# Credit Card Report 3

# 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)
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

```
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
```

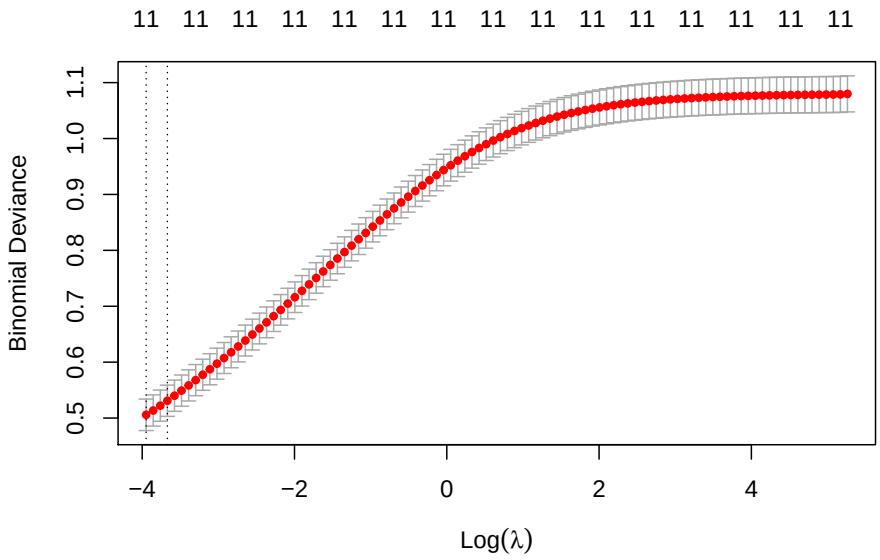
```
## [1] 0.2121212
```

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

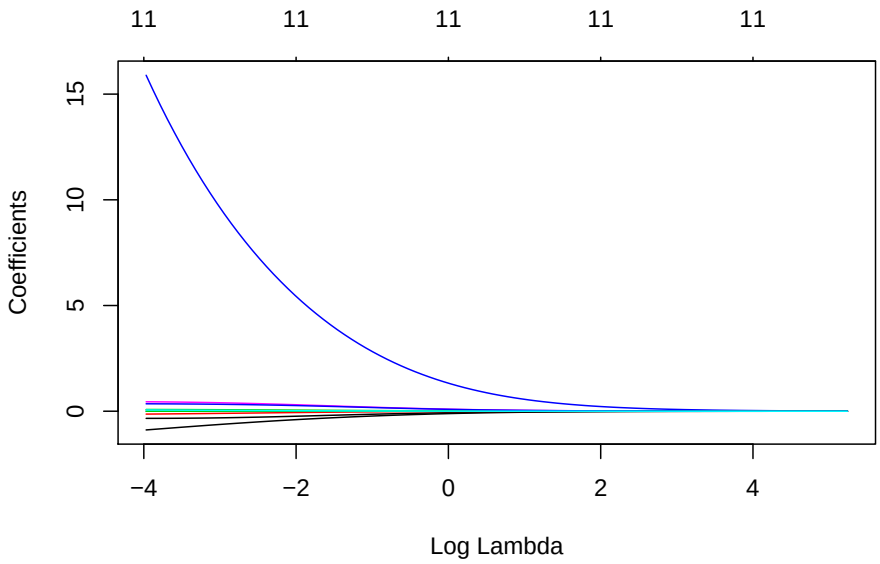
```
set.seed(123)
ridge_cv.out = cv.glmnet(x_train, y_train, alpha = 0, family = "binomial")
bestlam = ridge_cv.out$lambda.min
bestlam
```

