Machine Learning Engineer Nanodegree

Capstone Project

Andrae Delisser

September 28, 2017

Project Overview

A vast number of studies have been done on how to predict the future prices of stock to provide for the optimal decision-making in trading for investors. One of the first documented method of predicting future prices was in commodities namely rice; that method is called the candlestick charts. The   Candlestick charts are thought to have been developed in the 18th century by Munehisa Homma, a Japanese rice trader of financial instruments. [1] This would be regarded as technical analysis as it dealt only with the price movements and not necessarily the underlying causes of the moves. The contra to this method is fundamental analysis; a method of evaluating a security in an attempt to measure its intrinsic value, by examining related economic, financial and other qualitative and quantitative factors.[2]

In modern times with the advent of computers and the speed and power of processing complex calculations that it provides has increased the probability of a system that could be efficient in predicting stock price movements at least to some acceptable level of accuracy.   Recently some systems using machine learning methods such as artificial neural network achieved better performance than those using only conventional indicators. For this project most the data require classification for that ensemble method of applying algorithms can be quite useful.
.
Machine learning has been used to predict possible future stock prices and also as stock screeners. Having work in Financial Advisory for number of years I have had the desire to analyze a large set of stocks and be able to screen for potential buy and sell opportunities. In this Capstone project I will be Analyzing the Russell 3000 screening for potential stock buys and then using machine learning to predict the price of the Index itself. I will be zeroing on General Electric a large and diversified company that is a good representative of the companies that make up the Russell 3000.

Problem Statement

For this project, the task is to predict the Adjusted Close Price of a stock, I will be focusing on General Electric (GE) in the proposal for this project I suggested the Russell 3000 index however there wa not much data on its volume. I will be using five values illustrating movements in the price over one unit of time. The inputs would be trading data for the stock over 5 day period, 1 month and 1 year time period The key trading indicators are :

- Open: The starting price for a given trading day.
- Close:The final price on that day.
- High: The highest prices at which the stock traded on that day.
- Low: The lowest prices at which the stock traded on that day.
- Volume: The total number of shares traded before the market is closed on that day.

From these data I will generate new features associated with the price trend by computing ratios between each pair of average price in three different time frames. Example, the ratio between the average price over the past week and over that past year. The volume is another important factor that investors analyze. Similarly, we can generate new volume-based features by computing the average volumes in several different time frames and ratios between each pair of averaged values.

Stock volatility will be taken into consideration as well and as such we will use a standard deviation indicator; in this case we will use standard deviation of close prices in a particular time frame as well as standard deviation of volumes traded. The moving average of returns will also be an indicator I will employ.

Metrics

In this section, you will need to clearly define the metrics or calculations you will use to measure performance of a model or result in your project. These calculations and metrics should be justified based on the characteristics of the problem and problem domain. Questions to ask yourself when writing this section:
- _Are the metrics you've chosen to measure the performance of your models clearly discussed and defined?_
- _Have you provided reasonable justification for the metrics chosen based on the problem and solution?_

For this project the MSE/RMSE and R^2 of the prediction versus the actual result will be a good representation of how good the models are in being accurate. The MSE the smaller the value the better the regression model. The R^2 indicates the goodness of the fit of the regression model. It ranges from 0 to 1, meaning from no fit to perfect prediction.

MSE(Mean Squared Error)

The smaller the means squared error, the closer you are to finding the line of best fit. Depending on your data, it may be impossible to get a very small value for the mean squared error.

$$\mathrm{MSE}(\overline{X}) = \mathrm{E}((\overline{X} - \mu)^2) = \left(\frac{\sigma}{\sqrt{n}}\right)^2$$ [3]

The RMSE is calculated mathematically as such

$$\mathbf{RMSE}_{fo} = [\sum_{i=1}^{N} (z_{f_i} - z_{o_i})^2 / N]^{1/2}$$

Where:

- Σ = summation (“add up”)
- (zfi – Zoi)Sup>2 = differences, squared
- N = sample size. [4]

And *the* R^2 is calculated mathematically as such:
Step1. Find the correlation(r)

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[n\Sigma x^2 - (\Sigma x)^2][n\Sigma y^2 - (\Sigma y)^2]}}$$

Step 2: *Square the correlation coefficient.*[5]

