# Machine Learning for Time Series: Predicting a Recession

*Lester Pi*

*June 10, 2017*

```
knitr::opts_chunk$set(echo = TRUE)
```

## Introduction

This projects compares traditional time series forecasting methods with machine learning techniques. Since this is an offshoot of my capstone project, I will cover the testing methods and conclusions of my capstone in brief and build upon them. Skip to the "Project Extension" section if you would like to skip over the capstone work.

## Capstone Work

The premise of the capstone was to compare ARIMA with machine learning, more explicitly LASSO, decision trees, and neural networks, in a time series framework. The time series of choice is the volatility of the S&P 500. For machine learning, I introduced new variables into the data set including presidential approval ratings, interest rate, and others. The machine learning techniques surpassed the ARIMA model when comparing the MAPE (recursive 2.91353, rolling 2.93426) with a tuned neural netowrk performing with the lowest MAPE (2.2889). I concluded that the machine learning techniques, especially neural networks, can effectively take the place of ARIMA for time series forecasting.

The following code has had its output surpressed until the "Project Extension" section.

```
library('xts')
library("quantmod")
library('forecast')
library('dynlm')
library('vars')
library('tseries')
library('glmnet')
library('randomForest')
library('neuralnet')
library('plyr')
library('glarma')
library('caret')
```

```
setwd("C:/cygwin64/home/Lester/thesis")
```

```
#define functions
DateToInt = function(d){
  switch(d,January={return(1)},February={return(2)},March={return(3)},
         April={return(4)},May={return(5)},June={return(6)},
         July={return(7)},August={return(8)},September={return(9)},
         October={return(10)},November={return(11)},December={return(12)})
  return(NA)
```

```
}

IntToDate = function(i){
  switch(i,"1"={return("January")},"2"={return("February")},"3"={return("March")},
         "4"={return("April")},"5"={return("May")},"6"={return("June")},
         "7"={return("July")},"8"={return("August")},"9"={return("September")},
         "10"={return("October")},"11"={return("November")},"12"={return("December")})
  return(NA)
}

mape = function(y, yhat){
  return(mean(abs(y - yhat)/abs(y)*100))
}


backtest = function(ts, step_size, type){
  results = c()
  index = floor(2*length(ts)/3)

    y = c()
    y_hat = c()


  if(type == "recursive"){

    while(index < length(ts)-step_size){
      temp_mod = auto.arima(ts[1:index])
      temp_forecast = forecast(temp_mod,h=step_size)
      start = index+1
      end = index+step_size
      # results=c(results,mape(ts[start:end],temp_forecast$mean))

      y=c(y,ts[(index+1):(index+step_size)])
      y_hat=c(y_hat,temp_forecast$mean)

      index = index+1
    }
  }

  else if(type == "rolling"){
    count=0
    while(index < length(ts)-step_size){
      temp_mod = auto.arima(ts[(1+count):index])
      temp_forecast = forecast(temp_mod,h=step_size)
      # results=c(results,mape(ts[(index+1):(index+step_size)],temp_forecast$mean))
      y=c(y,ts[(index+1):(index+step_size)])
      y_hat=c(y_hat,temp_forecast$mean)

      index=index+1
      count=count+1
    }
  }
    #   print(y)
```

```r
    # print(y_hat)
  results = list(y,y_hat)
  return(results)
}
```

```r
options(scipen=999)


VIX = read.csv("^VIX.csv",stringsAsFactors=FALSE)
rownames(VIX)=as.Date(VIX$Date)
vix = VIX$Adj.Close
names(vix) = as.Date(VIX$Date)
vix = na.omit(vix)


GSPC = read.csv("^SP500TR.csv",stringsAsFactors=FALSE)
rownames(GSPC)=as.Date(GSPC$Date)
sp500 = GSPC$Adj.Close
names(sp500) = as.Date(GSPC$Date)
sp500 = na.omit(sp500)
print(length(vix))
```

```
## [1] 6901
```

```r
print(length(sp500))
```

```
## [1] 6901
```

```r
print(length(sp500))
```

```
## [1] 6901
```

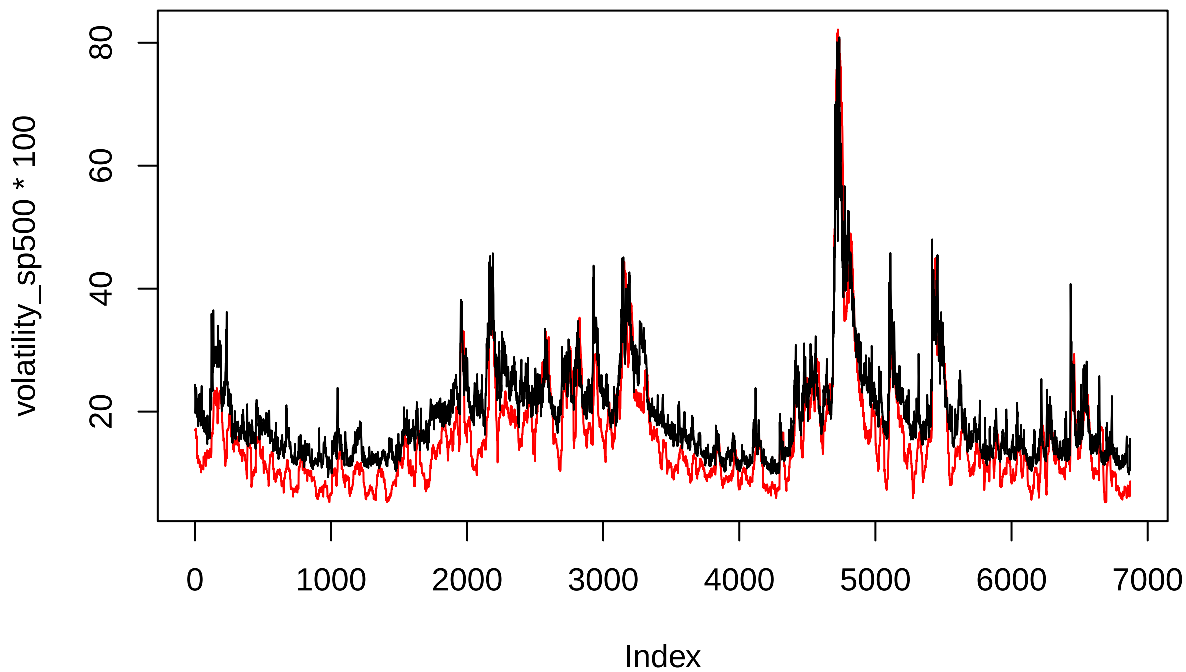
```r
print(length(vix))
```

```
## [1] 6901
```

```r
#transform into returns
sp500_returns = na.omit(diff(sp500)/sp500[-length(sp500)])

window_size = 30
volatility_sp500 = na.omit(volatility(sp500[1:length(sp500)], n=window_size))

pres_approval = read.csv("president_approval.csv",stringsAsFactors = FALSE)

pres_approval$republican = ifelse(pres_approval$President_Name=="Donald J. Trump"|
                               pres_approval$President_Name=="George W. Bush"|
                               pres_approval$President_Name=="George H.W. Bush",1,0)
pres_approval$End_Date = as.Date(pres_approval$End_Date,"%m/%d/%y")

pres_average = pres_approval

pres_average$Month <- months(pres_approval$End_Date)


pres_average$Year <- format(pres_approval$End_Date,format="%Y")
```

```
approval_average = aggregate( Approval ~ Month + Year,pres_average , mean )
disaproval_average = aggregate( Disapproval ~ Month + Year,pres_average , mean )
unknown_average = aggregate( Unsure.No_Data ~ Month + Year,pres_average , mean )
```

```
#make same length as volatility
vix_volatility=vix[30:length(vix)]


plot(volatility_sp500*100,col="red",type='l')
lines(vix_volatility)
```




```
ts_vix = ts(vix_volatility)
ts_vol = ts(volatility_sp500)

adf.test(ts_vol)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_vol
## Dickey-Fuller = -6.9402, Lag order = 19, p-value = 0.01
## alternative hypothesis: stationary
```

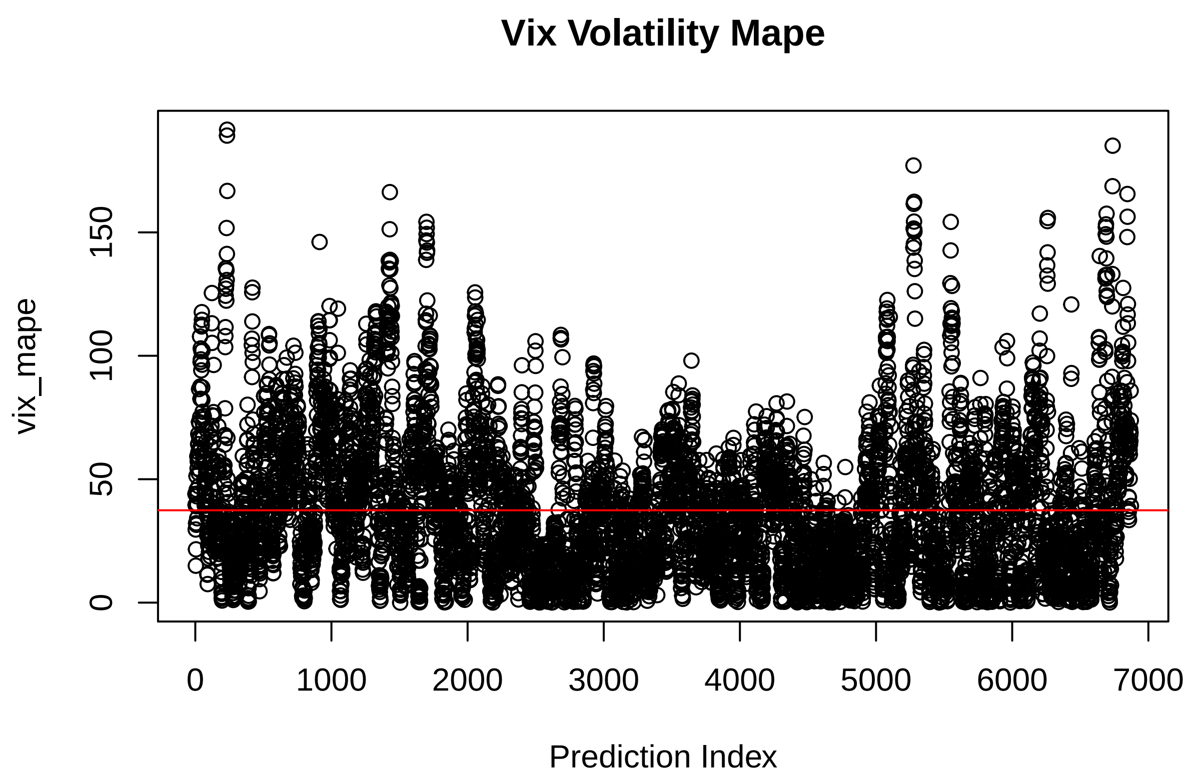
```
#benchmark
recursive=(backtest(ts_vol,1,"recursive"))
rolling=(backtest(ts_vol,1,"rolling"))
mape(recursive[[1]],recursive[[2]])
```

```
## [1] 2.913536
```

```r
mape(rolling[[1]],rolling[[2]])
```

```
## [1] 2.927933
```

```r
vix_mape=c()
for(i in 1:length(vix_volatility)){
  vix_mape=c(vix_mape,mape(volatility_sp500[i]*100,vix_volatility[i]))
}
plot(1:length(vix_mape),vix_mape,main="Vix Volatility Mape",xlab="Prediction Index")
abline(h=mean(vix_mape),col='red')
```




```r
print(mean(na.omit(vix_mape)))
```

```
## [1] 37.4075
```

```r
vol_df = data.frame(volatility_sp500)
vol_df$vix = vix_volatility
```

```r
#output file csv
filename = "out.csv"

if(file.exists(filename)){
  file.remove(filename)
}
```

```
## [1] TRUE
```

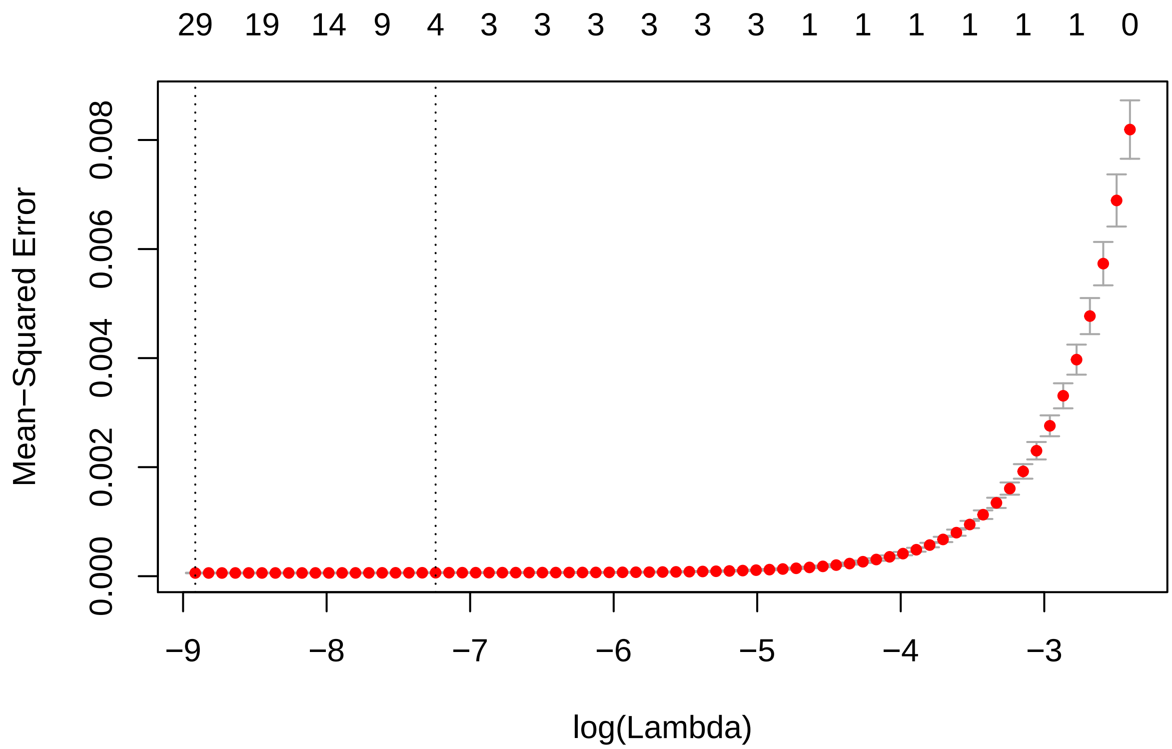
```
file.create(filename)
```

```
## [1] TRUE
```