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

```
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 $\lambda$ within 1 standard error is 0.01384886 in my Ridge Regression. The graph shows the relationship between the cross-validation error and log of $\lambda$ 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

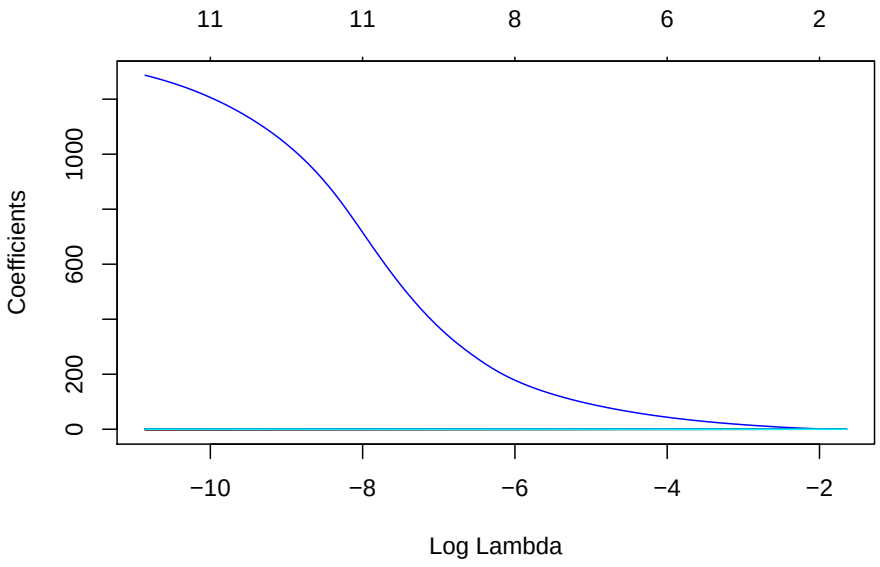
```
grid = 10^seq(10, -2, length = 100)
lasso_mod = glmnet(x_train, y_train, alpha=1, lambda = grid, family = "binomial")
lasso_pred = predict(lasso_mod, s = 4, newx = x_test)
y_pred = ifelse(lasso_pred>0.2,1,0)
lassoperf_matrix = table(y_pred, y_test)
lassoperf = (sum(diag(lassoperf_matrix)))/sum(lassoperf_matrix)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

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

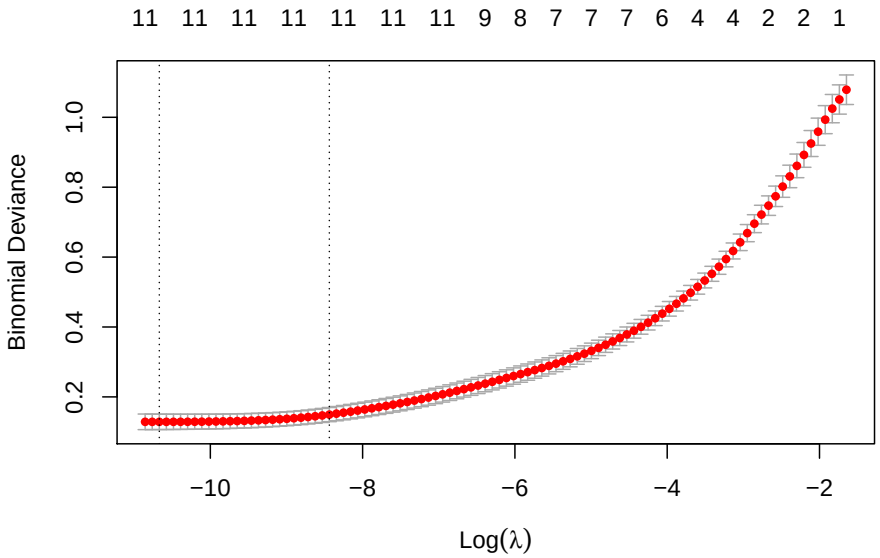
```
set.seed(123)
lasso_mod = glmnet(x_train,
                   y_train,
                   alpha = 1,
                   family = "binomial")
ls(lasso_mod)
```

```
##  [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
```

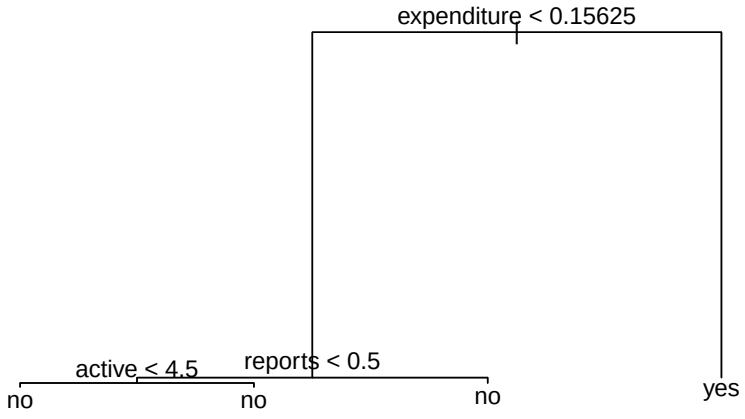
Lasso Regression has a 97.9% classification rate. Furthermore, we see how many of the coefficients in the graph are at zero, which is what Lasso regression does.

### 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)
```

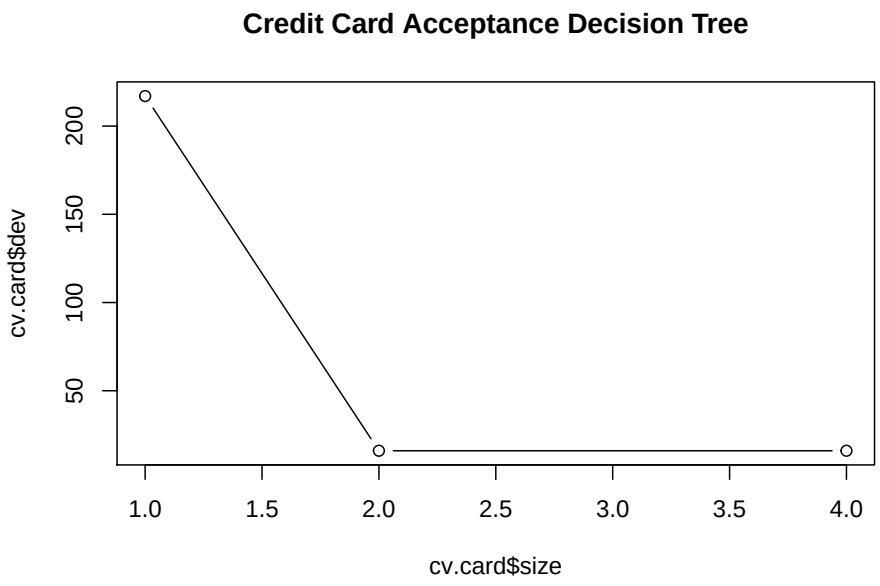


This tree has a depth of 3 and 4 leaves.

```
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.

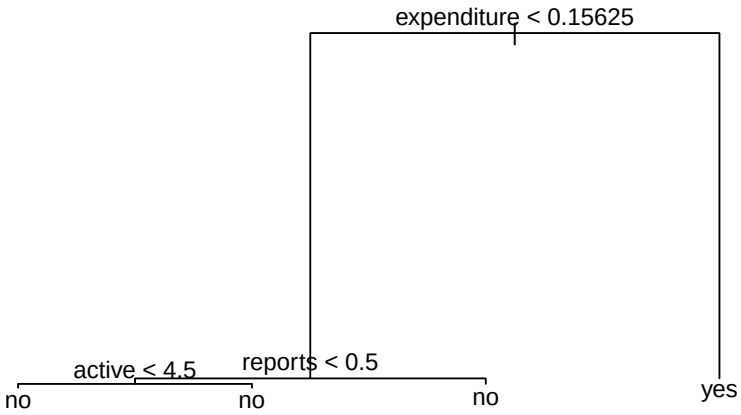
```
tree_table = table(tree_pred, test$card)
treeperf = (sum(diag(tree_table)))/sum(tree_table)
treeperf
```

```
## [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(ridgeperf, lassoperf, treeperf),ncol=1,
                      byrow=TRUE)
rownames(mlperformance)<-c("Ridge", "Lasso", "Decision Tree")
colnames(mlperformance)<-c("Performance")
mlperformance <- as.table(mlperformance)
mlperformance
```

```
##               Performance
## Ridge           0.9292929
## Lasso           0.9797980
## Decision Tree   0.9848485
```