HW5: Problem 1

```
library("jpeg")
library("plyr")
library(e1071)
library(tidyverse) # %>%
library(tree)
library(gbm)
library(randomForest)
library(glmnet)
library(ggplot2)
library(class)
library(xgboost)
library(ggcorrplot)
library(TTR)
library(quantmod)
```

Problem 1: Fault Classification

```
setwd("casting_100x100")

def_paths <- dir("def_front")
ok_paths <- dir("ok_front")

data_def = sapply(1:length(def_paths), function(x){
  tmp = readJPEG(paste0("def_front/", def_paths[x]))[,,1]
  tmp = as.vector(tmp)
  return(tmp)
})

data_ok = sapply(1:length(ok_paths), function(x){
  tmp = readJPEG(paste0("ok_front/", ok_paths[x]))[,,1]
  tmp = as.vector(tmp)
  return(tmp)
})

data = as.data.frame(rbind(t(data_def), t(data_ok)))
data$y = as.factor(c(rep(1,length(def_paths)), rep(0,length(ok_paths))))
dim(data)
```

```
## [1]  6633 10001
```

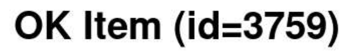
```
## Data Preprocessing

show_image = function(img_cast, col = gray(1:20/20), ...){
  image(matrix(as.matrix(img_cast), ncol = 100, byrow = TRUE)[, 100:1], col = col, ...)
}

## show images for a defect item and a good (OK) item
par(mfrow = c(1,1), pty="s")
show_image(data[1,-10001], col = gray(1:20/20), axes=F); box(); title("Defect Item (id=1)")
```

Defect Item (id=1)

```
show_image(data[length(def_paths)+1,-10001], col = gray(1:20/20), axes=F); box(); title("OK I
tem (id=3759)")
```




```
### For testing only!!!
#data <- rbind(data[1:10,],data[6623:6633,])
```

1. Dimension Reduction

It is really impossible to take all 10000 pixels into consideration. Hence, I prefer a PCA dimension reduction before any further classification effort.

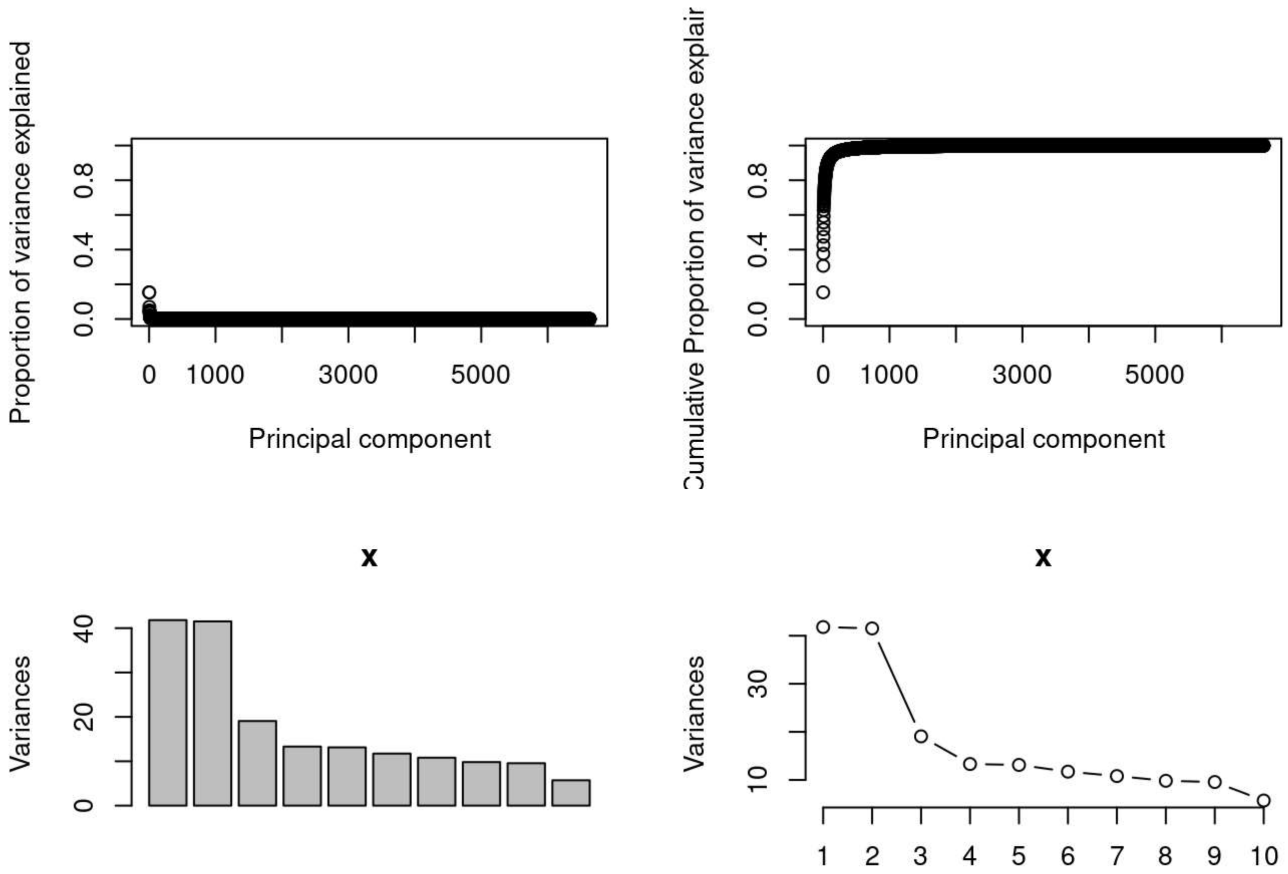
```
pcaCharts <- function(x) {
    x.var <- x$sdev ^ 2
    x.pvar <- x.var/sum(x.var)
    #print("proportions of variance:")
    #print(x.pvar)

    par(mfrow=c(2,2))
    plot(x.pvar,xlab="Principal component", ylab="Proportion of variance explained", ylim=c
(0,1), type='b')
    plot(cumsum(x.pvar),
         xlab="Principal component",
         ylab="Cumulative Proportion of variance explained",
         ylim=c(0,1),
         type='b')
    screeplot(x)
    screeplot(x,type="l")
    par(mfrow=c(1,1))
}
```

```
find_pc_num <- function(x) { # 60% of cumulative proportion of variance
    x.var <- x$sdev ^ 2
    x.pvar <- x.var/sum(x.var)
    min(which(cumsum(x.pvar) >0.6))
}
```

```
data.pca <- prcomp(data[,-10001])
```

```
pcaCharts(data.pca)
```



Here, I select the PCs such that they accumulate to 80% of variance.

```
pc.use <- find_pc_num(data.pca) # explains 60% of variance
data.pca <- data.pca$x[,1:pc.use] %*% t(data.pca$rotation[,1:pc.use])
```

