APAN5420 — HW 11, Stocks

Megan Wilder

8/10/18

Contents

---

1 Add Features

```
#In addition to features created in class,
#added EMA (exponential moving average) for 30 days and
#rolling 30 day average for open, high, low, close and volume

df2 <- gStockDF %>%
mutate(
Open_Close_PctChange = (Close - Open) / Open * 100,

Open_Close_Delt = Delt(Open, Close, k = 0) * 100,

Open_Change   = Open   - lag(Open),
High_Change   = High   - lag(High),
Low_Change    = Low    - lag(Low),
Close_Change  = Close  - lag(Close),
Volume_Change = Volume - lag(Volume),

Daily_Return  = Close / Open,

Open_PctChange   = Open_Change   / lag(Open)   * 100,
High_PctChange   = High_Change   / lag(High)   * 100,
Low_PctChange    = Low_Change    / lag(Low)    * 100,
Close_PctChange  = Close_Change  / lag(Close)  * 100,
Volume_PctChange = Volume_Change / lag(Volume) * 100,

Open_Mean10   = roll_mean(Open,   10, fill = NA, na.rm = TRUE),
High_Mean10   = roll_mean(High,   10, fill = NA, na.rm = TRUE),
Low_Mean10    = roll_mean(Low,    10, fill = NA, na.rm = TRUE),
Close_Mean10  = roll_mean(Close,  10, fill = NA, na.rm = TRUE),
```

```r
Volume_Mean10 = roll_mean(Volume, 10, fill = NA, na.rm = TRUE),

Open_Mean10_R   = roll_meanr(Open,   10, fill = NA, na.rm = TRUE),
High_Mean10_R   = roll_meanr(High,   10, fill = NA, na.rm = TRUE),
Low_Mean10_R    = roll_meanr(Low,    10, fill = NA, na.rm = TRUE),
Close_Mean10_R  = roll_meanr(Close,  10, fill = NA, na.rm = TRUE),
Volume_Mean10_R = roll_meanr(Volume, 10, fill = NA, na.rm = TRUE),

Open_Mean30   = roll_mean(Open,   30, fill = NA, na.rm = TRUE),
High_Mean30   = roll_mean(High,   30, fill = NA, na.rm = TRUE),
Low_Mean30    = roll_mean(Low,    30, fill = NA, na.rm = TRUE),
Close_Mean30  = roll_mean(Close,  30, fill = NA, na.rm = TRUE),
Volume_Mean30 = roll_mean(Volume, 30, fill = NA, na.rm = TRUE),

Open_Mean30_R   = roll_meanr(Open,   30, fill = NA, na.rm = TRUE),
High_Mean30_R   = roll_meanr(High,   30, fill = NA, na.rm = TRUE),
Low_Mean30_R    = roll_meanr(Low,    30, fill = NA, na.rm = TRUE),
Close_Mean30_R  = roll_meanr(Close,  30, fill = NA, na.rm = TRUE),
Volume_Mean30_R = roll_meanr(Volume, 30, fill = NA, na.rm = TRUE),

Open_SD30   = roll_sd(Open,   30, fill = NA, na.rm = TRUE),
High_SD30   = roll_sd(High,   30, fill = NA, na.rm = TRUE),
Low_SD30    = roll_sd(Low,    30, fill = NA, na.rm = TRUE),
Close_SD30  = roll_sd(Close,  30, fill = NA, na.rm = TRUE),
Volume_SD30 = roll_sd(Volume, 30, fill = NA, na.rm = TRUE),

Open_VAR30   = roll_var(Open,   30, fill = NA, na.rm = TRUE),
High_VAR30   = roll_var(High,   30, fill = NA, na.rm = TRUE),
Low_VAR30    = roll_var(Low,    30, fill = NA, na.rm = TRUE),
Close_VAR30  = roll_var(Close,  30, fill = NA, na.rm = TRUE),
Volume_VAR30 = roll_var(Volume, 30, fill = NA, na.rm = TRUE),

Open_EMA10   = EMA(Open,   n = 10),
High_EMA10   = EMA(High,   n = 10),
Low_EMA10    = EMA(Low,    n = 10),
Close_EMA10  = EMA(Close,  n = 10),
Volume_EMA10 = EMA(Volume, n = 10),

Open_EMA30   = EMA(Open,   n = 30),
High_EMA30   = EMA(High,   n = 30),
Low_EMA30    = EMA(Low,    n = 30),
Close_EMA30  = EMA(Close,  n = 30),
Volume_EMA30 = EMA(Volume, n = 30)

) %>%
arrange(Symbol, Date)

# add SP500 characteristics (sector)
df3 <- df2 %>% left_join(sp500Members, by = "Symbol")

#View number of stocks by sector
df3 %>%
group_by(Sector) %>%
```

```r
summarise(Num = length(unique(Symbol))) %>%
arrange(Num)

# Add day of week and quarter features
df3$DOW <- weekdays(df3$Date)
df3$Quarter <- quarters(df3$Date)

#view number of instances per quarter
df3 %>%
group_by(Quarter) %>%
summarise(Num = n()) %>%
arrange(Num)

#view number of instances per day
df3 %>%
group_by(DOW) %>%
summarise(Num = n()) %>%
arrange(Num)
```