Data Exploration

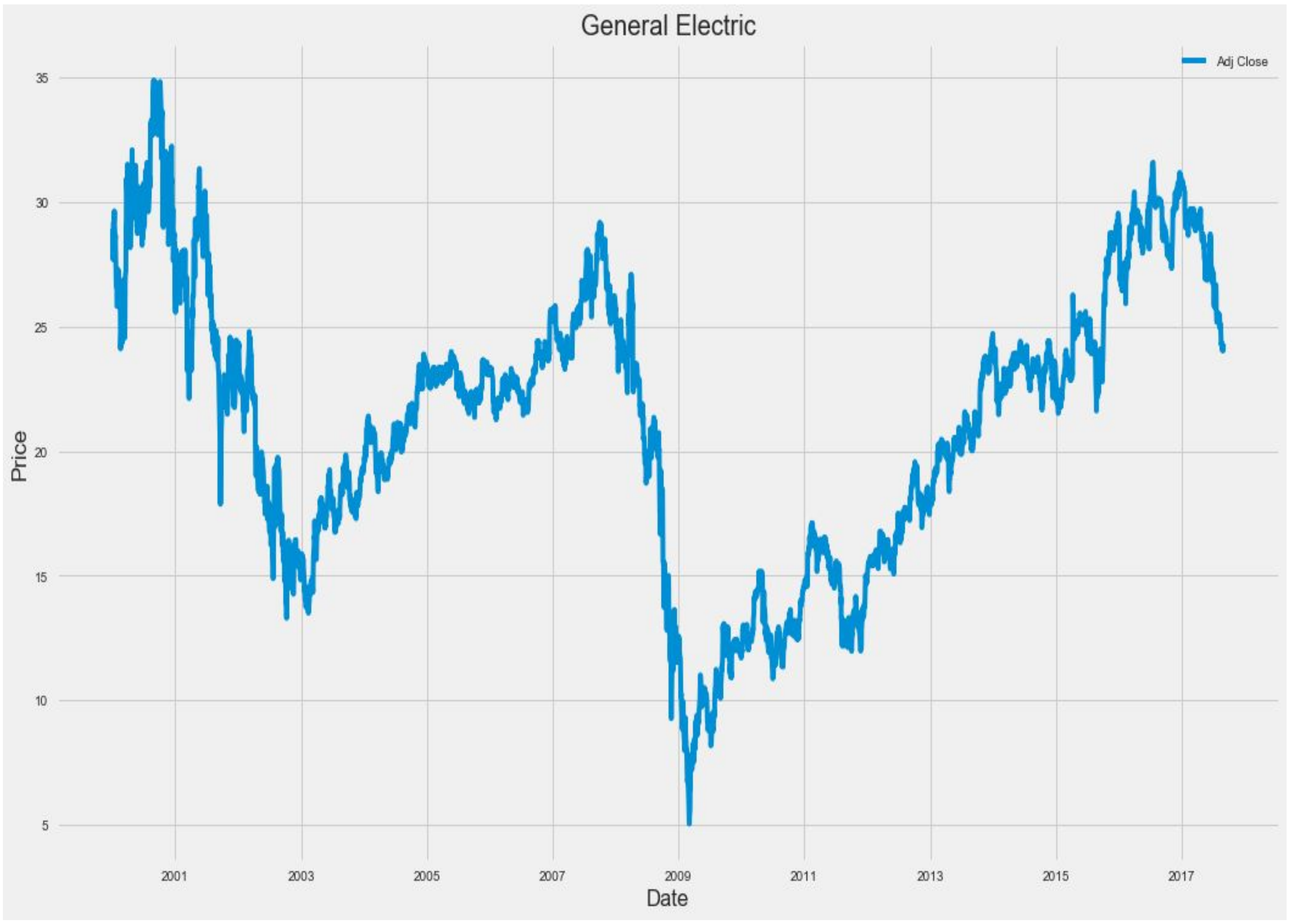
In my proposal I had intended to use the data for the Russell 3000 index as a security, however I was not able to receive the volume data so I opted to analyze the data for GE (General Electric). Any stock (granted it was trading from the initial date the data for this project was chosen) could be inputted and an analysis and prediction would be equally possible.

With the aid of the web.DataReader function and the datetime module I was able to get the data from yahoo that I used to complete this project. Below is the code I used to retrieve the data and a sample of the raw data as accessed:

```
start = dt.datetime(2000,1,1)   #The start date for the data
end = dt.datetime(2017,8,31)      # The end date for the purpose of this analysis
df= web.DataReader('GE', 'yahoo', start, end)# Getting the data for a stock in this case General
Electric from yahoo! Finance
print (df.head())# a representation of the first 5 set of data
print (df.tail())# a representation of the last 5 set of the data
```

```
×
                 Open       High        Low      Close  Adj Close    Volume
Date
2000-01-03  51.000000  51.229168  49.729168  50.000000  28.911114  22069800
2000-01-04  49.083332  49.333332  48.000000  48.000000  27.754667  22121400
2000-01-05  47.916668  49.000000  47.520832  47.916668  27.706484  27292800
2000-01-06  47.708332  48.979168  47.541668  48.557266  28.076889  19873200
2000-01-07  49.333332  50.625000  49.000000  50.437500  29.164083  20141400
                 Open       High        Low      Close  Adj Close    Volume
Date
2017-08-25  24.389999  24.600000  24.350000  24.490000  24.247725  22867800
2017-08-28  24.530001  24.670000  24.350000  24.469999  24.227922  23937600
2017-08-29  24.330000  24.459999  24.280001  24.440001  24.198219  23910100
2017-08-30  24.490000  24.490000  24.150000  24.280001  24.039803  33876000
2017-08-31  24.410000  24.700001  24.280001  24.549999  24.307131  55284300
```

Below is a visualization depicting the Adjusted Close for GE for the period 03-01-2000-31-08-2017; this period represent the dataset used for the project.



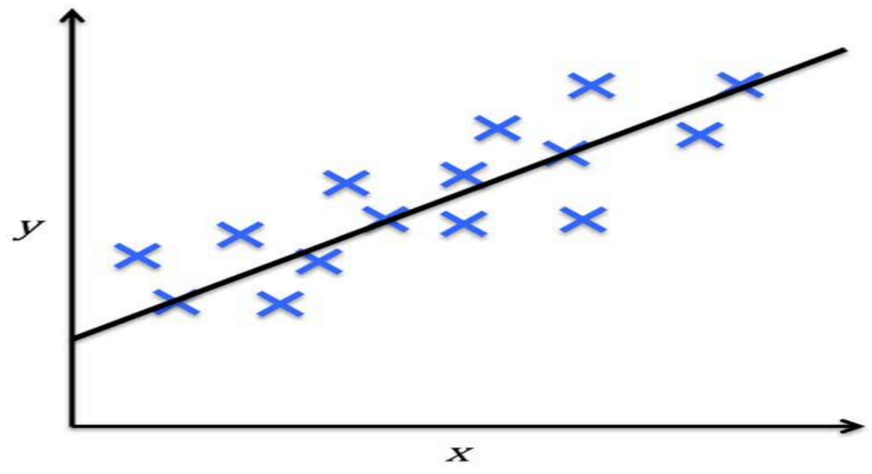
Algorithms and Techniques

I used three algorithms to create the model to make the price prediction from the existing data retrieved, namely: a Linear Regression(Stochastic Gradient Descent(SGD)), Random Forest and Support Vector Machine. I selected these algorithms because we are trying to infer what is the relation between the movement of a stock and the variables that may affect its movements; in essence how Y(the stock price) is affected by a bunch of x(variables such as momentum, CPI et al).