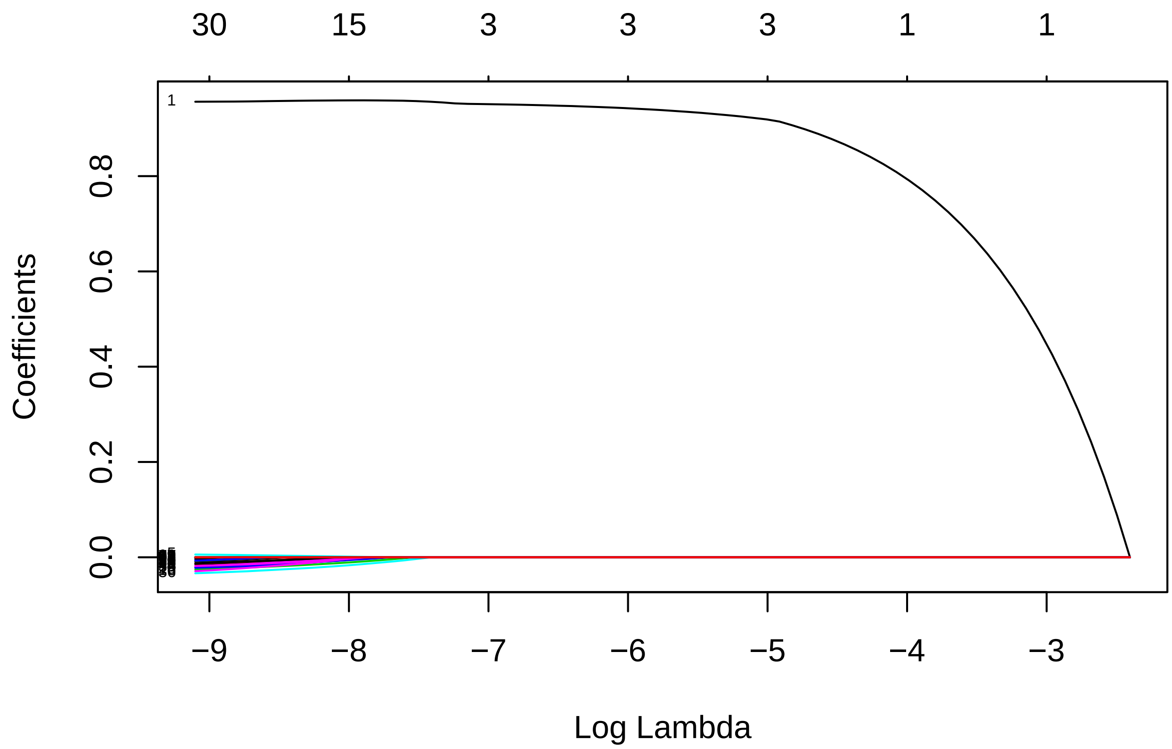
```
# outfile = file(filename)

#construct nn input data

names(volatility_sp500) = names(sp500[30:length(sp500)])

col_names = "target"

for(i in 30:length(volatility_sp500)){

  outString = ""
  for(j in 0:(window_size-2)){
    outString = paste(outString, volatility_sp500[i-j-1], sp500[i+j+1], vix[i+j], sp500_returns[i+j] ,se
    if(i==30){
      col_names = paste(col_names, paste(",volatilityL(",(j+1),")",sep=""), paste(",sp500L(",(i-(j+1)),
                      paste(",vixL(",(i-(j+1)),")",sep=""), paste(",sp500returnsL(",(i-(j+1)),")",sep=""))
    }
  }

  #get presidential info
  month_string = months(as.Date(names(volatility_sp500[i])))
  month_int = DateToInt(month_string)
  year = format(as.Date(names(volatility_sp500[i])),format="%Y")

  #current month's avg approval rating
  approval_avg = subset(approval_average$Approval,approval_average$Month==month_string&approval_average$
  disapproval_avg = subset(disaproval_average$Disapproval,disaproval_average$Month==month_string&disapro
  unknown_avg = subset(unknown_average$Unsure.No_Data,unknown_average$Month==month_string&unknown_averag

  #move back a month
  if(length(approval_avg)==0){
    tempM = month_int-1
    tempY = year
    if(tempM<0){
      tempM=12
      tempY=tempY-1
    }
    tempM_string = IntToDate(tempM)
    approval_avg = subset(approval_average$Approval,approval_average$Month==tempM_string&approval_averag
    disapproval_avg = subset(disaproval_average$Disapproval,disaproval_average$Month==tempM_string&disap
    unknown_avg = subset(unknown_average$Unsure.No_Data,unknown_average$Month==tempM_string&unknown_aver
  }

  if(i==30){
    col_names=paste(col_names,",pres_approv_avg,pres_disapprov_avg,pres_unknown_avg",sep="")
  }

  outString = paste(outString,approval_avg,disapproval_avg,unknown_avg,sep=",")

  #remove first comma
```

```
  outString = substring(outString,2,nchar(outString))
  #add on output
  outString = paste(volatility_sp500[i],outString,sep=",")

  #remove first comma
  outString = substring(outString,2,nchar(outString))

  #write to outfile
  if(i==30){
    cat(col_names,file=filename,append=TRUE,sep="\n")
  }
  cat(outString,file=filename,append=TRUE,sep="\n")

}
```

```
full_data = read.csv("out.csv",header = TRUE)
rownames(full_data) = names(volatility_sp500[30:length(volatility_sp500)])
# head(full_data)


#randomized vs timeseries?
#create training and test sets
## 66% of the sample size
smp_size <- floor(.66* nrow(full_data))

## set the seed to make your partition reproductible
set.seed(123)
train_ind <- sample(seq_len(nrow(full_data)), size = smp_size)

train <- full_data[train_ind, ]
test <- full_data[-train_ind, ]


#lasso
x <- model.matrix( ~ .-1, train[ , -1])
y <- data.matrix(train[, 1])

model.lasso <- cv.glmnet(x, y, family='gaussian', alpha=1, parallel=TRUE, standardize=TRUE)
plot(model.lasso)
```




```r
plot(model.lasso$glmnet.fit, xvar="lambda", label=TRUE)
```




```
model.lasso$lambda.min
```

```
## [1] 0.0001343292
```

```
model.lasso$lambda.1se
```

```
## [1] 0.0007168748
```

```
coef(model.lasso, s=model.lasso$lambda.min)
```

```
## 120 x 1 sparse Matrix of class "dgCMatrix"
##                                     1
## (Intercept)        -0.00177926258424
## volatilityL.1.      0.95642637046791
## sp500L.29.          0.00000009205663
## vixL.29.           -0.00033699922601
## sp500returnsL.29.  -0.01914870432711
## volatilityL.2.      0.00519352578274
## sp500L.28.          .
## vixL.28.            .
## sp500returnsL.28.  -0.01421590522341
## volatilityL.3.      .
## sp500L.27.          .
## vixL.27.            .
## sp500returnsL.27.  -0.01233767031519
## volatilityL.4.      .
## sp500L.26.          .
## vixL.26.            .
```

```
## sp500returnsL.26.  -0.00673039815353
## volatilityL.5.      .
## sp500L.25.          .
## vixL.25.            .
## sp500returnsL.25.  -0.02361380950902
## volatilityL.6.      .
## sp500L.24.          .
## vixL.24.            .
## sp500returnsL.24.   .
## volatilityL.7.      .
## sp500L.23.          .
## vixL.23.            .
## sp500returnsL.23.   .
## volatilityL.8.      .
## sp500L.22.          .
## vixL.22.            .
## sp500returnsL.22.  -0.02038680277062
## volatilityL.9.      .
## sp500L.21.          .
## vixL.21.            .
## sp500returnsL.21.  -0.03161976978960
## volatilityL.10.     .
## sp500L.20.          .
## vixL.20.            .
## sp500returnsL.20.   .
## volatilityL.11.     .
## sp500L.19.          .
## vixL.19.            .
## sp500returnsL.19.  -0.01673257894259
## volatilityL.12.     .
## sp500L.18.          .
## vixL.18.            .
## sp500returnsL.18.   .
## volatilityL.13.     .
## sp500L.17.          .
## vixL.17.            .
## sp500returnsL.17.   .
## volatilityL.14.     .
## sp500L.16.          .
## vixL.16.            .
## sp500returnsL.16.  -0.00676565273081
## volatilityL.15.     .
## sp500L.15.          .
## vixL.15.            .
## sp500returnsL.15.  -0.01107429841225
## volatilityL.16.     .
## sp500L.14.          .
## vixL.14.            .
## sp500returnsL.14.   .
## volatilityL.17.     .
## sp500L.13.          .
## vixL.13.            .
## sp500returnsL.13.   .
## volatilityL.18.     .
```

```
## sp500L.12.          .
## vixL.12.            .
## sp500returnsL.12.  -0.00177712945226
## volatilityL.19.     .
## sp500L.11.          .
## vixL.11.            .
## sp500returnsL.11.   .
## volatilityL.20.     .
## sp500L.10.          .
## vixL.10.            .
## sp500returnsL.10.   .
## volatilityL.21.    -0.00086768997266
## sp500L.9.           .
## vixL.9.             .
## sp500returnsL.9.   -0.01078157648860
## volatilityL.22.     .
## sp500L.8.           .
## vixL.8.             0.00000041016639
## sp500returnsL.8.    .
## volatilityL.23.     .
## sp500L.7.           .
## vixL.7.             .
## sp500returnsL.7.   -0.00503287650097
## volatilityL.24.     .
## sp500L.6.           .
## vixL.6.             .
## sp500returnsL.6.   -0.00188410618543
## volatilityL.25.     .
## sp500L.5.           .
## vixL.5.             0.00000480218178
## sp500returnsL.5.    .
## volatilityL.26.     .
## sp500L.4.           .
## vixL.4.             .
## sp500returnsL.4.   -0.00094712880775
## volatilityL.27.     .
## sp500L.3.           .
## vixL.3.             0.00001625230854
## sp500returnsL.3.    .
## volatilityL.28.     .
## sp500L.2.           .
## vixL.2.             0.00026993667035
## sp500returnsL.2.   -0.00494594567035
## volatilityL.29.     .
## sp500L.1.           .
## vixL.1.             0.00041424595211
## sp500returnsL.1.   -0.02605975927089
## pres_approv_avg     .
## pres_disapprov_avg  0.00001709317943
## pres_unknown_avg   -0.00000222178890
```

```r
#decision tree
tree_fit <- randomForest(target ~ .,   data=train)
print(tree_fit) # view results
```

```
##
## Call:
##  randomForest(formula = target ~ ., data = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 39
##
##           Mean of squared residuals: 0.00006493249
##                     % Var explained: 99.21
```

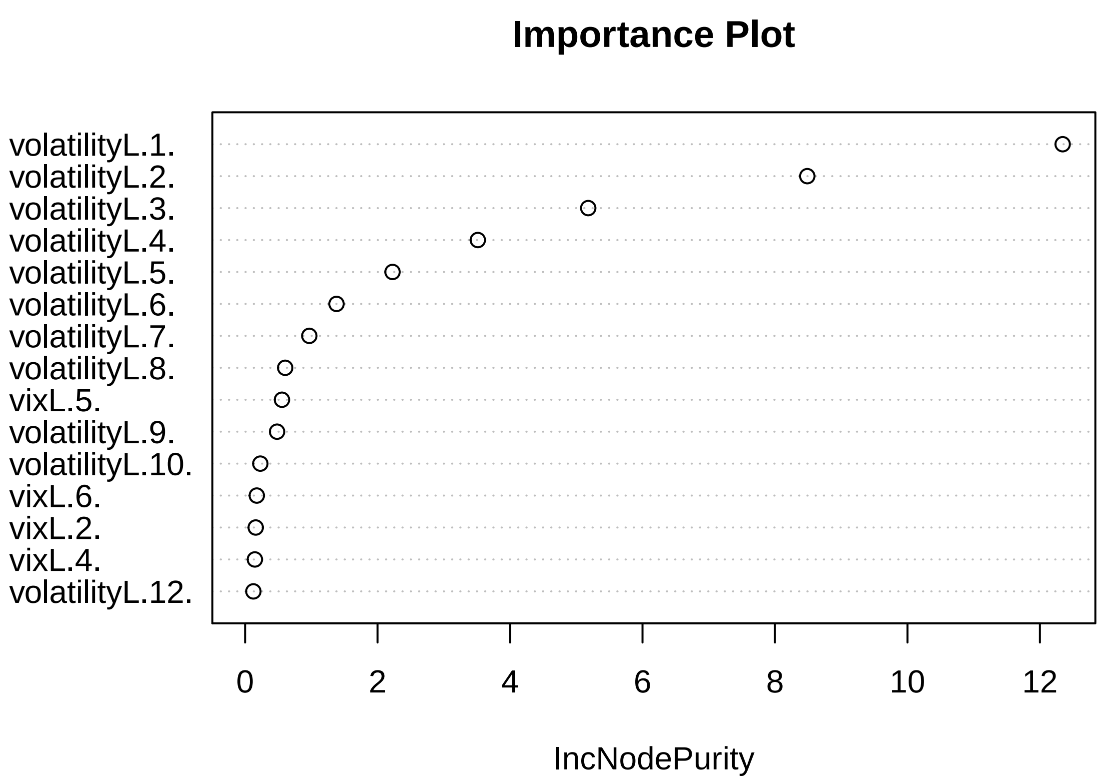
```r
importance(tree_fit) # importance of each predictor
```

```
##                    IncNodePurity
## volatilityL.1.      12.343363420
## sp500L.29.           0.002907938
## vixL.29.             0.006573063
## sp500returnsL.29.    0.004257896
## volatilityL.2.       8.487701435
## sp500L.28.           0.002222855
## vixL.28.             0.004417886
## sp500returnsL.28.    0.002843285
## volatilityL.3.       5.180417882
## sp500L.27.           0.002614262
## vixL.27.             0.002789618
## sp500returnsL.27.    0.002653648
## volatilityL.4.       3.513330378
## sp500L.26.           0.002372502
## vixL.26.             0.002744422
## sp500returnsL.26.    0.003100866
## volatilityL.5.       2.225965992
## sp500L.25.           0.002535472
## vixL.25.             0.002785621
## sp500returnsL.25.    0.003015163
## volatilityL.6.       1.380219458
## sp500L.24.           0.001873252
## vixL.24.             0.002159113
## sp500returnsL.24.    0.002593877
## volatilityL.7.       0.969617526
## sp500L.23.           0.002652445
## vixL.23.             0.003389400
## sp500returnsL.23.    0.002707145
## volatilityL.8.       0.605186111
## sp500L.22.           0.001771515
## vixL.22.             0.003271557
## sp500returnsL.22.    0.002591159
## volatilityL.9.       0.482546926
## sp500L.21.           0.002317024
## vixL.21.             0.003169917
## sp500returnsL.21.    0.002099969
## volatilityL.10.      0.229817703
## sp500L.20.           0.002603434
## vixL.20.             0.002931868
## sp500returnsL.20.    0.003093127
## volatilityL.11.      0.116406931
## sp500L.19.           0.002852469
```

```
## vixL.19.             0.002398153
## sp500returnsL.19.    0.002329112
## volatilityL.12.      0.124974883
## sp500L.18.           0.002382850
## vixL.18.             0.002465895
## sp500returnsL.18.    0.002440654
## volatilityL.13.      0.006951139
## sp500L.17.           0.002721866
## vixL.17.             0.002942837
## sp500returnsL.17.    0.002682080
## volatilityL.14.      0.003214515
## sp500L.16.           0.002514695
## vixL.16.             0.003574316
## sp500returnsL.16.    0.002577276
## volatilityL.15.      0.003030926
## sp500L.15.           0.003338094
## vixL.15.             0.014663160
## sp500returnsL.15.    0.002776402
## volatilityL.16.      0.002420446
## sp500L.14.           0.002460806
## vixL.14.             0.005673242
## sp500returnsL.14.    0.002560000
## volatilityL.17.      0.003215437
## sp500L.13.           0.001592148
## vixL.13.             0.003503700
## sp500returnsL.13.    0.002489128
## volatilityL.18.      0.002689306
## sp500L.12.           0.002230333
## vixL.12.             0.042456389
## sp500returnsL.12.    0.002603938
## volatilityL.19.      0.002741852
## sp500L.11.           0.001413582
## vixL.11.             0.018884300
## sp500returnsL.11.    0.002584080
## volatilityL.20.      0.003230590
## sp500L.10.           0.001625973
## vixL.10.             0.049262986
## sp500returnsL.10.    0.002875989
## volatilityL.21.      0.003462163
## sp500L.9.            0.001282256
## vixL.9.              0.008516229
## sp500returnsL.9.     0.002484239
## volatilityL.22.      0.002800285
## sp500L.8.            0.001386058
## vixL.8.              0.006555189
## sp500returnsL.8.     0.002528449
## volatilityL.23.      0.002438070
## sp500L.7.            0.002016509
## vixL.7.              0.084918480
## sp500returnsL.7.     0.003174561
## volatilityL.24.      0.003270305
## sp500L.6.            0.001472974
## vixL.6.              0.176217893
## sp500returnsL.6.     0.003071197
```

```
## volatilityL.25.       0.003632438
## sp500L.5.             0.001725917
## vixL.5.               0.556099244
## sp500returnsL.5.      0.004326901
## volatilityL.26.       0.003498375
## sp500L.4.             0.001324433
## vixL.4.               0.147317036
## sp500returnsL.4.      0.004967149
## volatilityL.27.       0.003434929
## sp500L.3.             0.002228060
## vixL.3.               0.090204766
## sp500returnsL.3.      0.011110958
## volatilityL.28.       0.005022498
## sp500L.2.             0.002419254
## vixL.2.               0.160189422
## sp500returnsL.2.      0.011909243
## volatilityL.29.       0.008742973
## sp500L.1.             0.002289866
## vixL.1.               0.045235116
## sp500returnsL.1.      0.043519283
## pres_approv_avg       0.013051748
## pres_disapprov_avg    0.004509175
## pres_unknown_avg      0.006070563
```

```r
varImpPlot(tree_fit, main = "Importance Plot", n.var = 15)
```



```r
test_x = test[,-1]
test_x_matrix = model.matrix( ~ .-1, test[,-1])
lasso_test = predict(model.lasso, newx=test_x_matrix,type="link")
tree_test = predict(tree_fit, newdata=test_x)


#mape
lasso_mape = mape(test$target,lasso_test)
lasso_mape
```

```
## [1] 2.743576
```

```r
tree_mape = mape(test$target,tree_test)
tree_mape
```

```
## [1] 2.73813
```

```r
set.seed(1)

#normalize data

maxs <- apply(full_data, 2, max)
mins <- apply(full_data, 2, min)

scaled <- as.data.frame(scale(full_data, center = mins, scale = maxs - mins))


train_ <- scaled[train_ind,]
test_ <- scaled[-train_ind,]


n <- names(train_)
f <- as.formula(paste("target ~", paste(n[!n %in% "target"], collapse = " + ")))
nn <- neuralnet(f,data=train_,hidden=c(100,70,60,50,40,30,20),linear.output=T)

pr.nn <- compute(nn,test_[,2:ncol(test_)])


pr.nn_ <- pr.nn$net.result*(max(full_data$target)-min(full_data$target))+min(full_data$target)
test.r <- (test_$target)*(max(full_data$target)-min(full_data$target))+min(full_data$target)

MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)


mape_nn = mape(test.r,pr.nn_)
mape_nn
```

```
## [1] 3.655849069
```

```r
#load and massage data
recession = read.csv("USRECD.csv")
interest = read.csv("DFF.csv")
rownames(recession) = as.Date(recession$DATE)
recession = recession[,2, drop=FALSE]
rownames(interest) = as.Date(interest$DATE)
interest = interest[,2, drop=FALSE]
```

```r
#momentums
interest_momentum = apply( interest , 2 , diff )
head(interest_momentum)
```

```
##              DFF
## 1990-02-12  0.00
## 1990-02-13  0.00
## 1990-02-14  0.00
## 1990-02-15  0.11
## 1990-02-16 -0.13
## 1990-02-17  0.00
```

```r
full_data_update <- merge(full_data, recession, by=0, all=TRUE)
full_data_update = full_data_update[ , !(names(full_data_update) %in% c("Row.names"))]
full_data_update = na.omit(full_data_update)
rownames(full_data_update) = names(volatility_sp500[30:length(volatility_sp500)])
full_data_update <- merge(full_data_update, interest, by=0, all=TRUE)
full_data_update = full_data_update[ , !(names(full_data_update) %in% c("Row.names"))]
full_data_update = na.omit(full_data_update)
rownames(full_data_update) = names(volatility_sp500[30:length(volatility_sp500)])
full_data_update <- merge(full_data_update, interest_momentum, by=0, all=TRUE)
full_data_update = full_data_update[ , !(names(full_data_update) %in% c("Row.names"))]
full_data_update = na.omit(full_data_update)
rownames(full_data_update) = names(volatility_sp500[30:length(volatility_sp500)])
head(full_data_update)
```

```
##                   target volatilityL.1. sp500L.29.  vixL.29.
## 1990-03-26 0.1177644377   0.1224088945 357.010010 24.379999
## 1990-03-27 0.1217657667   0.1177644377 358.100006 23.760000
## 1990-03-28 0.1216352031   0.1217657667 361.230011 22.049999
## 1990-03-29 0.1200872886   0.1216352031 358.989990 19.709999
## 1990-03-30 0.1185685714   0.1200872886 353.910004 20.780001
## 1990-04-02 0.1098959417   0.1185685714 353.570007 22.780001
##            sp500returnsL.29. volatilityL.2. sp500L.28.  vixL.28.
## 1990-03-26   0.0031470256337   0.1219198309 358.100006 23.760000
## 1990-03-27   0.0030531244768   0.1224088945 361.230011 22.049999
## 1990-03-28   0.0087405890744   0.1177644377 358.989990 19.709999
## 1990-03-29  -0.0062010932973   0.1217657667 353.910004 20.780001
## 1990-03-30  -0.0141507733962   0.1216352031 353.570007 22.780001
## 1990-04-02  -0.0009606877346   0.1200872886 351.480011 23.889999
##            sp500returnsL.28. volatilityL.3. sp500L.27.  vixL.27.
## 1990-03-26   0.0030531244768   0.1160746527 361.230011 22.049999
## 1990-03-27   0.0087405890744   0.1219198309 358.989990 19.709999
## 1990-03-28  -0.0062010932973   0.1224088945 353.910004 20.780001
## 1990-03-29  -0.0141507733962   0.1177644377 353.570007 22.780001
## 1990-03-30  -0.0009606877346   0.1217657667 351.480011 23.889999
## 1990-04-02  -0.0059111235643   0.1216352031 349.899994 22.540001
##            sp500returnsL.27. volatilityL.4. sp500L.26.  vixL.26.
## 1990-03-26   0.0087405890744   0.1196295907 358.989990 19.709999
## 1990-03-27  -0.0062010932973   0.1160746527 353.910004 20.780001
## 1990-03-28  -0.0141507733962   0.1219198309 353.570007 22.780001
## 1990-03-29  -0.0009606877346   0.1224088945 351.480011 23.889999
## 1990-03-30  -0.0059111235643   0.1177644377 349.899994 22.540001
```

```
## 1990-04-02  -0.0044953253401   0.1217657667 354.880005 23.690001
##            sp500returnsL.26. volatilityL.5. sp500L.25.  vixL.25.
## 1990-03-26  -0.0062010932973   0.1199812362 353.910004 20.780001
## 1990-03-27  -0.0141507733962   0.1196295907 353.570007 22.780001
## 1990-03-28  -0.0009606877346   0.1160746527 351.480011 23.889999
## 1990-03-29  -0.0059111235643   0.1219198309 349.899994 22.540001
## 1990-03-30  -0.0044953253401   0.1224088945 354.880005 23.690001
## 1990-04-02   0.0142326695782   0.1177644377 356.660004 23.559999
##            sp500returnsL.25. volatilityL.6. sp500L.24.  vixL.24.
## 1990-03-26  -0.0141507733962   0.1197046264 353.570007 22.780001
## 1990-03-27  -0.0009606877346   0.1199812362 351.480011 23.889999
## 1990-03-28  -0.0059111235643   0.1196295907 349.899994 22.540001
## 1990-03-29  -0.0044953253401   0.1160746527 354.880005 23.690001
## 1990-03-30   0.0142326695782   0.1219198309 356.660004 23.559999
## 1990-04-02   0.0050157770934   0.1224088945 358.500000 22.690001
##            sp500returnsL.24. volatilityL.7. sp500L.23.  vixL.23.
## 1990-03-26  -0.0009606877346   0.1168537736 351.480011 23.889999
## 1990-03-27  -0.0059111235643   0.1197046264 349.899994 22.540001
## 1990-03-28  -0.0044953253401   0.1199812362 354.880005 23.690001
## 1990-03-29   0.0142326695782   0.1196295907 356.660004 23.559999
## 1990-03-30   0.0050157770934   0.1160746527 358.500000 22.690001
## 1990-04-02   0.0051589636611   0.1219198309 359.459991 21.990000
##            sp500returnsL.23. volatilityL.8. sp500L.22.  vixL.22.
## 1990-03-26   -0.005911123564   0.1167314109 349.899994 22.540001
## 1990-03-27   -0.004495325340   0.1168537736 354.880005 23.690001
## 1990-03-28    0.014232669578   0.1197046264 356.660004 23.559999
## 1990-03-29    0.005015777093   0.1199812362 358.500000 22.690001
## 1990-03-30    0.005158963661   0.1196295907 359.459991 21.990000
## 1990-04-02    0.002677799163   0.1160746527 362.489990 21.900000
##            sp500returnsL.22. volatilityL.9. sp500L.21.  vixL.21.
## 1990-03-26   -0.004495325340   0.1281870545 354.880005 23.690001
## 1990-03-27    0.014232669578   0.1167314109 356.660004 23.559999
## 1990-03-28    0.005015777093   0.1168537736 358.500000 22.690001
## 1990-03-29    0.005158963661   0.1197046264 359.459991 21.990000
## 1990-03-30    0.002677799163   0.1199812362 362.489990 21.900000
## 1990-04-02    0.008429308062   0.1196295907 360.679993 21.340000
##            sp500returnsL.21. volatilityL.10. sp500L.20.  vixL.20.
## 1990-03-26    0.014232669578    0.1273584625 356.660004 23.559999
## 1990-03-27    0.005015777093    0.1281870545 358.500000 22.690001
## 1990-03-28    0.005158963661    0.1167314109 359.459991 21.990000
## 1990-03-29    0.002677799163    0.1168537736 362.489990 21.900000
## 1990-03-30    0.008429308062    0.1197046264 360.679993 21.340000
## 1990-04-02   -0.004993233055    0.1199812362 365.239990 22.030001
##            sp500returnsL.20. volatilityL.11. sp500L.19.  vixL.19.
## 1990-03-26    0.005015777093    0.1277096691 358.500000 22.690001
## 1990-03-27    0.005158963661    0.1273584625 359.459991 21.990000
## 1990-03-28    0.002677799163    0.1281870545 362.489990 21.900000
## 1990-03-29    0.008429308062    0.1167314109 360.679993 21.340000
## 1990-03-30   -0.004993233055    0.1168537736 365.239990 22.030001
## 1990-04-02    0.012642777777    0.1197046264 364.190002 20.549999
##            sp500returnsL.19. volatilityL.12. sp500L.18.  vixL.18.
## 1990-03-26    0.005158963661    0.1255476068 359.459991 21.990000
## 1990-03-27    0.002677799163    0.1277096691 362.489990 21.900000
## 1990-03-28    0.008429308062    0.1273584625 360.679993 21.340000
```

```
## 1990-03-29   -0.004993233055    0.1281870545 365.239990 22.030001
## 1990-03-30    0.012642777777    0.1167314109 364.190002 20.549999
## 1990-04-02   -0.002874789258    0.1168537736 367.790009 19.100000
##            sp500returnsL.18. volatilityL.13. sp500L.17.  vixL.17.
## 1990-03-26    0.002677799163    0.1237234717 362.489990 21.900000
## 1990-03-27    0.008429308062    0.1255476068 360.679993 21.340000
## 1990-03-28   -0.004993233055    0.1277096691 365.239990 22.030001
## 1990-03-29    0.012642777777    0.1273584625 364.190002 20.549999
## 1990-03-30   -0.002874789258    0.1281870545 367.790009 19.100000
## 1990-04-02    0.009884969330    0.1167314109 365.420013 19.740000
##            sp500returnsL.17. volatilityL.14. sp500L.16.  vixL.16.
## 1990-03-26    0.008429308062    0.1426638766 360.679993 21.340000
## 1990-03-27   -0.004993233055    0.1237234717 365.239990 22.030001
## 1990-03-28    0.012642777777    0.1255476068 364.190002 20.549999
## 1990-03-29   -0.002874789258    0.1277096691 367.790009 19.100000
## 1990-03-30    0.009884969330    0.1273584625 365.420013 19.740000
## 1990-04-02   -0.006443883580    0.1281870545 366.239990 20.299999
##            sp500returnsL.16. volatilityL.15. sp500L.15.  vixL.15.
## 1990-03-26   -0.004993233055    0.1383098721 365.239990 22.030001
## 1990-03-27    0.012642777777    0.1426638766 364.190002 20.549999
## 1990-03-28   -0.002874789258    0.1237234717 367.790009 19.100000
## 1990-03-29    0.009884969330    0.1255476068 365.420013 19.740000
## 1990-03-30   -0.006443883580    0.1277096691 366.239990 20.299999
## 1990-04-02    0.002243930192    0.1273584625 363.369995 20.070000
##            sp500returnsL.15. volatilityL.16. sp500L.14.  vixL.14.
## 1990-03-26    0.012642777777    0.1590434266 364.190002 20.549999
## 1990-03-27   -0.002874789258    0.1383098721 367.790009 19.100000
## 1990-03-28    0.009884969330    0.1426638766 365.420013 19.740000
## 1990-03-29   -0.006443883580    0.1237234717 366.239990 20.299999
## 1990-03-30    0.002243930192    0.1255476068 363.369995 20.070000
## 1990-04-02   -0.007836377999    0.1277096691 364.320007 21.049999
##            sp500returnsL.14. volatilityL.17. sp500L.13.  vixL.13.
## 1990-03-26   -0.002874789258    0.1571283221 367.790009 19.100000
## 1990-03-27    0.009884969330    0.1590434266 365.420013 19.740000
## 1990-03-28   -0.006443883580    0.1383098721 366.239990 20.299999
## 1990-03-29    0.002243930192    0.1426638766 363.369995 20.070000
## 1990-03-30   -0.007836377999    0.1237234717 364.320007 21.049999
## 1990-04-02    0.002614448119    0.1255476068 365.630005 19.650000
##            sp500returnsL.13. volatilityL.18. sp500L.12.  vixL.12.
## 1990-03-26    0.009884969330    0.1570671694 365.420013 19.740000
## 1990-03-27   -0.006443883580    0.1571283221 366.239990 20.299999
## 1990-03-28    0.002243930192    0.1590434266 363.369995 20.070000
## 1990-03-29   -0.007836377999    0.1383098721 364.320007 21.049999
## 1990-03-30    0.002614448119    0.1426638766 365.630005 19.650000
## 1990-04-02    0.003595734450    0.1237234717 369.790009 18.809999
##            sp500returnsL.12. volatilityL.19. sp500L.11.  vixL.11.
## 1990-03-26   -0.006443883580    0.1585610246 366.239990 20.299999
## 1990-03-27    0.002243930192    0.1570671694 363.369995 20.070000
## 1990-03-28   -0.007836377999    0.1571283221 364.320007 21.049999
## 1990-03-29    0.002614448119    0.1590434266 365.630005 19.650000
## 1990-03-30    0.003595734450    0.1383098721 369.790009 18.809999
## 1990-04-02    0.011377632971    0.1426638766 371.559998 17.620001
##            sp500returnsL.11. volatilityL.20. sp500L.10.  vixL.10.
## 1990-03-26    0.002243930192    0.1616869679 363.369995 20.070000
```

