Machine Learning Engineer Nanodegree

Capstone Project

Saurabh Kumar
December 24th, 2017

I. Definition

Project Overview

A Stock market is a place where humans and computers buy and sell shares of companies. Shares are small pieces of a company. There is a lot of money involved. Mathematicians and statisticians have been always interested in Stock Market given the amount of money involved. To predict the market behavior has always allured many. People try to find patterns in the behavior of stock market. However, given the huge amount of data and buy/sell decisions carried out every day makes it almost impossible to analyze the stock market manually.

Investment firms, hedge funds and even individuals have been using financial models to better understand market behavior and make profitable investments and trades. A wealth of information is available in the form of historical stock prices and company performance data, suitable for machine learning algorithms to process. In the recent few years, the emergence of organized data, high computational power and machine learning algorithms have made it a bit easier to predict the behavior of stock market

Problem Statement

The problem statement for me here is that, is it possible to apply regression machine learning techniques and predict the market trend. I wish to highlight here that there are many key indexes based on which stock market behavior can be predicted. For Example : Stock Market Opening Price or Closing Price. I have taken here another key index: “High” which is highest market price of the day.

Applying Supervised Machine Learning techniques and time series analysis model (ARIMA), I have tried to predict the stock market trend for a span of time. This time span is arbitrary. The problem statement is to capture the future trend. Can we predict about future stock market trend for years?

I have tried two approaches:

1. Prediction of feature “High” 5 days later.
2. Using ARIMA model prediction of feature “High”

Metrics

I have used three supervised learning techniques for my approach using sklearn models:

1. Linear Regression
2. K nearest neighbor regression
3. Gradient Boosting regression

Considering the problem as regression problem. For above-mentioned models, "`r2_score`" has been chosen as the metric to measure the performance of models. Basically, this metric is by default Regression model score function. It provides a measure of how well future samples are likely to be predicted by the model.

Working

If $\hat{y}_i$ is the predicted value of the i-th sample and y_i is the corresponding true value, then the score R^2 estimated over n_{samples} is defined as

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n_{\text{samples}}-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n_{\text{samples}}-1}(y_i - \bar{y})^2}$$

where $\bar{y} = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} y_i$.

Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0

My second approach was using ARIMA model (Time Series Analysis Model)

I would be using RMSE Root Mean Square Error

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. Root mean square error is commonly used in climatology, forecasting, and regression analysis to verify experimental results.
The formula is:

$$RMSE = \sqrt{\overline{(f - o)^2}}$$

Where:

- f = forecasts (expected values or unknown results),
- o = observed values (known results).

II. Analysis

Data Exploration

I have used historical data from S&P. The Standard & Poor's 500, often abbreviated as the S&P 500, or just the S&P, is an American stock market index based on the market capitalizations of 500 large companies having common stock listed on the NYSE or NASDAQ. I have downloaded the data using a link [1].

Inputs in the dataset

Date – Date of the day

Open – Opening market price of the day

High – Highest market price of the day

Low – Lowest market price of the day

Close – Closing market price of the day

Volume – Number of transaction on the given day

A sample snapshot

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
data = pd.read_csv('dataset.csv', index_col='Date',date_parser=dateparse)
print data.head()
```

```
                     Open  High   Low  Close  Volume
Date
1789-05-01 00:00:00  0.51  0.51  0.51   0.51     NaN
1789-06-01 00:00:00  0.51  0.51  0.51   0.51     NaN
1789-07-01 00:00:00  0.50  0.50  0.50   0.50     NaN
1789-08-01 00:00:00  0.50  0.51  0.50   0.51     NaN
1789-09-01 00:00:00  0.51  0.51  0.50   0.51     NaN
```

As we can see, the dataset begins from year 1789. The Volume column of the data is empty for starting years but for the latest years, we have complete data:

```
print data.tail()
#checking the datatype
print '\n Data Types:'
print data.dtypes
```