Linear Regression(Stochastic Gradient Descent)

In regression analysis, we are given a number of predictor (explanatory) variables and a continuous response variable (outcome), and we try to find a relationship between those variables that allows us to predict an outcome.

The following figure illustrates the concept of linear regression. Given a predictor variable x and a response variable y, we fit a straight line to this data that minimizes the distance—most commonly the average squared distance—between the sample points and the fitted line. We can now use the intercept and slope learned from this data to predict the outcome variable of new data:



[6]

For the Linear Regression I chose to use the Stochastic Gradient Descent mainly as it is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions. Below are the pros and cons :

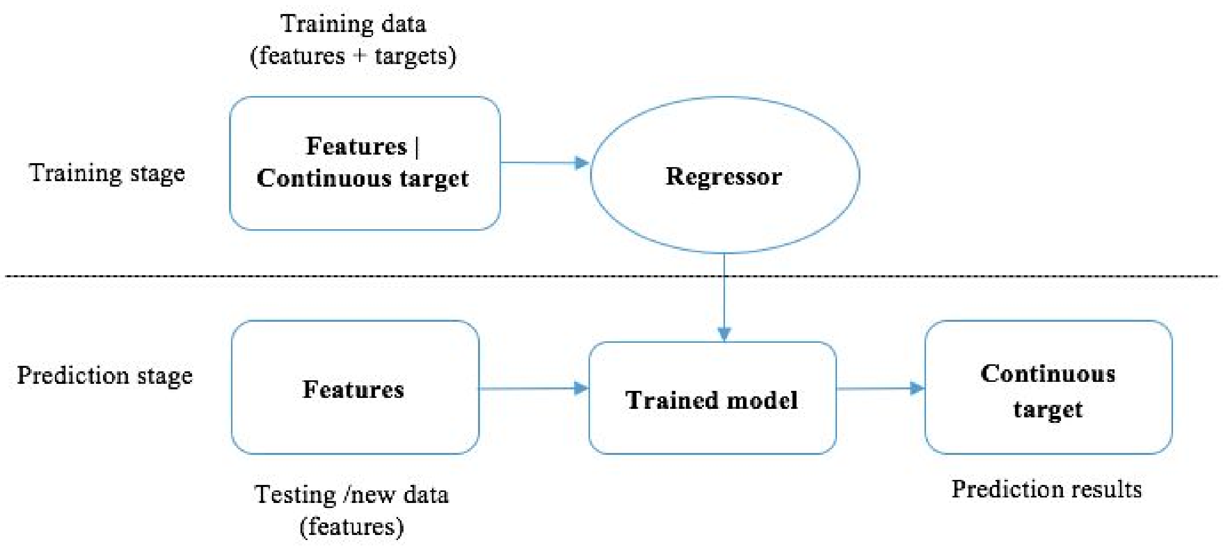
The advantages of Stochastic Gradient Descent are:

- Efficiency.
- Ease of implementation (lots of opportunities for code tuning).

The disadvantages of Stochastic Gradient Descent include:

- SGD requires a number of hyperparameters such as the regularization parameter and the number of iterations.
- SGD is sensitive to feature scaling. [7]

Below is a representation of how the input data was handled by the algorithm.



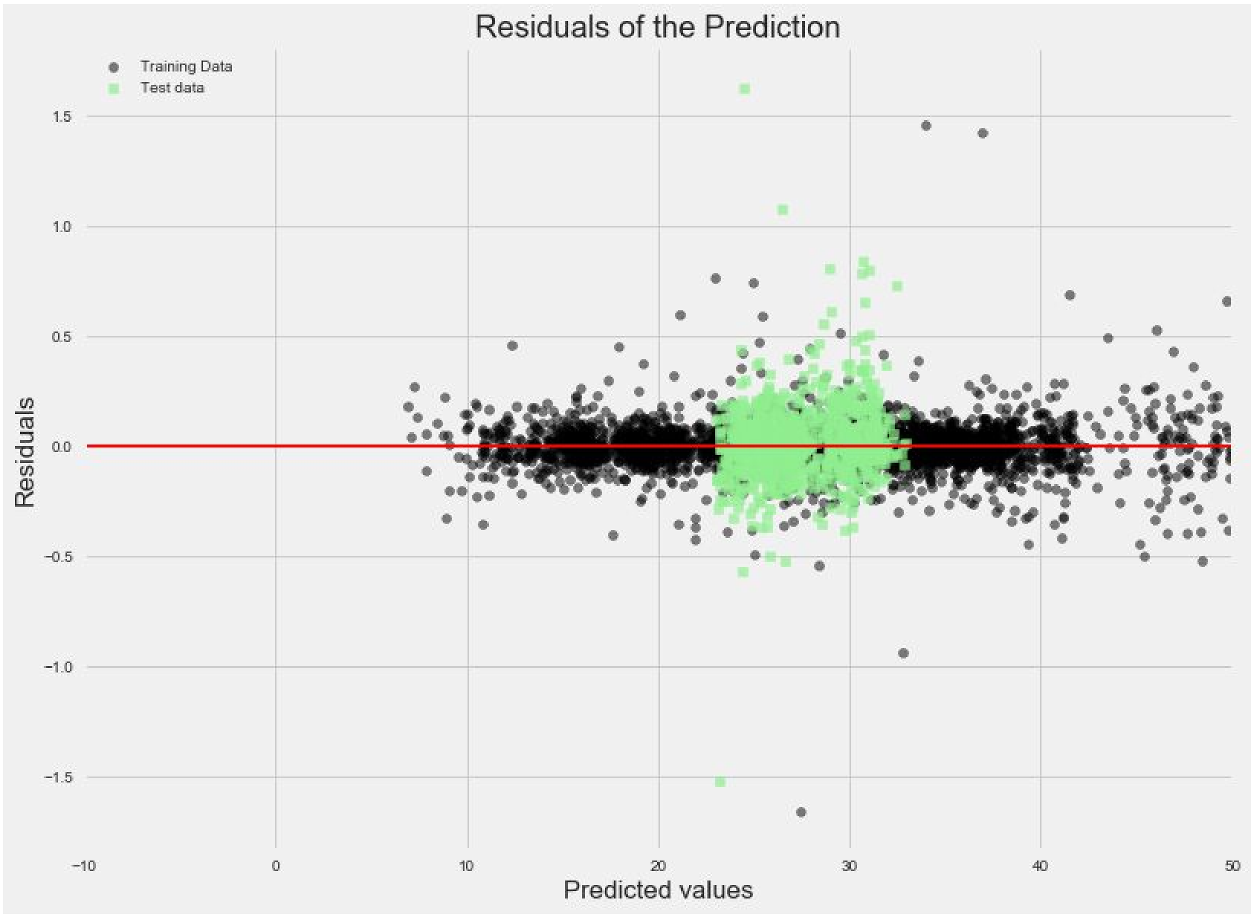
[8]