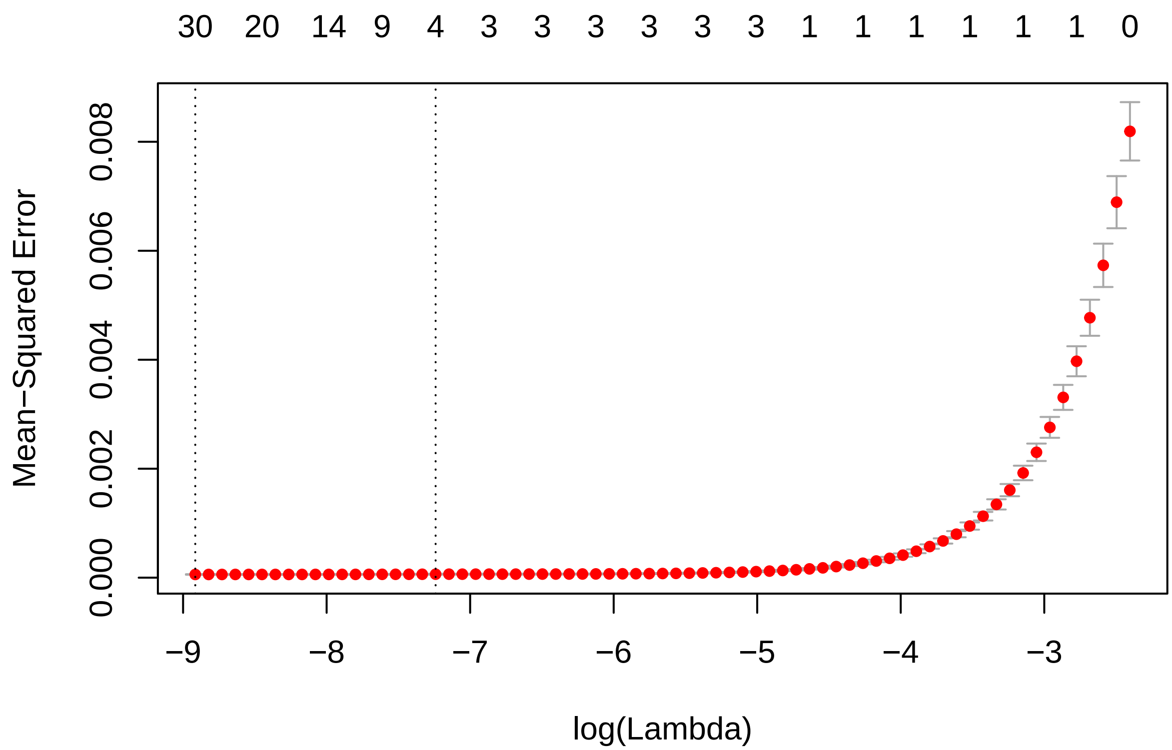
```
## 1990-03-27   -0.007836377999    0.1585610246 364.320007 21.049999
## 1990-03-28    0.002614448119    0.1570671694 365.630005 19.650000
## 1990-03-29    0.003595734450    0.1571283221 369.790009 18.809999
## 1990-03-30    0.011377632971    0.1590434266 371.559998 17.620001
## 1990-04-02    0.004786470583    0.1383098721 369.470001 18.290001
##             sp500returnsL.10. volatilityL.21.  sp500L.9.   vixL.9.
## 1990-03-26   -0.007836377999    0.1565938252 364.320007 21.049999
## 1990-03-27    0.002614448119    0.1616869679 365.630005 19.650000
## 1990-03-28    0.003595734450    0.1585610246 369.790009 18.809999
## 1990-03-29    0.011377632971    0.1570671694 371.559998 17.620001
## 1990-03-30    0.004786470583    0.1571283221 369.470001 18.290001
## 1990-04-02   -0.005624924672    0.1590434266 367.489990 19.059999
##             sp500returnsL.9. volatilityL.22.  sp500L.8.   vixL.8.
## 1990-03-26   0.002614448119    0.1713646351 365.630005 19.650000
## 1990-03-27   0.003595734450    0.1565938252 369.790009 18.809999
## 1990-03-28   0.011377632971    0.1616869679 371.559998 17.620001
## 1990-03-29   0.004786470583    0.1585610246 369.470001 18.290001
## 1990-03-30  -0.005624924672    0.1570671694 367.489990 19.059999
## 1990-04-02  -0.005359057554    0.1571283221 363.109985 20.100000
##             sp500returnsL.8. volatilityL.23.  sp500L.7.   vixL.7.
## 1990-03-26   0.003595734450    0.1717750792 369.790009 18.809999
## 1990-03-27   0.011377632971    0.1713646351 371.559998 17.620001
## 1990-03-28   0.004786470583    0.1565938252 369.470001 18.290001
## 1990-03-29  -0.005624924672    0.1616869679 367.489990 19.059999
## 1990-03-30  -0.005359057554    0.1585610246 363.109985 20.100000
## 1990-04-02  -0.011918705595    0.1570671694 364.769989 22.740000
##             sp500returnsL.7. volatilityL.24.  sp500L.6.   vixL.6.
## 1990-03-26   0.011377632971    0.1723317452 371.559998 17.620001
## 1990-03-27   0.004786470583    0.1717750792 369.470001 18.290001
## 1990-03-28  -0.005624924672    0.1713646351 367.489990 19.059999
## 1990-03-29  -0.005359057554    0.1565938252 363.109985 20.100000
## 1990-03-30  -0.011918705595    0.1616869679 364.769989 22.740000
## 1990-04-02   0.004571628621    0.1585610246 365.440002 20.459999
##             sp500returnsL.6. volatilityL.25.  sp500L.5.   vixL.5.
## 1990-03-26   0.004786470583    0.1709008106 369.470001 18.290001
## 1990-03-27  -0.005624924672    0.1723317452 367.489990 19.059999
## 1990-03-28  -0.005359057554    0.1717750792 363.109985 20.100000
## 1990-03-29  -0.011918705595    0.1713646351 364.769989 22.740000
## 1990-03-30   0.004571628621    0.1565938252 365.440002 20.459999
## 1990-04-02   0.001836809552    0.1616869679 369.640015 19.590000
##             sp500returnsL.5. volatilityL.26.  sp500L.4.   vixL.4.
## 1990-03-26  -0.005624924672    0.1714653754 367.489990 19.059999
## 1990-03-27  -0.005359057554    0.1709008106 363.109985 20.100000
## 1990-03-28  -0.011918705595    0.1723317452 364.769989 22.740000
## 1990-03-29   0.004571628621    0.1717750792 365.440002 20.459999
## 1990-03-30   0.001836809552    0.1713646351 369.640015 19.590000
## 1990-04-02   0.011493030257    0.1565938252 370.190002 21.010000
##             sp500returnsL.4. volatilityL.27.  sp500L.3.   vixL.3.
## 1990-03-26  -0.005359057554    0.1700886704 363.109985 20.100000
## 1990-03-27  -0.011918705595    0.1714653754 364.769989 22.740000
## 1990-03-28   0.004571628621    0.1709008106 365.440002 20.459999
## 1990-03-29   0.001836809552    0.1723317452 369.640015 19.590000
## 1990-03-30   0.011493030257    0.1717750792 370.190002 21.010000
## 1990-04-02   0.001487898977    0.1713646351 368.880005 19.770000
```

```
##            sp500returnsL.3. volatilityL.28.  sp500L.2.   vixL.2.
## 1990-03-26  -0.011918705595    0.1701857037 364.769989 22.740000
## 1990-03-27   0.004571628621    0.1700886704 365.440002 20.459999
## 1990-03-28   0.001836809552    0.1714653754 369.640015 19.590000
## 1990-03-29   0.011493030257    0.1709008106 370.190002 21.010000
## 1990-03-30   0.001487898977    0.1723317452 368.880005 19.770000
## 1990-04-02  -0.003538715235    0.1717750792 368.000000 18.459999
##            sp500returnsL.2. volatilityL.29.  sp500L.1.   vixL.1.
## 1990-03-26   0.004571628621    0.1692473886 365.440002 20.459999
## 1990-03-27   0.001836809552    0.1701857037 369.640015 19.590000
## 1990-03-28   0.011493030257    0.1700886704 370.190002 21.010000
## 1990-03-29   0.001487898977    0.1714653754 368.880005 19.770000
## 1990-03-30  -0.003538715235    0.1709008106 368.000000 18.459999
## 1990-04-02  -0.002385613175    0.1723317452 366.720001 19.730000
##            sp500returnsL.1. pres_approv_avg pres_disapprov_avg
## 1990-03-26   0.001836809552              71               17.0
## 1990-03-27   0.011493030257              71               17.0
## 1990-03-28   0.001487898977              71               17.0
## 1990-03-29  -0.003538715235              71               17.0
## 1990-03-30  -0.002385613175              71               17.0
## 1990-04-02  -0.003478258152              69               15.5
##            pres_unknown_avg USRECD DFF.x DFF.y
## 1990-03-26      10.66666667      0  8.25  0.00
## 1990-03-27      10.66666667      0  8.26  0.01
## 1990-03-28      10.66666667      0  8.30  0.04
## 1990-03-29      10.66666667      0  8.37  0.07
## 1990-03-30      10.66666667      0  8.30 -0.07
## 1990-04-02      14.50000000      0  8.34  0.04
```