2 Prepare Dataset

```r
#Subset Data to Only 1 Year
# add a year feature
df3$Year <- as.numeric(format(df3$Date, "%Y"))
df3$QtrYear <- sprintf("%s-%s", df3$Year, df3$Quarter)
df3$Month <- as.numeric(format(df3$Date, "%m"))

# use 2016 data
df4 <- filter(df3, Year == 2016)
```

```r
# Look for Zero or Near Zero Variance
nzv <- nearZeroVar(df4, saveMetrics = TRUE)
nzv  # year only variable with NZV
```

```r
# Look for Linear Combinations
#filter for only numerical columns
df4_num <- Filter(is.numeric, df4)

#remove rows with NA's
df4_num <- na.omit(df4_num)

#check for linear combinations
df4_linear <- findLinearCombos(df4_num)
df4_linear

#view columns identified
head(df4_num [, df4_linear$remove])

# remove the recommended columns from df4
df4$Open_Close_Delt <- NULL
df4$Year <- NULL
```

```r
#remove NA's
df4 <- na.omit(df4)
```

```r
#Split data into training and hold out test set
train <- filter(df4, Month >= 1 & Month <= 9)
holdout <- filter(df4, Month >= 10)
```

3 Models

3.1 First Model: Generalized Linear Model

```r
#GLM Model
#there are no tuning parameters for this model

myTimeControl <- trainControl(
method = "timeslice",
initialWindow = 100,
#5 months
horizon = 40,
#2 months
fixedWindow = TRUE
)

#train model with top 4 variables from Module 8 assignment
set.seed(123)
glm.mod <-
train(
Open_Close_PctChange ~ Open_EMA10 + Open_Change + High_Change + Open_Mean10_R,
data = train,
method = "glm",
family = "gaussian",
trControl = myTimeControl,
preProc = c("center", "scale")
)#Center and scale data
```

```r
#GLM model results
glm.mod
```

```
## Generalized Linear Model
##
## 93123 samples
##     4 predictor
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Rolling Forecasting Origin Resampling (40 held-out with a fixed window)
## Summary of sample sizes: 100, 100, 100, 100, 100, 100, ...
## Resampling results:
##
##   RMSE     Rsquared   MAE
##   1.46822  0.4016268  1.113504
```

```r
#view results table
glm.mod$results
```

```
##   parameter    RMSE  Rsquared      MAE  RMSESD RsquaredSD    MAESD
## 1      none 1.46822 0.4016268 1.113504 2.15421  0.1647838 1.729939
```

3.2 Second Model: Random Forest

```r
## Second Model: Random Forest
#RF Model
myTimeControl <- trainControl(
method = "timeslice",
initialWindow = 100,
#5 months
horizon = 40,
#2 months
fixedWindow = TRUE
)

#provide a grid of parameters
rf.grid <- expand.grid(mtry = c(2, 3))

#train model with top 4 variables from Module 8 assignment
set.seed(123)
rf.mod <-
train(
Open_Close_PctChange ~ Open_EMA10 + Open_Change + High_Change + Open_Mean10_R,
data = train,
method = "rf",
trControl = myTimeControl,
tuneGrid = rf.grid,
preProc = c("center", "scale")
)#Center and scale data
```

```r
#RF Model results
rf.mod
```

```
## Random Forest
##
## 93123 samples
##     4 predictor
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Rolling Forecasting Origin Resampling (40 held-out with a fixed window)
## Summary of sample sizes: 100, 100, 100, 100, 100, 100, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     1.301544  0.2625426  1.0026577
##   3     1.281251  0.2818192  0.9850602
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 3.
```

```r
#view results table
rf.mod$results
```

```
##   mtry     RMSE  Rsquared       MAE    RMSESD RsquaredSD     MAESD
## 1    2 1.301544 0.2625426 1.0026577 0.6628606  0.1488387 0.5173037
## 2    3 1.281251 0.2818192 0.9850602 0.6612670  0.1499822 0.5157663
```

```
#best moodel
rf.mod$bestTune
```

```
##   mtry
## 2    3
```

```
#RMSE was used to select the optimal model using the smallest value.
#The final value used for the model was mtry = 3.
```