I used the following parameters to tune the algorithm

- alpha=0.0001:used to compute learning rate)
- penalty='l2':The penalty (aka regularization term) to be used. Defaults to 'l2' which is the standard regularizer for linear SVM model.
- n_iter=1000: The number of passes over the training data (aka epochs). Defaults to None. Deprecated, will be removed in 0.21.
- eta0=0.01:The exponent for inverse scaling learning rate [default 0.5].

The next step was to use GridSearchCV to find the optimal combination of parameters. The graph below show that most of the distribution of the residuals does not seem to be completely random around the zero center point, indicating that the model is not able to
capture all the exploratory information.

As a result I thought it necessary to graphically represent the Residuals of the prediction. What this show as represented below is that as it was already summarized by the R^2 coefficient, we can see that the model fits the testing data relatively well as indicated by the outliers in the y axis direction. I think this was due to finding the best parameters
(grid_search.fit(X_scaled_train, y_train)) then using that to crunch the data.



The Random Forest Regressor

Intuitively, a random forest can be considered as an ensemble of decision trees. The idea behind ensemble learning is to combine weak learners to build a more robust model, a strong learner, that has a better generalization error and is less susceptible to overfitting. The random forest algorithm can be summarized in four simple steps:

1.Draw a random bootstrap sample of size *n* (randomly choose n samples from the training set with replacement).
2.Grow a decision tree from the bootstrap sample. At each node:

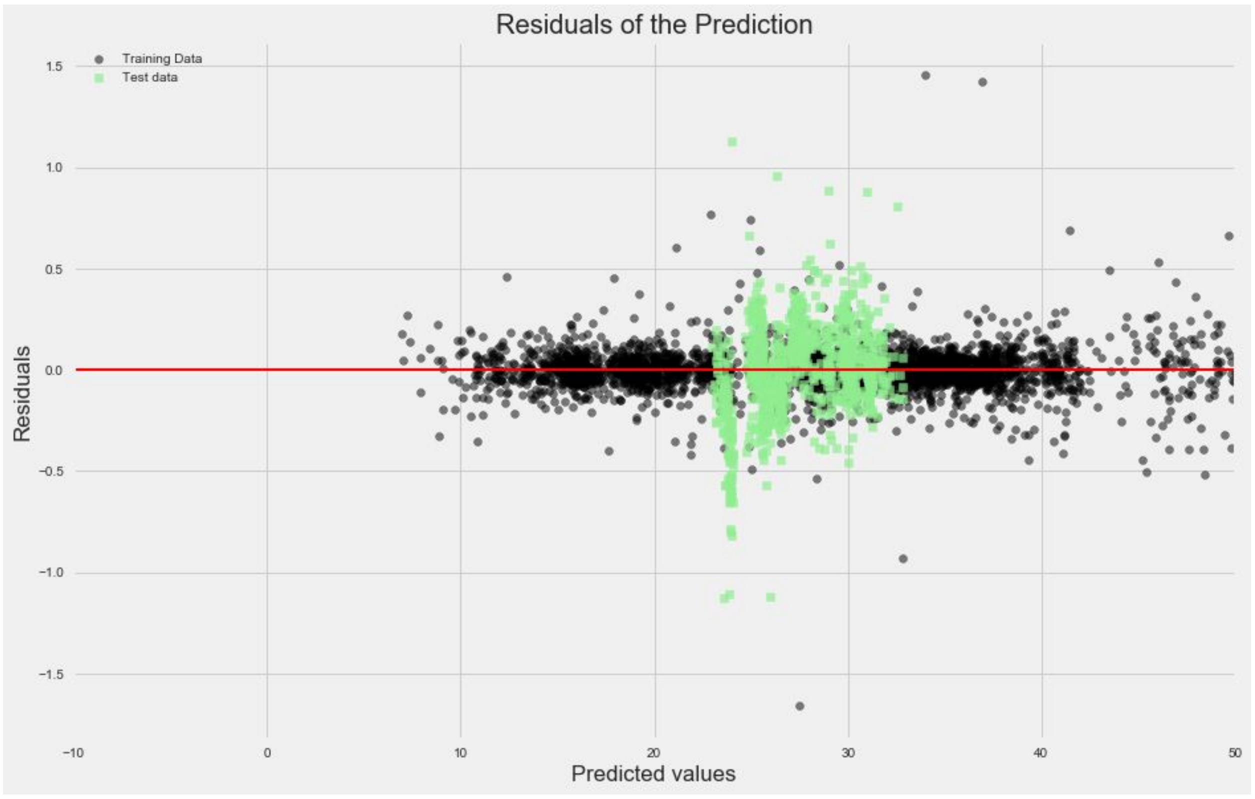
- Randomly select *d* features without replacement.
- Split the node using the feature that provides the best split according to the objective function, for instance, by maximizing the information gain.
- Repeat the steps 1 to 2 *k* times.
- Aggregate the prediction by each tree to assign the class label by majority vote. Majority voting (simply means that we select the class label that has been predicted by the majority of classifiers).