```
            Date     Open     High      Low    Close      Volume
37585  2017-12-18  2685.92  2694.97  2685.92  2690.16  608455168.0
37586  2017-12-19  2692.71  2694.44  2680.74  2681.47  556473472.0
37587  2017-12-20  2688.18  2691.01  2676.11  2679.25  521377568.0
37588  2017-12-21  2683.02  2692.64  2682.40  2684.57  511474976.0
37589  2017-12-22  2684.22  2685.35  2678.13  2683.34          NaN
```

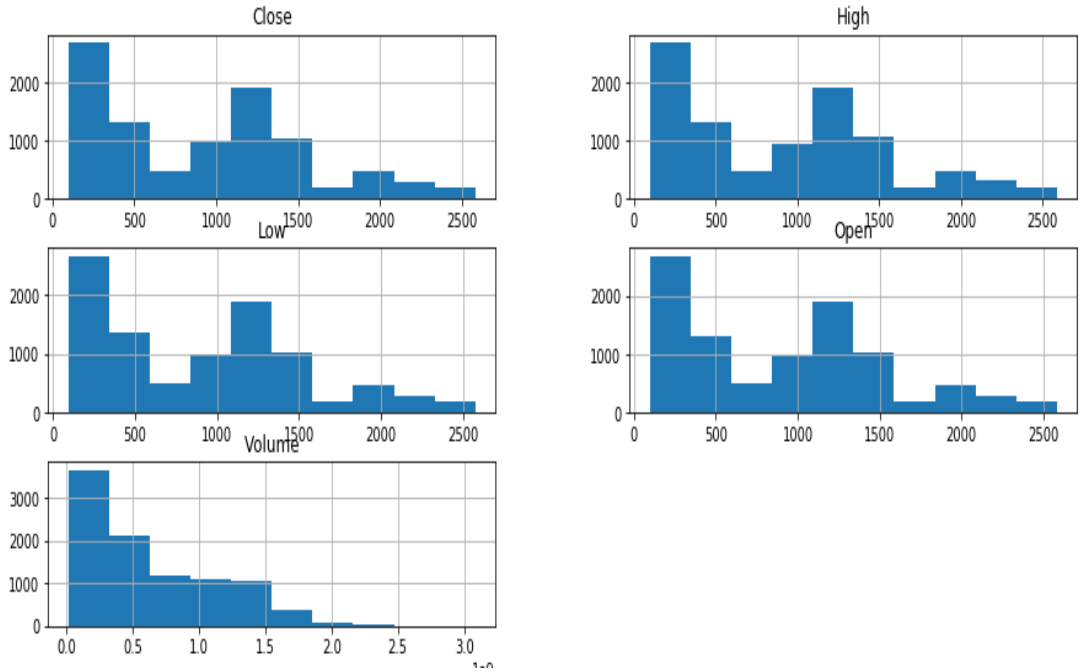
For the purpose of prediction and complete data, I have sliced the dataset and taken data only starting year 1980 in account. A total count of 9544 records.

```
In [96]: Actual_data = data['1980-01-01':'2017-11-01']

In [97]: Actual_data['High'].count()