3.3 Third Model: Parial Least Squares

```
#PLS Model
myTimeControl <- trainControl(
method = "timeslice",
initialWindow = 100,
#5 months
horizon = 40,
#2 months
fixedWindow = TRUE
)

#train model with top 4 variables from Module 8 assignment
set.seed(123)
pls.mod <-
train(
Open_Close_PctChange ~ Open_EMA10 + Open_Change + High_Change + Open_Mean10_R,
data = train,
method = 'pls',
trControl = myTimeControl,
tuneLength = 15,
preProc = c("center", "scale")
)#Center and scale data
```

```
#PLS Model results
pls.mod
```

```
## Partial Least Squares
##
## 93123 samples
##     4 predictor
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Rolling Forecasting Origin Resampling (40 held-out with a fixed window)
## Summary of sample sizes: 100, 100, 100, 100, 100, 100, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE      Rsquared   MAE
##   1      2.045068  0.2433134  1.593018
##   2      1.413780  0.3903871  1.072595
##   3      1.445177  0.4096575  1.093672
##
```

```
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 2.
```

```
#view results table
pls.mod$results
```

```
##   ncomp     RMSE  Rsquared      MAE   RMSESD RsquaredSD    MAESD
## 1     1 2.045068 0.2433134 1.593018 4.401038  0.1774641 3.599807
## 2     2 1.413780 0.3903871 1.072595 1.823926  0.1720738 1.516354
## 3     3 1.445177 0.4096575 1.093672 2.146610  0.1650551 1.703250
```

```
# best moodel
pls.mod$bestTune
```

```
##   ncomp
## 2     2
```

3.4 Fourth Model: GLMNET

```
#GLMNET Model
myTimeControl <- trainControl(
method = "timeslice",
initialWindow = 100,
#5 months
horizon = 40,
#2 months
fixedWindow = TRUE
)

#provide a grid of parameters
glmnet.grid <- expand.grid(expand.grid(
.alpha = c(0,
1),
.lambda = seq(0.02, 0.06, by = 0.02)
))

#train model with top 4 variables from Module 8 assignment
set.seed(123)
glmnet.mod <-
train(
Open_Close_PctChange ~ Open_EMA10 + Open_Change + High_Change + Open_Mean10_R,
data = train,
method = 'glmnet',
trControl = myTimeControl,
tuneGrid = glmnet.grid,
preProc = c("center", "scale")
)#Center and scale data
```

```
#GLM Model results
glmnet.mod
```

```
## glmnet
##
## 93123 samples
##     4 predictor
```

```
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Rolling Forecasting Origin Resampling (40 held-out with a fixed window)
## Summary of sample sizes: 100, 100, 100, 100, 100, 100, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda  RMSE      Rsquared   MAE
##   0      0.02    1.435135  0.4047321  1.0926691
##   0      0.04    1.434554  0.4046468  1.0923936
##   0      0.06    1.433379  0.4041753  1.0920805
##   1      0.02    1.395652  0.4097039  1.0568554
##   1      0.04    1.343480  0.4094883  1.0174803
##   1      0.06    1.300629  0.4077091  0.9854499
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 1 and lambda = 0.06.
```

```
#view results table
glmnet.mod$results
```

```
##   alpha lambda     RMSE  Rsquared       MAE   RMSESD RsquaredSD    MAESD
## 1     0   0.02 1.435135 0.4047321 1.0926691 2.148747  0.1696370 1.718987
## 2     0   0.04 1.434554 0.4046468 1.0923936 2.147456  0.1697857 1.718225
## 3     0   0.06 1.433379 0.4041753 1.0920805 2.145164  0.1704636 1.715368
## 4     1   0.02 1.395652 0.4097039 1.0568554 1.935325  0.1652529 1.537849
## 5     1   0.04 1.343480 0.4094883 1.0174803 1.703388  0.1651784 1.349577
## 6     1   0.06 1.300629 0.4077091 0.9854499 1.496837  0.1656079 1.183450
```

```
# best moodel
glmnet.mod$bestTune
```

```
##   alpha lambda
## 6     1   0.06
```