Then, conduct the train-test-split.

```
# Train test split
set.seed(48763)
train_index <- sample(1:nrow(data.pca),(0.7*nrow(data.pca)))
train <- data.pca[train_index,]
test <- data.pca[-train_index,]
label <- data[,10001]
x_train <- train[,-10001]
y_train <- label[train_index]
x_test <- test[,-10001]
y_test <- label[-train_index]
rm(train, test) # Throw to garbage
```

2. SVM

```
### tuning parameters (cost,gamma) via 10-fold CV
# set.seed(1)
# tune.out <- tune(svm,
#                  train.x = x_train,
#                  train.y = y_train,
#                  ranges = list( # No need to scale in this case
#                    cost = c(0.1, 1, 10, 100, 1000),
#                    gamma = c(0.5, 1, 2, 3, 4),
#                    kernel = c('radial', 'linear', 'polynomial', 'sigmoid')
#                    )
#                  )
# summary(tune.out)
```

```
# table(true = y_test,
#       pred = predict(tune.out$best.model, newdata = x_test))
```

```
svm_pca <- svm(x = x_train,
               y = y_train,
               kernel = 'linear',
               cost = 10,
               gamma = 1,
               scale = TRUE)
```

```
table(true = y_test, pred = predict(svm_pca, newdata = x_test))
```

```
##     pred
## true   0   1
##    0 753 116
##    1 172 949
```

```
mean(y_test==predict(svm_pca, newdata = x_test))
```

```
## [1] 0.8552764
```

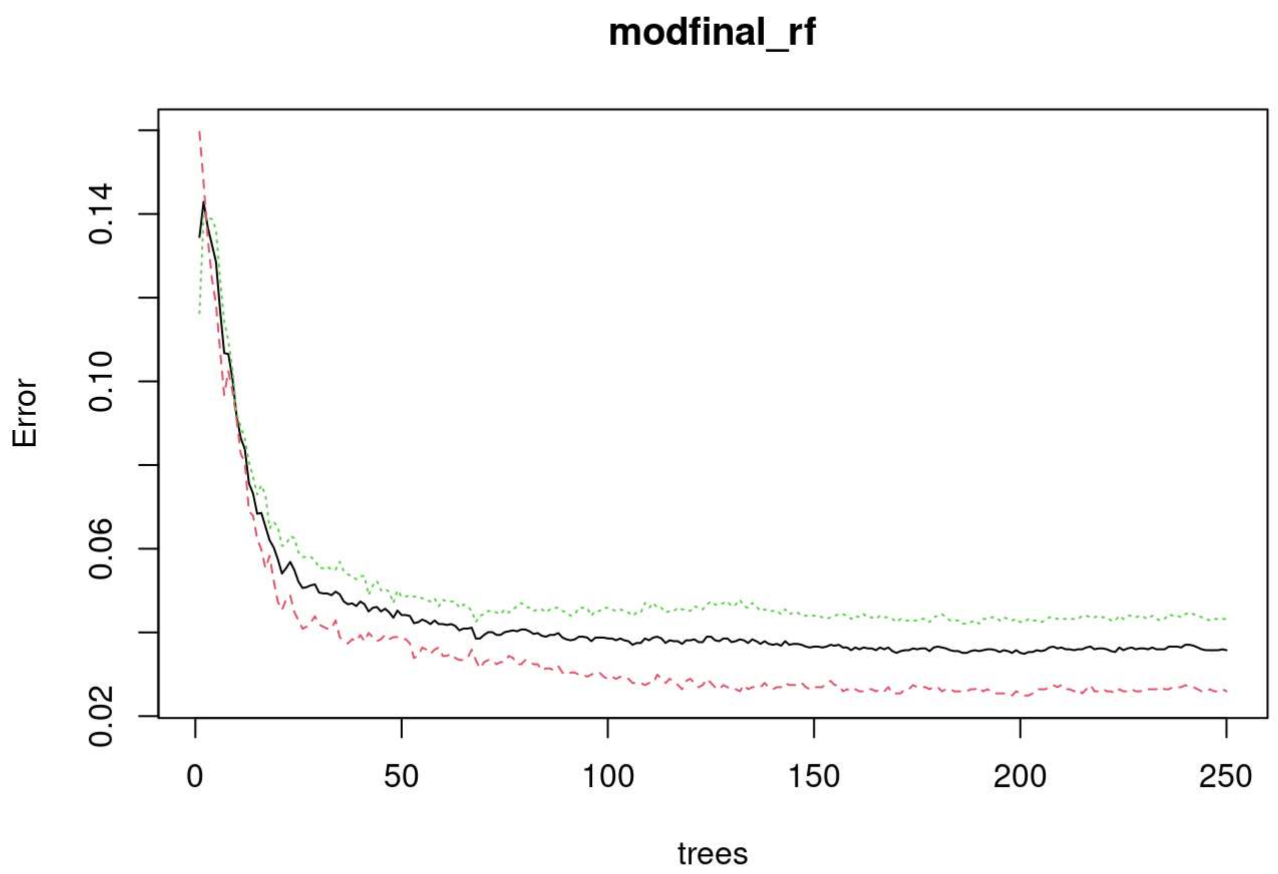
This parameter combination provides a 85.52% test accuracy.

3. Random Forest

```
train_rf <- data[train_index,]
test_rf <- data[-train_index,]
```

Since there are 10000 features, I take \(\texttt{mtry}=\lceil\sqrt{10000}\rceil=100\).

```
modfinal_rf <- randomForest(y~.,
                            data=train_rf,
                            mtry = 100,
                            ntree = 250,
                            importance = TRUE,
                            proximity = TRUE)
plot(modfinal_rf )
```




```
table(true = test_rf[,10001],
      pred = predict(modfinal_rf, newdata = test_rf[,-10001]))
```