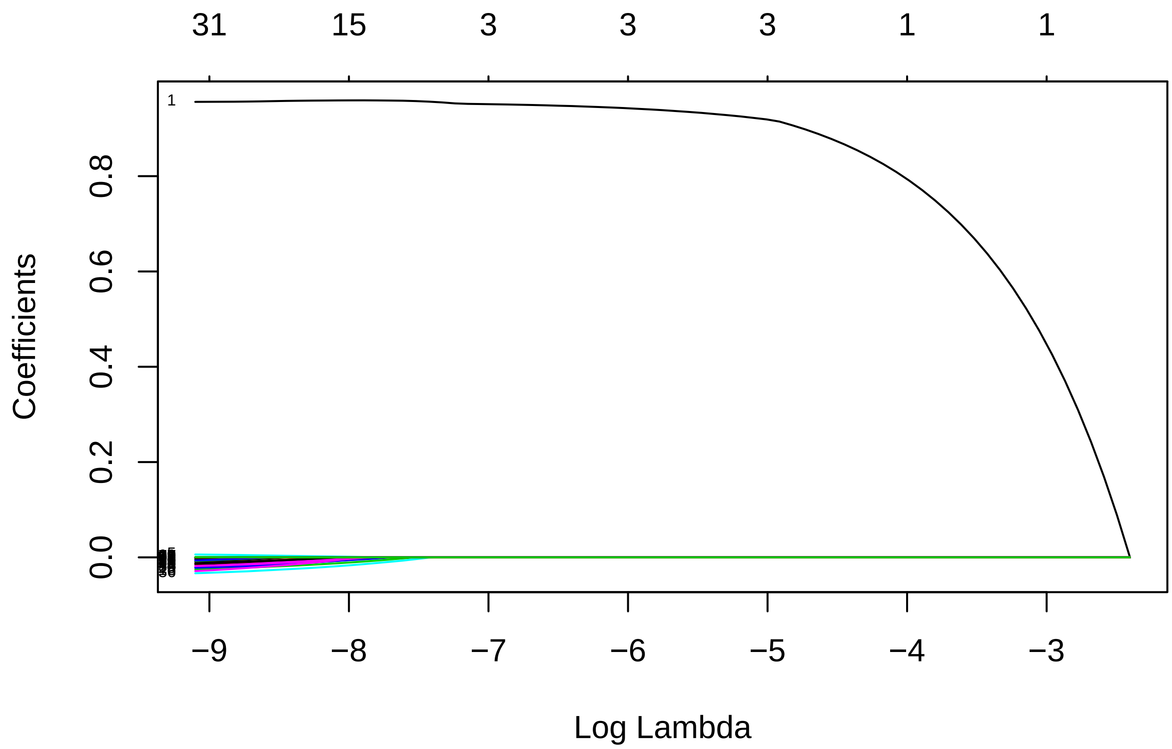
```r
#create training and test sets
## 66% of the sample size
smp_size <- floor(.66* nrow(full_data_update))

## set the seed to make your partition reproductible
set.seed(123)
train_ind <- sample(seq_len(nrow(full_data_update)), size = smp_size)

train <- full_data_update[train_ind, ]
test <- full_data_update[-train_ind, ]


maxs <- apply(full_data_update, 2, max)
mins <- apply(full_data_update, 2, min)

scaled <- as.data.frame(scale(full_data_update, center = mins, scale = maxs - mins))


train_ <- scaled[train_ind,]
test_ <- scaled[-train_ind,]
```

```r
#lasso
x <- model.matrix( ~ .-1, train[ , -1])
y <- data.matrix(train[, 1])

model.lasso <- cv.glmnet(x, y, family='gaussian', alpha=1, parallel=TRUE, standardize=TRUE)
```

```r
plot(model.lasso)
```




```r
plot(model.lasso$glmnet.fit, xvar="lambda", label=TRUE)
```




```
model.lasso$lambda.min
```

```
## [1] 0.0001343292408
```

```
model.lasso$lambda.1se
```

```
## [1] 0.0007168747562
```

```
coef(model.lasso, s=model.lasso$lambda.min)
```

```
## 123 x 1 sparse Matrix of class "dgCMatrix"
##                                      1
## (Intercept)        -0.00164671607885972
## volatilityL.1.      0.95614968436066483
## sp500L.29.          0.00000009677973602
## vixL.29.           -0.00033914485317262
## sp500returnsL.29.  -0.01904188032415464
## volatilityL.2.      0.00542667340453224
## sp500L.28.          .
## vixL.28.            .
## sp500returnsL.28.  -0.01402775844882055
## volatilityL.3.      .
## sp500L.27.          .
## vixL.27.            .
## sp500returnsL.27.  -0.01227179235823170
## volatilityL.4.      .
## sp500L.26.          .
## vixL.26.            .
```

```
## sp500returnsL.26.  -0.00665227197016906
## volatilityL.5.      .
## sp500L.25.          .
## vixL.25.            .
## sp500returnsL.25.  -0.02338501556407536
## volatilityL.6.      .
## sp500L.24.          .
## vixL.24.            .
## sp500returnsL.24.   .
## volatilityL.7.      .
## sp500L.23.          .
## vixL.23.            .
## sp500returnsL.23.   .
## volatilityL.8.      .
## sp500L.22.          .
## vixL.22.            .
## sp500returnsL.22.  -0.02020632589900393
## volatilityL.9.      .
## sp500L.21.          .
## vixL.21.            .
## sp500returnsL.21.  -0.03143087809172982
## volatilityL.10.     .
## sp500L.20.          .
## vixL.20.            .
## sp500returnsL.20.   .
## volatilityL.11.     .
## sp500L.19.          .
## vixL.19.            .
## sp500returnsL.19.  -0.01657949007305050
## volatilityL.12.     .
## sp500L.18.          .
## vixL.18.            .
## sp500returnsL.18.   .
## volatilityL.13.     .
## sp500L.17.          .
## vixL.17.            .
## sp500returnsL.17.   .
## volatilityL.14.     .
## sp500L.16.          .
## vixL.16.            .
## sp500returnsL.16.  -0.00668772187946690
## volatilityL.15.     .
## sp500L.15.          .
## vixL.15.            .
## sp500returnsL.15.  -0.01093318220483466
## volatilityL.16.     .
## sp500L.14.          .
## vixL.14.            .
## sp500returnsL.14.   .
## volatilityL.17.     .
## sp500L.13.          .
## vixL.13.            .
## sp500returnsL.13.   .
## volatilityL.18.     .
```

```
## sp500L.12.          .
## vixL.12.            .
## sp500returnsL.12.  -0.00170756139059227
## volatilityL.19.     .
## sp500L.11.          .
## vixL.11.            .
## sp500returnsL.11.   .
## volatilityL.20.     .
## sp500L.10.          .
## vixL.10.            .
## sp500returnsL.10.   .
## volatilityL.21.    -0.00102360209131944
## sp500L.9.           .
## vixL.9.             .
## sp500returnsL.9.   -0.01066279344413107
## volatilityL.22.     .
## sp500L.8.           .
## vixL.8.             0.00000023411488589
## sp500returnsL.8.    .
## volatilityL.23.     .
## sp500L.7.           .
## vixL.7.             .
## sp500returnsL.7.   -0.00514884537400726
## volatilityL.24.     .
## sp500L.6.           .
## vixL.6.             .
## sp500returnsL.6.   -0.00177363892888733
## volatilityL.25.     .
## sp500L.5.           .
## vixL.5.             0.00000485569464645
## sp500returnsL.5.    .
## volatilityL.26.     .
## sp500L.4.           .
## vixL.4.             .
## sp500returnsL.4.   -0.00088709377843585
## volatilityL.27.     .
## sp500L.3.           .
## vixL.3.             0.00001543338886448
## sp500returnsL.3.    .
## volatilityL.28.     .
## sp500L.2.           .
## vixL.2.             0.00026860910978826
## sp500returnsL.2.   -0.00448574101470899
## volatilityL.29.     .
## sp500L.1.           .
## vixL.1.             0.00041420979778702
## sp500returnsL.1.   -0.02576837460159772
## pres_approv_avg     .
## pres_disapprov_avg  0.00001634939420384
## pres_unknown_avg   -0.00000663653548162
## USRECD              0.00033609688951726
## DFF.x               .
## DFF.y               .
```

```r
#decision tree
tree_fit <- randomForest(target ~ .,   data=train)
print(tree_fit) # view results
```

```
##
## Call:
##  randomForest(formula = target ~ ., data = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 40
##
##           Mean of squared residuals: 0.00006623889338
##                     % Var explained: 99.2
```

```r
importance(tree_fit) # importance of each predictor
```

```
##                      IncNodePurity
## volatilityL.1.     11.5993882855227
## sp500L.29.          0.0028584162157
## vixL.29.            0.0070198563420
## sp500returnsL.29.   0.0049171303871
## volatilityL.2.      7.8392025666736
## sp500L.28.          0.0025688079130
## vixL.28.            0.0042940641124
## sp500returnsL.28.   0.0028682892991
## volatilityL.3.      5.5993729458787
## sp500L.27.          0.0025847891969
## vixL.27.            0.0031540678926
## sp500returnsL.27.   0.0023849434649
## volatilityL.4.      3.5762263726769
## sp500L.26.          0.0017042793271
## vixL.26.            0.0026930642822
## sp500returnsL.26.   0.0030160425075
## volatilityL.5.      2.9513630990647
## sp500L.25.          0.0022678653003
## vixL.25.            0.0027637294081
## sp500returnsL.25.   0.0027708575636
## volatilityL.6.      1.6305626406370
## sp500L.24.          0.0012886594276
## vixL.24.            0.0021496009030
## sp500returnsL.24.   0.0023998476747
## volatilityL.7.      0.9945755579402
## sp500L.23.          0.0030993066118
## vixL.23.            0.0029298347847
## sp500returnsL.23.   0.0029060813537
## volatilityL.8.      0.5283542487153
## sp500L.22.          0.0017869827312
## vixL.22.            0.0027745104189
## sp500returnsL.22.   0.0020811439501
## volatilityL.9.      0.4414009409361
## sp500L.21.          0.0021425603672
## vixL.21.            0.0028954087758
## sp500returnsL.21.   0.0022118722554
## volatilityL.10.     0.3002951065708
```

```
## sp500L.20.          0.0030123811256
## vixL.20.            0.0025416180225
## sp500returnsL.20.   0.0028475618604
## volatilityL.11.     0.2345771434350
## sp500L.19.          0.0032239217485
## vixL.19.            0.0037691844253
## sp500returnsL.19.   0.0026839123407
## volatilityL.12.     0.0560761255915
## sp500L.18.          0.0028186415120
## vixL.18.            0.0025202905621
## sp500returnsL.18.   0.0024337095513
## volatilityL.13.     0.0040395765691
## sp500L.17.          0.0034151971114
## vixL.17.            0.0031169993934
## sp500returnsL.17.   0.0026140170706
## volatilityL.14.     0.0029177978596
## sp500L.16.          0.0019268690249
## vixL.16.            0.0046912545797
## sp500returnsL.16.   0.0024886808834
## volatilityL.15.     0.0034445858969
## sp500L.15.          0.0026500358151
## vixL.15.            0.0041499402527
## sp500returnsL.15.   0.0025893975648
## volatilityL.16.     0.0023276054586
## sp500L.14.          0.0024048725641
## vixL.14.            0.0052923504043
## sp500returnsL.14.   0.0025517072028
## volatilityL.17.     0.0024852543319
## sp500L.13.          0.0019517895774
## vixL.13.            0.0087686185150
## sp500returnsL.13.   0.0029287834043
## volatilityL.18.     0.0027844680591
## sp500L.12.          0.0022011593003
## vixL.12.            0.0098054611552
## sp500returnsL.12.   0.0021753845594
## volatilityL.19.     0.0024306620013
## sp500L.11.          0.0014264385736
## vixL.11.            0.0141487892875
## sp500returnsL.11.   0.0028059871361
## volatilityL.20.     0.0031088717652
## sp500L.10.          0.0015232339738
## vixL.10.            0.0405837410257
## sp500returnsL.10.   0.0035960777247
## volatilityL.21.     0.0035611222268
## sp500L.9.           0.0014357679305
## vixL.9.             0.0103713858364
## sp500returnsL.9.    0.0023299442226
## volatilityL.22.     0.0028759356874
## sp500L.8.           0.0015897865526
## vixL.8.             0.0429071180233
## sp500returnsL.8.    0.0027075040972
## volatilityL.23.     0.0023376200589
## sp500L.7.           0.0016014569043
## vixL.7.             0.0159729507420
```

```
## sp500returnsL.7.     0.0031444962953
## volatilityL.24.      0.0030304516509
## sp500L.6.            0.0014763984738
## vixL.6.              0.0963609381141
## sp500returnsL.6.     0.0029718149325
## volatilityL.25.      0.0029669518375
## sp500L.5.            0.0014270923754
## vixL.5.              0.4933205175792
## sp500returnsL.5.     0.0037944375813
## volatilityL.26.      0.0030851241256
## sp500L.4.            0.0015196329034
## vixL.4.              0.2155329738625
## sp500returnsL.4.     0.0053632783374
## volatilityL.27.      0.0038033464996
## sp500L.3.            0.0019932923442
## vixL.3.              0.0378191501145
## sp500returnsL.3.     0.0122705670004
## volatilityL.28.      0.0050254994429
## sp500L.2.            0.0021687129726
## vixL.2.              0.1098927186149
## sp500returnsL.2.     0.0127479579406
## volatilityL.29.      0.0095828565601
## sp500L.1.            0.0020425282236
## vixL.1.              0.0708180033747
## sp500returnsL.1.     0.0439975980425
## pres_approv_avg      0.0172227368524
## pres_disapprov_avg   0.0087904132564
## pres_unknown_avg     0.0121193430024
## USRECD               0.0002842490065
## DFF.x                0.0070511890712
## DFF.y                0.0044928664828
```

```
varImpPlot(tree_fit, n.var = 15)
```


```r
#nn
set.seed(1)

n <- names(train_)
f <- as.formula(paste("target ~", paste(n[!n %in% "target"], collapse = " + ")))
nn <- neuralnet(f,data=train_,hidden=c(100,70,60,50,40,30,20),linear.output=T)
```

```r
test_x = test[,-1]
test_x_matrix = model.matrix( ~ .-1, test[,-1])
lasso_test = predict(model.lasso, newx=test_x_matrix,type="link")
tree_test = predict(tree_fit, newdata=test_x)

lasso_mape = mape(test$target,lasso_test)
lasso_mape
```

```
## [1] 2.743576272
```

```r
tree_mape = mape(test$target,tree_test)
tree_mape
```

```
## [1] 2.741516149
```

```r
#nn
pr.nn <- compute(nn,test_[,2:ncol(test_)])
# pr.nn
pr.nn_ <- pr.nn$net.result*(max(full_data$target)-min(full_data$target))+min(full_data$target)
test.r <- (test_$target)*(max(full_data$target)-min(full_data$target))+min(full_data$target)
MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)
```

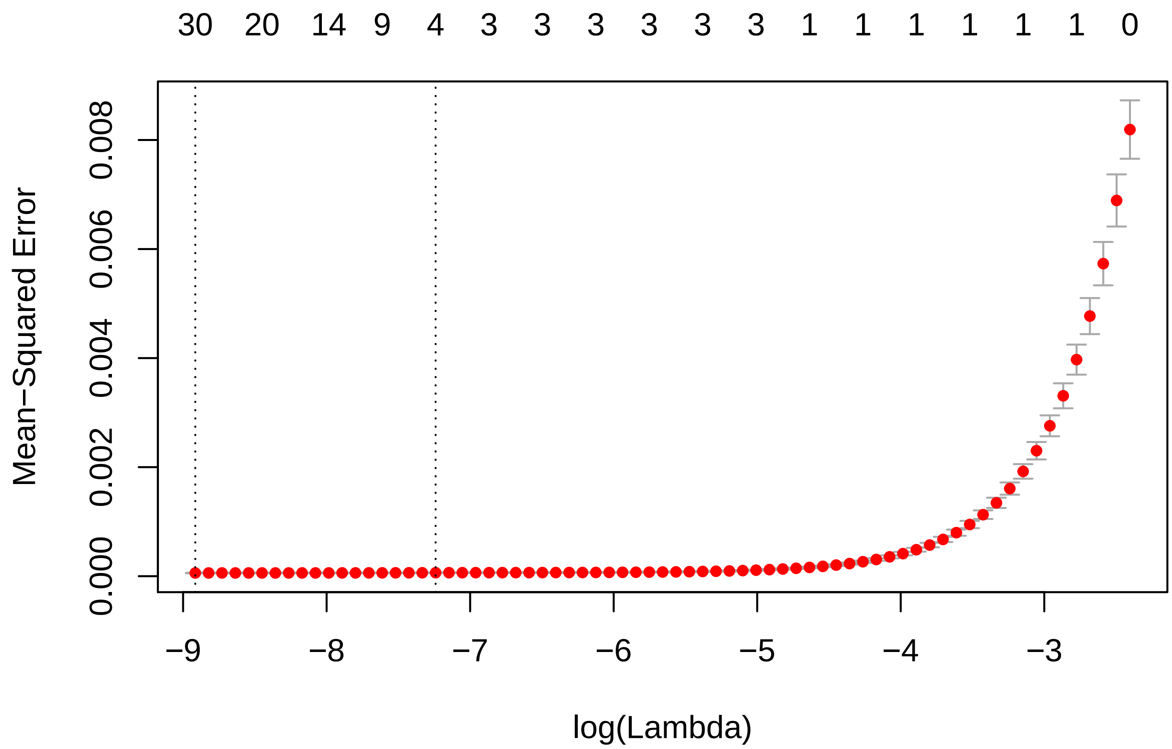
```
# MSE.nn
mape_nn = mape(test.r,pr.nn_)
mape_nn
```

```
## [1] 4.060531678
```