Out[97]: 9544
```

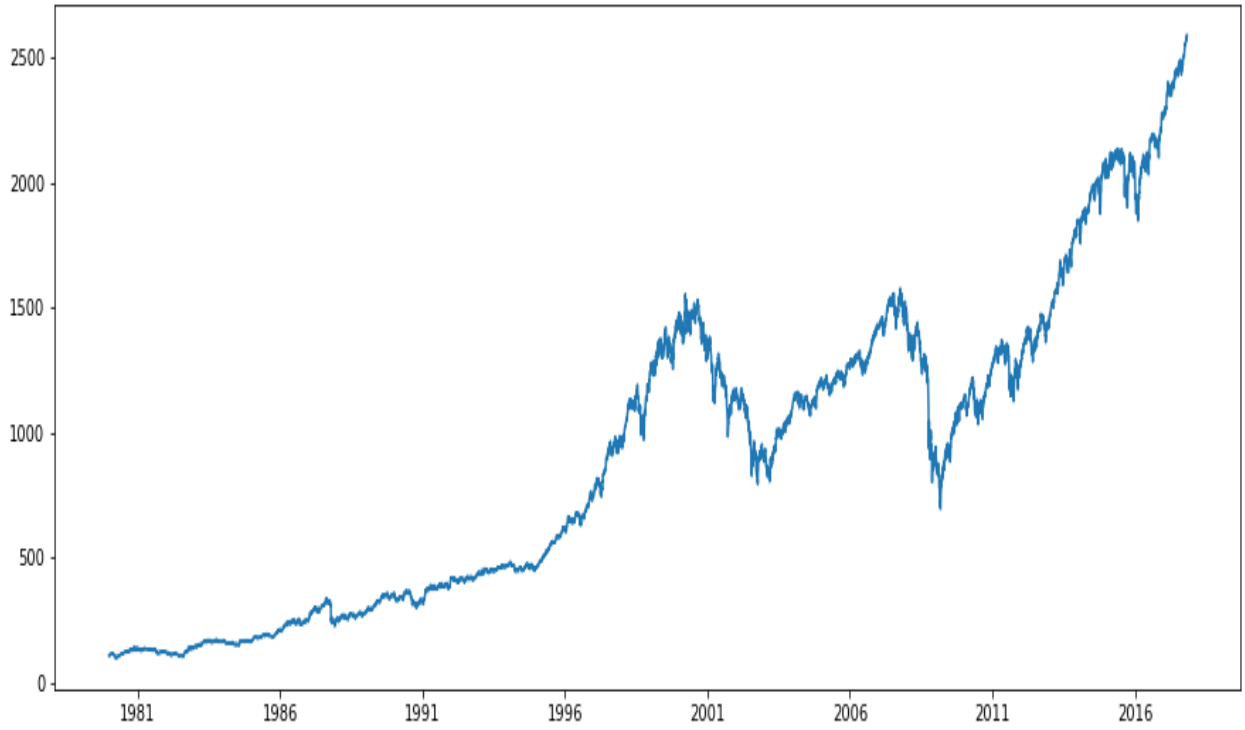
I did this to keep the prediction of stock market trend relevant to current years rather than past centuries.

Exploratory Visualization

A simple histogram of the dataset gives us clue of the values of each feature and their spread:



As I wish to focus on the feature „High, Higesht Stock Price“of the dataset. I did a basic plot of feature against Datetime index



- y-Axis is the value of Highest Stock Market Price of the day
- x- Axis is the time
- The plot shows value of feature “High” for each day

Algorithms and Techniques

For this project, I will be using three supervised learning algorithms to "predict" stock prices:

A) sklearn's **Linear Regression**
Sklearn’s linear regression fits the best line to the given data. The equation for this algorithm is y = mx + b. The algorithm processes a dataset to produce the m and b variables. Once complete, the algorithm now relies on the y = mx + b equation to produce a prediction. In this project, the target prediction is the price of a stock 5 days later. For the prediction, the x features are fed into the querying function and the result is a predictive y variable. In this model, the data is “thrown away” after the algorithm has been trained with a subset of the data. After the training phase, the algorithm solely relies on the constructed y = mx + b equation for predictions. This project, I will use the KNN and Gradient Boost model to compare and contrast results against the linear regression model.

B) sklearn's **KNN Regression**
KNN regression is different than linear regression in many ways. Firstly, KNN does not “throw away” the data after processing a subset of the data during the training phase. In fact, the knn regression model is not really required to be trained to prep itself for predicting a result with unseen data. This is because knn is an algorithm that simply processes surrounding information

to make a prediction. This is known as taking an instance-based approach to predictive modeling. In other words, the raw data is used for querying a prediction to a future stock price as opposed to running the queried data through an algorithmic function. My hypothesis is that the knn regression algorithm will produce a better predictive score than the linear regression algorithm.

c) sklearn **Gradient boosting Regressor**

If linear regression was a Car, then gradient boosting would be a Helicopter.

Gradient Boosting builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

And a loss function or cost function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event

d) **ARIMA**

In statistics and econometrics, and in particular in time series analysis, an autoregressive integrated moving average (ARIMA) model is a generalization of an autoregressive moving average (ARMA) model. Both of these models are fit to time series data either to better understand the data or to predict future points in the series (forecasting). ARIMA models are applied in some cases where data show evidence of non-stationarity, where an initial differencing step (corresponding to the "integrated" part of the model) can be applied one or more times to eliminate the non-stationarity.

The AR part of ARIMA indicates that the evolving variable of interest is regressed on its own lagged (i.e., prior) values. The MA part indicates that the regression error is actually a linear combination of error terms whose values occurred contemporaneously and at various times in the past. The I (for "integrated") indicates that the data values have been replaced with the difference between their values and the previous values (and this differencing process may have been performed more than once). The purpose of each of these features is to make the model fit the data as well as possible.

The ARIMA forecasting for a stationary time series is nothing but a linear (like a linear regression) equation. The predictors depend on the parameters (p,d,q) of the ARIMA model:

1. Number of AR (Auto-Regressive) terms (p): AR terms are just lags of dependent variable. For instance if p is 5, the predictors for x(t) will be x(t-1)….x(t-5).
2. Number of MA (Moving Average) terms (q): MA terms are lagged forecast errors in prediction equation. For instance if q is 5, the predictors for x(t) will be e(t-1)….e(t-5) where e(i) is the difference between the moving average at ith instant and actual value.
3. Number of Differences (d): These are the number of nonseasonal differences, i.e. in this case we took the first order difference. So either we can pass that variable and put d=0 or pass the original variable and put d=1. Both will generate same results.

Techniques:

1. For sklearn Supervised Model idea is to
    - process the data
    - Add an extra column with “High 5 days later”
    - Target Value is “High 5 days later”
    - use the model to train and predict the target value
    - Validate the model

2. For ARIMA approach, I will
    - Check if Dataset or Timeseries is Stationary
    - Make Timeseries Sationary
    - Forecast the target value
    - Validate the model

Benchmark

I will be taking here sklearn Linear Regressor as the Benchmark model. I would comparing the results and score of each model to the Linear Regressor model result.

III. Methodology

Data Preprocessing

For supervised learning:

The following pre-processing steps were taken:

1. Slicing the dataset and keeping data only from year 1980.
2. Use of dateparse function to read the Date column of feature and use this column for Indexing.
3. Adding a new column in dataset High_after_5_days
4. Keeping ['Open', ‘High’, 'Low', 'Close', 'Volume'] as features and ‘High_after_5_days’ as Target value.

For ARIMA model:

1. Slicing the dataset and keeping data only from year 1980.
2. Use of dateparse function to read the Date column of feature and use this column for Indexing.

For both approaches, I have not done any data scaling or normalization. I wish to check if the models are capable enough to predict real values. For ARIMA approach data preprocessing involved a lot of steps. I will be covering that in implementation section of this report.

Implementation

Supervised learning implementation:

1. Loading the dataset and using date_parser to read 'Date' column.

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
data = pd.read_csv('dataset.csv', index_col='Date',date_parser=dateparse)
print data.tail()
```

```
              Open      High       Low     Close        Volume
Date
2017-12-18  2685.92  2694.97  2685.92  2690.16  608455168.0
2017-12-19  2692.71  2694.44  2680.74  2681.47  556473472.0
2017-12-20  2688.18  2691.01  2676.11  2679.25  521377568.0
2017-12-21  2683.02  2692.64  2682.40  2684.57  511474976.0
2017-12-22  2684.22  2685.35  2678.13  2683.34          NaN
```

2. Slicing the dataset Actual_data = data['1980-01-01':'2017-11-30'].
   This was done to keep the relevant dataset and remove non-available values. Relevant data for predicting the Stock market Trend of current years. Hence, leaving out the dataset before 1980.