3.5 Fifth Model: SVM Radial

```
#SVM Radial Model
myTimeControl <- trainControl(
method = "timeslice",
initialWindow = 100,
#5 months
horizon = 40,
#2 months
fixedWindow = TRUE
)

#Train and Tune the SVM with default parameters
#(for computation reasons I did not re-run this when knitting the file)
#svm.tune <-
#train(
#Open_Close_PctChange ~ Open_EMA10 + Open_Change + High_Change + Open_Mean10_R,
#data = train,
#method = "svmRadial",
# Radial kernel
```

```r
#preProc = c("center", "scale"),
#Center and scale data
#trControl = myTimeControl
#)

#svm.tune

## In the second pass, having seen the parameter values selected in the
#first pass, we use train()'s tuneGrid parameter to do some sensitivity
#analysis around the values C = 0.5 and sigma = 2.425959 that produced
#the best model with the default settings.

#provide a grid of parameters
svm.grid <- expand.grid( sigma = c(2, 2.5, 3),
             C = c(.25, .5, 1))

#train model with top 4 variables from Module 8 assignment
set.seed(123)
svm.mod <- train(Open_Close_PctChange ~ Open_EMA10 + Open_Change + High_Change + Open_Mean10_R,
 data = train,
 method = "svmRadial",
 trControl = myTimeControl,
 tuneGrid = svm.grid,
 preProc = c("center", "scale"))
```

```r
#SVM Model results
svm.mod
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 93123 samples
##     4 predictor
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Rolling Forecasting Origin Resampling (40 held-out with a fixed window)
## Summary of sample sizes: 100, 100, 100, 100, 100, 100, ...
## Resampling results across tuning parameters:
##
##   sigma  C     RMSE      Rsquared   MAE
##   2.0    0.25  1.306556  0.1645712  0.9852048
##   2.0    0.50  1.296290  0.1685287  0.9775275
##   2.0    1.00  1.301741  0.1672264  0.9850014
##   2.5    0.25  1.313317  0.1511548  0.9910470
##   2.5    0.50  1.303658  0.1547305  0.9838595
##   2.5    1.00  1.309096  0.1532505  0.9912022
##   3.0    0.25  1.318804  0.1402568  0.9958529
##   3.0    0.50  1.309611  0.1435958  0.9890413
##   3.0    1.00  1.314957  0.1420075  0.9961171
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 2 and C = 0.5.
```

```r
#view results table
svm.mod$results
```

```
##   sigma    C     RMSE  Rsquared       MAE    RMSESD RsquaredSD     MAESD
## 1   2.0 0.25 1.306556 0.1645712 0.9852048 0.6313638  0.1455851 0.4715866
## 2   2.0 0.50 1.296290 0.1685287 0.9775275 0.6297323  0.1484219 0.4698309
## 3   2.0 1.00 1.301741 0.1672264 0.9850014 0.6296994  0.1485625 0.4703788
## 4   2.5 0.25 1.313317 0.1511548 0.9910470 0.6315653  0.1396164 0.4719700
## 5   2.5 0.50 1.303658 0.1547305 0.9838595 0.6298838  0.1422990 0.4701798
## 6   2.5 1.00 1.309096 0.1532505 0.9912022 0.6294416  0.1422608 0.4701857
## 7   3.0 0.25 1.318804 0.1402568 0.9958529 0.6317782  0.1344613 0.4723012
## 8   3.0 0.50 1.309611 0.1435958 0.9890413 0.6301072  0.1370417 0.4705283
## 9   3.0 1.00 1.314957 0.1420075 0.9961171 0.6294789  0.1368163 0.4702373
```

```r
# best moodel
svm.mod$bestTune #sigma = 2, C = 0.5
```

```
##   sigma   C
## 2     2 0.5
```

4 Compare All Models We've Trained

```r
resamps <- resamples(
 list(
   GLM = glm.mod,
   RF = rf.mod,
   PLS = pls.mod,
   GLM.Net = glmnet.mod,
   SVM = svm.mod) )
```

