Predicting US Equities Trends Using Random Forests

Ahmad Mohammad

Jul 1, 2016

Introduction

One of the main reasons I started studying machine learning is to apply it stock market and this is my first post to do so. Specifically, we are going to predict some U.S. stocks using machine leaning models.

Depending on whether we are trying to predict the price trend or the exact price, stock market prediction can be a classification problem or a regression one. But we are only going to deal with predicting the price trend as a starting point in this post. The machine learning model we are going to use is random forests. One of the main advantages of the random forests model is that we do not need any data scaling or normalization before training it. Also the model does not strictly need parameter tuning such as in the case of support vector machine (SVM) and neural networks (NN) models. However, research indicates that SVM and NN achieved astonishing results in predicting stock price movements (1). But we will leave them for another separate post. Manojlovic and Staduhar (2) provides a great implementation of random forests for stock price prediction. This post is a semi-replication of their paper with few differences. They used the model to predict the stock direction of Zagreb stock exchange 5 and 10 days ahead achieving accuracies ranging from

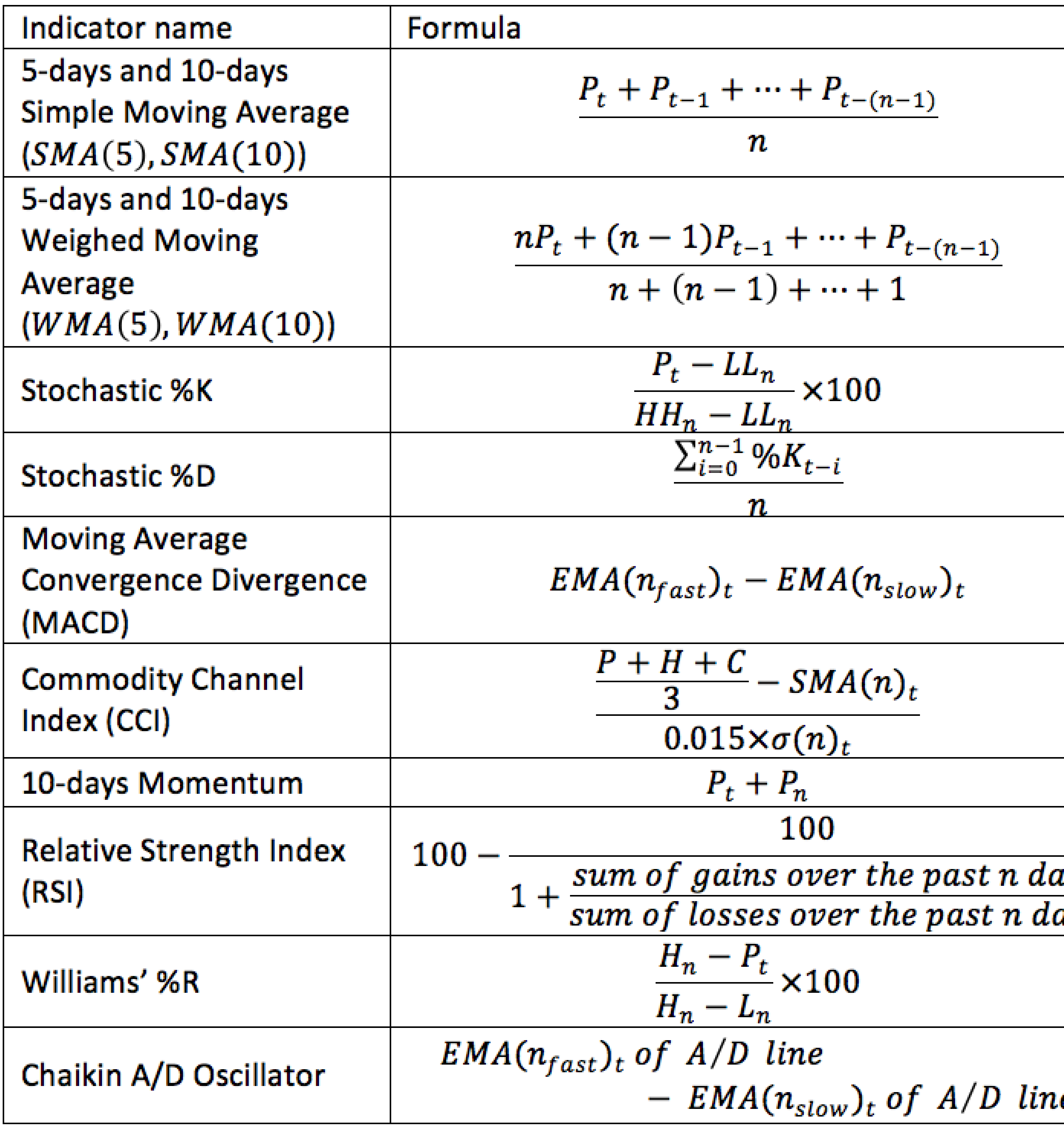
0.76 to 0.816. We are going to use the same methods as the ones in the paper with similar technical indicators (only two different ones) to predict the US stock market movement instead of Zagreb stock exchange and varying the days ahead from 1 to 20 days head instead of just 5 and 10 days ahead.