3. Adding a new column 'High_after_5_days'

```
new_data['High_after_5_days'] = new_data['High']
new_data['High_after_5_days'] = new_data['High_after_5_days'].shift(-5)
print new_data.tail(10)
```

```
              Open      High       Low     Close        Volume  \
Date
2017-12-11  2652.19  2660.33  2651.47  2659.99  5.104737e+08
2017-12-12  2661.73  2669.72  2659.78  2664.11  5.536614e+08
```

   Here using shift(-5) gives us the 5 Days later value of column 'High'

4. Creating feature and target lists
   I have taken features_list = ['Open','High','Low','Close','Volume']
   And target_list = 'High_after_5_days', as this value will be predicted

5. Creating manually Training, Validation and Test set.
   I tried using sklearn TimeSeriesSplit but I was failing so bad using that. I understood that Timeseries is a special dataset where to use a simple sklearn Cross validation split is cheating as it randomaly splits the dataset and assigns a portion as Test and Traing sets.
   I want to keep my test set in future in comparison to training set, which actually makes sense if I wish to predict stock market. Same logic for Validation set. Hence, I calculated where

dataset is splitting into 80-20% and divided the dataset for Train and Test. Again, I repeated the procedure for Train and Validation sets.

The slicing of dataset was easy using dates.

```
# Manually slicing 80-20 percent Training and target dataset
X_train = features['1980-01-01':'2010-04-28']
y_train = target['1980-01-01':'2010-04-28']
X_test = features['2010-04-29':]
y_test = target['2010-04-29':]
```

```
# Manually slicing 80-20 percent Training and validation dataset
X_val = X_train['2004-03-12':]
y_val = y_train['2004-03-12':]
X_train = X_train['1980-01-01':'2004-03-11']
y_train = y_train['1980-01-01':'2004-03-11']
```

6. Fitting the model

Linear regression

```
#Implementation of linear regression
from sklearn.linear_model import LinearRegression
my_linear_model = LinearRegression()
my_linear_train = my_linear_model.fit(X_train,y_train)
```

Gradient Boosting

```
#Implementing Gradient Boosting Regressor
from sklearn.ensemble import GradientBoostingRegressor
my_GB_Regressor = GradientBoostingRegressor(learning_rate=0.1, max_depth=3, random_state=None)
```

```
#training the Gradient Boost model
my_GB_train = my_GB_Regressor.fit(X_train,y_train)
```

KNN Regressor

```
#Implementing KNN Regressor
from sklearn import neighbors
my_KNN_regressor = neighbors.KNeighborsRegressor(n_neighbors = 5, weights = 'distance')
```

```
my_KNN_regressor_train = my_KNN_regressor.fit(X_train,y_train)
```

ARIMA Implementation:

1. Loading the dataset and using date_parser to read ‘Date’ column.

2. Slicing the dataset to keep only relevant years.
   TimeSeries_data = data['1980-01-01':'2017-11-01']

3. Taking the column ‘High’ as relevant data.
   TimeSeries_data = TimeSeries_data['High']

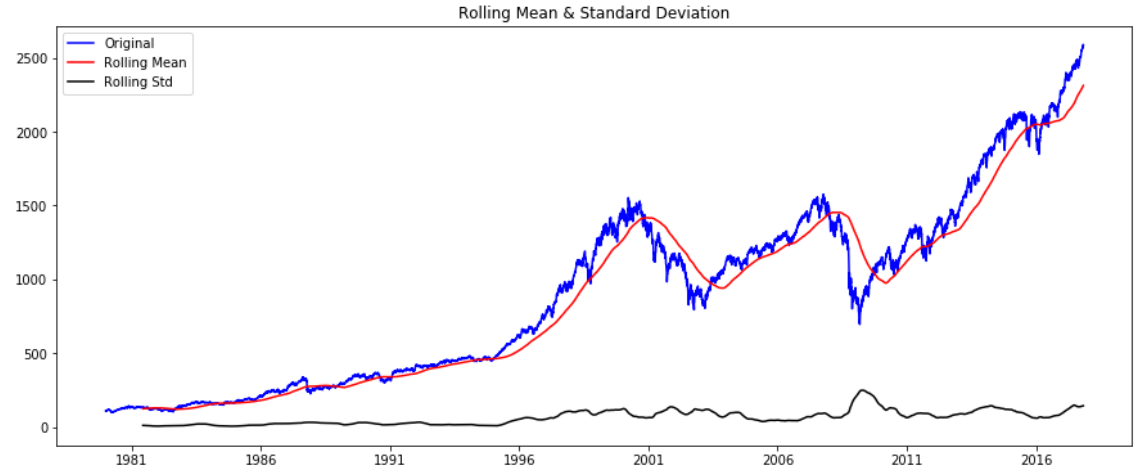
4. From here on the complexity of the approach increases. In ARIMA model we need to understand if Time Series is Stationary and if not then make it Stationary. A Time Series is not stationary if we can see Trends(varying mean over time) and Seasonality (variations at specific time-frames. Example people might have a tendency to buy cars in a particular month because of pay increment or festivals).
   So this step involves checking if Time Series is Stationary.

We can two tests to identify if TS is Stationary

a. **Plotting Rolling Statistics:** We can plot the moving average or moving variance and see if it varies with time. By moving average/variance I mean that at any instant 't', we'll take the average/variance of the last year, i.e. last 1 year.

b. **Dickey-Fuller Test:** This is one of the statistical tests for checking stationarity. Here the null hypothesis is that the TS is non-stationary. The test results comprise of a **Test Statistic** and some **Critical Values**for difference confidence levels. If the 'Test Statistic' is less than the 'Critical Value', we can reject the null hypothesis and say that the series is stationary

I have used 'adfuller' [3] stats model to compute Dickey Fuller Test. Exact Code is in Ipython file. Here is the visualization




```
Results of Dickey-Fuller Test:
Test Statistic                    1.429210
p-value                           0.997243
#Lags Used                       36.000000
Number of Observations Used    9507.000000
Critical Value (5%)              -2.861844
Critical Value (1%)              -3.431038
Critical Value (10%)             -2.566932
```

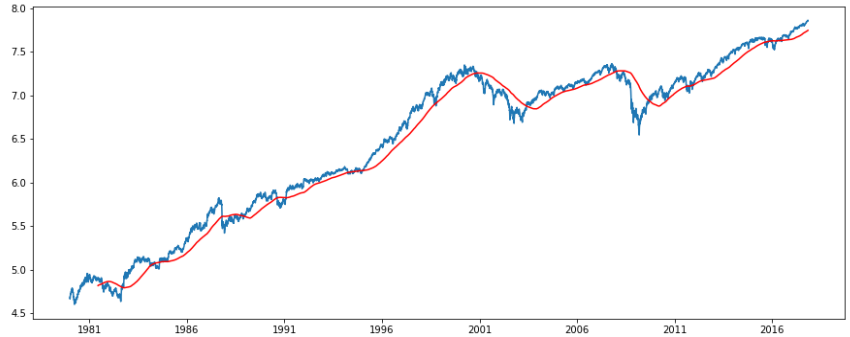
Here the 'Test Statistic' is not less than the 'Critical Value'(taking signed value), we can not reject the null hypothesis of Dickey-Fuller Test. Hence, the Timeseries is non-stationary as per the null hypothesis.

5. How to make Time Series stationary

We need to estimate and eliminate Trend and Seasonality.

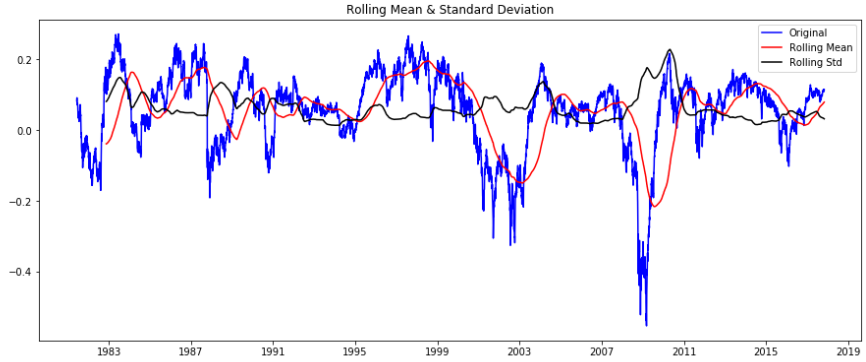
a. To counter Trend we can take log of the Time series
TimeSeries_log = np.log(TimeSeries_data)

b. For Seasonality, we can take moving average of 'k' consecutive values and then eliminate this from actual Time Series.
moving_avg = pd.rolling_mean(TimeSeries_log,365)



```
#The red line shows the rolling mean. Lets subtract this from the original series.
```

ts_log_moving_avg_diff = TimeSeries_log - moving_avg
Dropping all the NaN values and doing the stationary test again

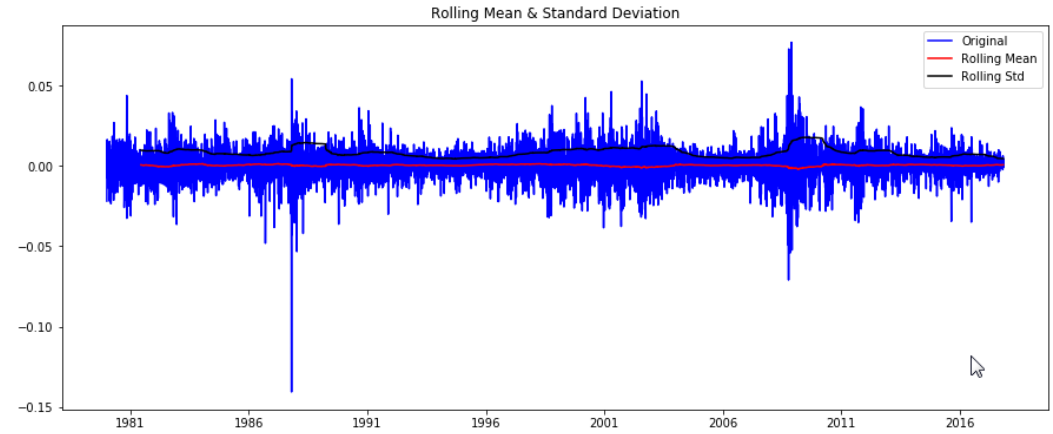



```
Results of Dickey-Fuller Test:
Test Statistic                  -4.144529
p-value                          0.000817
#Lags Used                      17.000000
Number of Observations Used    9162.000000
Critical Value (5%)              -2.861856
Critical Value (1%)              -3.431064
Critical Value (10%)             -2.566938
```

6. Here I want that more recent values are given a higher weight. A popular technique is **exponentially weighted moving average** where weights are assigned to all the previous values with a decay factor.

   expwighted_avg = pd.ewma(TimeSeries_log, halflife=365)
   Note that here the parameter ‘halflife’ is used to define the amount of exponential decay which 365 days or one year.
7. Again to eliminate Trend and Seasonality and get better result for Dicky Fuller Test, I apply Differencing
   ts_log_diff = TimeSeries_log - TimeSeries_log.shift()
   Dropping the NaN values, I get

```
ts_log_diff.dropna(inplace=True)
test_stationarity(ts_log_diff)
```



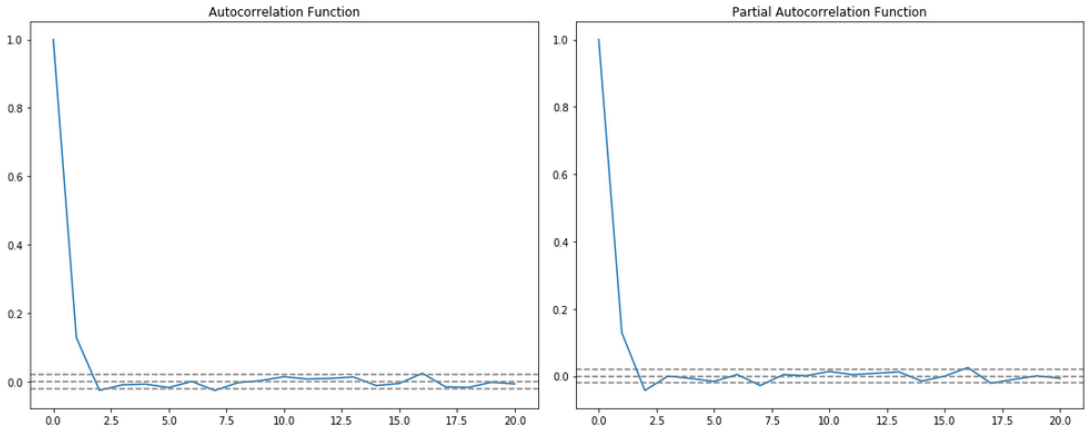

```
Results of Dickey-Fuller Test:
Test Statistic                -38.065387
p-value                         0.000000
#Lags Used                      6.000000
Number of Observations Used  9536.000000
Critical Value (5%)            -2.861843
Critical Value (1%)            -3.431036
Critical Value (10%)           -2.566931
```

We can see that the mean and std variations have small variations with time. In addition, the Dickey-Fuller test statistic is less than the 10% critical value, thus the TS is stationary with 90% confidence.

8. Last step is Forecasting the Time Series but before we need to find internal values “p”,”q” and “d” of ARIMA function given the Time Series. We use two plots to determine these numbers.
    a. **Autocorrelation Function (ACF):** It is a measure of the correlation between the TimeSeries with a lagged version of itself. The analysis of autocorrelation is a mathematical tool for finding repeating patterns, such as the presence of a periodic signal obscured by noise, or identifying the missing fundamental frequency in a signal implied by its harmonic frequencies.

    b. **Partial Autocorrelation Function (PACF):** This measures the correlation between the TimeSeries with a lagged version of itself but after eliminating the variations already explained by the intervening comparisons. Example at time lag 5, it will check the correlation but remove the effects already explained by time lags 1 to 4. It contrasts with the autocorrelation function, which does not control for other lags.

The code is there in Ipython file and for the report lets not go too much in mathematics. For diving deeper here is a reference [2]



1. p – The lag value where the PACF chart crosses the upper confidence interval for the first time. If you notice closely, in this case p=2.
2. q – The lag value where the ACF chart crosses the upper confidence interval for the first time. If you notice closely, in this case q=2

After this ARIMA model is applied using the internal values and before prediction, the Time series is scaled back to original scale, using cumulative summing and inverse log.

```
#ARIMA model
model = ARIMA(TimeSeries_log, order=(2, 1, 2))
results_ARIMA = model.fit(disp=-1)
```

```
predictions_ARIMA_log = pd.Series(TimeSeries_log.ix[0], index=TimeSeries_log.index)
predictions_ARIMA_log = predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum,fill_value=0)
```

Refinement

For Supervised Learning models, I have chosen default parameters and I am happy with the Linear Reegression result. I did not tweak other supervised models further.

For Example:

KNeighborsRegressor(n_neighbors = 5, weights = 'distance')

I just used this model with weights set to distance.

In case of ARIMA approach , in the case of eliminating Trends and Seasonality, I found use of Differencing(taking the difference with particular time lag) giving me good result for Dickey Fuller Test.

ts_log_diff = TimeSeries_log - TimeSeries_log.shift()

IV. Results

Model Evaluation and Validation

Supervised Approach:

I used R2 score for evaluation for each model. In addition, I checked Validation score for validation set.