There is a slight modification in step 2 when we are training the individual decision trees: instead of evaluating all features to determine the best split at each node, we only consider a random subset of those.
Although random forests don't offer the same level of interpretability as decision trees, a big advantage of random forests is that we don't have to worry so much about choosing good hyperparameter values. We typically don't need to prune the random forest since the ensemble model is quite robust to noise from the individual decision trees. The only parameter that we really need to care about in practice is the number of trees k (step 3) that we choose for the random forest. Typically, the larger the number of trees, the better the performance of the random forest classifier at the expense of an increased computational cost. [9]

In the case of this implementation by increasing the max_depth(max depth of the tree) to 50 I was able to greatly improve the predictive power, however the time taken to run the algorithm increased drastically. The other parameters I used were:

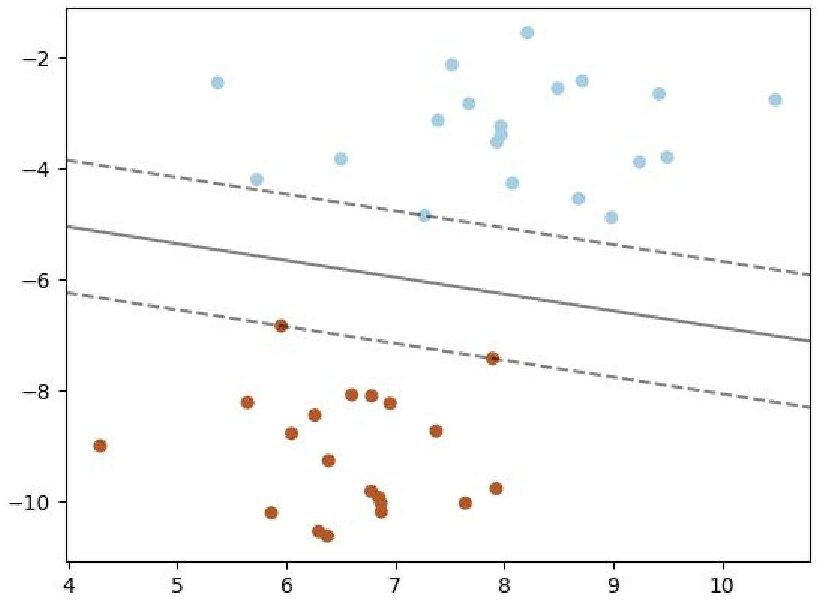
- ❖ n_estimators=1000 this is the number of trees in the forest.
- ❖ random_state=1, If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.
- ❖ n_jobs=1,The number of jobs to run in parallel for both fit and predict

The Random forest overfit the training data as the R^2 was a 1.0, it did a better job with the testing data as the R^2 (0.991) was similar to the other two algorithm scores. As a result I thought it necessary to graphically represent the Residuals of the prediction. What this show as represented below is that as it was already summarized by the R^2 coefficient, we can see that the model fits the training data better than the test data, as indicated by the outliers in the y axis direction. Also, the distribution of the residuals does not seem to be completely random around the zero center point, indicating that the model is not able to capture all the exploratory information.



The Support Vector Machine

I used the support vector regressor as the third algorithm to create a model for predicting the stock's price. A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