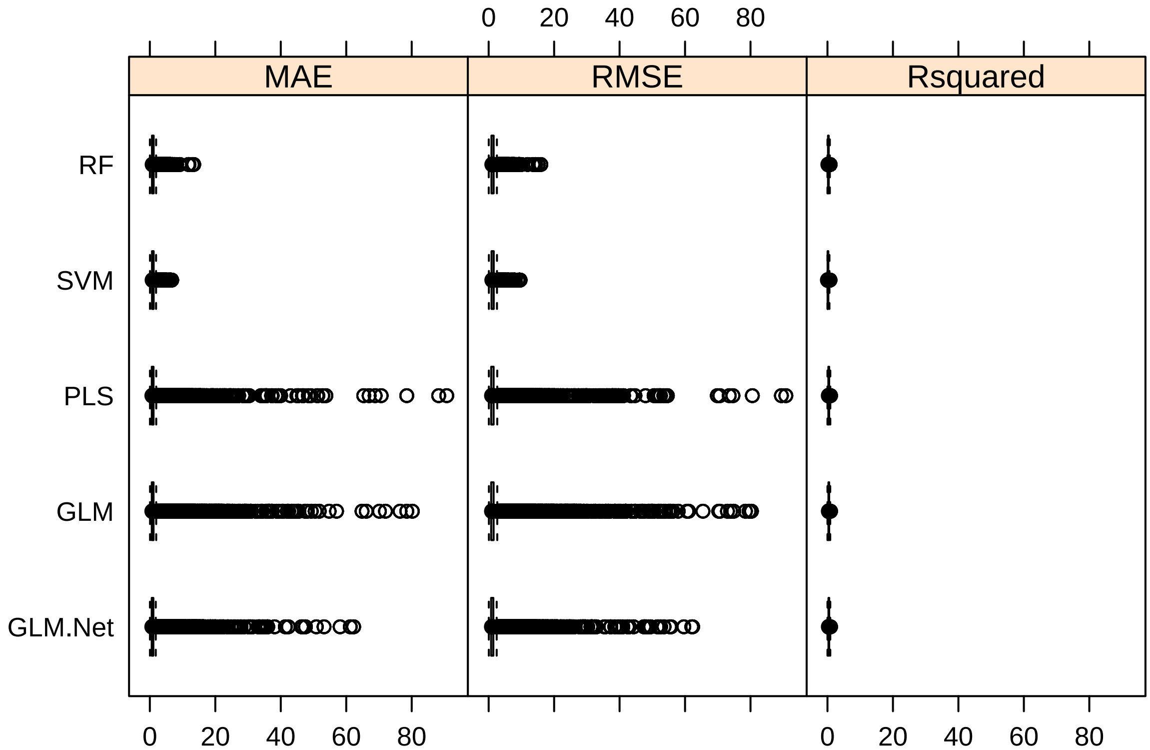
```r
resamps
```

```
##
## Call:
## resamples.default(x = list(GLM = glm.mod, RF = rf.mod, PLS =
##  pls.mod, GLM.Net = glmnet.mod, SVM = svm.mod))
##
## Models: GLM, RF, PLS, GLM.Net, SVM
## Number of resamples: 557904
## Performance metrics: MAE, RMSE, Rsquared
## Time estimates for: everything, final model fit
```

4.1 Box Plots of Metrics

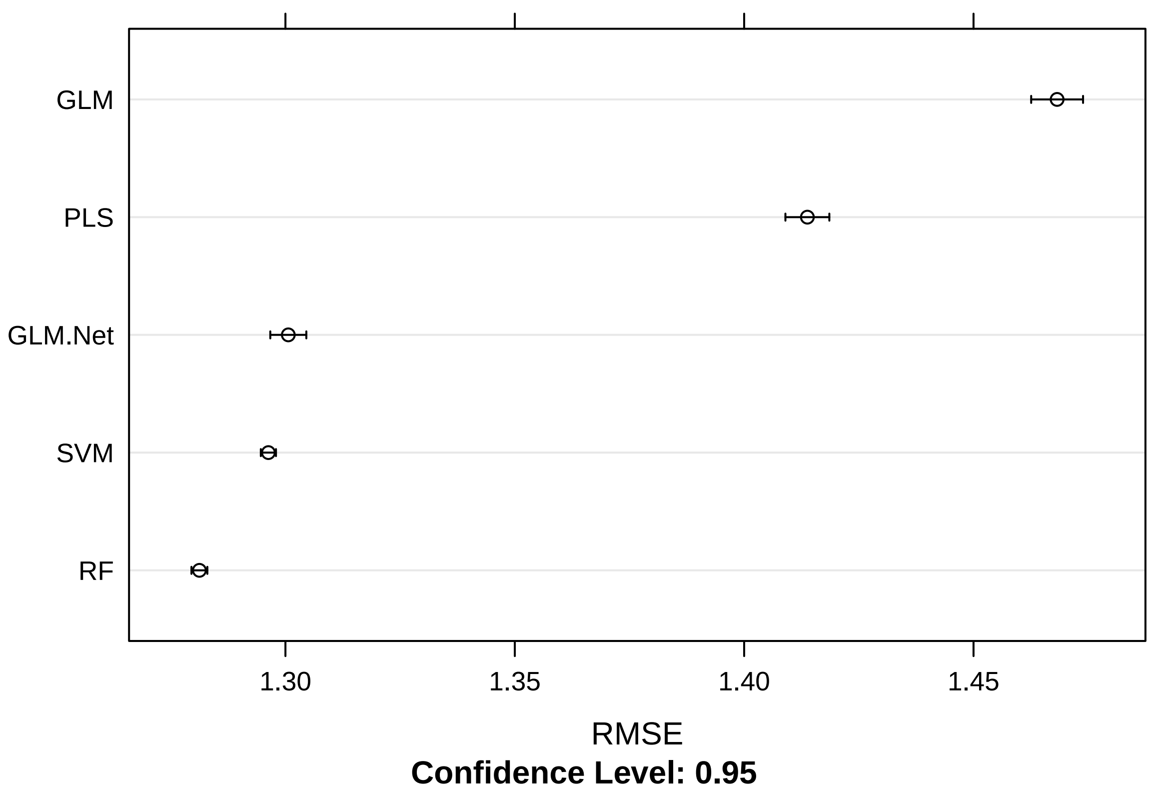
```r
trellis.par.set( caretTheme())

bwplot(resamps, layout = c(3, 1))
```