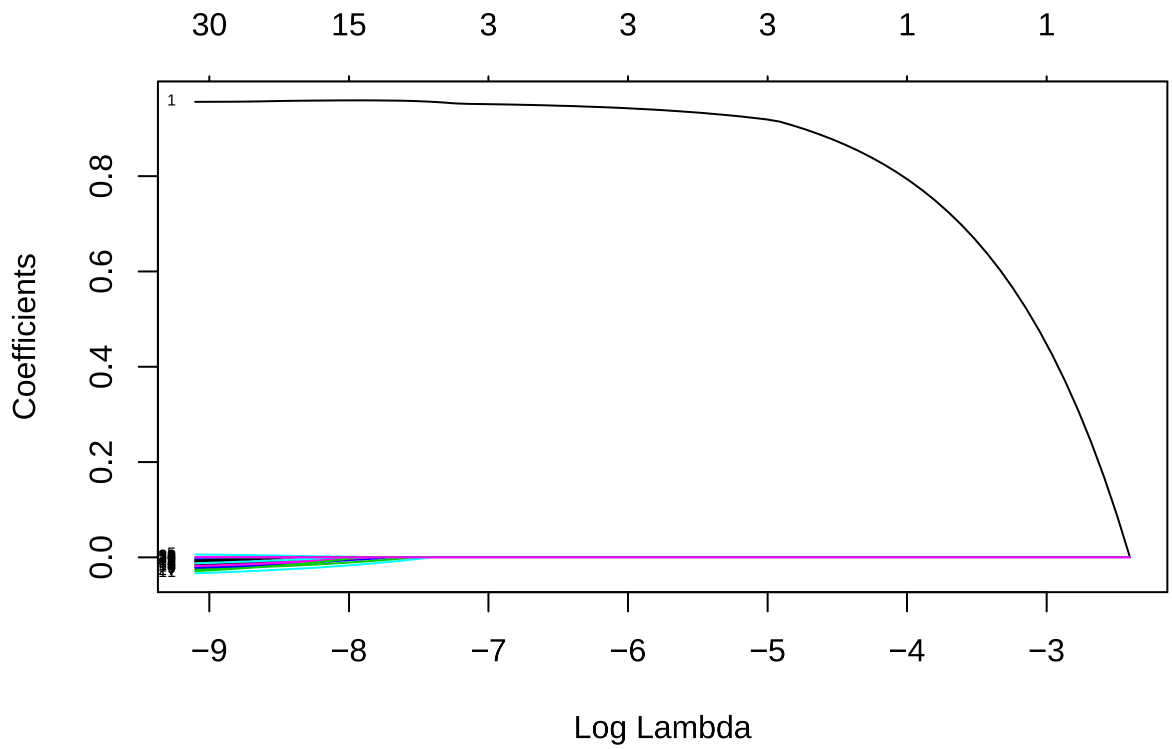
```
#extract non 0s from lasso
coefs = coef(model.lasso, s=model.lasso$lambda.min)

non_0_coefs=c()
for( i in 2: length(coefs) ){
  if(coefs[i]!=0){
    non_0_coefs = c(non_0_coefs,rownames(coefs)[i])
  }
}

#add in target
non_0_coefs = c("target",non_0_coefs)
```

```
# recreate training and test sets
full_data_minimized = full_data_update[ , which(names(full_data_update) %in% non_0_coefs)]

#randomized vs timeseries?
#create training and test sets
## 66% of the sample size
smp_size <- floor(.66* nrow(full_data_minimized))

## set the seed to make your partition reproductible
set.seed(123)
train_ind <- sample(seq_len(nrow(full_data_minimized)), size = smp_size)

train <- full_data_minimized[train_ind, ]
test <- full_data_minimized[-train_ind, ]

#normalize data for nn

maxs <- apply(full_data_minimized, 2, max)
mins <- apply(full_data_minimized, 2, min)

scaled <- as.data.frame(scale(full_data_minimized, center = mins, scale = maxs - mins))


train_ <- scaled[train_ind,]
test_ <- scaled[-train_ind,]
```

```
#lasso
x <- model.matrix( ~ .-1, train[ , -1])
y <- data.matrix(train[, 1])

model.lasso <- cv.glmnet(x, y, family='gaussian', alpha=1, parallel=TRUE, standardize=TRUE)
plot(model.lasso)
```




```r
plot(model.lasso$glmnet.fit, xvar="lambda", label=TRUE)
```




```r
model.lasso$lambda.min
```

```
## [1] 0.0001343292408
```

```r
model.lasso$lambda.1se
```

```
## [1] 0.0007168747562
```

```r
coef(model.lasso, s=model.lasso$lambda.min)
```

```
## 31 x 1 sparse Matrix of class "dgCMatrix"
##                                     1
## (Intercept)        -0.00164681770362576
## volatilityL.1.      0.95615000525714477
## sp500L.29.          0.00000009678586914
## vixL.29.           -0.00033914333367402
## sp500returnsL.29.  -0.01904145324310854
## volatilityL.2.      0.00542628780856293
## sp500returnsL.28.  -0.01402732507694284
## sp500returnsL.27.  -0.01227132527419695
## sp500returnsL.26.  -0.00665165647003229
## sp500returnsL.25.  -0.02338451984523382
## sp500returnsL.22.  -0.02020554677547412
## sp500returnsL.21.  -0.03142996409108745
## sp500returnsL.19.  -0.01657854307618097
## sp500returnsL.16.  -0.00668675791171637
## sp500returnsL.15.  -0.01093247564280158
## sp500returnsL.12.  -0.00170679657864188
```

```
## volatilityL.21.    -0.00102476310533367
## sp500returnsL.9.   -0.01066187203634715
## vixL.8.             0.00000023024722876
## sp500returnsL.7.   -0.00514768935308629
## sp500returnsL.6.   -0.00177254255475355
## vixL.5.             0.00000486401050231
## sp500returnsL.4.   -0.00088654558017649
## vixL.3.             0.00001543674315533
## vixL.2.             0.00026861201897751
## sp500returnsL.2.   -0.00448584170573145
## vixL.1.             0.00041420995438159
## sp500returnsL.1.   -0.02576836137519164
## pres_disapprov_avg  0.00001635007992736
## pres_unknown_avg   -0.00000663565226761
## USRECD              0.00033612885053441
```

```r
#decision tree
tree_fit <- randomForest(target ~ .,   data=train)
print(tree_fit) # view results
```

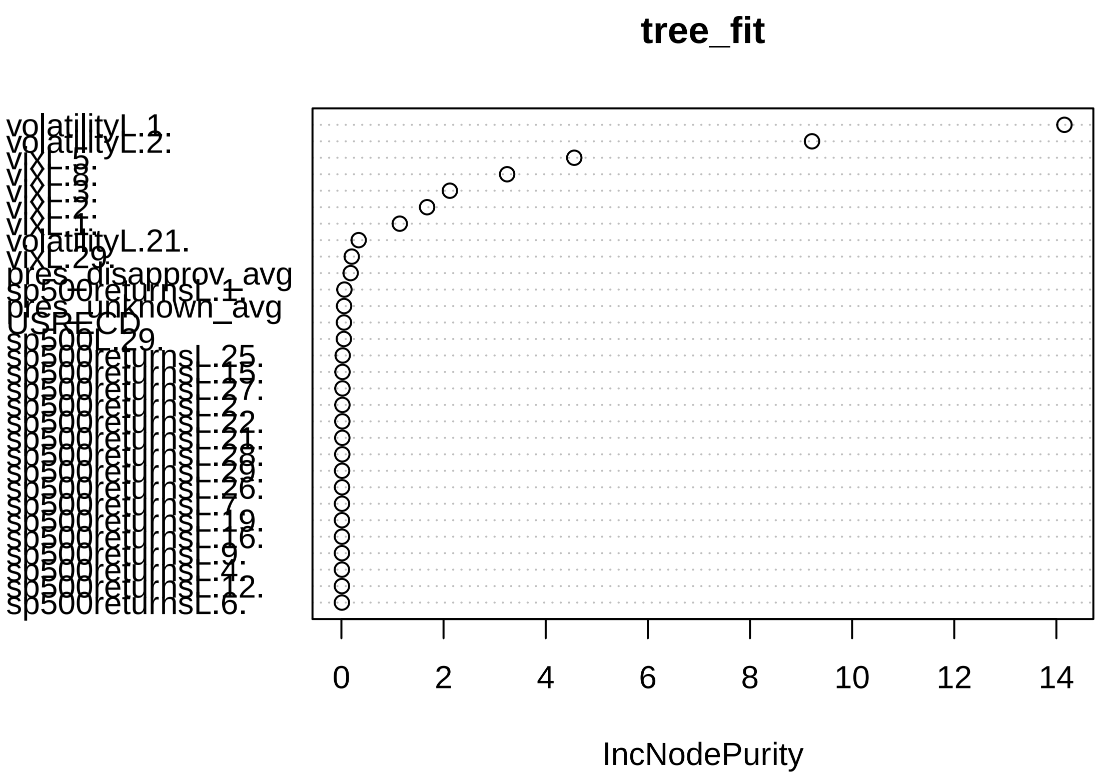
```
##
## Call:
##  randomForest(formula = target ~ ., data = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 0.000060460118
##                     % Var explained: 99.27
```

```r
importance(tree_fit) # importance of each predictor
```

```
##                     IncNodePurity
## volatilityL.1.     14.157485924676
## sp500L.29.          0.048098998086
## vixL.29.            0.202180122222
## sp500returnsL.29.   0.013730179794
## volatilityL.2.      9.214392635123
## sp500returnsL.28.   0.016744082286
## sp500returnsL.27.   0.019936815987
## sp500returnsL.26.   0.012626122528
## sp500returnsL.25.   0.025281752593
## sp500returnsL.22.   0.018055796588
## sp500returnsL.21.   0.017440491035
## sp500returnsL.19.   0.011138625075
## sp500returnsL.16.   0.010886152843
## sp500returnsL.15.   0.021162065387
## sp500returnsL.12.   0.009814930310
## volatilityL.21.     0.337965157967
## sp500returnsL.9.    0.010595776367
## vixL.8.             3.245446978384
## sp500returnsL.7.    0.011516802178
## sp500returnsL.6.    0.009586764366
## vixL.5.             4.559757755894
## sp500returnsL.4.    0.009897074354
```

```
## vixL.3.              2.123628621402
## vixL.2.              1.678156895864
## sp500returnsL.2.     0.019661260986
## vixL.1.              1.142729288338
## sp500returnsL.1.     0.058342871854
## pres_disapprov_avg   0.180852655765
## pres_unknown_avg     0.052153413168
## USRECD               0.049741156649
```

```r
varImpPlot(tree_fit)
```




```r
#nn
set.seed(1)

n <- names(train_)
f <- as.formula(paste("target ~", paste(n[!n %in% "target"], collapse = " + ")))
nn <- neuralnet(f,data=train_,hidden=c(100,70,60,50,40,30,20),linear.output=T)
```

```r
test_x = test[,-1]
test_x_matrix = model.matrix( ~ .-1, test[,-1])
lasso_test = predict(model.lasso, newx=test_x_matrix,type="link")
tree_test = predict(tree_fit, newdata=test_x)

lasso_mape = mape(test$target,lasso_test)
lasso_mape
```

```
## [1] 2.743576272
```

```r
tree_mape = mape(test$target,tree_test)
tree_mape
```

```
## [1] 2.732716043
```

```r
#nn
pr.nn <- compute(nn,test_[,2:ncol(test_)])
# pr.nn
pr.nn_ <- pr.nn$net.result*(max(full_data$target)-min(full_data$target))+min(full_data$target)
test.r <- (test_$target)*(max(full_data$target)-min(full_data$target))+min(full_data$target)
MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)
# MSE.nn
mape_nn = mape(test.r,pr.nn_)
mape_nn
```

```
## [1] 3.309412937
```

```r
set.seed(1)
nn <- neuralnet(f,data=train_,hidden=c(30,15,6),linear.output=T)
pr.nn <- compute(nn,test_[,2:ncol(test_)])
# pr.nn
pr.nn_ <- pr.nn$net.result*(max(full_data$target)-min(full_data$target))+min(full_data$target)
test.r <- (test_$target)*(max(full_data$target)-min(full_data$target))+min(full_data$target)
MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)
# MSE.nn
mape_nn = mape(test.r,pr.nn_)
mape_nn
```

```
## [1] 3.361479453
```

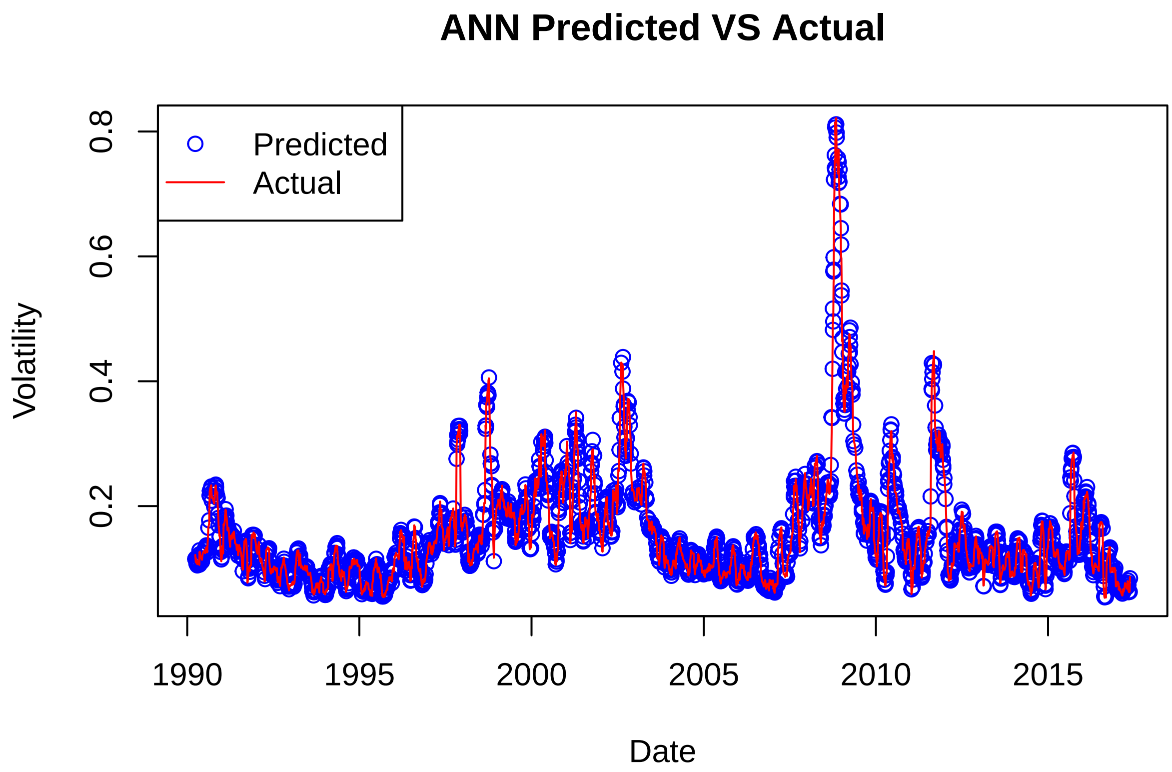
```r
nn_func=function(nodes){
  set.seed(1)
  nn <- neuralnet(f,data=train_,hidden=nodes,linear.output=T)
  pr.nn <- compute(nn,test_[,2:ncol(test_)])
  # pr.nn
  pr.nn_ <- pr.nn$net.result*(max(full_data$target)-min(full_data$target))+min(full_data$target)
  test.r <- (test_$target)*(max(full_data$target)-min(full_data$target))+min(full_data$target)
  MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)
  # MSE.nn
  mape_nn = mape(test.r,pr.nn_)
  return(mape_nn)
}

##comment out when not in use

# iterations = 10
# #randomly select values around 30, 15, 6
# for(i in 1:iterations){
#
#   x1 <- floor(runif(1, 15, 25))
#   x2 <- floor(runif(1, 10, 16))
#   x3 <- floor(runif(1, -2, 3))
#
#   node_list=c()
#
#   if(x3>0){
```

```
#     node_list = c(x1,x2,x3)
#   }
#   else{
#     node_list = c(x1,x2)
#   }
#   mape1 = nn_func(node_list)
#   if(mape1<=2.24627944){
#     print(node_list)
#     print(mape1)
#   }
#
# }
```

```
set.seed(1)
nn <- neuralnet(f,data=train_,hidden=c(20,13,1),linear.output=T)
pr.nn <- compute(nn,test_[,2:ncol(test_)])
# pr.nn
pr.nn_ <- pr.nn$net.result*(max(full_data$target)-min(full_data$target))+min(full_data$target)
test.r <- (test_$target)*(max(full_data$target)-min(full_data$target))+min(full_data$target)
MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)
# MSE.nn
mape_nn = mape(test.r,pr.nn_)
print(mape_nn)
```

```
## [1] 2.288863265
```

```
plot(x=as.Date(rownames(pr.nn_)),y=pr.nn_, type='p',col="blue", ylab="Volatility",xlab="Date",main="ANN
lines(x=as.Date(rownames(pr.nn_)),test.r,col="red")
legend("topleft",c("Predicted","Actual"),lty=c(0,1), col = c('blue','red'), pch=c(1,NA))
```




```
#normalize data

maxs <- apply(full_data_minimized, 2, max)
mins <- apply(full_data_minimized, 2, min)

scaled <- as.data.frame(scale(full_data_minimized, center = mins, scale = maxs - mins))

set.seed(34)

cv.error <- NULL
mape_nn_cv = NULL
k <- 5


pbar <- create_progress_bar('text')
pbar$init(k)
```

```
##
  |
  |                                                                 |   0%
```

```
for(i in 1:k){
    index <- sample(1:nrow(scaled),round(0.8*nrow(scaled)))
    train.cv <- scaled[index,]
    test.cv <- scaled[-index,]

    nn <- neuralnet(f,data=train.cv,hidden=c(20,13,1),linear.output=T)
```

```r
    pr.nn <- compute(nn,test.cv[,2:ncol(test.cv)])
    pr.nn <- pr.nn$net.result*(max(full_data$target)-min(full_data$target))+min(full_data$target)

    test.cv.r <- (test.cv$target)*(max(full_data$target)-min(full_data$target))+min(full_data$target)

    cv.error[i] <- sum((test.cv.r - pr.nn)^2)/nrow(test.cv)

    mape_nn_cv[i] = mape(test.cv.r,pr.nn)

    pbar$step()
}
```

```
##
  |
  |=============                                                    |  20%
  |
  |==========================                                       |  40%
  |
  |=======================================                          |  60%
  |
  |====================================================             |  80%
  |
  |=================================================================| 100%
```

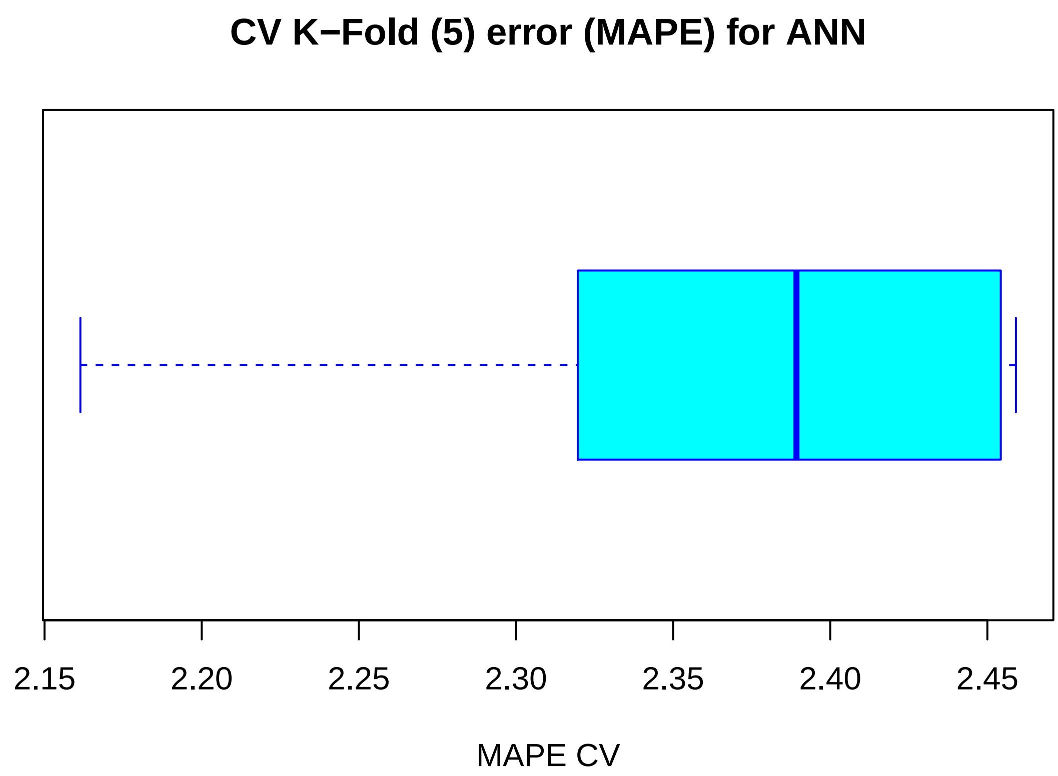
```r
mean(cv.error)
```

```
## [1] 0.00006429047282
```

```r
mean(mape_nn_cv)
```

```
## [1] 2.356741945
```

```r
boxplot(mape_nn_cv,xlab='MAPE CV',col='cyan',
        border='blue',names='CV error (MAPE)',
        main='CV K-Fold (5) error (MAPE) for ANN',horizontal=TRUE)
```




```
var(mape_nn_cv)
```

```
## [1] 0.01514343111
```

## Project Extension

The previous capstone work confirmed that machine learning can outperform ARIMA models in a linear regression time series format. However, this project will extend the applications into binary time series prediction/classification. For this project, I use the same data set.

The new proposal is to compare ARIMA with the machine learning algorithms by using the recession as a time series and the target for the algorithms.