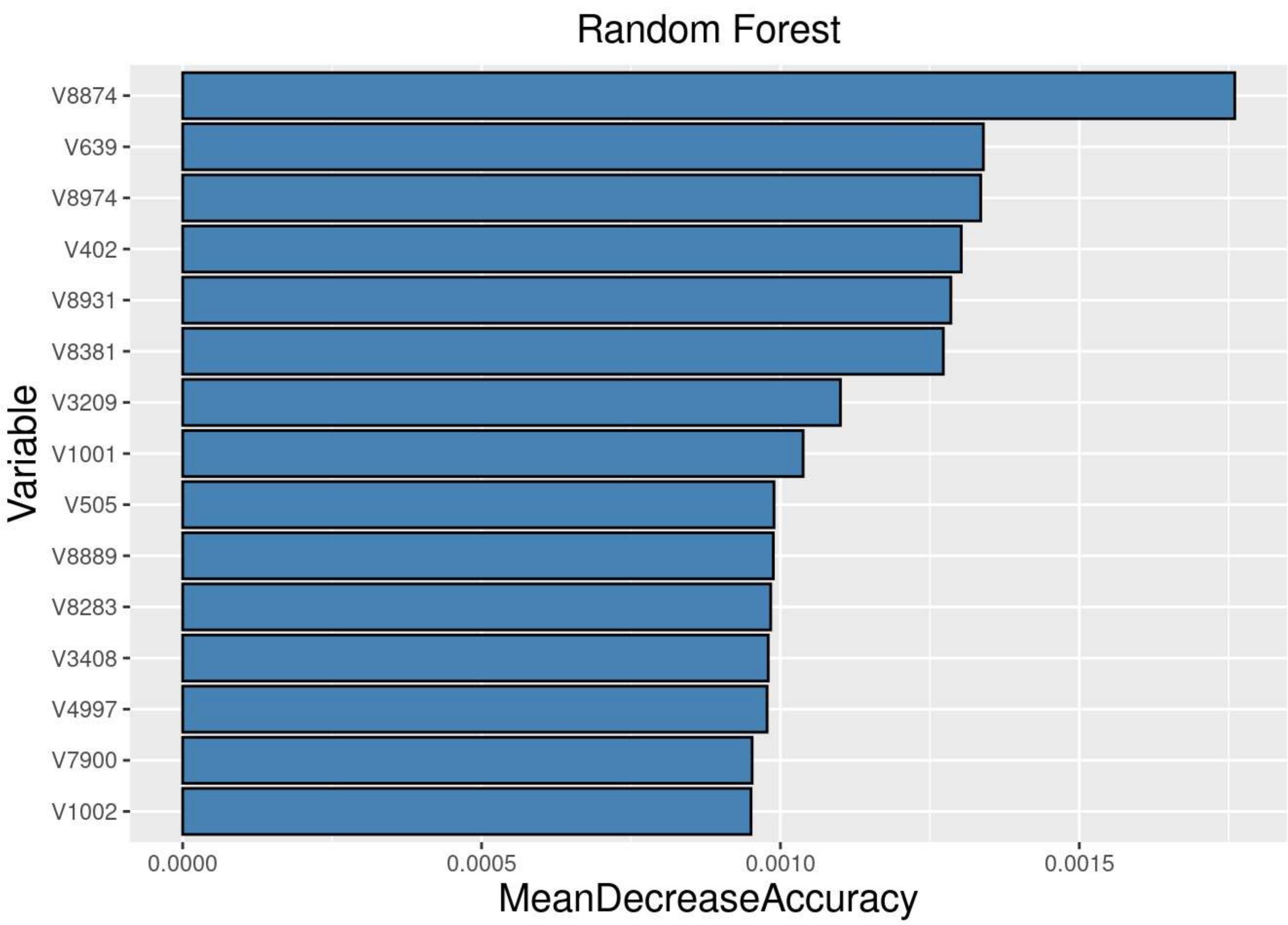
```
##     pred
## true    0    1
##    0  844   25
##    1   44 1077
```

```
(844+1077)/(844+1077+25+44)
```

```
## [1] 0.9653266
```

The random forest outperforms SVM. It gives a 96.53% test accuracy. Next, I investigate the variable importance, and interpret in the summary.

```
modfinal_rf$importance %>%
  as.matrix() %>%
  as.data.frame() %>%
  add_rownames() %>%
  `colnames<-`(c("varname",'No','Yes',"MeanDecreaseAccuracy",'MeanDecreaseGini')) %>%
  arrange(desc(MeanDecreaseAccuracy)) %>%
  top_n(15,wt = MeanDecreaseAccuracy) %>%
  ggplot(aes(x = reorder(varname, MeanDecreaseAccuracy),y = MeanDecreaseAccuracy)) +
  geom_col(fill = 'steelblue', color = 'black') +
  coord_flip() +
  ggtitle(label = "Random Forest") +
  xlab('Variable') +
  ylab('MeanDecreaseAccuracy') +
  theme(plot.title=element_text(hjust=0.5,size=15),
        axis.title=element_text(size=15))
```



4. Summary

To detect the defected items, we do not need to take The defect items do not need every pixeld into consideration. Actually, By the above variable importance plots, 7~9 PCs should be enough for random forest.

```
rm(x_train, y_train, x_test, y_test, data_def, data_ok, train_rf, test_rf, data, data.pca)
```

HW5: Problem 2)

```r
library("jpeg")
library("plyr")
library(e1071)
library(tidyverse) # %>%
library(tree)
library(gbm)
library(randomForest)
library(glmnet)
library(ggplot2)
library(class)
library(xgboost)
library(ggcorrplot)
library(TTR)
library(quantmod)
```

1 Data preprossesing

```r
getSymbols("SPY", src = "yahoo", from = as.Date("2010-01-01"), to = as.Date("2022-12-31"))
```

(1) Get data

```
## [1] "SPY"
```

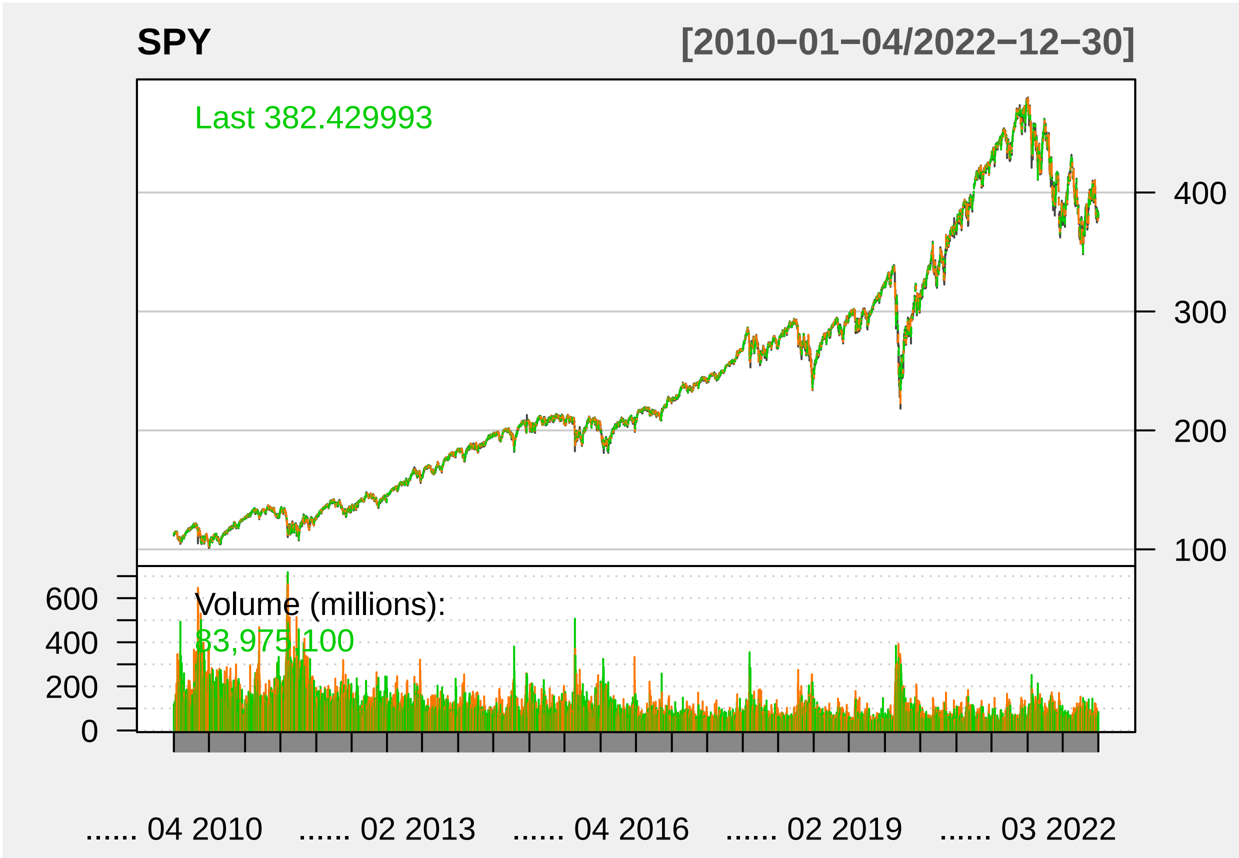
```r
head(SPY);
```

```
##            SPY.Open SPY.High SPY.Low SPY.Close SPY.Volume SPY.Adjusted
## 2010-01-04   112.37   113.39  111.51    113.33  118944600     88.45418
## 2010-01-05   113.26   113.68  112.85    113.63  111579900     88.68835
## 2010-01-06   113.52   113.99  113.43    113.71  116074400     88.75079
## 2010-01-07   113.50   114.33  113.18    114.19  131091100     89.12543
## 2010-01-08   113.89   114.62  113.66    114.57  126402800     89.42203
## 2010-01-11   115.08   115.13  114.24    114.73  106375700     89.54690
```

```r
tail(SPY)
```

```
##            SPY.Open SPY.High SPY.Low SPY.Close SPY.Volume SPY.Adjusted
## 2022-12-22   383.05   386.21  374.77    380.72  100120900       380.72
## 2022-12-23   379.65   383.06  378.03    382.91   59857300       382.91
## 2022-12-27   382.79   383.15  379.65    381.40   51638200       381.40
## 2022-12-28   381.33   383.39  376.42    376.66   70911500       376.66
## 2022-12-29   379.63   384.35  379.08    383.44   66970900       383.44
## 2022-12-30   380.64   382.58  378.43    382.43   83975100       382.43
```

```
chartSeries(SPY, theme = chartTheme("white"))
```