4.2 RMSE Metrics

```
dotplot(resamps, metric = "RMSE")
```



5 Train with the Best Model

```
r eval = FALSE
}
#Create a new trainControl object for training the full model
#Method - none = only fits one model to the entire training set
#tuneGrid = Can pass the bestTune from the training session

#RF had lowest RMSE use as best model.

finalFitControl <- trainControl(method = "none")

set.seed(123)
rfFitFinal <-
train(
Open_Close_PctChange ~ Open_EMA10 + Open_Change + High_Change + Open_Mean10_R,
data = train,
method = "rf",
trControl = finalFitControl,
verbose = FALSE,
## Only a single model can be passed to the
## function when no resampling is used:
tuneGrid = rf.mod$bestTune
)
rfFitFinal
```

```r
rfFitFinal
```

```
## Random Forest
##
## 93123 samples
##     4 predictor
##
## No pre-processing
## Resampling: None
```

```r
#predict on test set
rf.pred <- predict(rfFitFinal, newdata = holdout)

#change to dataframes
rf.df <- as.data.frame(rf.pred)
holdout.df <- as.data.frame(holdout)

#attached predicted values to test dataframe
final <- cbind(holdout.df, rf.df)
```

6 Find Outliers

```r
# define a function to find outliers
FindOutliers <- function(data) {
lowerq = quantile(data)[2]
upperq = quantile(data)[4]
iqr = upperq - lowerq
```

```r
extreme.threshold.upper = (iqr * 200) + upperq
extreme.threshold.lower = lowerq - (iqr * 200)
result <-
which(data > extreme.threshold.upper |
data < extreme.threshold.lower)
}
```

```r
#ape computes the elementwise absolute percent
#difference between two numeric vectors
final$APE <- ape(final$Open_Close_PctChange, final$rf.pred)
# use the function to identify outliers
outliers <- FindOutliers(final$APE)
# remove non outliers
RF.Outliers <- final[outliers, ]
#remove rows with APE of Inf
RF.Outliers <- RF.Outliers[is.finite(RF.Outliers$APE),]
```

CMI: On 2016-11-11 CMI's closing price was flat versus its opening price. However, my model predicted a 2% decline. It's difficult to say that my model is correct and that the stock should have traded down instead of flat. Particularly given that the stock traded between -.2% and +3% in the week prior to 1/12/12 and the week after.

ISRG: On 2016-12-13 ISRG's closing price was flat versus its opening price. However, my model predicted a 2% increase. It's difficult to say that my model is correct and that the stock should have traded down instead of flat. Particularly given that the stock traded in a narrow band the week prior to 2016-12-13 and the week after (-.7% - +2%).

REGN: On 2016-10-27 REGN's closing price was flat versus its opening price. However, my model predicted a 1% increase. It's difficult to say that my model is correct and that the stock should have traded down instead of flat. Particularly given that the stock traded between -3% and +5% in the week prior to 2016-10-27 and the week after.