First, I start by massaging the data set to make the recession the new target.

```
#drop target
target_dropped = full_data_update[,2:length(full_data_update)]
#use recession
recession_target = target_dropped
recession_target$target = target_dropped$USRECD
recession_target = recession_target[ , -which(names(recession_target) %in% c("USRECD"))]

prediction_error = function(y,yhat){
  results = ifelse(yhat>.05,1,0)
  print(confusionMatrix(results,y))
  return(mean(results != y))
```

```
}
```

# ARIMA

First, I run ARIMA to see how well it performs in backtests in both the recursive and rolling window setting. I do not expect it to do well.

```
ts_r = ts(recession_target$target)

#benchmark
recursive=(backtest(ts_r,1,"recursive"))
rolling=(backtest(ts_r,1,"rolling"))
print(prediction_error(recursive[[1]],recursive[[2]]))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1984    0
##          1    1  295
##
##                Accuracy : 0.9995614
##                  95% CI : (0.9975587, 0.9999889)
##     No Information Rate : 0.870614
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.998056
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9994962
##             Specificity : 1.0000000
##          Pos Pred Value : 1.0000000
##          Neg Pred Value : 0.9966216
##              Prevalence : 0.8706140
##          Detection Rate : 0.8701754
##    Detection Prevalence : 0.8701754
##       Balanced Accuracy : 0.9997481
##
##        'Positive' Class : 0
##
## [1] 0.0004385964912
```

```
print(prediction_error(rolling[[1]],rolling[[2]]))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1984    0
##          1    1  295
##
##                Accuracy : 0.9995614
##                  95% CI : (0.9975587, 0.9999889)
```

```
##     No Information Rate : 0.870614
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.998056
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9994962
##             Specificity : 1.0000000
##          Pos Pred Value : 1.0000000
##          Neg Pred Value : 0.9966216
##              Prevalence : 0.8706140
##          Detection Rate : 0.8701754
##    Detection Prevalence : 0.8701754
##       Balanced Accuracy : 0.9997481
##
##        'Positive' Class : 0
##
## [1] 0.0004385964912
```

The results show the opposite and show that ARIMA does an excellent job in classifying the recession. On closer inspection, however, the ARIMA model is predicting a recession tomorrow if there is a recession today and similar for non-recessions. The test set was split from the training set in the middle of a recession, therefore it starts with a string of recession days with no other recessions after this one ends. This does not give us a good method to predict oncoming recessions and only tells us we are likely to be in a recession tomorrow if we are in one today. Therefore, ARIMA fails at the task of giving a liklihood estimate and the accuracy rating is misleading.

# Machine Learning

The techniques used in this section are the same as the capstone portion and are translatable into the binary classifcation setting. Before running our machine learning algorithms, the data needs to be randomly split into 2/3 training and 1/3 test.

```
#create training and test sets
## 66% of the sample size
smp_size <- floor(.66* nrow(recession_target))

## set the seed to make your partition reproductible
set.seed(123)
train_ind <- sample(seq_len(nrow(recession_target)), size = smp_size)

train <- recession_target[train_ind, ]
test <- recession_target[-train_ind, ]


#normalize data for nn
maxs <- apply(recession_target, 2, max)
mins <- apply(recession_target, 2, min)
scaled <- as.data.frame(scale(recession_target, center = mins, scale = maxs - mins))
train_ <- scaled[train_ind,]
test_ <- scaled[-train_ind,]
```

```r
#y_index
y_index = length(recession_target)
```

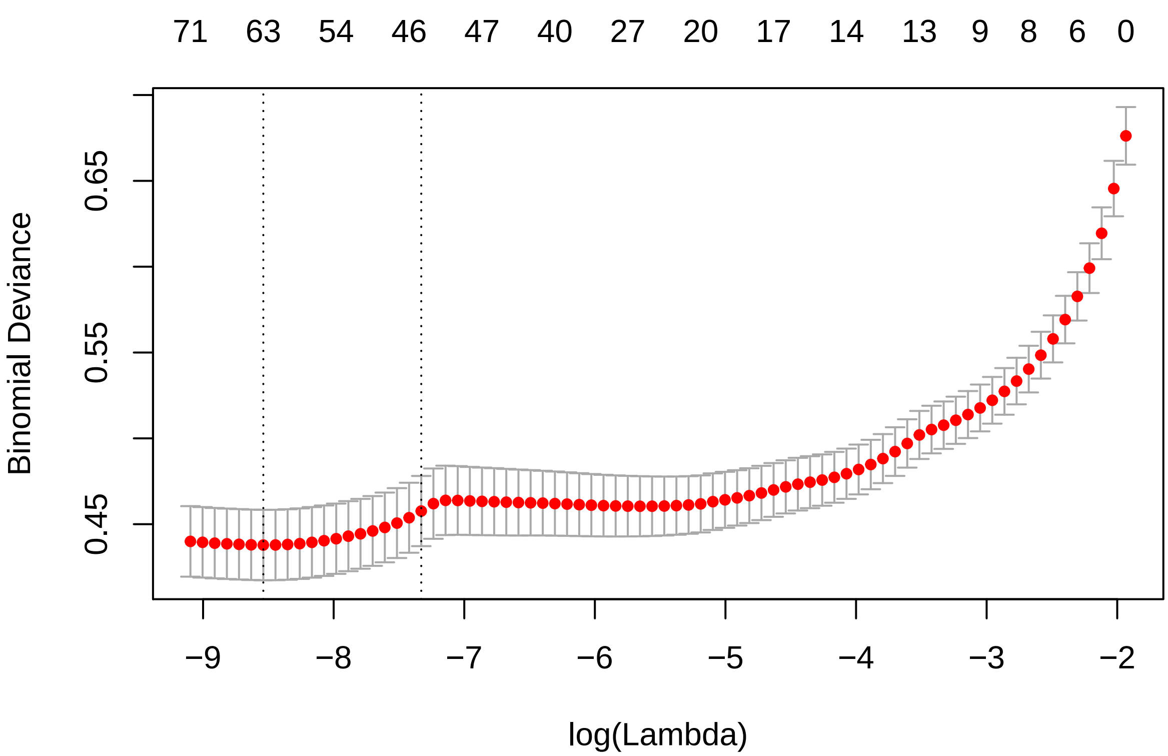
## LASSO

There are two main purposes for LASSO:

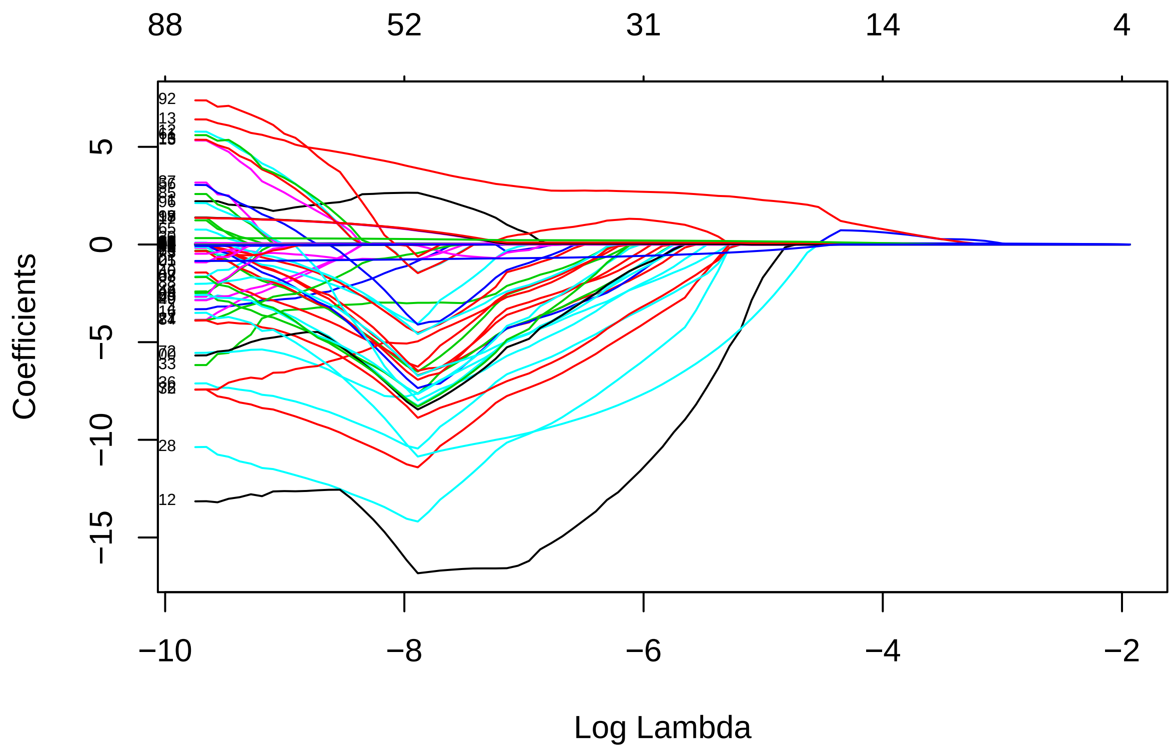
1. Prediction
2. Shrinkage

LASSO is capable of giving us a good binomial prediction, but it is also able to shrink our data set to remove irrelevant variables that will help for neural network tuning.

```r
#lasso
x <- model.matrix( ~ .-1, train[ , -y_index])
y <- data.matrix(train[, y_index])

model.lasso <- cv.glmnet(x, y, family='binomial', alpha=1, parallel=TRUE, standardize=TRUE)
plot(model.lasso)
```




```r
plot(model.lasso$glmnet.fit, xvar="lambda", label=TRUE)
```




```
model.lasso$lambda.min
```

```
## [1] 0.000195731992
```

```
model.lasso$lambda.1se
```

```
## [1] 0.0006560158632
```

```
coef(model.lasso, s=model.lasso$lambda.min)
```

```
## 122 x 1 sparse Matrix of class "dgCMatrix"
##                                        1
## (Intercept)        -115.2244119672416787
## volatilityL.1.        2.1723799331228824
## sp500L.29.            0.0035197861250741
## vixL.29.              0.0612346116896018
## sp500returnsL.29.    -2.2037753831604157
## volatilityL.2.        .
## sp500L.28.            .
## vixL.28.              0.0377393261509109
## sp500returnsL.28.    -3.2606311705494040
## volatilityL.3.        .
## sp500L.27.            .
## vixL.27.              0.0040282224883304
## sp500returnsL.27.     1.0066177921631259
## volatilityL.4.        1.0857887963071067
## sp500L.26.            .
## vixL.26.              0.0138874308567433
```

```
## sp500returnsL.26.      0.9319791872572156
## volatilityL.5.         .
## sp500L.25.             .
## vixL.25.               0.0222031784918754
## sp500returnsL.25.     -5.1263888895675231
## volatilityL.6.        -0.7123494938751496
## sp500L.24.             .
## vixL.24.               .
## sp500returnsL.24.     -3.7022669889846047
## volatilityL.7.         .
## sp500L.23.             .
## vixL.23.               0.0449274206260172
## sp500returnsL.23.    -12.5166730607321544
## volatilityL.8.        -0.7338231154173670
## sp500L.22.             .
## vixL.22.               .
## sp500returnsL.22.     -9.6297874095317582
## volatilityL.9.        -3.0992795908537532
## sp500L.21.             .
## vixL.21.               .
## sp500returnsL.21.     -8.7866335541191312
## volatilityL.10.        .
## sp500L.20.             .
## vixL.20.               .
## sp500returnsL.20.     -4.1847281968047820
## volatilityL.11.        .
## sp500L.19.             .
## vixL.19.               .
## sp500returnsL.19.     -5.2406622609744753
## volatilityL.12.        .
## sp500L.18.             .
## vixL.18.               .
## sp500returnsL.18.     -2.1784510639050709
## volatilityL.13.       -0.6498405742503764
## sp500L.17.             .
## vixL.17.              -0.0263610603346937
## sp500returnsL.17.     -2.9609794830105542
## volatilityL.14.        .
## sp500L.16.             .
## vixL.16.               .
## sp500returnsL.16.     -0.3588438087809170
## volatilityL.15.        .
## sp500L.15.             .
## vixL.15.               .
## sp500returnsL.15.     -3.6065713606728842
## volatilityL.16.        1.3934433076788293
## sp500L.14.             .
## vixL.14.               .
## sp500returnsL.14.     -3.6288791653539554
## volatilityL.17.        .
## sp500L.13.             .
## vixL.13.               0.0255872552575021
## sp500returnsL.13.     -1.9554858587921922
## volatilityL.18.        .
```

```
## sp500L.12.                 .
## vixL.12.                  0.0225982755688940
## sp500returnsL.12.        -6.7299788447879472
## volatilityL.19.            .
## sp500L.11.                 .
## vixL.11.                  0.0181638660649899
## sp500returnsL.11.        -5.8440897044580709
## volatilityL.20.          -1.5995891190157134
## sp500L.10.                 .
## vixL.10.                 -0.0392102693275391
## sp500returnsL.10.        -1.8368167911417910
## volatilityL.21.            .
## sp500L.9.                  .
## vixL.9.                    .
## sp500returnsL.9.         -5.6969183925222797
## volatilityL.22.            .
## sp500L.8.                  .
## vixL.8.                    .
## sp500returnsL.8.         -3.3429407165988674
## volatilityL.23.            .
## sp500L.7.                  .
## vixL.7.                  -0.0156140691365199
## sp500returnsL.7.          3.7157985288544659
## volatilityL.24.            .
## sp500L.6.                  .
## vixL.6.                  -0.0000489121943011
## sp500returnsL.6.         -3.0648145787189018
## volatilityL.25.            .
## sp500L.5.                  .
## vixL.5.                   0.0206333616592860
## sp500returnsL.5.         -5.1794296672655147
## volatilityL.26.            .
## sp500L.4.                  .
## vixL.4.                    .
## sp500returnsL.4.         -5.1446261192790592
## volatilityL.27.            .
## sp500L.3.                -0.0000003777710657
## vixL.3.                    .
## sp500returnsL.3.         -5.3842704374962302
## volatilityL.28.            .
## sp500L.2.                -0.0000010602550352
## vixL.2.                   0.0132590837332285
## sp500returnsL.2.        -12.5545593819526449
## volatilityL.29.           4.7166776498684744
## sp500L.1.                -0.0044318058084628
## vixL.1.                  -0.0009009528768962
## sp500returnsL.1.         -6.6714493996023165
## pres_approv_avg           1.0841982524496878
## pres_disapprov_avg        1.1014921725550426
## pres_unknown_avg          1.1014423460667671
## DFF.x                     0.3038233644718105
## DFF.y                    -0.7984323211897200
```

```r
test_x = test[,-1]
test_x_matrix = model.matrix( ~ .-1, test[,-y_index])
lasso_test = predict(model.lasso, newx=test_x_matrix,type="link")


lasso_error = prediction_error(test$target, lasso_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2058  155
##          1   40   74
##
##                Accuracy : 0.9162011
##                  95% CI : (0.9041969, 0.9271447)
##     No Information Rate : 0.90159
##     P-Value [Acc > NIR] : 0.008752342
##
##                   Kappa : 0.3916943
##  Mcnemar's Test P-Value : 0.000000000000003248895
##
##             Sensitivity : 0.9809342
##             Specificity : 0.3231441
##          Pos Pred Value : 0.9299593
##          Neg Pred Value : 0.6491228
##              Prevalence : 0.9015900
##          Detection Rate : 0.8844005
##    Detection Prevalence : 0.9510099
##       Balanced Accuracy : 0.6520392
##
##        'Positive' Class : 0
##
```

```r
print(lasso_error)
```

```
## [1] 0.08379888268
```

We can see the results from LASSO have a high accuracy. However, the accuracy is misleading as it was in ARIMA because there is a disproportionate ammount of 0s compared to 1s in the data set. It is important to look at the confusion matrix. We can see that LASSO actually misclassified 1s a lot. It wrongly estimated a 0 155 times and wrongly estimated a 1 40 times and only correctly estimated a 1 74 times. This is still better than ARIMA for giving us a liklihood estimate, but it is not strong enough.

# Decision Trees

We use random forests so we don't have correlated trees or high variance from sample.

```r
#decision tree
tree_fit <- randomForest(target ~ .,   data=train)
print(tree_fit) # view results
```

```
##
## Call:
```

```
##  randomForest(formula = target ~ ., data = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 40
##
##           Mean of squared residuals: 0.003277210193
##                     % Var explained: 96.56
```