(2) Data Preprocessing I'm currently investing on stocks, so I knew some indicators. My favorites are KD and MACD, so lets introduce them into our dataset.

```
# Add MACD and RSI
macd  <- MACD(SPY[,"SPY.Close"], 12, 26, 9, maType="EMA" )
rsi <- RSI(SPY[,"SPY.Close"])

# compute daily returns:
rr <- exp(diff(log(SPY$SPY.Close)))-1
Direction <- ifelse(rr >= 0, "Up", "Down")
names(Direction) <- "Direction"

# create lag variables (you may use more lags or other info from SPY object)
rr.Lag <- Lag(rr, 1:5)
Volume.Lag <- Lag(SPY$SPY.Volume,1:5)

#For Task 1 (respones variable: rr):
SPY.return <- data.frame(rr, rr.Lag, Volume.Lag/10^9) #rescale Volume such that all scales are comparab

SPY.return <- cbind(SPY.return, macd, rsi)
names(SPY.return) <- c("r", "r.Lag.1", "r.Lag.2", "r.Lag.3",  "r.Lag.4", "r.Lag.5",
                       "v.Lag.1", "v.Lag.2", "v.Lag.3",  "v.Lag.4", "v.Lag.5",
```

```r
                        "macd", "signal", "rsi")

SPY.return = na.omit(SPY.return) #remove NA's
apply(SPY.return[,-1:-3],2,mean)
```

```
##      r.Lag.3      r.Lag.4      r.Lag.5      v.Lag.1      v.Lag.2      v.Lag.3
## 4.468412e-04 4.529167e-04 4.508838e-04 1.172993e-01 1.173385e-01 1.173687e-01
##      v.Lag.4      v.Lag.5         macd       signal          rsi
## 1.174020e-01 1.174775e-01 2.611090e-01 2.593340e-01 5.537146e+01
```

```r
apply(SPY.return[,-1:-3],2,sd)
```

```
##     r.Lag.3     r.Lag.4     r.Lag.5     v.Lag.1     v.Lag.2     v.Lag.3
##  0.01112860  0.01113158  0.01113121  0.07048627  0.07049350  0.07049458
##     v.Lag.4     v.Lag.5        macd      signal         rsi
##  0.07048896  0.07055842  1.04797556  0.97478767 11.24619183
```

```r
head(SPY.return)
```

```
##                        r       r.Lag.1       r.Lag.2       r.Lag.3
## 2010-02-22  0.0001799982  0.0020737083  0.0058951749  0.0047385093
## 2010-02-23 -0.0121447099  0.0001799982  0.0020737083  0.0058951749
## 2010-02-24  0.0091977235 -0.0121447099  0.0001799982  0.0020737083
## 2010-02-25 -0.0013535643  0.0091977235 -0.0121447099  0.0001799982
## 2010-02-26  0.0006325111 -0.0013535643  0.0091977235 -0.0121447099
## 2010-03-01  0.0103846941  0.0006325111 -0.0013535643  0.0091977235
##                  r.Lag.4       r.Lag.5   v.Lag.1   v.Lag.2   v.Lag.3   v.Lag.4
## 2010-02-22  0.0157348851 -0.0008322945 0.2226849 0.1937086 0.1688451 0.1593175
## 2010-02-23  0.0047385093  0.0157348851 0.1323469 0.2226849 0.1937086 0.1688451
## 2010-02-24  0.0058951749  0.0047385093 0.2074970 0.1323469 0.2226849 0.1937086
## 2010-02-25  0.0020737083  0.0058951749 0.1763507 0.2074970 0.1323469 0.2226849
## 2010-02-26  0.0001799982  0.0020737083 0.2596347 0.1763507 0.2074970 0.1323469
## 2010-03-01 -0.0121447099  0.0001799982 0.1735893 0.2596347 0.1763507 0.2074970
##              v.Lag.5       macd     signal      rsi
## 2010-02-22 0.3046221 -0.8137820 -1.7200123 55.16748
## 2010-02-23 0.1593175 -0.7428162 -1.5245730 49.03458
## 2010-02-24 0.1688451 -0.6053894 -1.3407363 53.22420
## 2010-02-25 0.1937086 -0.5017218 -1.1729334 52.53349
## 2010-02-26 0.2226849 -0.4097214 -1.0202910 52.84106
## 2010-03-01 0.1323469 -0.2498190 -0.8661966 57.69130
```

```r
#For Task 2 (response variable: Direction):
SPY.trend <- data.frame(Direction, rr.Lag, Volume.Lag/10^9)
SPY.trend <- cbind(SPY.trend, macd, rsi)
names(SPY.trend) <- c("Direction", "r.Lag.1", "r.Lag.2", "r.Lag.3",  "r.Lag.4", "r.Lag.5",
                        "v.Lag.1", "v.Lag.2", "v.Lag.3",  "v.Lag.4", "v.Lag.5",
                        "macd", "signal", "rsi")
SPY.trend = na.omit(SPY.trend) #remove NA's
head(SPY.trend)
```

```
##            Direction       r.Lag.1       r.Lag.2       r.Lag.3       r.Lag.4
```

```
## 2010-02-22         Up  0.0020737083  0.0058951749  0.0047385093  0.0157348851
## 2010-02-23       Down  0.0001799982  0.0020737083  0.0058951749  0.0047385093
## 2010-02-24         Up -0.0121447099  0.0001799982  0.0020737083  0.0058951749
## 2010-02-25       Down  0.0091977235 -0.0121447099  0.0001799982  0.0020737083
## 2010-02-26         Up -0.0013535643  0.0091977235 -0.0121447099  0.0001799982
## 2010-03-01         Up  0.0006325111 -0.0013535643  0.0091977235 -0.0121447099
##                  r.Lag.5   v.Lag.1   v.Lag.2   v.Lag.3   v.Lag.4   v.Lag.5
## 2010-02-22 -0.0008322945 0.2226849 0.1937086 0.1688451 0.1593175 0.3046221
## 2010-02-23  0.0157348851 0.1323469 0.2226849 0.1937086 0.1688451 0.1593175
## 2010-02-24  0.0047385093 0.2074970 0.1323469 0.2226849 0.1937086 0.1688451
## 2010-02-25  0.0058951749 0.1763507 0.2074970 0.1323469 0.2226849 0.1937086
## 2010-02-26  0.0020737083 0.2596347 0.1763507 0.2074970 0.1323469 0.2226849
## 2010-03-01  0.0001799982 0.1735893 0.2596347 0.1763507 0.2074970 0.1323469
##                  macd     signal      rsi
## 2010-02-22 -0.8137820 -1.7200123 55.16748
## 2010-02-23 -0.7428162 -1.5245730 49.03458
## 2010-02-24 -0.6053894 -1.3407363 53.22420
## 2010-02-25 -0.5017218 -1.1729334 52.53349
## 2010-02-26 -0.4097214 -1.0202910 52.84106
## 2010-03-01 -0.2498190 -0.8661966 57.69130
```

```r
# Train test split
train_index2 <- seq(1,2988)

# For regression
train_reg <- SPY.return[train_index2,] # before 2021-12-31
test_reg <- SPY.return[-train_index2,] # after 2022-01-03
x_train_reg <- train_reg[,-1]
x_test_reg <- test_reg[,-1]
y_train_reg <- train_reg[,1]
y_test_reg <- test_reg[,1]

# For SVM
train_class <- SPY.trend[train_index2,] # before 2021-12-31
test_class <- SPY.trend[-train_index2,] # after 2022-01-03
x_train_class <- train_class[,-1]
y_train_class <- factor(train_class[,1])
x_test_class <- test_class[,-1]
y_test_class <- factor(test_class[,1])

# For random forest
train_dummy_X <- model.matrix(~.,train_class[,-1])[,-1]
test_dummy_X <- model.matrix(~.,test_class[,-1])[,-1]
```