Research Data

I chose 8 of the top companies in the S&P 500 in terms of market cap: AAP, BRK-B, GE, JNJ, MSFT, T, VZ, XOM.

The data sets for all the stocks are from May 5th, 1998 to May 4th, 2015 with total of 4277 days (the figure above shows a higher range). Since we have 8 stocks and we are going to predict the price movement from 1 to 20 days ahead, we will have a total of 160 data sets to train and evaluate. But before proceeding with training the data, we had to check weather the data are balanced. The figure below shows the percentage of positive returns instances for each day and for each stock. Fortunately, the data does not need to be balanced since they are almost evenly split for all the stocks.

The technical indicators were calculated with their default parameters settings using the awesome `TA-Lib` python package. They are summarized in the table below where P_tPt is the closing price at the day t, H_tHt is the high price at day t, L_tLt is the low price at day t, HH_nHHn is the highest high during the last n days, LL_tLLt is the lowest low during the last n days, and $EMA(n)$EMA(n) is the exponential moving average.



Indicator name	Formula
5-days and 10-days Simple Moving Average $(SMA(5), SMA(10))$	$\frac{P_t + P_{t-1} + \cdots + P_{t-(n-1)}}{n}$
5-days and 10-days Weighed Moving Average $(WMA(5), WMA(10))$	$\frac{nP_t + (n-1)P_{t-1} + \cdots + P_{t-(n-1)}}{n + (n-1) + \cdots + 1}$
Stochastic %K	$\frac{P_t - LL_n}{HH_n - LL_n} \times 100$
Stochastic %D	$\frac{\sum_{i=0}^{n-1} \%K_{t-i}}{n}$
Moving Average Convergence Divergence (MACD)	$EMA(n_{fast})_t - EMA(n_{slow})_t$
Commodity Channel Index (CCI)	$\frac{\frac{P+H+C}{3} - SMA(n)_t}{0.015 \times \sigma(n)_t}$
10-days Momentum	$P_t + P_n$
Relative Strength Index (RSI)	$100 - \frac{100}{1 + \frac{sum\ of\ gains\ over\ the\ past\ n\ da}{sum\ of\ losses\ over\ the\ past\ n\ da}}$
Williams' %R	$\frac{H_n - P_t}{H_n - L_n} \times 100$
Chaikin A/D Oscillator	$EMA(n_{fast})_t\ of\ A/D\ line - EMA(n_{slow})_t\ of\ A/D\ lin$

Model and Method

As mentioned above, one of the advantages of random forests is that it does not strictly need parameter tuning. Random forests, first introduced by breidman (3), is an aggregation of another weaker machine learning model, decision trees. First, a bootstrapped sample is taken from the training set. Then, a random number of features are chosen to form a decision tree. Finally, each tree is trained and grown to the fullest extend possible without pruning. Those three steps are repeated n times form random decision trees. Each tree gives a classification and the classification that has the most votes is chosen. For the number of trees in the random forests, I chose 300 trees. I could go for a higher number but according to research, a larger number of trees does not always give better performance and only increases the computational cost (4). Since we will not be tuning the model's parameters, we are only going to split the data to train and test set (no validation set). For the scores, I used the accuracy score and the f1 score. The accuracy score is simply the percentage (or fraction) of the instances correctly classified. The f1 score calculated by

$$F1 = 2\frac{precision \times recall}{precision + recall}$$

F1=2precision×recallprecision+recall

$$precision = \frac{tp}{tp + fp}$$

precision=tptp+fp

$$recall = \frac{tp}{tp + fn}$$

recall=tptp+fn

where tptp is the number of positive instances classifier as positive, fpfp is the number of negative instances classified as positive and fnfn is the number of positive instances classified as negative. Because of the randomness of the model, each train set is trained 5 times and the average of the scores on the test set is the final score. All of the calculation were done by python's `scikit-learn` library.

