Random Forest

Jesse Keränen

12/8/2021

Prologue

One thing every investor would like to know is whether price of the given asset goes up or down next day. If investor would also know the magnitude of the change trading would become pretty easy. One attempt to solve this problem has been to use machine learning algorithms and volatility factors to predict the sign of the change of asset. Some researches go so far that they even try to estimate how much the asset is going to change in given time period. This makes them possible to use in portfolio optimization. In this file I am only going to try to estimate if price of one asset is going go up or down using random forest algorithm.

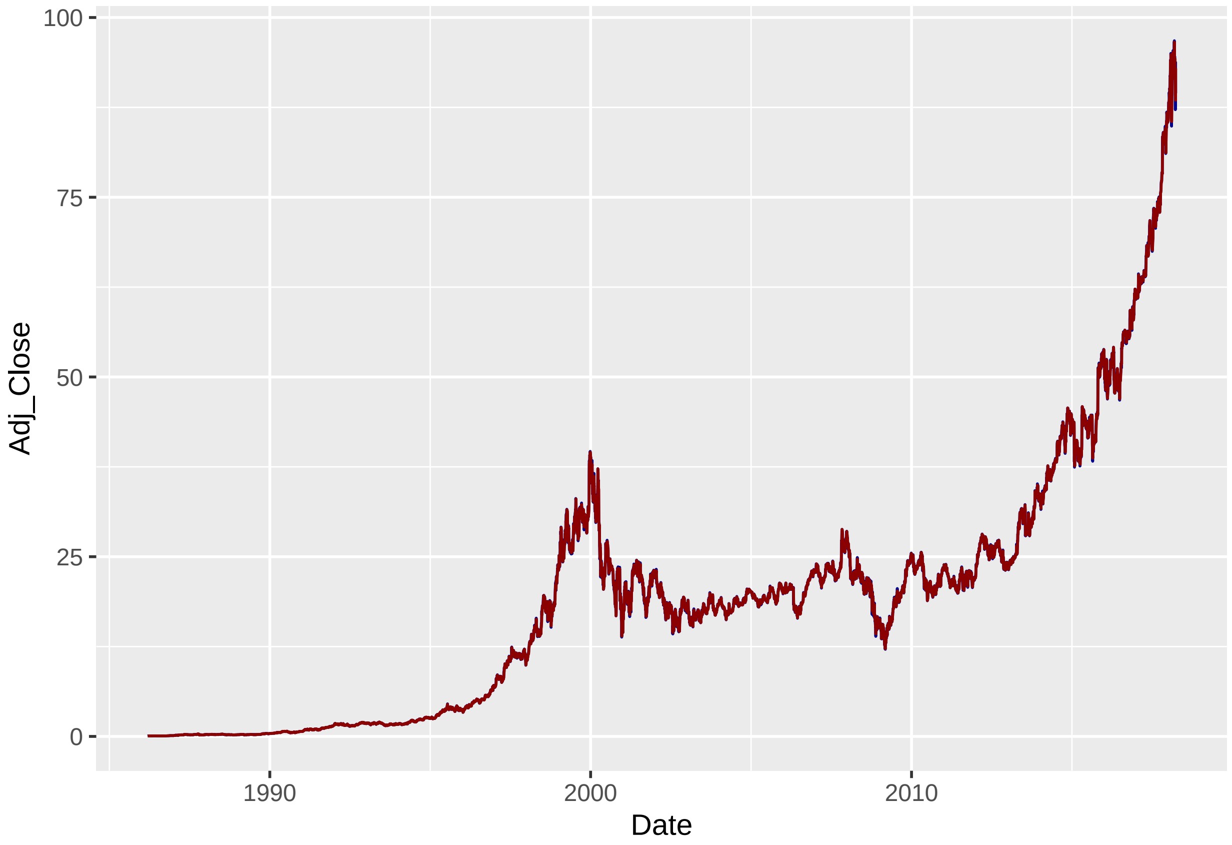
```r
library(ggplot2)
library(Quandl)
library(tidyverse)
library(data.table)
library(zoo)
library(pracma)
library(caret)
library(rpart)
library(randomForest)
library(lubridate)
library(plotROC)
library(quantmod)


Quandl.api_key("bx1qdehfWXg6SNKnicQC")

# For monthly data use collapse = "monthly"
price <- as.data.table(Quandl(c("WIKI/MSFT"), start_date
        = "1983-01-01", end_date = "2021-01-01", collapse = "daily"))

price <- price[, .(Date,`Adj. Open`, `Adj. High`, `Adj. Low`, `Adj. Close`,
                   `Adj. Volume`)]

price <- na.omit(price)
colnames(price) <- c("Date", "Adj_Open", "Adj_High", "Adj_Low", "Adj_Close", "Adj_Volume")

price <- price[order(Date)]
```