(3) Train-Test Split

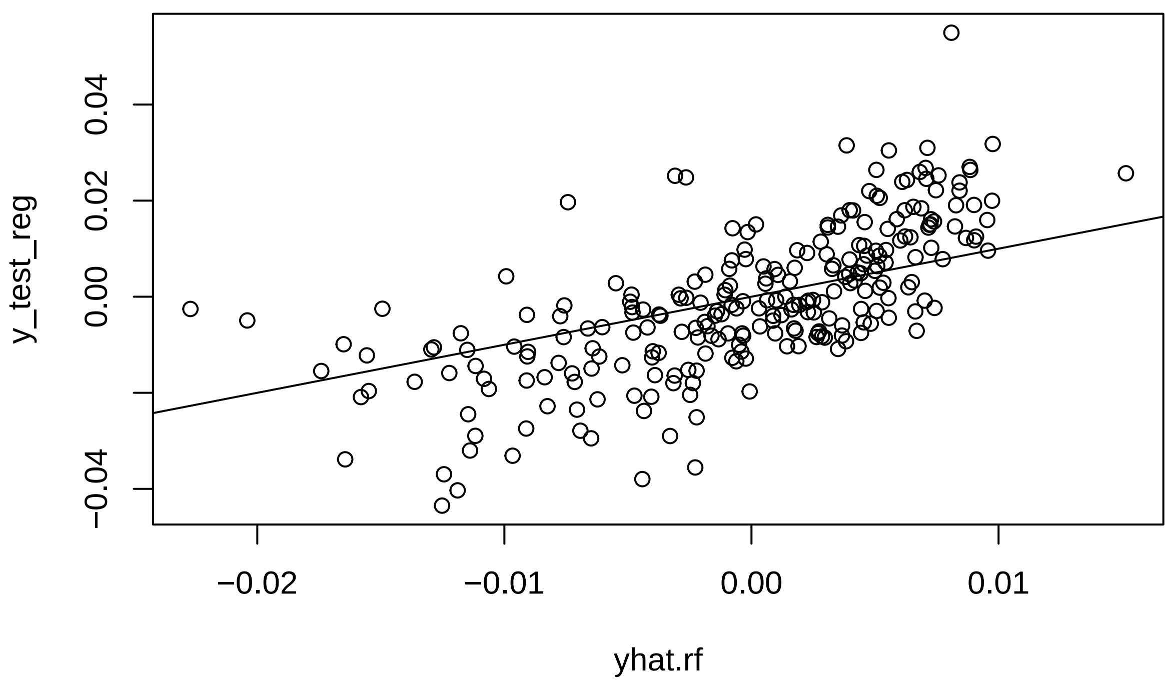
2. Regression

(1) Random Forest Since there are 14 features, I take `mtry` $= \lceil\sqrt{14}\rceil = 4$.

```r
set.seed(1)
rf_stock <- randomForest(r ~ .,
                         data = train_reg,
                         mtry = 4,
                         ntree = 500,
                         importance = TRUE)
rf_stock
```

```
##
## Call:
##  randomForest(formula = r ~ ., data = train_reg, mtry = 4, ntree = 500,      importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 6.841268e-05
##                     % Var explained: 40.34
```

```r
yhat.rf <- predict(rf_stock, newdata = x_test_reg)
plot(yhat.rf, y_test_reg)
abline(0, 1)
```