Results

As seen from the two figures above, we get poor results for small number of days ahead (from 1 to 4) and greater results as the number of days ahead increases afterworlds. For almost all the stocks for both scores, the highest scores are in the range of 17 to 20-days ahead.

Conclusion

In this post, we demonstrated the use of one machine learning model, random forests, to predict the price movement (positive or negative) of some of the major US equities. We got satisfying results with ten technical indicators when predicting the price movement mostly within 14 to 20 days ahead witch scores ranging from 0.78 to 0.84 for the accuracy scores and from 0.76 to 0.87 for the f1 scores. We probably could have better results if we add more features that includes fundamentals and macro economic variables. Also, since commodities are highly leveraged, we could use minute by minute data to predict the movement of commodity prices at the end of the day. Actually this was my initial choice of data. But I could not find a source that provides minute by minute data for free.

References

1. Yakup Kara, Melek Acar Boyacioglu, Ömer Kaan Baykan, Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange, Expert Systems with Applications, Volume 38, Issue 5, May 2011, Pages 5311-5319.

2. Manojlovic, T., & Stajduhar, I.. (2015). Predicting stock market trends using random forests: A sample of the Zagreb stock exchange. MIPRO.

3. L. Breiman, Random forests, Machine Learning, 45(1):5–32, 2001

4. Oshiro, Thais Mayumi, Pedro Santoro Perez, and José Augusto Baranauskas. “How Many Trees in a Random Forest?” Machine Learning and Data Mining in Pattern Recognition Lecture Notes in Computer Science (2012): 154-68. Web.

Code to produce all the results above:

```
=IFERROR(IF(BK6=1, if(AV6=1,
                                        IF(AT6<C6*(1-$AT$2)*(1-$DA$3),1,0)),"  "),

IF(BK6=-1,if(av6=1,IF(AS6>C6*(1+$AT$2)*(1+$DA$3),1,0)," ")," ")," ")," ")
```

```
import pandas as pd
import pandas_datareader.data as web
import datetime
import numpy as np
from talib.abstract import *
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from plotly.offline import download_plotlyjs, init_notebook_mode, iplot
import cufflinks as cf
init_notebook_mode()
```

```
start = datetime.datetime(1998, 1, 1)
end = datetime.datetime(2016, 6, 30)
top_500 = ['AAPL', 'MSFT', 'XOM', 'JNJ', 'GE', 'BRK-B', 'T', 'VZ']
```

```
f = web.DataReader(top_500, 'yahoo',start,end)
cleanData = f.ix['Adj Close']
stock_data = pd.DataFrame(cleanData)
```

```
stock_data.iplot(dimensions=(950,400), yTitle='Daily Price ($)')
```

```
stocks = {}
for i in top_500:
    stocks[i] = web.DataReader(i, 'yahoo',start,end)
```

```
for i,j in enumerate(stocks):
    stocks[j].columns = [s.lower() for s in stocks[j].columns]
    stocks[j].volume = stocks[j].volume.apply(lambda x: float(x))
```

```
ocks[j].volume = stocks[j].volume.apply(lambda x: float(x))
```

In [7]:

```
def get_indicators(stocks, period):
    stocks_indicators = {}
    for i in stocks:
        features = pd.DataFrame(SMA(stocks[i], timeperiod=5))
        features.columns = ['sma_5']
        features['sma_10'] = pd.DataFrame(SMA(stocks[i], timeperiod=10))
        features['mom_10'] = pd.DataFrame(MOM(stocks[i],10))
        features['wma_10'] = pd.DataFrame(WMA(stocks[i],10))
        features['wma_5'] = pd.DataFrame(WMA(stocks[i],5))
        features = pd.concat([features,STOCHF(stocks[i],
                                          fastk_period=14,
                                          fastd_period=3)],
                             axis=1)
        features['macd'] = pd.DataFrame(MACD(stocks[i], fastperiod=12,
slowperiod=26)['macd'])
        features['rsi'] = pd.DataFrame(RSI(stocks[i], timeperiod=14))
        features['willr'] = pd.DataFrame(WILLR(stocks[i], timeperiod=14))
        features['cci'] = pd.DataFrame(CCI(stocks[i], timeperiod=14))
        features['adosc'] = pd.DataFrame(ADOSC(stocks[i], fastperiod=3,
slowperiod=10))
        features['pct_change'] = ROC(stocks[i], timeperiod=period)
        features['pct_change'] = features['pct_change'].shift(-period)
        features['pct_change'] = features['pct_change'].apply(lambda x: '1'
if x > 0 else '0' if x <= 0 else np.nan)
        features = features.dropna()
        features = features.iloc[np.where(features.index=='1998-5-5')[0]
[0]:np.where(features.index=='2015-5-5')[0][0]]
        stocks_indicators[i] = features
    return stocks_indicators
```

```python
stocks_indicators = get_indicators(stocks, 1)
```

```python
stocks_indicators['AAPL'].head()
```