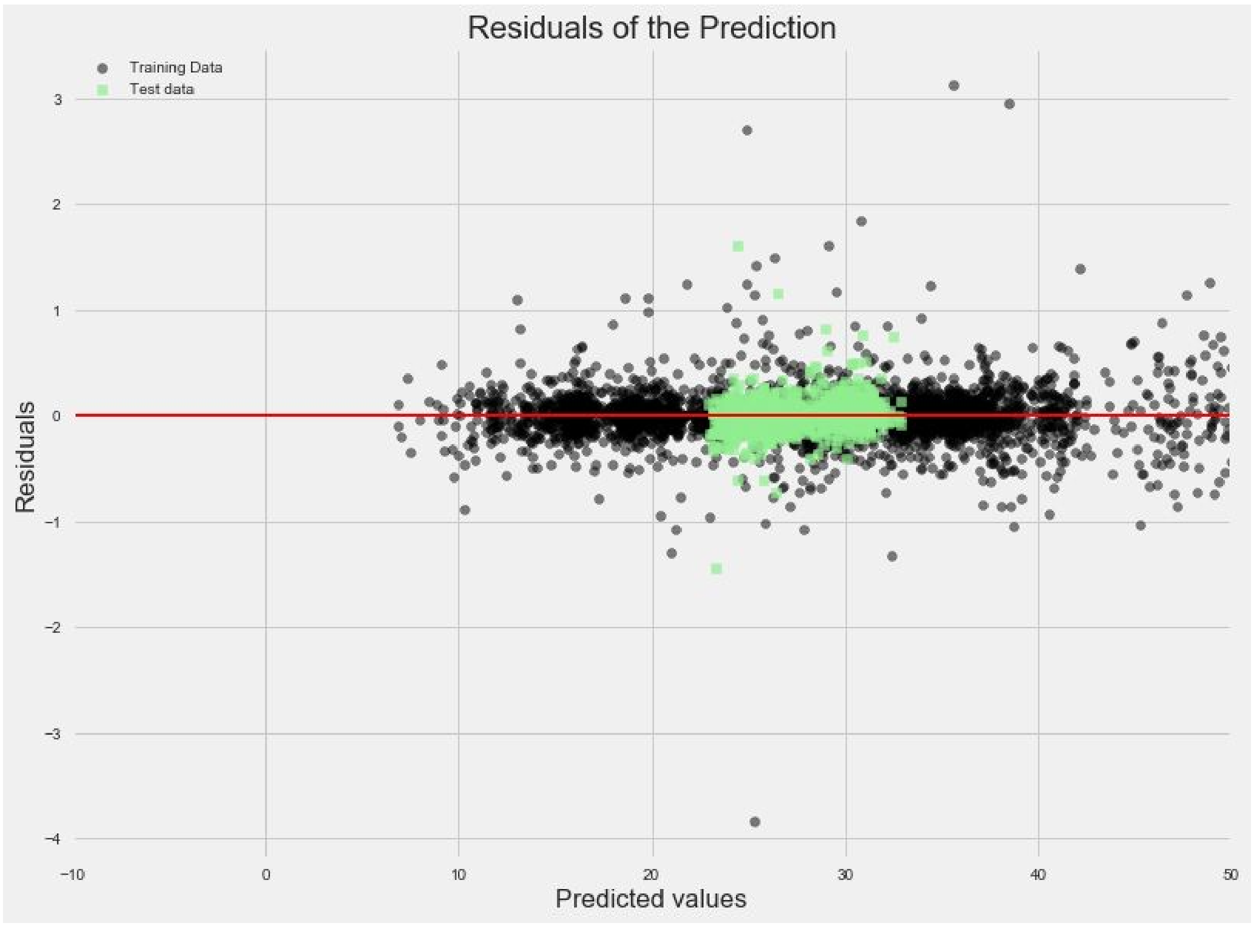


[10]

The parameters I chose and the tuning I implementer for the Support Vector Regressor were :

- (kernel='linear': Specifies the kernel type to be used in the algorithm.,
- gamma=100.0: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. If gamma is 'auto' then 1/n_features will be used instead.
- C=1.0:Penalty parameter C of the error term.

Below is a representation of the residuals of the prediction the numbers of outliers for the testing data was minimal, the algorithm seemed to have done a good job fitting the testing data.



Benchmark

As a benchmark for this project I will be using a Naive Forecast namely the autoregressive integrated moving average(ARIMA)

Naïve forecasts are the most cost-effective forecasting model, and provide a benchmark against which more sophisticated models can be compared. This forecasting method is only suitable for time series data. Using the naïve approach, forecasts are produced that are equal to the last observed value. This method works quite well for economic and financial time series, which often have patterns that are difficult to reliably and accurately predict. If the time series is believed to have seasonality, seasonal naïve approach may be more appropriate where the forecasts are equal to the value from last season. The naïve method may also use a drift, which will take the last observation plus the average change from the first observation to the last observation. [11]

ARIMA

This acronym is descriptive, capturing the key aspects of the model itself. Briefly, they are:

- **AR**: *Autoregression*. A model that uses the dependent relationship between an observation and some number of lagged observations.
- **I**: *Integrated*. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
- **MA**: *Moving Average*. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

The parameters of the ARIMA model are defined as follows:

- **p**: The number of lag observations included in the model, also called the lag order.
- **d**: The number of times that the raw observations are differenced, also called the degree of differencing.
- **q**: The size of the moving average window, also called the order of moving average. [12]

I will be using the similar dataset as I trained the three Algorithms on to fit to the Arima model.

Methodology

Data Preprocessing

In this section, all of your preprocessing steps will need to be clearly documented, if any were necessary. From the previous section, any of the abnormalities or characteristics that you identified about the dataset will be addressed and corrected here. Questions to ask yourself when writing this section:
- _If the algorithms chosen require preprocessing steps like feature selection or feature transformations, have they been properly documented?_
- _Based on the **Data Exploration** section, if there were abnormalities or characteristics that needed to be addressed, have they been properly corrected?_
- _If no preprocessing is needed, has it been made clear why?_

Before we train the model it is noteworthy that SGD-based algorithms are sensitive to data with features at largely different scales, for example we may have a 'open' price of $50.00 and a 'moving_avg_365' of 0.002. Hence we need to standardize the features into the same or comparable scale. Here, we will use a feature scaling method called standardization, which gives our data the property of a standard normal distribution. The mean of each feature is centered at value 0 and the feature column has a standard deviation of 1. For example, to standardize the j th feature, we simply need to subtract the sample mean μ from every training sample and divide it by its standard deviation , Formula:

$$\frac{X - \mu}{\sigma}$$

For this project we implement feature standardization using the 'StandardScaler ' form scikit-learn:

```
from sklearn.preprocessing import StandardScaler

sc=StandardScaler() #Using standardization

sc.fit(X_train)
```

For the reason listed above Feature scaling is a crucial step in our preprocessing pipeline.

Implementation

After retrieving the data from yahoo, the next important step was feature Engineering.