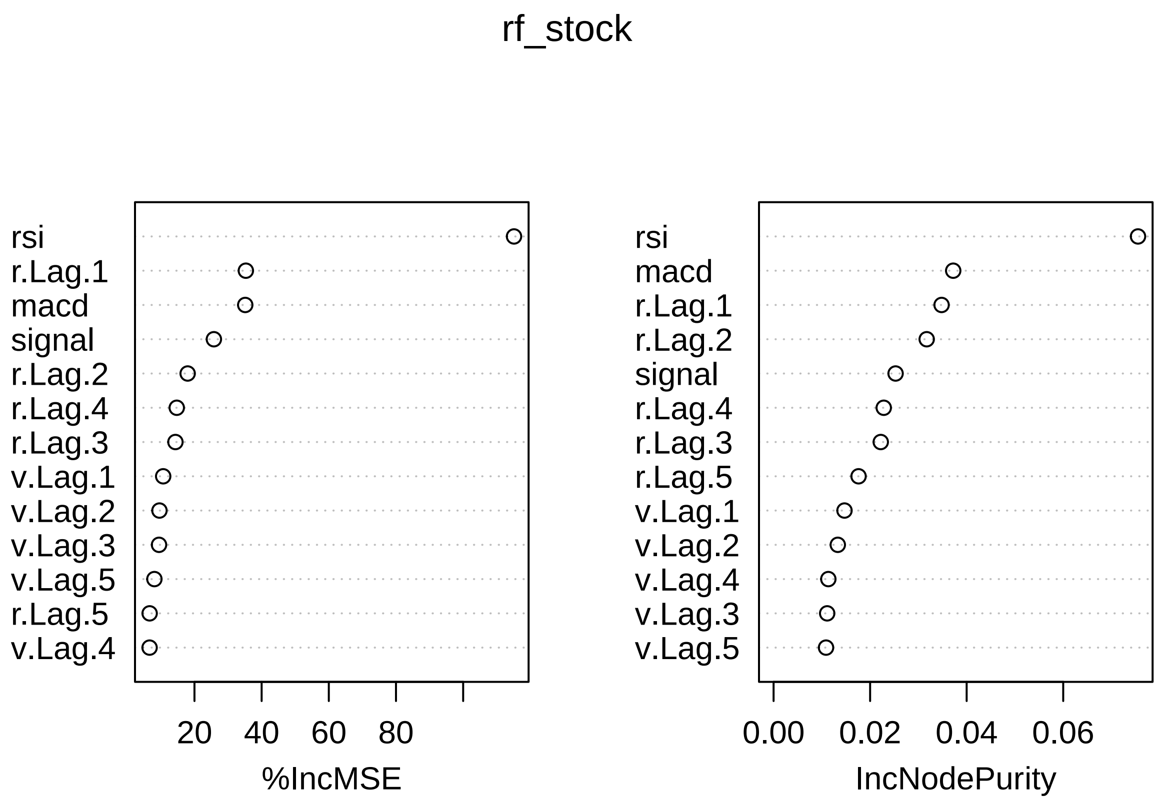



```r
mean((yhat.rf - y_test_reg)^2)
```

```
## [1] 0.0001376877
```

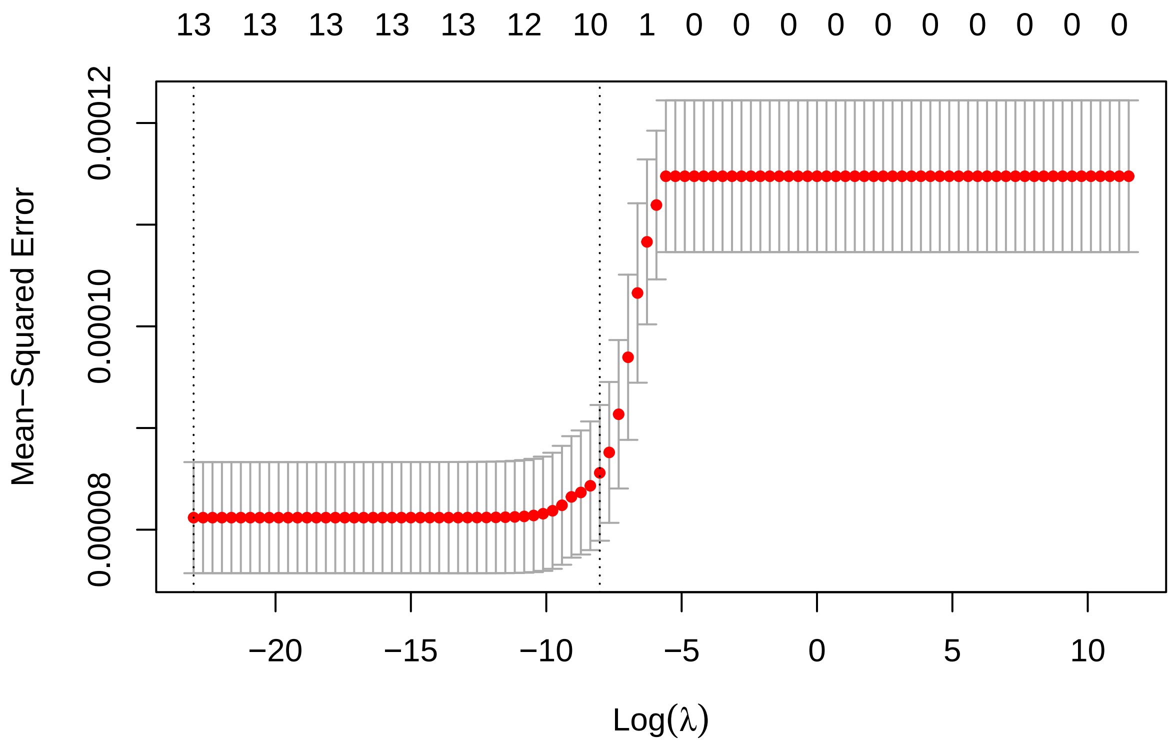
The regression result looks good (MSE=0.000138).

```
varImpPlot(rf_stock)
```



For MSE, yesterday's close price, RSI, MACD (along with the signal line) determines today's price. For interpret, MACD gives the moving average and trend of stock price, and RSI indicates whether the stocks go up or down to what extend.

```
set.seed(1)
grid <- 10^seq(5, -10, length = 100) # use grid search to find lambda
cv.out <- cv.glmnet(train_dummy_X, y_train_reg, alpha = 1, nfolds=5, lambda=grid) # LASSO
plot(cv.out)
```



(2) LASSO

Choose the best LASSO λ value.

```
bestlam_lasso <- cv.out$lambda.min # best lambda
bestlam_lasso
```

```
## [1] 1e-10
```

```
# retrain the model with the best lambda
lasso.stock <- glmnet(train_dummy_X, y_train_reg, alpha = 1, lambda = grid)

# training performance
lasso.pred <- predict(lasso.stock, s = bestlam_lasso, newx = train_dummy_X)
MSE_train <- mean((lasso.pred - y_train_reg)^2)

# testing performance
lasso.pred <- predict(lasso.stock, s = bestlam_lasso, newx = test_dummy_X)
MSE_test <- mean((lasso.pred - y_test_reg)^2)

# results
MSE_train
```

```
## [1] 7.770079e-05
```

```
MSE_test
```