Out[40]:

```
sma_5sma_10mom_10wma_10wma_5fastkfastdmacdrsiwillrcciadoscpct_changeDate1998-05-
0528.225027.893750.68749928.15340928.69583394.82755976.6709240.68195865.507369-
5.172441149.90075812066108.52568511998-05-
0628.887528.175002.81250028.59318229.39166797.01494991.7252900.81065168.116574-
2.985051159.42351371289716.76583101998-05-
0729.450028.425002.50000028.95909129.82500090.00000093.9475020.89226967.024556-
10.000000137.25490481706570.12778411998-05-
0829.937528.675002.49999929.32500030.15416695.71426394.2430700.96598968.125177-
4.285737116.66667495295085.32583911998-05-
1130.312528.993753.18750129.73636430.48750087.20930290.9745221.05262570.262954-
12.790698134.90850761962542.8772400
```

```python
len(stocks_indicators['AAPL'])
```

```python
def weighs_tabale(stocks, period):
    table = pd.DataFrame()
    for j in stocks:
        weighs_1 = []
        for i in range(1,period+1):
            stocks_indicators = get_indicators(stocks, i)
            weighs_1.append((len(stocks_indicators[j]
[stocks_indicators[j]['pct_change']=='1'])/\
                                float(len(stocks_indicators[j])))*100)
        table = pd.concat([table, pd.DataFrame(weighs_1)], axis=1)
    table.index = range(1,period+1)
    table.columns = stocks.keys()
    return table
```

```python
table = weighs_tabale(stocks, 20)
```

```python
table.iplot(kind='bar', subplots=True, dimensions=(950,500),
title='Percentage of the Increase Data Points')
```

```python
def avg_score(x_train, y_train,x_test,y_test,trees):
    accuracy = []
    f1 = []
    rf_model = RandomForestClassifier(trees)
    for i in range(5):
        rf_model.fit(x_train,y_train)
        accuracy.append(rf_model.score(x_test,y_test))
        f1.append(f1_score(y_test,rf_model.predict(x_test),
pos_label='1'))
    avg_accuracy = sum(accuracy)/len(accuracy)
    avg_f1 = sum(f1)/len(f1)
    return avg_accuracy, avg_f1
```

```
def accuracy(stocks, trees, period):
    table_accuracy = pd.DataFrame()
    table_f1 = pd.DataFrame()
    for j in stocks:
        accuracy_values = []
        f1_values = []
        for i in range(1,period+1):
            stocks_indicators = get_indicators(stocks, i)
            train, test = train_test_split(stocks_indicators[j])
            accuracy, f1 = avg_score(train.iloc[:,:-1],train.iloc[:,-
1],test.iloc[:,:-1],test.iloc[:,-1],trees)
            accuracy_values.append(accuracy)
            f1_values.append(f1)
        table_accuracy = pd.concat([table_accuracy, pd.DataFrame({j :
accuracy_values})], axis=1)
        table_f1 = pd.concat([table_f1, pd.DataFrame({j : f1_values})],
axis=1)
    table_accuracy.index = range(1,period+1)
    table_f1.index = range(1,period+1)
    return table_accuracy, table_f1
```

In [16]:

```
accuracy_table, f1_table = accuracy(stocks, 300, 20)
```

```
accuracy_table.iplot(dimensions=(950,400), xTitle='Days Ahead',
yTitle='Average Score', title='Accuracy scores')
```

```
f1_table.iplot(dimensions=(950,400), xTitle='Days Ahead', yTitle='Average
Score', title='F1 scores')
```

```
def highlight_max(s):
    '''
    highlight the maximum in a Series yellow.
    '''
    is_max = s == s.max()
    return ['background-color: yellow' if v else '' for v in is_max
```

```
accuracy_table.style.apply(highlight_max, axis=0)
```

Out[38]:

```
f1_table.style.apply(highlight_max, axis=0)
```

Out[39]: