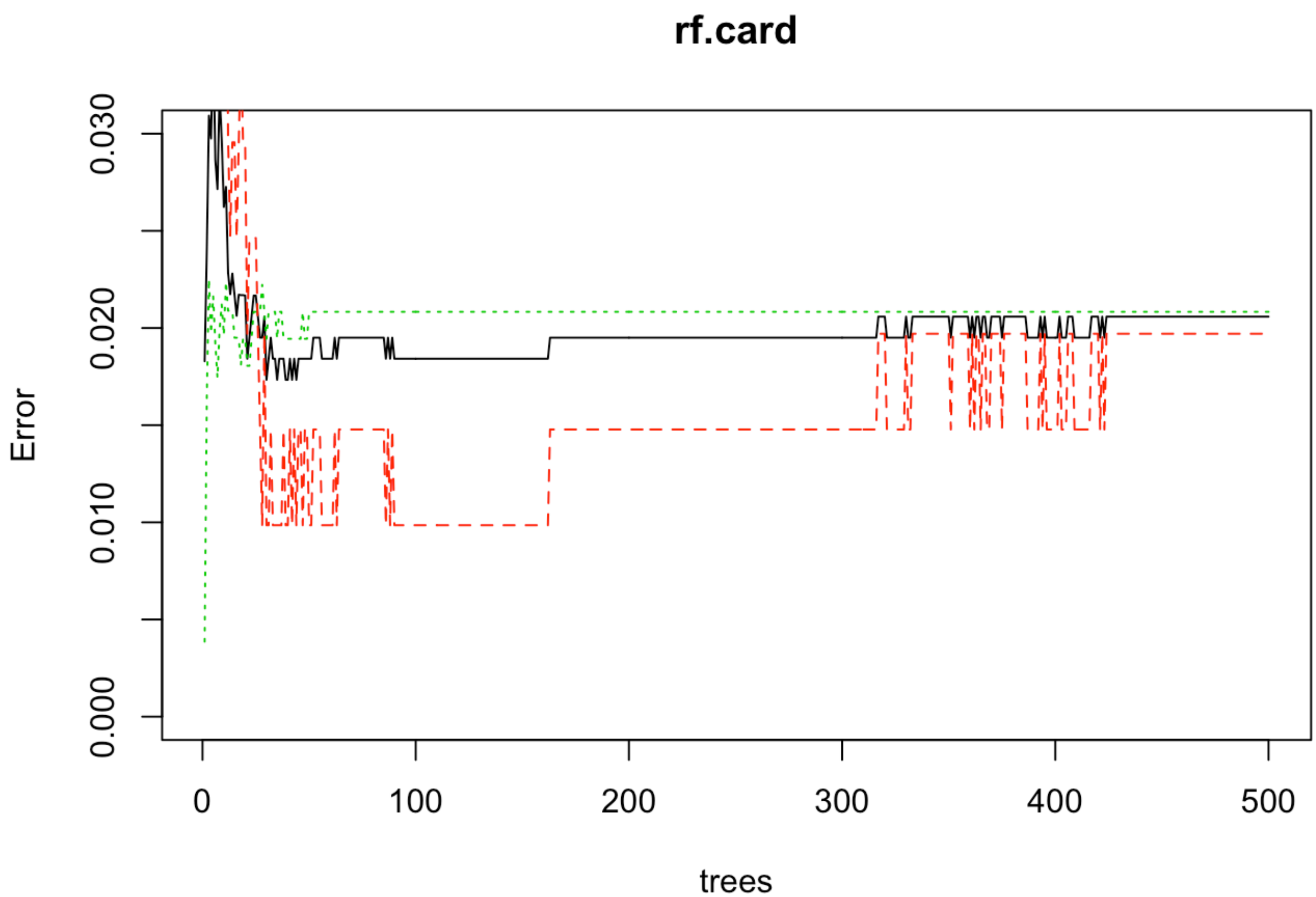
3.Fitting a Random Forest Regression

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

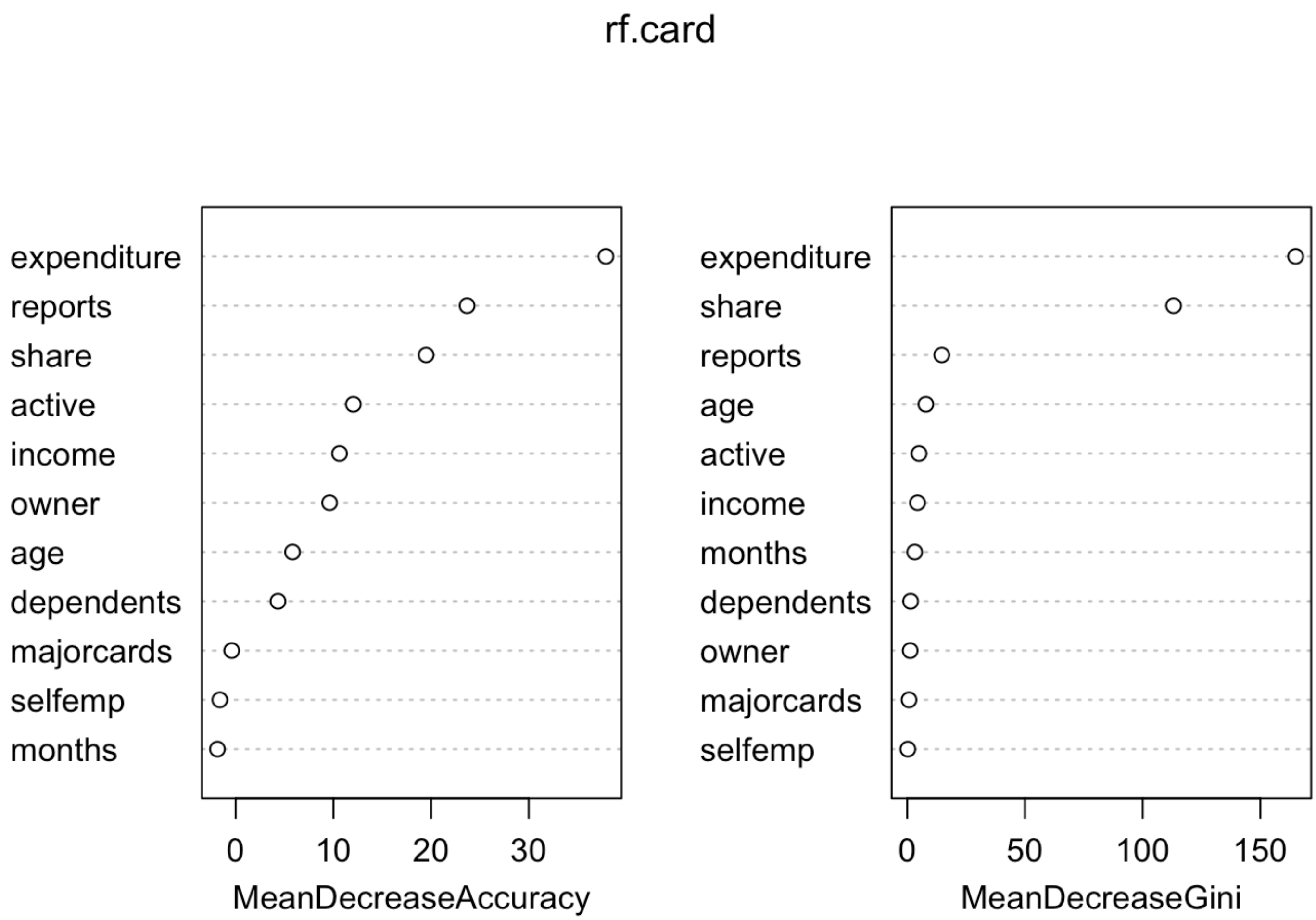
The Random Forest Error rate was 97.9798% and the error rate for bagging was 97.9415%, for LASSO was 97.3%, and for Ridge was 96.9%. Random Forest may have been better at classifying my data because my some of my variables and my response variable was categorical. Random Forest can automatically model 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

```
library(xgboost)
```

Preparing data for XGBoost Model

```
#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