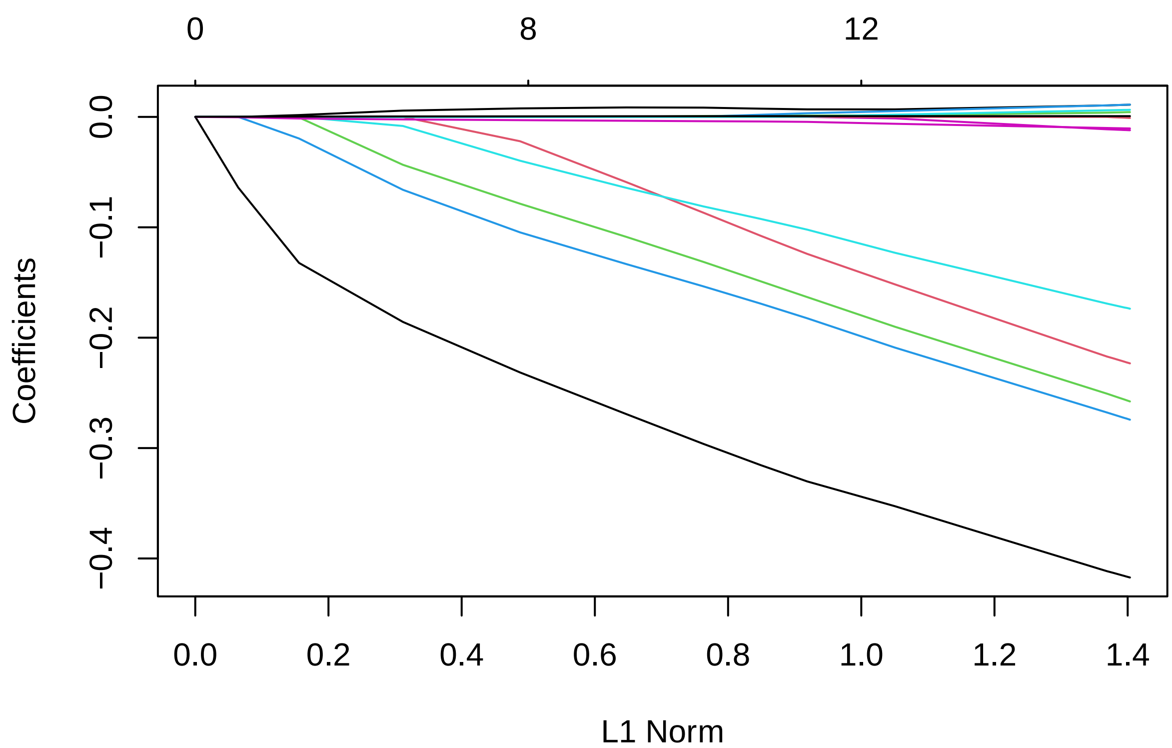
```
## [1] 0.0001617365
```

The test MSE is 0.00016, it does not differs a lot from random forest.

```
lasso_coeff <- predict(lasso.stock,
                       s = bestlam_lasso,
                       exact = T,
                       type = "coefficients",
                       x = train_dummy_X,
                       y =  y_train_reg)
lasso_coeff[1:14,][which(lasso_coeff!=0)]
```

```
##   (Intercept)       r.Lag.1       r.Lag.2       r.Lag.3       r.Lag.4
## -0.0379581194 -0.4172315754 -0.2232732955 -0.2577592987 -0.2743132632
##       r.Lag.5       v.Lag.1       v.Lag.2       v.Lag.3       v.Lag.4
## -0.1737519507 -0.0120741935  0.0110385816 -0.0009606937  0.0043217940
##       v.Lag.5          macd        signal           rsi
##  0.0111500336  0.0064134333 -0.0104553776  0.0006941517
```

```
plot(lasso.stock)
```



For LASSO, the past closed value (in 5 days) are more important than they are for random forest. Besides, the volume does not play a big role.

3. Classification

```r
### tuning parameters (cost,gamma) via 10-fold CV
set.seed(1)
tune_stock.out <- tune(svm,
                       train.x = x_train_class,
                       train.y = y_train_class,
                       ranges = list(
                         cost = c(0.1, 1, 10, 100),
                         gamma = c(0.1, 1, 2, 3, 4),
                         kernel = 'linear',
                         scale = TRUE # Need to scale in this case
                         )
                       )
summary(tune_stock.out)
```

(1) SVM

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma kernel scale
##   100   0.1 linear  TRUE
##
## - best performance: 0.2520134
##
## - Detailed performance results:
##     cost gamma kernel scale     error dispersion
## 1    0.1   0.1 linear  TRUE 0.2536890 0.03590093
## 2    1.0   0.1 linear  TRUE 0.2526823 0.03818408
## 3   10.0   0.1 linear  TRUE 0.2523479 0.03908488
## 4  100.0   0.1 linear  TRUE 0.2520134 0.03908177
## 5    0.1   1.0 linear  TRUE 0.2536890 0.03590093
## 6    1.0   1.0 linear  TRUE 0.2526823 0.03818408
## 7   10.0   1.0 linear  TRUE 0.2523479 0.03908488
## 8  100.0   1.0 linear  TRUE 0.2520134 0.03908177
## 9    0.1   2.0 linear  TRUE 0.2536890 0.03590093
## 10   1.0   2.0 linear  TRUE 0.2526823 0.03818408
## 11  10.0   2.0 linear  TRUE 0.2523479 0.03908488
## 12 100.0   2.0 linear  TRUE 0.2520134 0.03908177
## 13   0.1   3.0 linear  TRUE 0.2536890 0.03590093
## 14   1.0   3.0 linear  TRUE 0.2526823 0.03818408
## 15  10.0   3.0 linear  TRUE 0.2523479 0.03908488
## 16 100.0   3.0 linear  TRUE 0.2520134 0.03908177
## 17   0.1   4.0 linear  TRUE 0.2536890 0.03590093
## 18   1.0   4.0 linear  TRUE 0.2526823 0.03818408
## 19  10.0   4.0 linear  TRUE 0.2523479 0.03908488
## 20 100.0   4.0 linear  TRUE 0.2520134 0.03908177
```

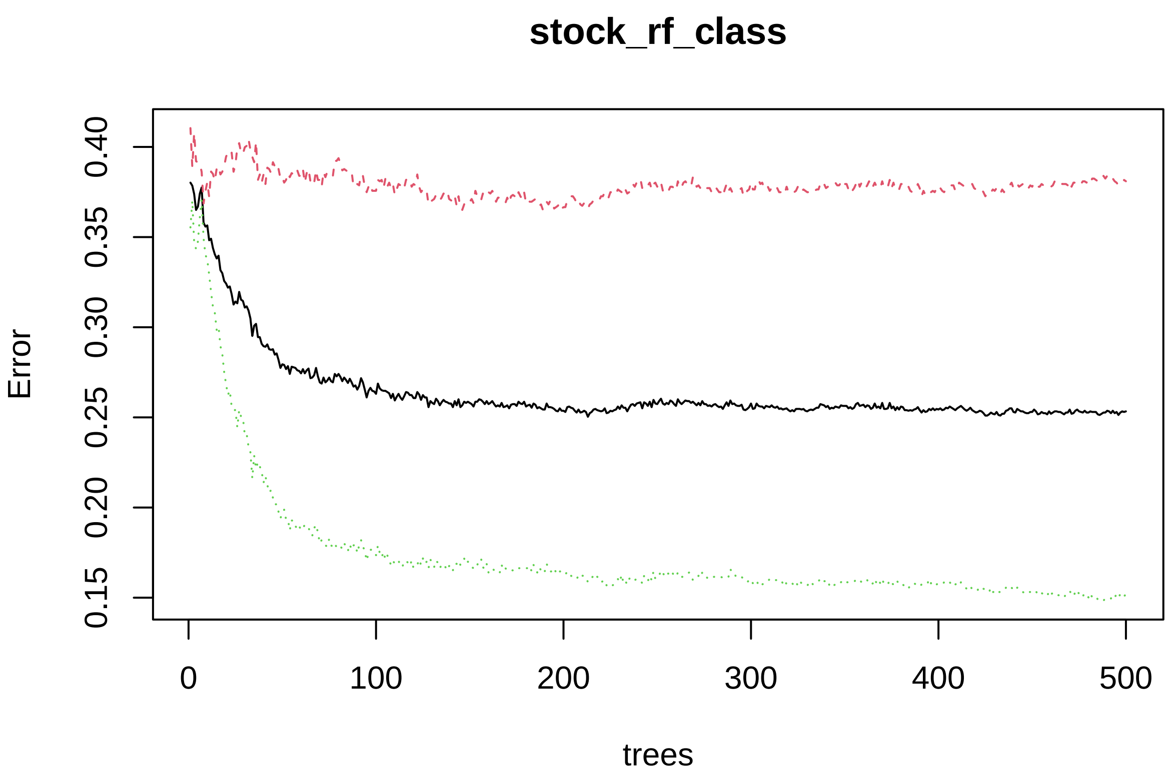
```r
###
table(true = y_test_class,
      pred = predict(tune_stock.out$best.model, newdata = x_test_class))
```

```
##       pred
## true   Down Up
##   Down   93 49
##   Up     24 85
```

The best accuracy is 70.92%.

(2) Random Forest Since there are 14 features, I take `mtry` $= \lceil\sqrt{14}\rceil = 4$.

```r
stock_rf_class <- randomForest(x=x_train_class,
                               y=y_train_class,
                               mtry = 4, # number of variables tried at each split
                               importance = TRUE,
                               proximity = TRUE)
plot(stock_rf_class)
```




```r
modfinal_rf_stock <- randomForest(x=x_train_class,
                                  y=y_train_class,
                                  mtry = 4,
```

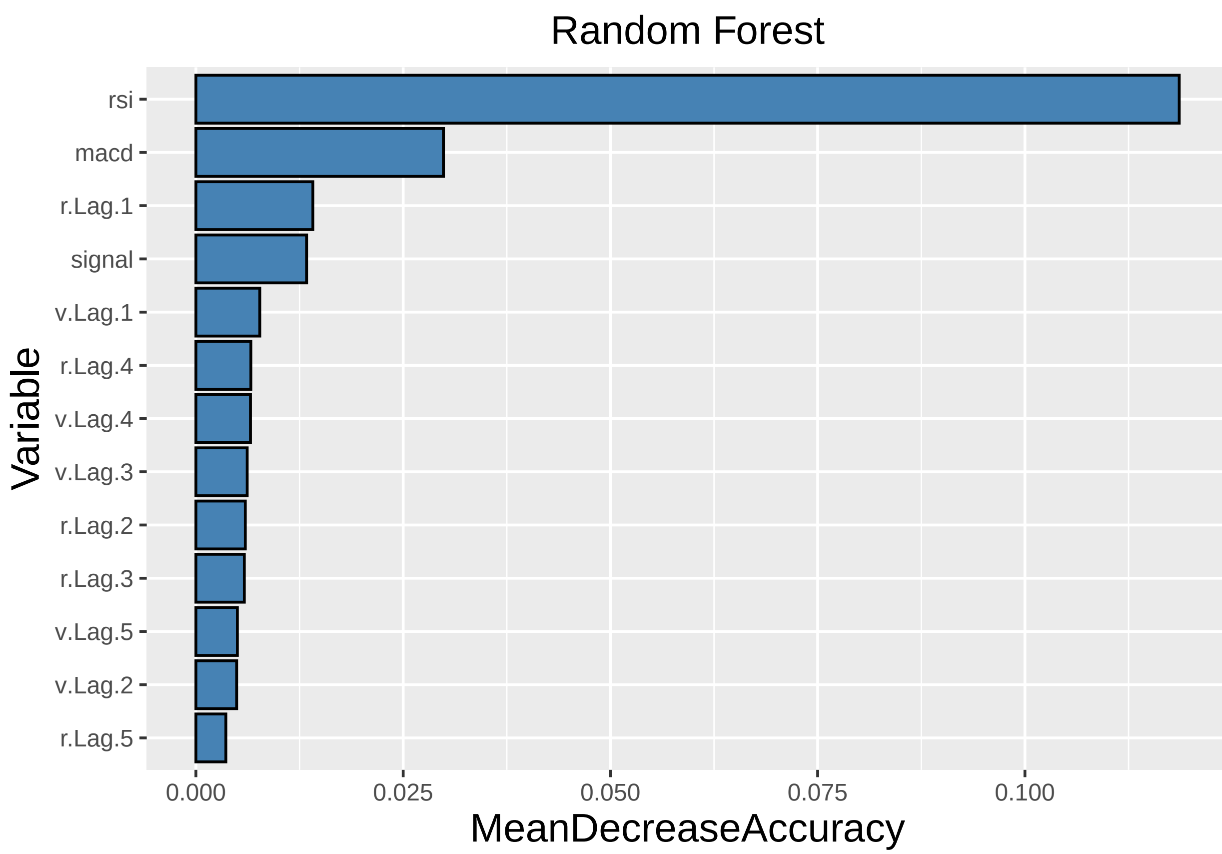
```r
                                          ntree = 150,
                                          importance = TRUE,
                                          proximity = TRUE)
```

```r
table(true = y_test_class,
      pred = predict(modfinal_rf_stock, newdata = x_test_class))
```

```
##       pred
## true   Down  Up
##   Down  104  38
##   Up     27  82
```

The accuracy is 0.74. Next, we investigate the variable importance and interpret in the summary.

```r
modfinal_rf_stock$importance %>%
  as.matrix() %>%
  as.data.frame() %>%
  add_rownames() %>%
  `colnames<-`(c("varname",'No','Yes',"MeanDecreaseAccuracy",'MeanDecreaseGini')) %>%
  arrange(desc(MeanDecreaseAccuracy)) %>%
  top_n(15,wt = MeanDecreaseAccuracy) %>%
  ggplot(aes(x = reorder(varname, MeanDecreaseAccuracy),y = MeanDecreaseAccuracy)) +
  geom_col(fill = 'steelblue', color = 'black') +
  coord_flip() +
  ggtitle(label = "Random Forest") +
  xlab('Variable') +
  ylab('MeanDecreaseAccuracy') +
  theme(plot.title=element_text(hjust=0.5,size=15),
        axis.title=element_text(size=15))
```



4. Summary

The accuracy is about 74% to 75% for both randomforest and SVM, to predict the value and the trend, random forest prefers the technical indicators such as RSI and MACD. However, LASSO relys a lot from past prices. The fine-tuned performance of random forest is better than SVM.