- Linear Regression

```
: val_score_linear = my_linear_model.score(X_val,y_val)
  print "Linear regression Validation Score {}.".format(val_score_linea
```

```
Linear regression Validation Score 0.981476206506.
```

```
: my_linear_score = my_linear_model.score(X_test,y_test)
  print "Linear regression Test Score {}.".format(my_linear_score)
```

```
Linear regression Test Score 0.995873747423.
```

- Gradient Boosting Regression

```
val_score_Gradient = my_GB_Regressor.score(X_val,y_val)
print "Gradient Boosting regression Validation Score {}.".format(val_score_Gradient)
```

```
Gradient Boosting regression Validation Score 0.978639653062.
```

```
my_GB_Regressor_score = my_GB_Regressor.score(X_test,y_test)
print "Gradient Boosting regression Test Score {}.".format(my_GB_Regressor_score)
```

```
Gradient Boosting regression Test Score -0.273216917914.
```

- K nearest neighbor(n=5)

```
val_score_KNN_regressor = my_KNN_regressor.score(X_val,y_val)
print "KNN regression Validation Score {}.".format(val_score_KNN_regressor)
```

```
KNN regression Validation Score -0.704391977323.
```

```
my_KNN_regressor_score = my_KNN_regressor.score(X_test,y_test)
print "KNN regression Test Score {}.".format(my_KNN_regressor_score)
```

```
KNN regression Test Score -5.09586587554.
```

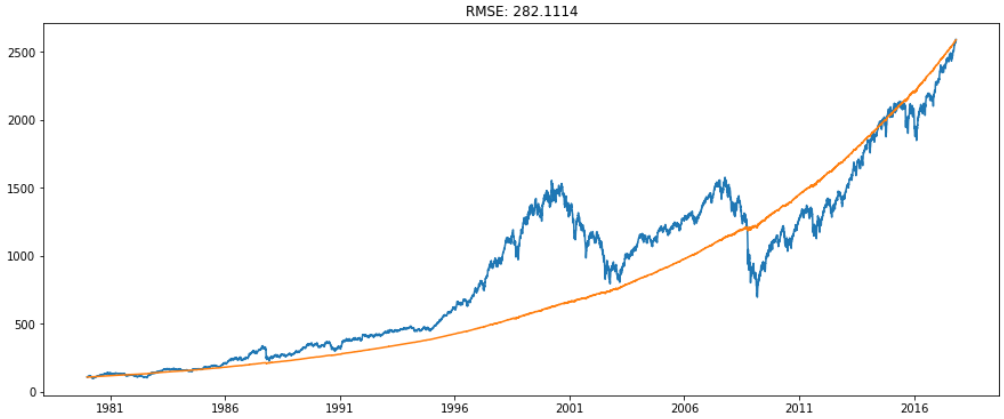
We can see here that Linear Regression test score of 0.9958 is extra ordinary. Gradient Boosting and K nearest neighbor scores are in negative meaning the models failed at fitting the dataset.

For ARIMA approach:

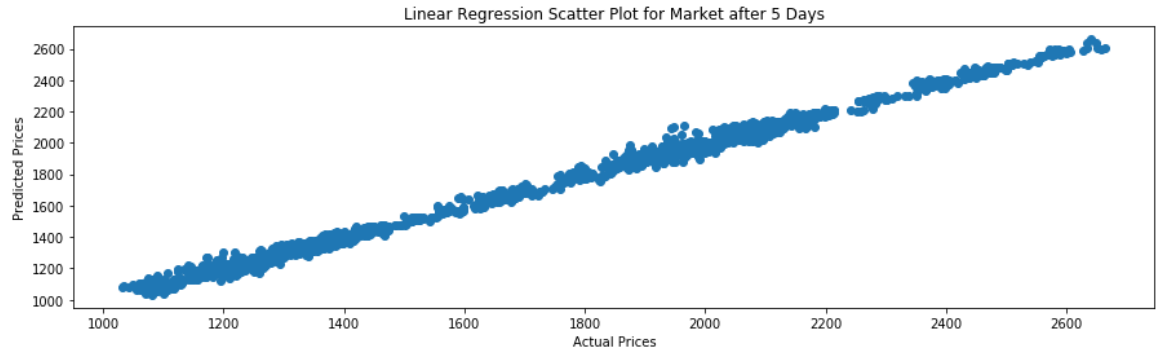
Metric used is RMSE. The RMSE score is 282.11 which is kind of satisfactory given the range of dependent variable(Value of High) from approx. 180 to 2600

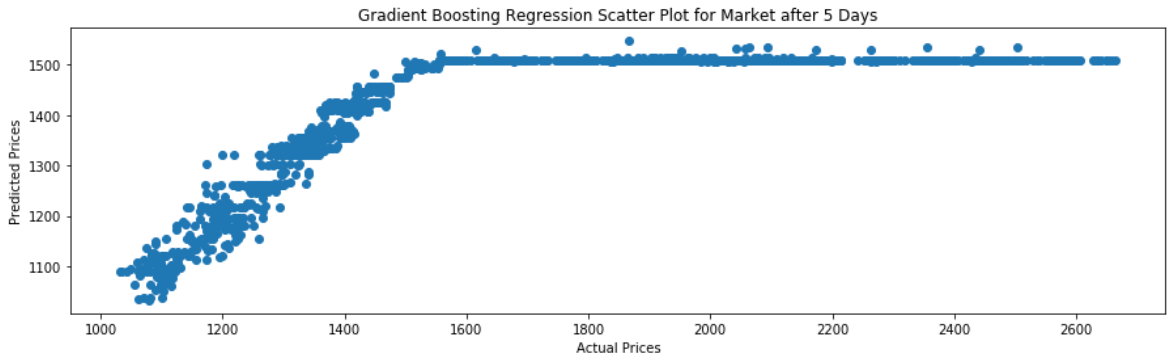
```
predictions_ARIMA = np.exp(predictions_ARIMA_log)
plt.plot(TimeSeries_data)
plt.plot(predictions_ARIMA)
plt.title('RMSE: %.4f'% np.sqrt(sum((predictions_ARIMA-TimeSeries_data)**2)/len(TimeSeries_data)))
```