Data smoothing

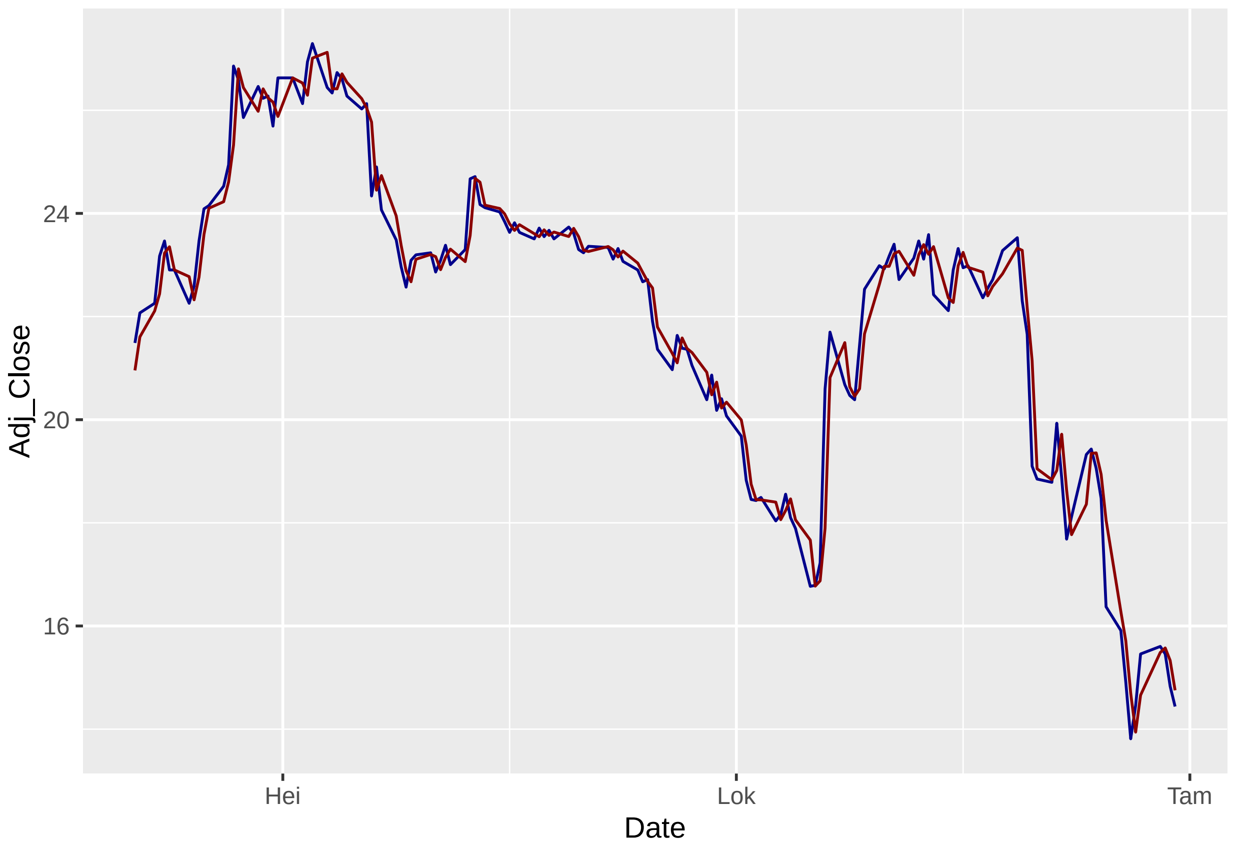
Financial data normally consists lots of noise. Impact of this noise can be reduced by smoothing the data. Prices are smoothed using exponential smoothing which means that sets the price of time t to be smoothed average of observed value at time t and smoothed value at time $t-1$. Smoothing factor α can be set. I have read other studies using α value of 0.2 so I will use the same.

$$\hat{P}_0 = P_0,$$
$$\hat{P}_{t+1} = \alpha P_{t+1} + (1-\alpha)\hat{P}_{t+1}$$

```
alpha <- 0.2
price[, Adj_Close_Smooth := alpha * Adj_Close + (1-alpha)*shift(Adj_Close)]
price[1, Adj_Close_Smooth := Adj_Close]

price[, Adj_Return := Adj_Close_Smooth/shift(Adj_Close_Smooth)-1]

# In the big picture it is hard to see any difference between smoothed and not
# smoothed prices
ggplot(price, aes(Date, Adj_Close)) + geom_line(color = "darkblue") + geom_line(aes(Date, Adj_Close_Smoo
```



```
# When you look shorter period, effect of smoothing can be seen
ggplot(price[year(Date) == 2000 & month(Date) >= 1 & month(Date) >= 6], aes(Date, Adj_Close)) + geom_li
```



Explanatory variables

So that our algorithm can make reasonable estimates, we need to give it relevant information on which it can base it's predictions. We are going to use set of trading indicators. I am not too familiar with quantitative trading so I am going to use same indicators I have seen in other studies. Using these indicators is convenient for us since they base largely on trading volume of asset. We can easily download trading volume from same data base as price data. Some accounting variables like book value we can't get from Quandl data base.

On Balance Volume

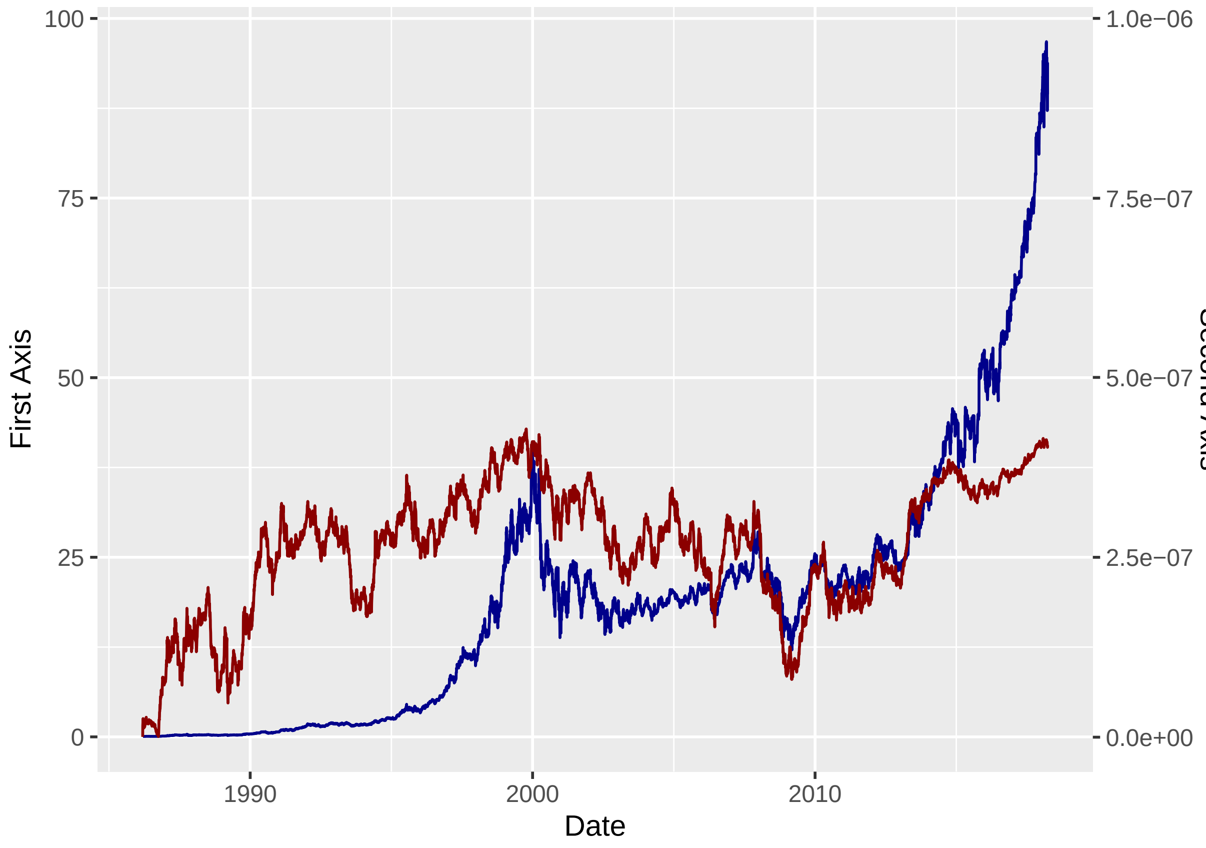
On Balance Volume is a cumulative trading pressure measure. OBV can remain same or it can be calculated by adding or reducing trading volume from last OBV. Different scenarios are show below.

$$OBV_t = OBV_{t-1} + \begin{cases} volume, & if\ Price_t > Price_{t-1} \\ 0, & if\ Price_t = Price_{t-1} \\ -volume, & if\ Price_t < Price_{t-1} \end{cases}$$

Idea of the On Balance Volume is that volume proceeds price. I think OBV as pressure for stock price. When volumes are high and prices increase there is still pressure for assets price to increase. Again when volumes and prices are decreasing prices are more likely to still decrease.

```
price[, OBV := 0]
for(i in 2:price[, .N]){
  price$OBV[i] = price$OBV[i-1] + sign(price$Adj_Return[i]) *
```

```
    price$Adj_Volume[i]
}

# There would be easier way
# price[, OBV := OBV(Adj_Close_Smooth, Adj_Volume)]

ggplot(price, aes(x = Date)) + geom_line(aes(y = Adj_Close), color = "darkblue") + geom_line(aes(y = 
  scale_y_continuous(
    # Features of the first axis
    name = "First Axis",
    # Add a second axis and specify its features
    sec.axis = sec_axis( trans=~.*10^(-8), name="Second Axis"))
```



Stochastic Oscillator %K

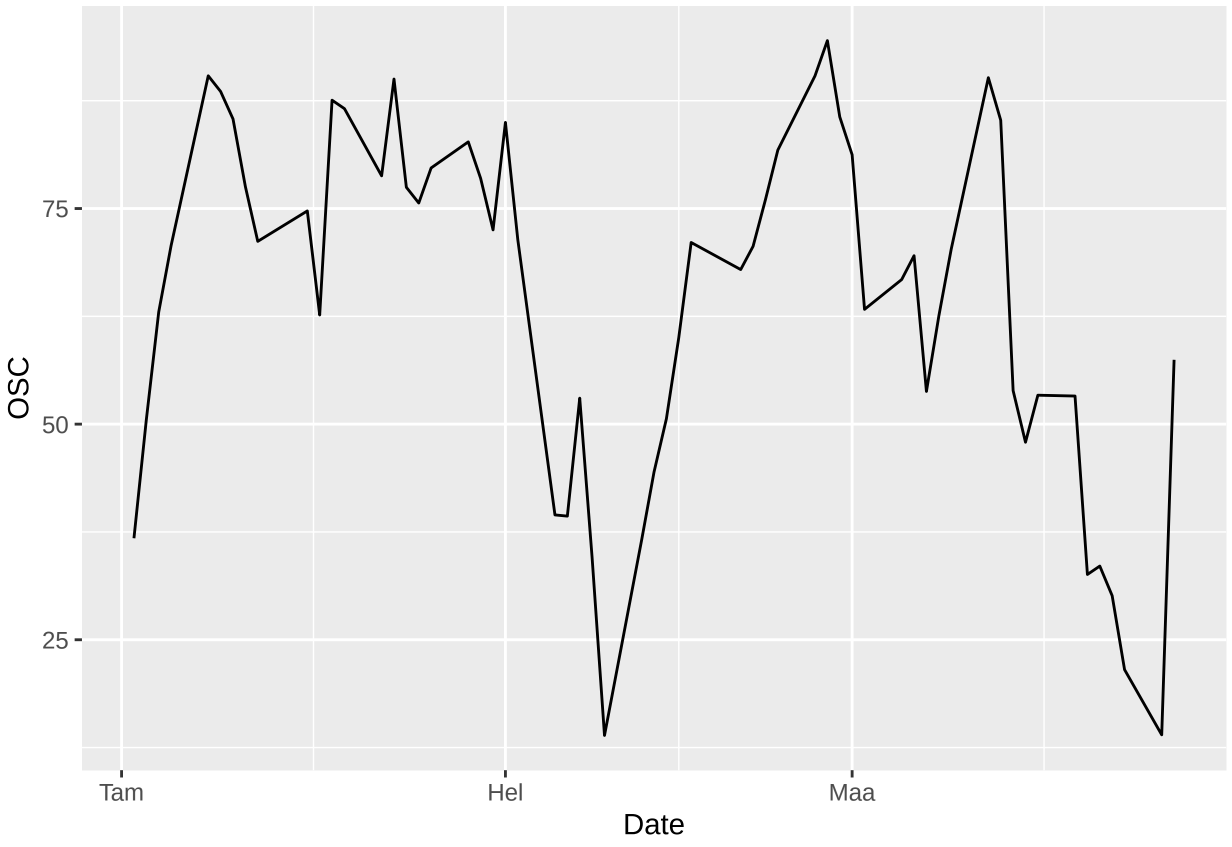
Stochastic Oscillator is a measure indicating overbought and oversold situations. It will vary between 0 and 100. It will tell you how high is the current price of the asset compared to it's high and low values within K last days. I will use 14 last day, which again seems to be quite usual in literature. If Stochastic Oscillator gets values close to 100 it means that current price is close highest high value within last two weeks. When value is close to 0 it means that current value of the asset is close to its minimum low value.

$$K = 100 * \frac{P_t - Low_K}{High_K - Low_K}$$

Stochastic Oscillator tries to predict turning point of price movement. Many times is said that asset is overbought when Stochastic Oscillator get values greater than 80 and oversold when it gets values below 20.

It is important to remember that asset can remain overbought or oversold for long periods if the price of the asset is trending up or down. I have understood that this measure indicates possible overshooting in the markets.

```r
stoch_osc <- function(data, t, K){
  P <- data[Date == t, Adj_Close_Smooth]
  dt <- tail(data[Date <= t, .(Date, Adj_Close_Smooth, Adj_High, Adj_Low)], K)
  return(100*(P-min(dt$Adj_Low))/(max(dt$Adj_High)-min(dt$Adj_Low)))
}

K <- 14
price[, OSC := as.numeric(rbind(lapply(price$Date, function(t){stoch_osc(price,
      t, K)})))]

# Better way with quantmod
# lk <- stoch(as.matrix(price[, .(Adj_High, Adj_Low, Adj_Close_Smooth)]),
#                         nFastK = 14)*100
# price[, OSC := lk[, "fastK"]]

ggplot(price[year(Date) > 2017], aes(Date, OSC)) + geom_line()
```



Moving Average Convergence Divergence

Moving Average Convergence Divergence calculates difference between two Moving Averages. I will use 12 and 26 day Exponential Moving Averages. Exponential moving average differences from normal moving average by giving more weight on recent observations.

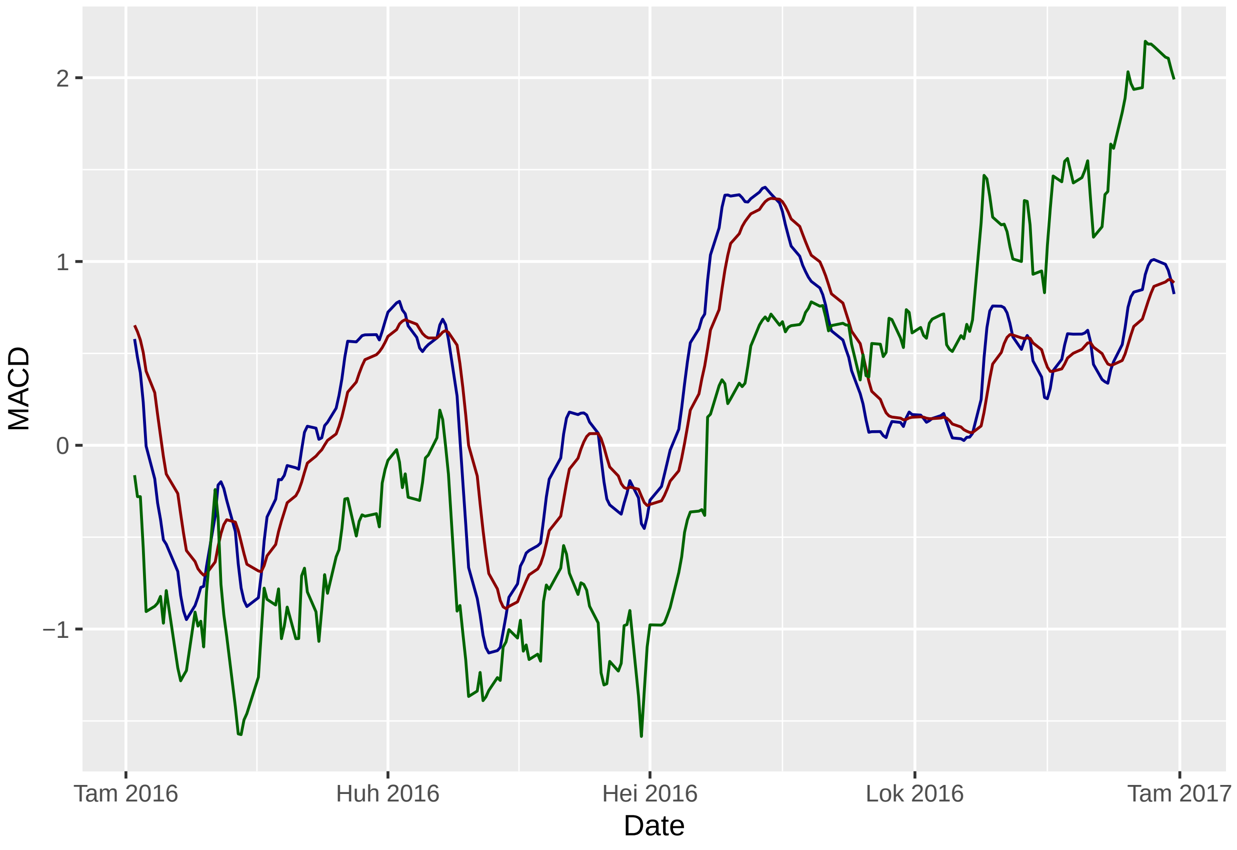
$$EMA = P_i(\frac{2}{n+1}) + EMA_{i-1}(1 - (\frac{2}{n+1}))$$

Where P_i is current closing price and i is the number of days. Then we can calculate our MACD and signal MACD.

$$MACD = EMA_{12} - EMA_{26}$$
$$SignalMACD = EMA_9(MACD)$$

Investor can get buying and selling signals by looking at intercepts of the MACD and it's signal line. Signal line is smoother than MACD line and reacts with lag to price changes.

```
price[, EMA12 := movavg(Adj_Close_Smooth, 12, type="e")]
price[, EMA26 := movavg(Adj_Close_Smooth, 26, type="e")]
price[, MACD := EMA12 - EMA26]
price[, SignalMACD := movavg(MACD, 9 , type ="e")]

# With quantmod
# MACD <- MACD(price$Adj_Close_Smooth, nFast = 12, nSlow = 26, nSig = 9)
# price[, MACD := MACD[, "macd"]]
# price[, SignalMACD := MACD[, "signal"]]

ggplot(price[year(Date) == 2016], aes(x = Date)) + geom_line(aes(y = MACD),
      color = "darkblue") + geom_line(aes(y = SignalMACD), color = "darkred") +
      geom_line(aes(y = scale(Adj_Close_Smooth)), color = "darkgreen")
```



Relative Strength Index

Relative strength index is once again a momentum indicator used by investors to recognize overbought and oversold situations. It measures magnitude of price changes within given time period. Many times 14 days is used as time period. As a rule of thumb asset is considered overbought (over valued) if RSI is over 70 and oversold (under valued) when it is less than 30.

$If\ close > close_{t-1}\ then,$

$up = close - close_{t-1}$

$down = 0$

$Else$

$up = 0$

$down = close_{t-1} - close$

```
n <- 14
price[, RSI := NA]
price[, RS := NA]

price[, Change := Adj_Close_Smooth - shift(Adj_Close_Smooth)]
price[, Up := pmax(Change, 0)]
price[, Down := abs(pmin(Change, 0))]
price[1:14, Av_UM := mean(Up, na.rm = T)]
price[1:14, Av_DM := mean(Down, na.rm = T)]


for (i in (n+1):price[, .N]){
  price$Av_UM[i] <- (price$Av_UM[i-1]*(n-1) + price$Up[i])/n
  price$Av_DM[i] <- (price$Av_DM[i-1]*(n-1) + price$Down[i])/n

  price$RS[i] <- price$Av_UM[i]/price$Av_DM[i]
  price$RSI[i] <- 100 -  (100/(1+(price$RS[i])))
}

# Could have been made much easier and faste by:
# price[, RSI :=  RSI(Adj_Close_Smooth, n=14)]

price <- drop_na(price)
head(price)
```

```
         Date   Adj_Open   Adj_High    Adj_Low  Adj_Close Adj_Volume
1: 1986-04-03 0.06414212 0.06587569 0.06414212 0.06414212   23040000
2: 1986-04-04 0.06414212 0.06471998 0.06414212 0.06414212   26582400
3: 1986-04-07 0.06414212 0.06471998 0.06183069 0.06298641   16560000
4: 1986-04-08 0.06298641 0.06471998 0.06298641 0.06356427   10252800
5: 1986-04-09 0.06356427 0.06529784 0.06356427 0.06471998   12153600
6: 1986-04-10 0.06471998 0.06587569 0.06356427 0.06529784   13881600
   Adj_Close_Smooth   Adj_Return       OBV      OSC      EMA12      EMA26
1:       0.06367984  0.009157509 282816000 45.00000 0.06354324 0.06401297
2:       0.06414212  0.007259528 309398400 50.00000 0.06363537 0.06402254
3:       0.06391098 -0.003603604 292838400 47.50000 0.06367777 0.06401427
4:       0.06310198 -0.012658228 282585600 47.69231 0.06358919 0.06394670
5:       0.06379541  0.010989011 294739200 67.27273 0.06362092 0.06393549
```

```
6:       0.06483555  0.016304348 308620800 83.63636 0.06380778 0.06400216
             MACD    SignalMACD      RSI        RS        Change           Up
1: -0.0004697335 -0.0003926349 46.71533 0.8767123  0.0005778570 0.0005778570
2: -0.0003871641 -0.0003915407 48.33968 0.9357218  0.0004622856 0.0004622856
3: -0.0003364996 -0.0003805325 47.55902 0.9069055 -0.0002311428 0.0000000000
4: -0.0003575060 -0.0003759272 44.83013 0.8125836 -0.0008089997 0.0000000000
5: -0.0003145737 -0.0003636565 47.60523 0.9085874  0.0006934284 0.0006934284
6: -0.0001943780 -0.0003298008 51.54265 1.0636704  0.0010401425 0.0010401425
           Down        Av_UM        Av_DM
1: 0.0000000000 0.0005283264 0.0006026223
2: 0.0000000000 0.0005236092 0.0005595778
3: 0.0002311428 0.0004862085 0.0005361182
4: 0.0008089997 0.0004514793 0.0005556097
5: 0.0000000000 0.0004687614 0.0005159233
6: 0.0000000000 0.0005095743 0.0004790716
```

Random forest

Before we can start training our model we have to change create our outcome variable. We define that it can have values +1 or -1, depending on direction where smoothed price has moved in given time period. We will use first 20 days period. This is a parameter for which we probably should try different values. I believe that something like 20 days would be good starting point. I have seen other studies where longer time horizon improves prediction capability of the model. On the other hand due to transaction costs I would hope model to predict trend for longer period than couple of days.

```
price[, Sign := sign(log(shift(Adj_Close_Smooth, 20, type = "lead")/Adj_Close_Smooth))]
```

```
Warning in ‘[.data.table‘(price, , ‘:=‘(Sign, sign(log(shift(Adj_Close_Smooth, :
Invalid .internal.selfref detected and fixed by taking a (shallow) copy of the
data.table so that := can add this new column by reference. At an earlier point,
this data.table has been copied by R (or was created manually using structure()
or similar). Avoid names<- and attr<- which in R currently (and oddly) may
copy the whole data.table. Use set* syntax instead to avoid copying: ?set, ?
setnames and ?setattr. If this message doesn’t help, please report your use case
to the data.table issue tracker so the root cause can be fixed or this message
improved.
```

```
price[Sign == 0, Sign := 1]
price[, Sign := as.factor(Sign)]
class(price$Sign)
```

```
[1] "factor"
```

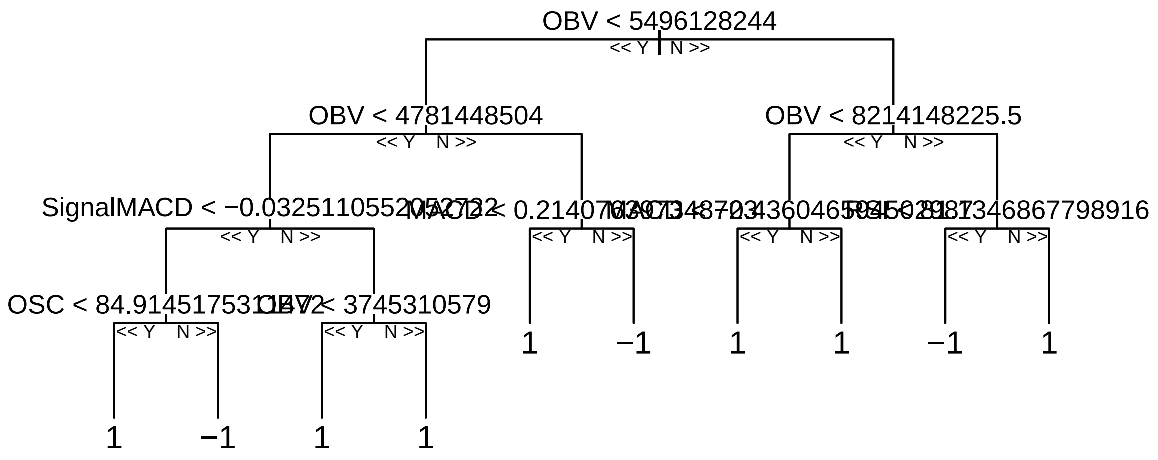
Before we can use random forest we need to take a look on decision trees. A decision tree is a supervised machine learning algorithm. Idea of the decision trees is to understand data by dividing it based on the characteristics. It will split the data until it can not classify it more clearly anymore. How this is measured? There are several methods. One of them is to minimize Gini impurity.

$$Gini\ impurity = 1 - \sum p_i^2$$

If we think of a classification problem, let's say with values 0 and 1. Then p_0 would be the percentage of 0 of the all observations entered to that node. We can easily see that Gini impurity gets it's lowest value when all observations in the node are homogeneous. It will get values between 0.5 and 0. Algorithm keeps on splitting the data if it can find threshold values for splitting that increase Gini impurity. In classification problem class that node classifies observations is based on majority of classes of outcomes in training data in that node. So if we have binary outcome variable and majority of observation allocated to node have outcome one, then this node will predict one for observation that end up to this node.

There is one major drawback with decision trees. They usually do very well explaining the data that they are constructed from, but when they are used for new data they perform way worse. This is called overfitting. One solution to this problem is to use Random Forest method. Random forest is a tree based machine learning algorithm. As one could guess from the name it is based on decision trees. In Random Forest instead of using whole data set, boot strap data set is created where observations are picked randomly. After that only subset of the variables is randomly chosen to build a decision tree in each step. This procedure will be repeated as many times as is declared, for example 200 times. When tested with new data, different decision trees in our forest probably give different results. Final result is again decided based on majority of results.

```r
library(devtools)
library(randomForest)
library(reprtree)

price <- na.omit(price)

feature_vars <- c("OBV", "OSC", "MACD", "SignalMACD", "RSI")
full_formula <- as.formula(paste(c("Sign ~ ", paste(feature_vars, collapse = " +
                                  ")), collapse = ""))

smp_size <- floor(0.8 * nrow(price))

temp <- copy(price)
train_indices <- sample(seq_len(nrow(temp)), size = smp_size)

temp[train_indices, dataset:="train"]
temp[-train_indices, dataset:="test"]


# Example Random Forest tree, just for illustration purposes
example_rf_classifier <- randomForest(full_formula, data = temp[train_indices],
                            importance=TRUE, ntree = 1, maxnodes=10, mtry = 3)

reprtree:::plot.getTree(example_rf_classifier)
```



There are several ways to measure performance of Random Forest model or other classification models. As mentioned we have a classification problem. This means that our response variable can get only two values. If model predicts that stock return is $Return_i \geq 0$ it will get value 1 and if it predicts that stock price goes down $Return_i < 0$ it will give value -1. We also know the realized returns. When we factorize them they will also only get values -1 and 1. After we have done our predictions there are four different options: True positive means that model predicted price increase and price actually increased. True negative means that model predicted price decrease and price actually decreased. False positive means that model predicted price increase, but price actually decreased. False negative means situation where model predicts price decrease, but price increases. From these values we can derive performance metrics for our model, namely accuracy, sensitivity, specificity and precision.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

Accuracy is pretty intuitive measure. It measures how many observations model predicted correctly compared to all observations. Naturally we usually want to reach as high accuracy as possible.

$$Sensitivity = \frac{TP}{TP+FN}$$

Sensitivity, sometime referred as recall, measures ratio between observations model correctly predicted to be positive and total number of positive observations. In this case it would mean how many of the stock increase periods model correctly predicted to be positive.

$$Specificity = \frac{TN}{TN+FP}$$

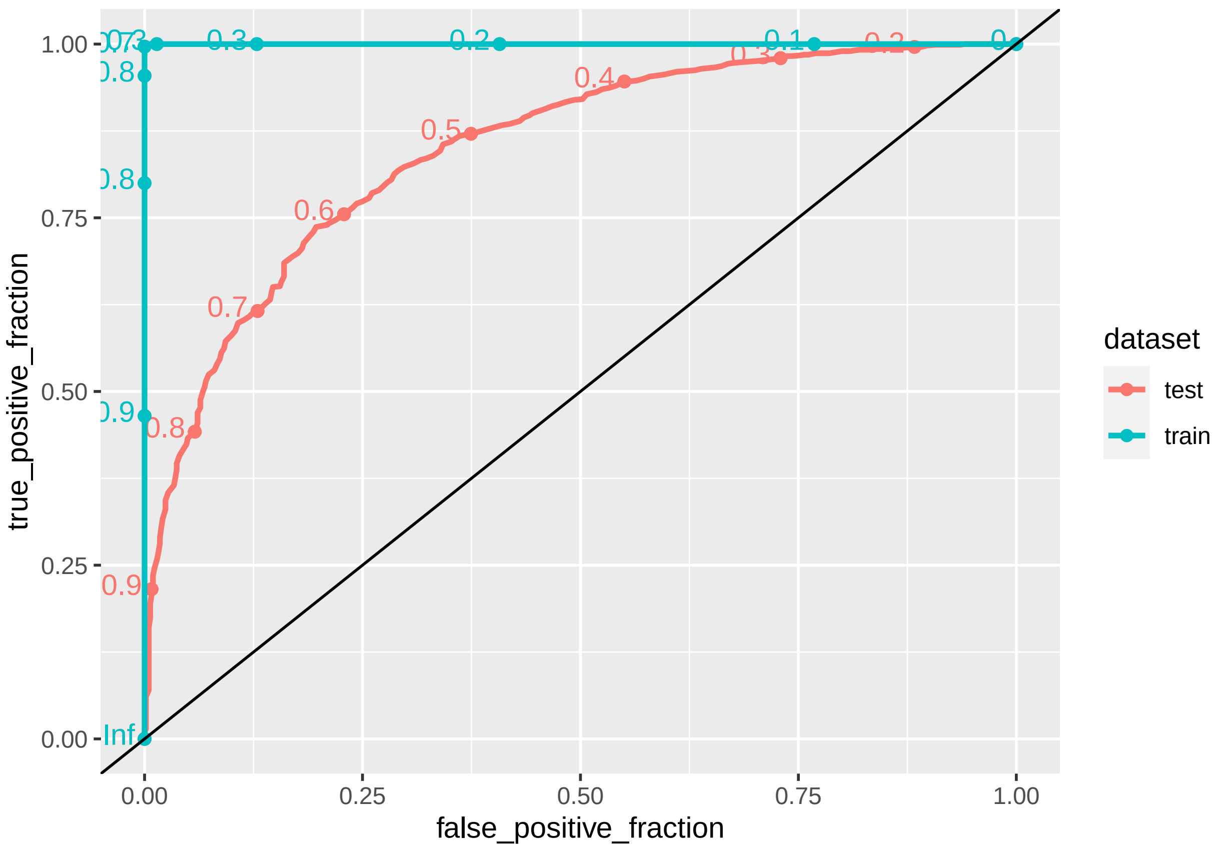
Specificity can also be called true negative rate. This is opposite to sensitivity. It measures number of correctly predicted negatives compared to total negatives. Depending of the goal of the study, one can decide whether to focus on sensitivity or specificity. For example if it is important to predict all the positives correctly, but it doesn't harm to have false positives, one should focus on sensitivity.

$$Precision = \frac{TP}{TP+FP}$$

Precision measures how many of the models positive predictions actually were positive. In stock return case precision would tell us how many percent of models price increase predictions actually were true.

Before simulating the investment decision and dividing the data, let's first see how well our model works on the whole data. One way to visualize performance of the model is to plot a ROC curve. The receiver operating characteristic curve is a way of measuring model performance with different cutoff values. On the y axis it has sensitivity of the model and on x axis it has false positive rate with different cutoff values. What are these cutoff values?

```r
rf_classifier <- randomForest(full_formula, data = temp[train_indices],
                              importance=TRUE, ntree = 200, mtry = 3)

temp[, Pred_Prob := predict(rf_classifier, type="prob", newdata = price)[, 2]]
temp[, Pred_Outcome := predict(rf_classifier, type="class", newdata = price)]

roc <- ggplot(temp, aes(d=Sign, m=Pred_Prob, color = dataset)) + geom_roc() + geom_abline()
roc
```



I have to admit that I am not sure if I have understood intuition of ROC curve for random forest correctly. Nevertheless this is how I have understood it. As mentioned algorithm splits the data based on Gini impurity. It is possible that the final leaf isn't pure. Meaning that there could be observations with different outcomes in same leaf. By default majority of outcomes defines the leafs value. If you type in R predict function argument "type="prob" " it will give you probabilities of outcomes. So on default leafs with probability lower than 0.5 for up get down value and leafs with greater probability get up value. This 0.5 is called threshold value.

What the ROC curve does is that it changes the threshold values that define leaf's value and plot above mentioned values on these threshold values. One extreme case being that all outcome leafs would get positive value. Then all positives get positive values and sensitivity value is 1, but also all negatives get positive value and false positive fraction would get value of 1. We would end up in the up right corner. So the ROC curve plots the trade off between fraction of ups predicted correctly and fraction of downs predicted wrong. More homogeneous (meaning more extreme probabilities) leafs are, more towards the up left corner the curve is, since if majority of observations in leaf have same outcome chancing the threshold value doesn't change the leaf's value.

Our ROC curve for train set seems pretty perfect. This isn't surprise since we didn't restrict the height of the tree. On the other hand model didn't perform as good with the test data, but it still did much better than random guess indicated by straight line. With same threshold values training and test data can have different values. It is possible that when we trained our model all observations that ended up in a leaf had positive value, which result homogeneous leaf. Nevertheless when we run test data observation that ends up to that leaf can have negative outcome value and therefore result a false positive. Basically ROC curve plot many confusion matrices once.

```
# Confusion matrix with default threshold
confusionMatrix(temp[-train_indices, Sign], temp[-train_indices, Pred_Outcome])$table
```

```
          Reference
Prediction  -1   1
        -1 413 212
        1  148 836
```

Some times instead of plotting ROC curve it could be convenient to just report a single number indicating the performance of our model. Luckily we have this kind of number. It is called area under curve. As the name indicates it measures the are under ROC curve. It will usually have values between 0.5 and 1. 1 being the best possible.

```
calc_auc(roc)
```

```
Warning in verify_d(data$d): D not labeled 0/1, assuming -1 = 0 and 1 = 1!
```

```
  PANEL group       AUC
1     1     1 0.8483545
2     1     2 1.0000000
```

Our goal is to deploy random forest for tool to investor. Above we saw some metrics how well our algorithm was capable to forecast stock movement direction. Next we are going to train our algorithm as before and after training we are going to give it next days information and let it forecast the direction. We do this for each month in our data set (not for the 16 first years). Then we check how many month algorithm forecast correctly. This is a bit like simulating investors decision making in the past. It is interesting to see how well or badly our algorithm worked compared to buy and forget strategy. Meaning in this set that you would every month expect stock to go up.

```
start_date <- "2015-01-02"
smh <- price[Date >= start_date]
dates <- smh[endpoints(Date, "month")]$Date

smp_size <- floor(0.8 * nrow(price[Date < dates[1]]))

rand_forest <- function(dt, date){
  new_dt <- dt[Date == date]
  data <- dt[Date < date & Date >= as.Date(date) %m-% years(16)]
  smp_size <- floor(0.8 * nrow(data))
  train_indices <- sample(seq_len(nrow(data)), size = smp_size)
  rf_classifier <- randomForest(full_formula, data = data[train_indices],
                          importance=TRUE, ntree = 200, mtry = 3)
  return(data.table(Date = date, pred=predict(rf_classifier, newdata=new_dt,
                                         type = "class")))
}
```

```r
pred <- rbindlist(lapply(dates, function(t){rand_forest(price, t)}))

price[, Mean := mean(Adj_Return)]

smh <- merge(price, pred, by = "Date")

confusionMatrix(smh$Sign, smh$pred)
```

```
Confusion Matrix and Statistics

          Reference
Prediction -1  1
        -1  8  5
        1   6 19

               Accuracy : 0.7105
                 95% CI : (0.541, 0.8458)
    No Information Rate : 0.6316
    P-Value [Acc > NIR] : 0.2016

                  Kappa : 0.3686

 Mcnemar's Test P-Value : 1.0000

            Sensitivity : 0.5714
            Specificity : 0.7917
         Pos Pred Value : 0.6154
         Neg Pred Value : 0.7600
             Prevalence : 0.3684
         Detection Rate : 0.2105
   Detection Prevalence : 0.3421
      Balanced Accuracy : 0.6815

       'Positive' Class : -1
```

```r
smh[, ones := as.factor(replicate(length(smh$pred), 1))]
confusionMatrix(smh$ones, smh$pred)
```

```
Warning in confusionMatrix.default(smh$ones, smh$pred): Levels are not in the
same order for reference and data. Refactoring data to match.

Confusion Matrix and Statistics

          Reference
Prediction -1  1
        -1  0  0
        1  14 24

               Accuracy : 0.6316
                 95% CI : (0.4599, 0.7819)
    No Information Rate : 0.6316
    P-Value [Acc > NIR] : 0.572310

                  Kappa : 0

 Mcnemar's Test P-Value : 0.000512

            Sensitivity : 0.0000
            Specificity : 1.0000
         Pos Pred Value :    NaN
         Neg Pred Value : 0.6316
             Prevalence : 0.3684
         Detection Rate : 0.0000
   Detection Prevalence : 0.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : -1
```

```
# Save the results for later usage
saveRDS(smh, file = "extdata/rnd_forest_result.Rda")
```