Importance of Feature Engineering

The features in your data will directly influence the predictive models you use and the results you can achieve.You can say that: the better the features that you prepare and choose, the better the results you will achieve. It is true, but it also misleading.

The results you achieve are a factor of the model you choose, the data you have available and the features you prepared. Even your framing of the problem and objective measures you're using to estimate accuracy play a part. Your results are dependent on many inter-dependent properties.

You need great features that describe the structures inherent in your data.

Better features means flexibility.

You can choose "the wrong models" (less than optimal) and still get good results. Most models can pick up on good structure in data. The flexibility of good features will allow you to use less complex models that are faster to run, easier to understand and easier to maintain. This is very desirable.

Better features means simpler models.

With well engineered features, you can choose "the wrong parameters" (less than optimal) and still get good results, for much the same reasons. You do not need to work as hard to pick the right models and the most optimized parameters.

With good features, you are closer to the underlying problem and a representation of all the data you have available and could use to best characterize that underlying problem.

Better features means better results.[13]

To ensure underfitting does not occur I will be generating 31 sets of features along with the original features; Open, Close , High, Low, Volume and Adjusted close. The 31 sets of features included:

- The average price over past 5 days, months and year and the ratio between these time period prices. Price movements represent sentiment and also according to the **efficient-market hypothesis** (**EMH**) a theory in financial economics that states that an asset's prices fully reflect all available information.
- The average and ratio of the volumes between the time frames will also be a set of feature. By knowing the total volume on a day, you can understand the power of influence on a given stock. The greater the volume, the greater the influence for the price to change. This allows us to identify accumulation and distribution days on a stock chart which can be used to identify current momentum and predict future price movements.
- The volatility of stock prices are a key indicator for predicting stock prices, I will therefore be examining the volatility ratio(in this case the standard deviation) between the various time frames.
- The final feature(indicator) will be the Moving average of the returns between the three time frames. Return is the percentage of gain or loss of close price for a stock/index in a particular period. The most effort will be spent on these features as they will be needed for solving the problem.

Here is an excerpt of the implementation:

```
# Average price
    #The window sizes are rounded to 5 days, 21 days and 252 to represent the number of
trading days in a week, month and year.


    df_new['avg_price_5']= pd.Series.rolling(df['Close'],window=5,center=False).mean().shift(1)

    #rolling_mean calculates the moving average given a window {(Example
[1,2,1,,4,3,2,1,4]->[N/A,N/A,N/A,N/A,2.2,2.42.2,2.8])}

    df_new['avg_price_30']=pd.Series.rolling(df['Close'], window=21,
center=False).mean().shift(1)
    df_new['avg_price_365']=pd.Series.rolling(df['Close'], window=252,
center=False).mean().shift(1)
```

After deciding on the features and implementing them , the next step was to split the data into training and testing set. Because we are using with time sensitive data it was advisable to use the Module TimeSeriesSplit provided by scikit-learn. ‘TimeSeriesSplit’ Provides train/test indices to split time series data samples that are observed at fixed time intervals,   in train/test sets. In each split, test indices must be higher than before, and thus shuffling in cross validation is inappropriate. [14] It was implemented as follows:

```
from sklearn.model_selection import TimeSeriesSplit
X,y= data_train.iloc[:, 1:].values, data_train.iloc[:,0].values #Assigning values to the X(Training set) and y(Target values)

#print(X)

#print(y)

dataList=TimeSeriesSplit(n_splits=3)

print(dataList)

for train_index, test_index in dataList.split(X):

    print("TRAIN:", train_index, "TEST:", test_index)

    X_train, X_test = X[train_index], X[test_index]

    y_train, y_test = y[train_index], y[test_index]
```

The next step was preprocessing the data as discussed earlier above and we ready to assign the training and testing data to the different models. The detailed process is already outlined for each algorithm chosen.

Refinement

The use of data standardization, ‘TimeSeriesSplit’ (for the main reason we are dealing with time sensitive data) and the use of finding the best parameters for each algorithm used were the steps I took in obtaining the best possible solution to the problem statement.

The results of the RandomForest gradually improved as the Maxdepth was increased, I realized the optimal number without increasing too much the time cost was 50.

IV. Results

Model Evaluation and Validation

I chose to use regression and supervised machine learning technique to predict stock prices as this was clearly a problem of inferences; meaning Y(the stock price) is affected by myriad of x(variables such as volume traded, that could determine what the future price should be.)

After tuning for the best parameters below are the results:

Linear Regression (SGD)

Metric	Training Results	Testing Results
MSE	0.068	0.027
RMSE	0.260	0.166
R^2	0.99226	0.9951264

RandomForest

Varying the max_depth resulted in changes in the power of the prediction. This I find by varying the numbers from 2 up to 50.

Metric	**Training Results**	**Testing Results**
MSE	0.012	0.050

RMSE	0.112	0.224
R^2	1.000	0.991

<u>SVM</u>

Metric	Training Results	Testing Results
MSE	0.067	0.025
RMSE	0.260	0.157
R^2	0.9992302	0.9956108

<u>Benchmark-Naive Forecast(ARIMA)</u>

Metric	Testing Results
MSE	0.089
RMSE	0.299
R^2	0.990

Overall the model holds up well. I varied the number of training data and testing percentages for a change from 30% to 20% of the total number of the data dedicated to testing the change in the MSE for the three algorithmic model and the benchmark was approximately 2-3% difference. This prove the model was robust and could hold up well to scrutiny. I would say once the tuning parameters are at their optimal levels for each algorithm the model can be trusted.

Justification

The Results from the models used were better than the benchmark, not by much but all three performed better. The lowest mean squared error (MSE) was for the Random Forest (0.012) being an ensemble of learners it was expected. The benchmark testing MSE was 0.089.