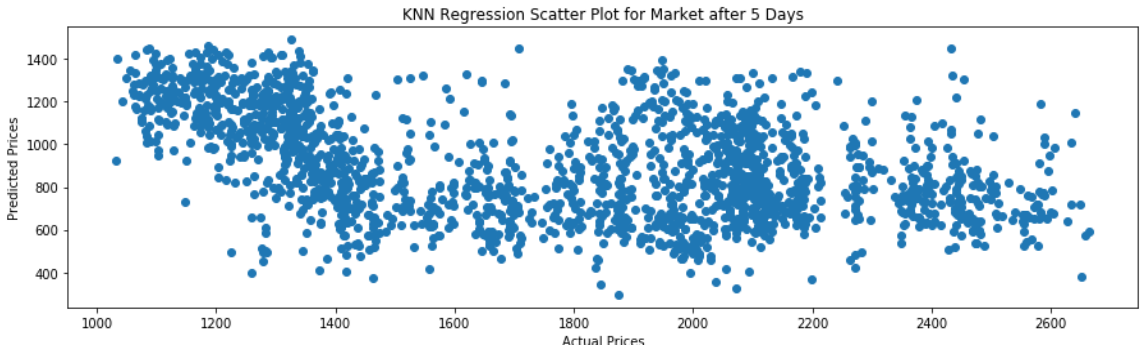
```
Text(0.5,1,u'RMSE: 282.1114')
```



Justification







From above graphs, we can see that only Linear Regression has the best prediction. Actual and Predicted prices for Linear regression follow equation of line y = mx +c.

- Y –axis Predicted value of High after 5 days
- X- Axis Actual value of High after 5 days

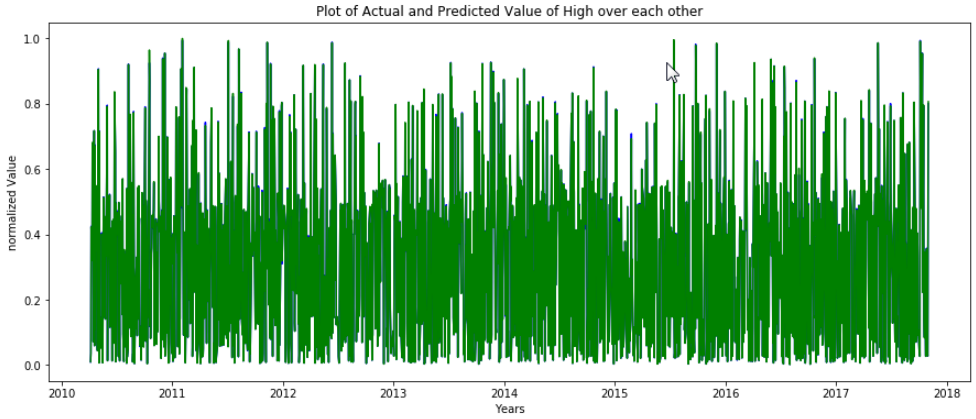
Justification: other models apart from Linear Regression were bad at learning due to Trends and seasonality of Time Series. On the other hand, ARIMA approach giving RMSE of 282 is also not a very good score but we can compare here even though.

V. Conclusion

Free-Form Visualization

I am just going to explain here my first naïve approach, which is till now not present in report and then compare, why I went with other approaches.

To begin with, I tried to tackle this problem without considering the dataset as Time Series and simply using sk-learn cross validation techniques. I got good score for my supervised models.



- X-axis – Years
- Y-axis – normalized value of feature High
- Graph is of KNN predicted values(blue) and actual values(green) of course normalized between 0 and 1
- The actual values and predicted values cover each other perfectly.
- There is only green color in the graph

I realized later that this approach is not that relevant, as I need to predict something from future. This requires splitting of test data and training data such that Test data is in future.

In above sections of report one can see how my other approaches turned out in comparison. Supervised Learning with prediction of “High after 5 days” was good only with Linear Regression and ARIMA approach residue prediction is just ok not that great.

Reflection

Reflecting upon the project, I feel that there is still a lot of strategies that needs to be discovered to handle Time Series data. ARIMA approach of making Time Series stationary and then Forecasting is really good in theory but it is kind of hard to understand each statistical concept. On the other hand, classic approach of supervised models are easy to implement but they do not do justice to Trends and Seasonality (Curves) present in Time Series data. I think I need to search and implement more strategies but given the time constraint, I need to finish this project.

Improvement

I did read about using LSTM network with Tensor flow to predict Time Series. I wish to devote some time later applying the strategy and improving. For now I am running out of Time and money.

Refrences:

1. https://stooq.com/q/d/l/?s=^spx&i=d
2. https://onlinecourses.science.psu.edu/stat510/node/60
3. http://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.adfuller.html