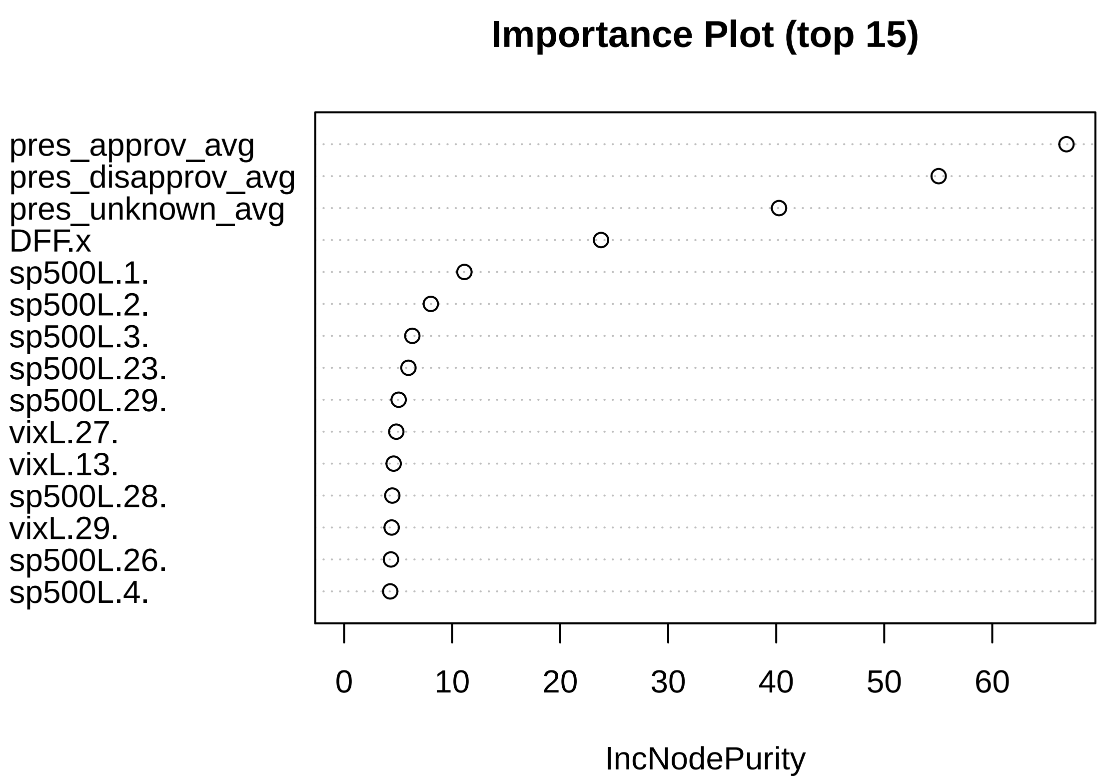
```r
importance(tree_fit) # importance of each predictor
```

```
##                    IncNodePurity
## volatilityL.1.      4.1480614452
## sp500L.29.          5.0509299869
## vixL.29.            4.4020446320
## sp500returnsL.29.   0.1514691125
## volatilityL.2.      2.2141652595
## sp500L.28.          4.4614428577
## vixL.28.            3.0013700539
## sp500returnsL.28.   0.1335122537
## volatilityL.3.      1.9369395455
## sp500L.27.          3.1155940924
## vixL.27.            4.8316896308
## sp500returnsL.27.   0.1813834853
## volatilityL.4.      1.4335287785
## sp500L.26.          4.3366048411
## vixL.26.            1.6481347070
## sp500returnsL.26.   0.1659017298
## volatilityL.5.      1.4417431984
## sp500L.25.          3.8326718227
## vixL.25.            2.0998368778
## sp500returnsL.25.   0.1610409363
## volatilityL.6.      1.6095496692
## sp500L.24.          3.1904790840
## vixL.24.            2.4619615173
## sp500returnsL.24.   0.1458026900
## volatilityL.7.      1.4765628313
## sp500L.23.          5.9565135109
## vixL.23.            2.0806952869
## sp500returnsL.23.   0.1031253577
## volatilityL.8.      2.5908483043
## sp500L.22.          3.0933378221
## vixL.22.            3.9439481467
## sp500returnsL.22.   0.1606892328
## volatilityL.9.      1.1818548345
## sp500L.21.          4.0080216155
## vixL.21.            3.7264840771
## sp500returnsL.21.   0.2642562658
## volatilityL.10.     2.2102705687
## sp500L.20.          3.4041191276
## vixL.20.            0.7754129054
## sp500returnsL.20.   0.1404572837
## volatilityL.11.     0.9535817045
## sp500L.19.          3.5358150594
## vixL.19.            0.7060412490
## sp500returnsL.19.   0.1440550066
```

```
## volatilityL.12.     1.3580438863
## sp500L.18.          2.0028441661
## vixL.18.            1.1539423584
## sp500returnsL.18.   0.1855170776
## volatilityL.13.     1.6963390114
## sp500L.17.          2.2271346976
## vixL.17.            0.7636194211
## sp500returnsL.17.   0.2221985252
## volatilityL.14.     2.2300113856
## sp500L.16.          1.7219162436
## vixL.16.            0.7508266981
## sp500returnsL.16.   0.1679102699
## volatilityL.15.     1.6289172920
## sp500L.15.          2.1466794123
## vixL.15.            1.1282139546
## sp500returnsL.15.   0.1986533505
## volatilityL.16.     3.3432765897
## sp500L.14.          1.9219868612
## vixL.14.            1.5210471837
## sp500returnsL.14.   0.2164478452
## volatilityL.17.     2.6569895280
## sp500L.13.          2.0198657256
## vixL.13.            4.5867248353
## sp500returnsL.13.   0.1979153027
## volatilityL.18.     1.7975374669
## sp500L.12.          2.7297625816
## vixL.12.            1.3247534686
## sp500returnsL.12.   0.1770699426
## volatilityL.19.     2.9857712816
## sp500L.11.          3.1226332907
## vixL.11.            2.4191565750
## sp500returnsL.11.   0.2006253387
## volatilityL.20.     1.6933828275
## sp500L.10.          2.6685444553
## vixL.10.            1.6797866453
## sp500returnsL.10.   0.1869562561
## volatilityL.21.     2.0464018864
## sp500L.9.           2.4691004315
## vixL.9.             1.8457998792
## sp500returnsL.9.    0.2277311089
## volatilityL.22.     1.6852660218
## sp500L.8.           2.8700346283
## vixL.8.             1.4311488117
## sp500returnsL.8.    0.1843443688
## volatilityL.23.     2.2314663475
## sp500L.7.           3.7648792905
## vixL.7.             2.6854719761
## sp500returnsL.7.    0.1349273437
## volatilityL.24.     1.7718255758
## sp500L.6.           3.3019238248
## vixL.6.             0.8318974077
## sp500returnsL.6.    0.3189863864
## volatilityL.25.     1.9814609971
## sp500L.5.           3.7422087658
```

```
## vixL.5.              1.7541765894
## sp500returnsL.5.     0.2712134543
## volatilityL.26.      2.4516834236
## sp500L.4.            4.2695643559
## vixL.4.              1.2013297284
## sp500returnsL.4.     0.1803181827
## volatilityL.27.      2.2621159401
## sp500L.3.            6.3134779289
## vixL.3.              3.5010836160
## sp500returnsL.3.     0.3169315475
## volatilityL.28.      3.1297346481
## sp500L.2.            8.0242229792
## vixL.2.              2.4214716774
## sp500returnsL.2.     0.4130253501
## volatilityL.29.      3.5652243763
## sp500L.1.           11.1356197625
## vixL.1.              2.6696215083
## sp500returnsL.1.     0.4488788097
## pres_approv_avg     66.8679552137
## pres_disapprov_avg 55.0371764197
## pres_unknown_avg    40.2606897364
## DFF.x               23.7836551237
## DFF.y                0.4822133425
```

```
varImpPlot(tree_fit, main = "Importance Plot (top 15)", n.var = 15)
```



```r
test_x = test[,-y_index]
test_x_matrix = model.matrix( ~ .-1, test[,-y_index])
tree_test = predict(tree_fit, newdata=test_x)

tree_mape = mape(test$target,tree_test)
tree_mape
```

```
## [1] Inf
```

```r
tree_error = prediction_error(test$target,tree_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2003    0
##          1   95  229
##
##                Accuracy : 0.9591749
##                  95% CI : (0.9503216, 0.9668468)
##     No Information Rate : 0.90159
##     P-Value [Acc > NIR] : < 0.00000000000000022204
##
##                   Kappa : 0.8058174
##  Mcnemar's Test P-Value : < 0.00000000000000022204
##
##             Sensitivity : 0.9547188
##             Specificity : 1.0000000
##          Pos Pred Value : 1.0000000
##          Neg Pred Value : 0.7067901
##              Prevalence : 0.9015900
##          Detection Rate : 0.8607649
##    Detection Prevalence : 0.8607649
##       Balanced Accuracy : 0.9773594
##
##        'Positive' Class : 0
##
```

```r
print(tree_error)
```

```
## [1] 0.04082509669
```

This gives us a very good prediction accuracy, but we must remember that the prediction accuracy is deceiving. Therefore, we must look at the confusion matrix. From the confusion matrix, we see that it does not wrongly predict a 0 when the actual was a 1. It does however predict a 1 95 times when it should have been a 0. It correctly predicts a recession 229 times. This is much better than the LASSO results. Based on these results, we can take this as a cautious approach in estimating a recession prediction model.

# Neural Network

Since neural networks proved to be very powerful in the capstone section, there is a chance that they may prove extremely capable of recession prediction. I start by using the same neural network setup from the untuned capstone network as a starting point.

```r
n <- names(train_)
f <- as.formula(paste("target ~", paste(n[!n %in% "target"], collapse = " + ")))
set.seed(1)
nn <- neuralnet(f,data=train_,hidden=c(100,80,70,60,50,40,30,20),linear.output = FALSE)
pr.nn <- compute(nn,test_[,2:ncol(test_)])
# pr.nn
pr.nn_ <- pr.nn$net.result*(max(recession_target$target)-min(recession_target$target))+min(recession_ta
test.r <- (test_$target)*(max(recession_target$target)-min(recession_target$target))+min(recession_targ


nn_error = prediction_error(test.r,pr.nn_)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0  194  120
##          1 1904  109
##
##                Accuracy : 0.1302106
##                  95% CI : (0.1167941, 0.1445653)
##     No Information Rate : 0.90159
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.0965426
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.09246902
##             Specificity : 0.47598253
##          Pos Pred Value : 0.61783439
##          Neg Pred Value : 0.05414804
##              Prevalence : 0.90159003
##          Detection Rate : 0.08336914
##    Detection Prevalence : 0.13493769
##       Balanced Accuracy : 0.28422578
##
##        'Positive' Class : 0
##
```

```r
print(nn_error)
```

```
## [1] 0.8697894284
```

The results are not assuring as there is very poor accuracy all around. The network needs to be tuned.

# Nueral Network Tuning

I start by dropping variables that were shrunk from LASSO. This will help reduce noise from irrelevant variables.

```r
#extract non 0s from lasso
coefs = coef(model.lasso, s=model.lasso$lambda.min)

non_0_coefs=c()
```

```r
for( i in 2: length(coefs) ){
  if(coefs[i]!=0){
    non_0_coefs = c(non_0_coefs,rownames(coefs)[i])
  }
}

#add in target
non_0_coefs = c("target",non_0_coefs)
rec_data_minimized = recession_target[ , which(names(recession_target) %in% non_0_coefs)]


#recreate training/test

#create training and test sets
## 66% of the sample size
smp_size <- floor(.66* nrow(rec_data_minimized))

## set the seed to make your partition reproductible
set.seed(123)
train_ind <- sample(seq_len(nrow(rec_data_minimized)), size = smp_size)

train <- rec_data_minimized[train_ind, ]
test <- rec_data_minimized[-train_ind, ]


#normalize data for nn
maxs <- apply(rec_data_minimized, 2, max)
mins <- apply(rec_data_minimized, 2, min)
scaled <- as.data.frame(scale(rec_data_minimized, center = mins, scale = maxs - mins))
train_ <- scaled[train_ind,]
test_ <- scaled[-train_ind,]


#y_index
y_index = length(rec_data_minimized)
```

After shrinking the data, I select a set of hidden layers and nodes at each layer through guesswork.

```r
n <- names(train_)
f <- as.formula(paste("target ~", paste(n[!n %in% "target"], collapse = " + ")))
set.seed(1)
nn <- neuralnet(f,data=train_,hidden=c(5,3,1),linear.output = FALSE,act.fct = "logistic")
pr.nn <- compute(nn,test_[,2:ncol(test_)])
# pr.nn
pr.nn_ <- pr.nn$net.result*(max(recession_target$target)-min(recession_target$target))+min(recession_ta
test.r <- (test_$target)*(max(recession_target$target)-min(recession_target$target))+min(recession_targe


nn_error = prediction_error(test.r,pr.nn_)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
```

```
##          0  422   81
##          1 1676  148
##
##                Accuracy : 0.2449506
##                  95% CI : (0.2275921, 0.2629534)
##     No Information Rate : 0.90159
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.0371898
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.20114395
##             Specificity : 0.64628821
##          Pos Pred Value : 0.83896620
##          Neg Pred Value : 0.08114035
##              Prevalence : 0.90159003
##          Detection Rate : 0.18134938
##    Detection Prevalence : 0.21615814
##       Balanced Accuracy : 0.42371608
##
##        'Positive' Class : 0
##
```

```r
print(nn_error)
```

```
## [1] 0.7550494199
```

It's still not doing well. To further tune the network, the hidden layer setup needs to be optimized. By running a random sample of hidden layers and nodes, we can hope to find one that fits the data better without over or underfitting. The code is commented out for computational reasons.

```r
#
# nn_func=function(nodes){
#   set.seed(1)
#   nn <- neuralnet(f,data=train_,hidden=nodes)
#   pr.nn <- compute(nn,test_[,2:ncol(test_)])
#   # pr.nn
#   pr.nn_ <- pr.nn$net.result*(max(recession_target$target)-min(recession_target$target))+min(recessio
#   test.r <- (test_$target)*(max(recession_target$target)-min(recession_target$target))+min(recession_
#   MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)
#   # MSE.nn
#   nn_error = prediction_error(test.r,pr.nn_)
#   return(nn_error)
# }
#
# iterations = 10
# #randomly select values around 30, 15, 6
# for(i in 1:iterations){
#
#   x1 <- floor(runif(1, 2, 25))
#   x2 <- floor(runif(1, 1, 16))
#   x3 <- floor(runif(1, -10, 10))
#
#   node_list=c()
#
#   if(x3>0){
```

```
#     node_list = c(x1,x2,x3)
#   }
#   else{
#     node_list = c(x1,x2)
#   }
#   nn_f = nn_func(node_list)
#   print(node_list)
#   print(nn_f)
# }
```

After running the random sampling of hidden layer setup, the best setup that was found is a two layer (19,3) setup.

```
n <- names(train_)
f <- as.formula(paste("target ~", paste(n[!n %in% "target"], collapse = " + ")))
set.seed(1)
nn <- neuralnet(f,data=train_,hidden=c(19,3),linear.output = FALSE,act.fct = "logistic")
pr.nn <- compute(nn,test_[,2:ncol(test_)])
# pr.nn
pr.nn_ <- pr.nn$net.result*(max(recession_target$target)-min(recession_target$target))+min(recession_ta
test.r <- (test_$target)*(max(recession_target$target)-min(recession_target$target))+min(recession_targe


nn_error = prediction_error(test.r,pr.nn_)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1269   99
##          1  829  130
##
##                Accuracy : 0.6012033
##                  95% CI : (0.5809765, 0.6211748)
##     No Information Rate : 0.90159
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0713032
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.6048618
##             Specificity : 0.5676856
##          Pos Pred Value : 0.9276316
##          Neg Pred Value : 0.1355579
##              Prevalence : 0.9015900
##          Detection Rate : 0.5453373
##    Detection Prevalence : 0.5878814
##       Balanced Accuracy : 0.5862737
##
##        'Positive' Class : 0
##
```

```
print(nn_error)
```

```
## [1] 0.398796734
```

After running our "best" setup, we still do not reach results as good as the previous methods. It actually has a lower accuracy rating than if we were to predict all 0s. This could be due to many reasons including, but not limitted to, over/underfitting and not enough data. The neural network does not seem fit for this data.

## Cross Validation

By running cross validation on each of our three models, we can get a better idea of how they perform. The distribution of the prediction error is also plotted.

```
#normalize data

maxs <- apply(rec_data_minimized, 2, max)
mins <- apply(rec_data_minimized, 2, min)

scaled <- as.data.frame(scale(rec_data_minimized, center = mins, scale = maxs - mins))

set.seed(34)


pred_error = NULL
k <- 5


pbar <- create_progress_bar('text')
pbar$init(k)
```

```
##
  |
  |                                                                  |   0%
```

```
for(i in 1:k){
    index <- sample(1:nrow(scaled),round(0.8*nrow(scaled)))
    train.cv <- scaled[index,]
    test.cv <- scaled[-index,]

    nn <- neuralnet(f,data=train.cv,hidden=c(19,3),linear.output=FALSE)

    pr.nn <- compute(nn,test.cv[,2:ncol(test.cv)])
    pr.nn <- pr.nn$net.result*(max(rec_data_minimized$target)-min(rec_data_minimized$target))+min(rec_da

    test.cv.r <- (test.cv$target)*(max(rec_data_minimized$target)-min(rec_data_minimized$target))+min(re

    pred_error[i] = prediction_error(test.cv.r,pr.nn)


    pbar$step()
}
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 517  69
##          1 712  71
```

```
##
##                Accuracy : 0.4295106
##                  95% CI : (0.4030992, 0.4562252)
##     No Information Rate : 0.8977356
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.0237871
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.42066721
##             Specificity : 0.50714286
##          Pos Pred Value : 0.88225256
##          Neg Pred Value : 0.09067688
##              Prevalence : 0.89773557
##          Detection Rate : 0.37764792
##    Detection Prevalence : 0.42804967
##       Balanced Accuracy : 0.46390503
##
##        'Positive' Class : 0
##
##
  |
  |=============                                                        |  20%Confusion Matrix and Statisti
##
##           Reference
## Prediction   0   1
##          0 527  56
##          1 712  74
##
##                Accuracy : 0.4390066
##                  95% CI : (0.4125081, 0.4657674)
##     No Information Rate : 0.9050402
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.0016653
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.42534302
##             Specificity : 0.56923077
##          Pos Pred Value : 0.90394511
##          Neg Pred Value : 0.09414758
##              Prevalence : 0.90504018
##          Detection Rate : 0.38495252
##    Detection Prevalence : 0.42585829
##       Balanced Accuracy : 0.49728689
##
##        'Positive' Class : 0
##
##
  |
  |==========================                                           |  40%Confusion Matrix and Statisti
##
##           Reference
## Prediction    0    1
```

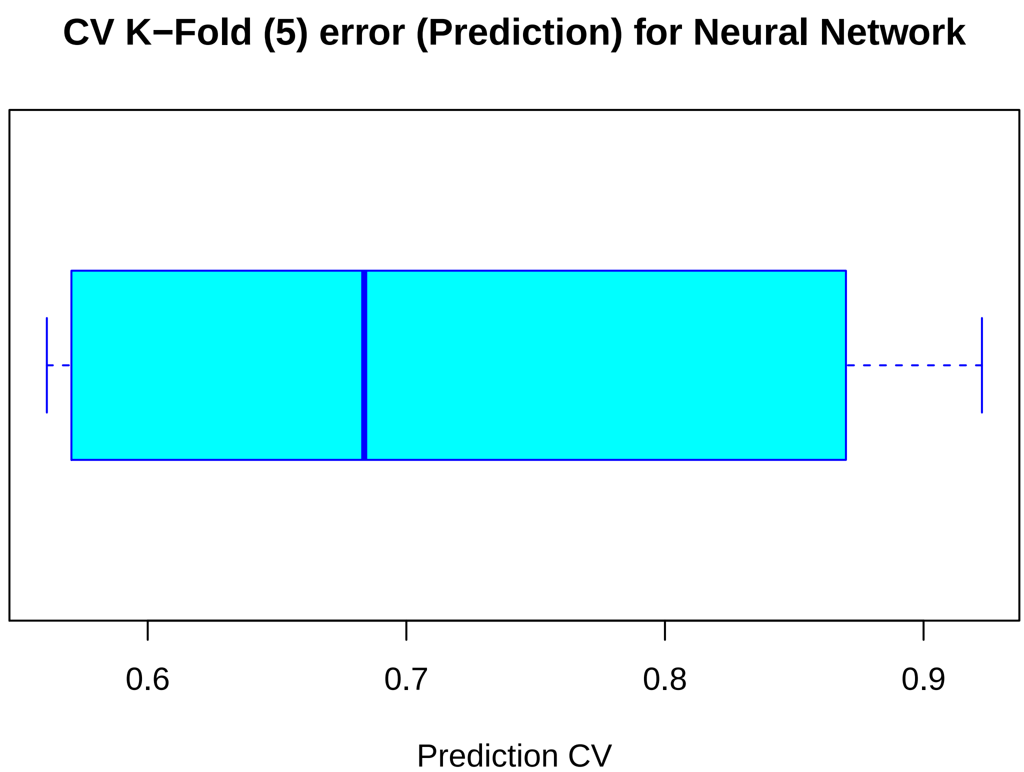
```
##          0   88   46
##          1 1145   90
##
##                Accuracy : 0.1300219
##                  95% CI : (0.1126494, 0.1489991)
##     No Information Rate : 0.9006574
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.05808
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.07137064
##             Specificity : 0.66176471
##          Pos Pred Value : 0.65671642
##          Neg Pred Value : 0.07287449
##              Prevalence : 0.90065741
##          Detection Rate : 0.06428050
##    Detection Prevalence : 0.09788167
##       Balanced Accuracy : 0.36656767
##
##        'Positive' Class : 0
##
##
  |
  |=======================================                           |  60%Confusion Matrix and Statisti
##
##           Reference
## Prediction    0    1
##          0   28   62
##          1 1201   78
##
##                Accuracy : 0.0774288
##                  95% CI : (0.0638248, 0.0928778)
##     No Information Rate : 0.8977356
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.0912318
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.02278275
##             Specificity : 0.55714286
##          Pos Pred Value : 0.31111111
##          Neg Pred Value : 0.06098514
##              Prevalence : 0.89773557
##          Detection Rate : 0.02045289
##    Detection Prevalence : 0.06574142
##       Balanced Accuracy : 0.28996280
##
##        'Positive' Class : 0
##
##
  |
  |====================================================              |  80%Confusion Matrix and Statisti
##
```

```
##           Reference
## Prediction   0   1
##          0 363  58
##          1 878  70
##
##                Accuracy : 0.3162893
##                  95% CI : (0.2917069, 0.3416628)
##     No Information Rate : 0.9065011
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.0414742
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.29250604
##             Specificity : 0.54687500
##          Pos Pred Value : 0.86223278
##          Neg Pred Value : 0.07383966
##              Prevalence : 0.90650110
##          Detection Rate : 0.26515705
##    Detection Prevalence : 0.30752374
##       Balanced Accuracy : 0.41969052
##
##        'Positive' Class : 0
##
##
  |
  |=================================================================| 100%
```

```r
mean(pred_error)
```

```
## [1] 0.7215485756
```

```r
boxplot(pred_error,xlab='Prediction CV',col='cyan',
        border='blue',names='CV error (Prediction Error)',
        main='CV K-Fold (5) error (Prediction) for Neural Network',horizontal=TRUE)
```




```r
var(pred_error)
```

```
## [1] 0.02811748831
```

```r
set.seed(34)


pred_error = NULL
k <- 5


pbar <- create_progress_bar('text')
pbar$init(k)
```

```
##
  |
  |                                                                  |   0%
```

```r
for(i in 1:k){
    index <- sample(1:nrow(rec_data_minimized),round(0.8*nrow(rec_data_minimized)))
    train.cv <- rec_data_minimized[index,]
    test.cv <- rec_data_minimized[-index,]
    x <- model.matrix( ~ .-1, train.cv[ , -y_index])
    y <- data.matrix(train.cv[, y_index])
    test_x = test.cv[,-1]
    test_x_matrix = model.matrix( ~ .-1, test.cv[,-y_index])
    model.lasso <- cv.glmnet(x, y, family='binomial', alpha=1, parallel=TRUE, standardize=TRUE)
```

```r
    lasso_test = predict(model.lasso, newx=test_x_matrix,type="link")
    pred_error[i] = prediction_error(test.cv$target, lasso_test)
    pbar$step()
}
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1202   96
##          1   27   44
##
##                Accuracy : 0.9101534
##                  95% CI : (0.8937461, 0.9247736)
##     No Information Rate : 0.8977356
##     P-Value [Acc > NIR] : 0.06855261
##
##                   Kappa : 0.3739772
##  Mcnemar's Test P-Value : 0.0000000008713454
##
##             Sensitivity : 0.9780309
##             Specificity : 0.3142857
##          Pos Pred Value : 0.9260401
##          Neg Pred Value : 0.6197183
##              Prevalence : 0.8977356
##          Detection Rate : 0.8780131
##    Detection Prevalence : 0.9481373
##       Balanced Accuracy : 0.6461583
##
##        'Positive' Class : 0
##
##
  |
  |=============                                                    |  20%Confusion Matrix and Statisti
##
##           Reference
## Prediction    0    1
##          0 1204  100
##          1   23   42
##
##                Accuracy : 0.9101534
##                  95% CI : (0.8937461, 0.9247736)
##     No Information Rate : 0.8962747
##     P-Value [Acc > NIR] : 0.04833643
##
##                   Kappa : 0.3643927
##  Mcnemar's Test P-Value : 0.000000000007247442
##
##             Sensitivity : 0.9812551
##             Specificity : 0.2957746
##          Pos Pred Value : 0.9233129
##          Neg Pred Value : 0.6461538
##              Prevalence : 0.8962747
##          Detection Rate : 0.8794741
```

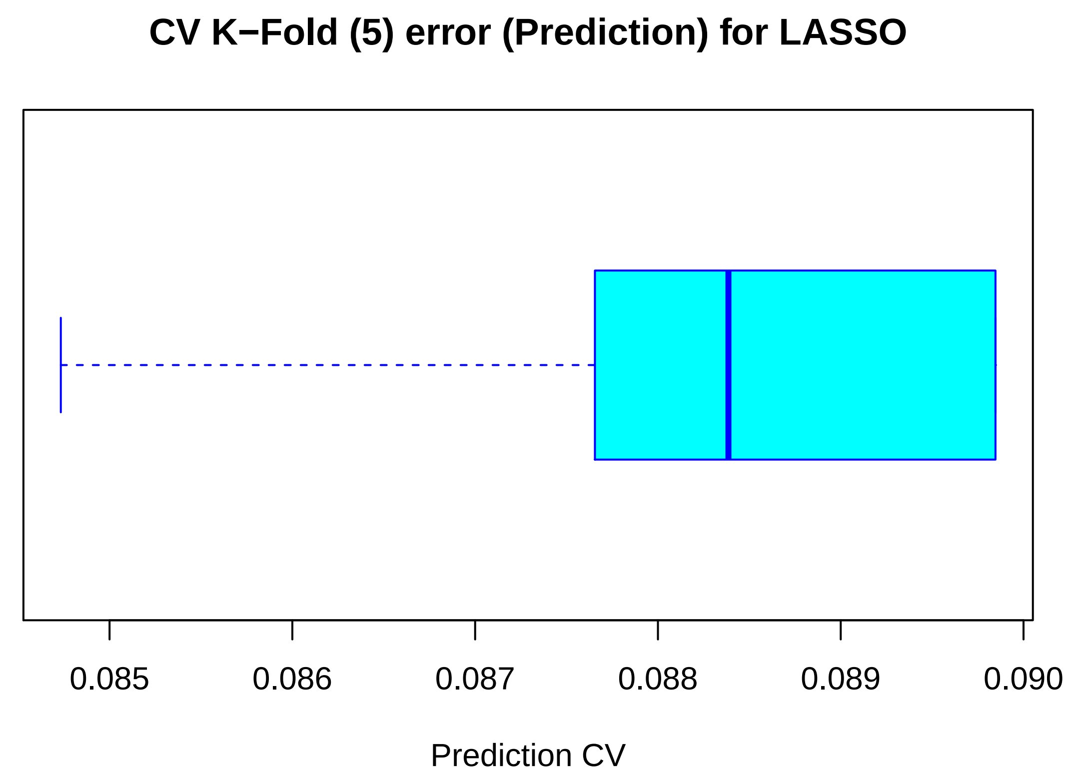
```
##    Detection Prevalence : 0.9525201
##       Balanced Accuracy : 0.6385149
##
##        'Positive' Class : 0
##
##
  |
  |==========================                                        |  40%Confusion Matrix and Statisti
##
##           Reference
## Prediction    0    1
##          0 1203   95
##          1   25   46
##
##                Accuracy : 0.9123448
##                  95% CI : (0.8960998, 0.9267925)
##     No Information Rate : 0.8970051
##     P-Value [Acc > NIR] : 0.03198076
##
##                   Kappa : 0.3920194
##  Mcnemar's Test P-Value : 0.0000000002999405
##
##             Sensitivity : 0.9796417
##             Specificity : 0.3262411
##          Pos Pred Value : 0.9268105
##          Neg Pred Value : 0.6478873
##              Prevalence : 0.8970051
##          Detection Rate : 0.8787436
##    Detection Prevalence : 0.9481373
##       Balanced Accuracy : 0.6529414
##
##        'Positive' Class : 0
##
##
  |
  |=======================================                           |  60%Confusion Matrix and Statisti
##
##           Reference
## Prediction    0    1
##          0 1216   93
##          1   23   37
##
##                Accuracy : 0.9152666
##                  95% CI : (0.8992424, 0.92948)
##     No Information Rate : 0.9050402
##     P-Value [Acc > NIR] : 0.1052756
##
##                   Kappa : 0.3505215
##  Mcnemar's Test P-Value : 0.0000000001489087
##
##             Sensitivity : 0.9814366
##             Specificity : 0.2846154
##          Pos Pred Value : 0.9289534
##          Neg Pred Value : 0.6166667
```

```
##              Prevalence : 0.9050402
##          Detection Rate : 0.8882396
##    Detection Prevalence : 0.9561724
##       Balanced Accuracy : 0.6330260
##
##        'Positive' Class : 0
##
##
  |
  |====================================================             |  80%Confusion Matrix and Statisti
##
##           Reference
## Prediction    0    1
##          0 1208  110
##          1   11   40
##
##                Accuracy : 0.9116143
##                  95% CI : (0.8953149, 0.9261199)
##     No Information Rate : 0.890431
##     P-Value [Acc > NIR] : 0.00571802
##
##                   Kappa : 0.3625673
##  Mcnemar's Test P-Value : < 0.000000000000000222
##
##             Sensitivity : 0.9909762
##             Specificity : 0.2666667
##          Pos Pred Value : 0.9165402
##          Neg Pred Value : 0.7843137
##              Prevalence : 0.8904310
##          Detection Rate : 0.8823959
##    Detection Prevalence : 0.9627465
##       Balanced Accuracy : 0.6288214
##
##        'Positive' Class : 0
##
##
  |
  |=================================================================| 100%
```

```r
mean(pred_error)
```

```
## [1] 0.0880934989
```

```r
boxplot(pred_error,xlab='Prediction CV',col='cyan',
        border='blue',names='CV error (Prediction Error)',
        main='CV K-Fold (5) error (Prediction) for LASSO',horizontal=TRUE)
```




```r
var(pred_error)
```

```
## [1] 0.000004428648339
```

```r
set.seed(34)


pred_error = NULL
k <- 5


pbar <- create_progress_bar('text')
pbar$init(k)
```

```
##
  |
  |                                                                      |   0%
```

```r
for(i in 1:k){
    index <- sample(1:nrow(rec_data_minimized),round(0.8*nrow(rec_data_minimized)))
    train.cv <- rec_data_minimized[index,]
    test.cv <- rec_data_minimized[-index,]
    tree_fit <- randomForest(target ~ .,   data=train.cv)
    tree_test = predict(tree_fit, newdata=test.cv)
    pred_error[i] = prediction_error(test.cv$target,tree_test)
    pbar$step()
}
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1180    0
##          1   49  140
##
##                Accuracy : 0.9642075
##                  95% CI : (0.9529544, 0.9734054)
##     No Information Rate : 0.8977356
##     P-Value [Acc > NIR] : < 0.00000000000000022204
##
##                   Kappa : 0.8312347
##  Mcnemar's Test P-Value : 0.000000000007025137
##
##             Sensitivity : 0.9601302
##             Specificity : 1.0000000
##          Pos Pred Value : 1.0000000
##          Neg Pred Value : 0.7407407
##              Prevalence : 0.8977356
##          Detection Rate : 0.8619430
##    Detection Prevalence : 0.8619430
##       Balanced Accuracy : 0.9800651
##
##        'Positive' Class : 0
##
##
  |
  |=============                                                     |  20%Confusion Matrix and Statisti
##
##           Reference
## Prediction    0    1
##          0 1175    0
##          1   45  149
##
##                Accuracy : 0.9671293
##                  95% CI : (0.9562618, 0.9759249)
##     No Information Rate : 0.8911614
##     P-Value [Acc > NIR] : < 0.00000000000000022204
##
##                   Kappa : 0.8503843
##  Mcnemar's Test P-Value : 0.00000000005412161
##
##             Sensitivity : 0.9631148
##             Specificity : 1.0000000
##          Pos Pred Value : 1.0000000
##          Neg Pred Value : 0.7680412
##              Prevalence : 0.8911614
##          Detection Rate : 0.8582907
##    Detection Prevalence : 0.8582907
##       Balanced Accuracy : 0.9815574
##
##        'Positive' Class : 0
##
```

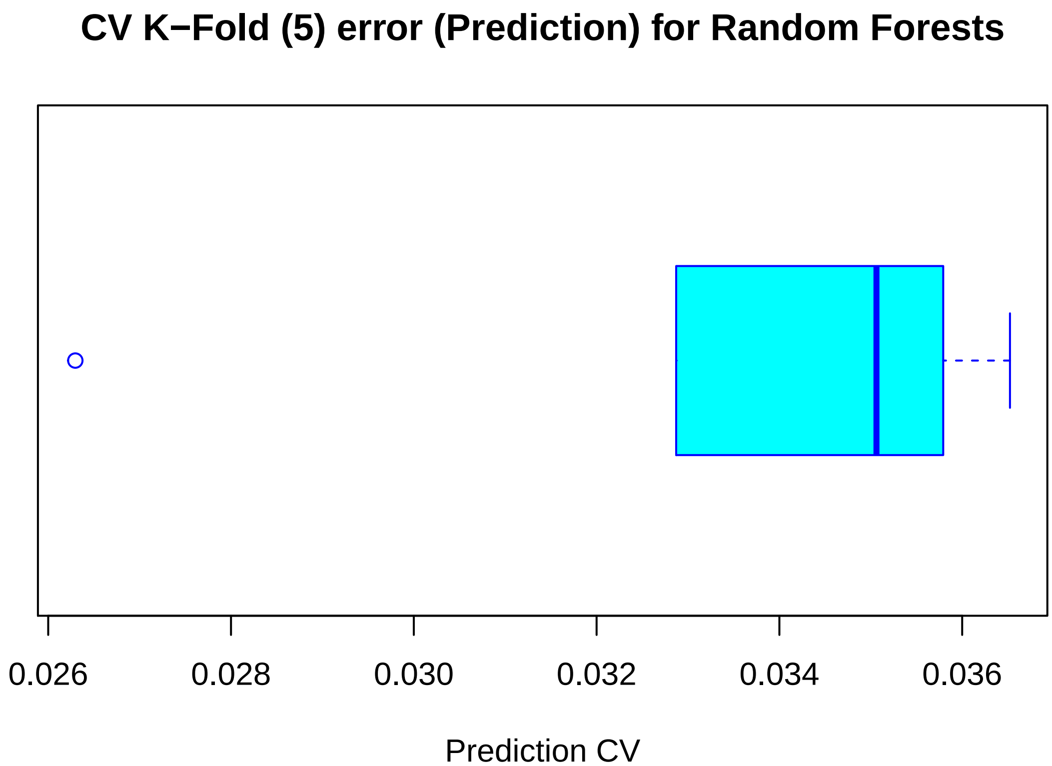
```
##
  |
  |==========================                                        |  40%Confusion Matrix and Statisti
##
##           Reference
## Prediction    0    1
##          0 1168    0
##          1   48  153
##
##                Accuracy : 0.9649379
##                  95% CI : (0.9537796, 0.974037)
##     No Information Rate : 0.8882396
##     P-Value [Acc > NIR] : < 0.00000000000000022204
##
##                   Kappa : 0.8446965
##  Mcnemar's Test P-Value : 0.00000000001170021
##
##             Sensitivity : 0.9605263
##             Specificity : 1.0000000
##          Pos Pred Value : 1.0000000
##          Neg Pred Value : 0.7611940
##              Prevalence : 0.8882396
##          Detection Rate : 0.8531775
##    Detection Prevalence : 0.8531775
##       Balanced Accuracy : 0.9802632
##
##        'Positive' Class : 0
##
##
  |
  |=======================================                           |  60%Confusion Matrix and Statisti
##
##           Reference
## Prediction    0    1
##          0 1189    0
##          1   50  130
##
##                Accuracy : 0.963477
##                  95% CI : (0.9521303, 0.9727727)
##     No Information Rate : 0.9050402
##     P-Value [Acc > NIR] : < 0.00000000000000022204
##
##                   Kappa : 0.8187187
##  Mcnemar's Test P-Value : 0.000000000004218937
##
##             Sensitivity : 0.9596449
##             Specificity : 1.0000000
##          Pos Pred Value : 1.0000000
##          Neg Pred Value : 0.7222222
##              Prevalence : 0.9050402
##          Detection Rate : 0.8685172
##    Detection Prevalence : 0.8685172
##       Balanced Accuracy : 0.9798224
##
```

```
##        'Positive' Class : 0
##
##
  |
  |====================================================             |  80%Confusion Matrix and Statisti
##
##           Reference
## Prediction    0    1
##          0 1203    0
##          1   36  130
##
##                Accuracy : 0.9737034
##                  95% CI : (0.963779, 0.9815158)
##     No Information Rate : 0.9050402
##     P-Value [Acc > NIR] : < 0.00000000000000022204
##
##                   Kappa : 0.8638804
##  Mcnemar's Test P-Value : 0.000000005433087
##
##             Sensitivity : 0.9709443
##             Specificity : 1.0000000
##          Pos Pred Value : 1.0000000
##          Neg Pred Value : 0.7831325
##              Prevalence : 0.9050402
##          Detection Rate : 0.8787436
##    Detection Prevalence : 0.8787436
##       Balanced Accuracy : 0.9854722
##
##        'Positive' Class : 0
##
##
  |
  |=================================================================| 100%
```

```r
mean(pred_error)
```

```
## [1] 0.03330898466
```

```r
boxplot(pred_error,xlab='Prediction CV',col='cyan',
        border='blue',names='CV error (Prediction Error)',
        main='CV K-Fold (5) error (Prediction) for Random Forests',horizontal=TRUE)
```




```r
var(pred_error)
```

```
## [1] 0.00001723437848
```

We are seeing similar results we saw in the test set. The neural network is performing very poorly. LASSO looks to be doing good, but when only considering correctly and incorrectly classifying 1 (excluding predicting 0 corrrectly), it does not perform well. Decision trees are also overclassifying 1s, but that is not a bad thing as it does not missclassify a 0 as a 1 and is taking a "cautious", as defined by being rather safe than sorry, approach.

## Comparing Machine Learning Methods

By looking at each machine learning methods' confusion matrix and their cross validation results, the decision trees with a random forest implementation performs the best. It was able to correctly classify most of the recession days and did not classify an actual recession day as being not in a recession. It did overclassify the recession, however, predicting there would be a recession day when in actuallity there was not a recession that day. This makes it a more cautious approach from the human perspective. The computer does not know that 1 is bad and 0 is good which is why we need humans to look at the confusion matrix and conclude that it is a cautious model.

## Conclusion

In conclusion, we should definately be using machine learning opposed to ARIMA for predicting an oncoming recession. ARIMA only tells us that we will be in a recession tomorrow if we are in one today and we will not be in a recession tomorrow if we are not in one today. Being economists, statisticians, and data scientists, we know that this cannot be used to predict an oncoming recession, but rather just tells us what we already know. This is why machine learning is very powerful. It can give us a probability that there is a recession regardless of the previous day's recession status.

Through the different machine learning methods, it was determined that random forests were the best for classifying a recession. The confusion matricies showed how accurate it actually was in correctly predicting a recession. After combining the results with the economic domain knowledge, the random forest model is considered a "cautious" model that it incorrectly classifies days as a recession when it actually was a non-recession day, but does not incorrectly mark days a non-recessionary days when they actually were. The importance plot from the random forest model shows that the biggest determinants for a recession is the presidential approval and dissaproval ratings.

Combing these results with the results from my capstone project, the final conclusion is that ARIMA is not as powerful as machine learning and my prediction is that ARIMA will one day be phased out and surpassed completely by machine learning.