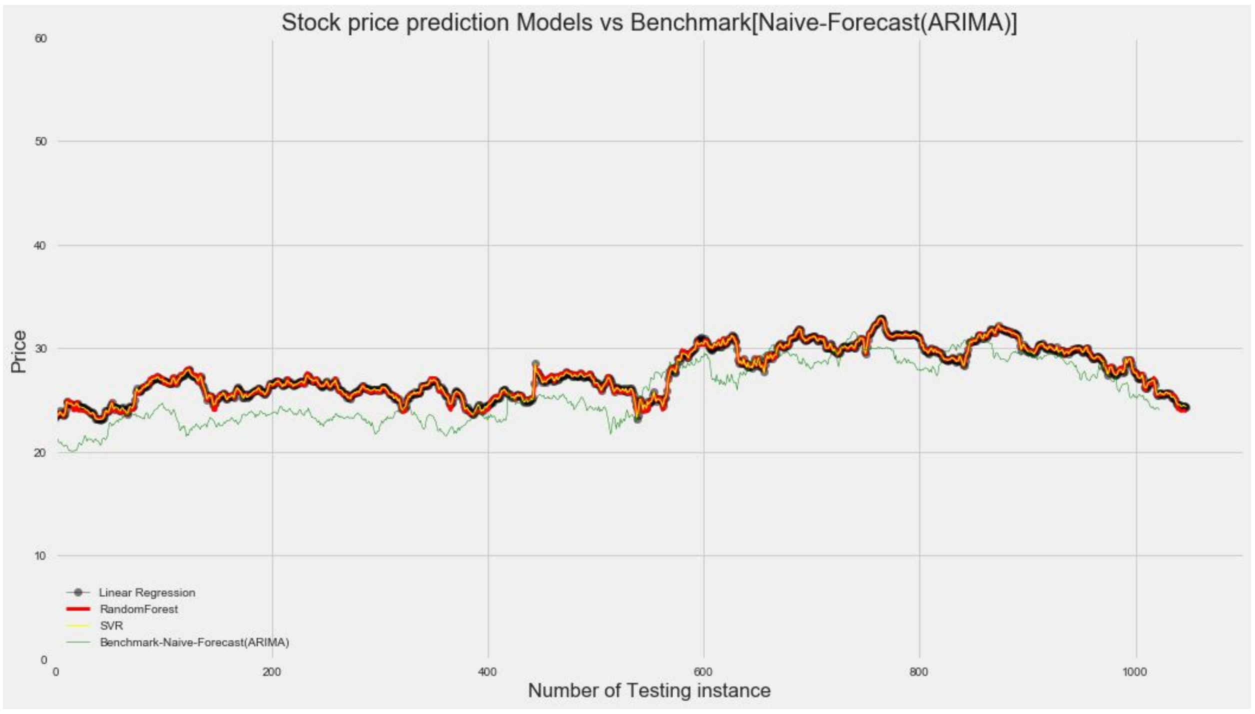
Stock prices while they appear random seem to reflect the value placed on them by investors. Some of these investors are speculators while most are seeking value. Most institutional investors are seeking valuable assets to put their capital in; as a result intricate work is done to value stocks and to estimate possible price movements over period of time. The Features I generated were mostly based on the concept of technical analysis which state that the stock's value is built in its price. Therefore by studying volumes and price trends we were able to create a model using the various algorithms I mentioned earlier to make a predictive model of the stock prices. I was generally satisfied with the results and with more time to do further analysis and even employ more relevant features this model could be very accurate in predicting the possible movement of a stock based on analyzing a wide cross section of available price data.

V. Conclusion

One of my main goal of this project was to have the various algorithmic models and the benchmark final solution being in sync. While the results in terms of the MSE and R^2 were similar i wanted to see how plotting the predictions of the testing data set would look together.

Below is a chart that shows the predictions of the dataset for the three algorithms and the benchmark juxtaposed together. The patterns were similar and that was the outcome I desired from the beginning of the implementation process.

The chart I want to emphasize is my comparative chart of merging my models with the benchmark



Reflection

The aim of this project was to use machine learning to predict to a fair degree of accuracy future stock prices. There are two main method employed by investors to analyze stock prices; fundamental and technical analysis. While I personally hold the view that using both methods can be most effective I am also more leaning to the concept that a stock price reflects all the known data that could affects it movements. This latter view is the bedrock of technical analysis and so I generated features using the price dataset for an extended period for a stock(little over 16 years).

After retrieving the data and then generating features from it, the next step was to standardize the data. This step is critical to ensure large or very small values does not skew the data and thus makes the result untenable.

I decided to use Supervisory type algorithm for this project as I would qualify the goal that we seek to accomplish as one that requires using inferences. That is how does some variable(s) (example momentum or sentiment) affects the result of a targetvalue (stock price). I also explore a number of other methods; I considered using Markov's models and Neural network algorithms in the end I decided this problem might be better of with regression type algorithms. I had difficulty implementing the Markov type algorithm using the various features(indicators), I have decided to further look into these methods on my own.

I would say given the metrics' results I employed to validate the models' results I am satisfied with the solution to the problem I seek to solve. The project took a much longer time than I would have hoped but it was great learning experience.

Improvement

This project does have lots of ways it could be improved. With more time and research more algorithms could be tried and then the most optimal ones selected. I will use the SVR model as a benchmark as it had the lowest MSE results. The naive forecast (ARIMA) results despite it not being as sophisticated as the algorithms i chose were better than I expected.

I think a model that uses both fundamental, technical and social media chatter weighted in proportions then fed to the algorithm that best processes that type of data (example Regression or Markov for technical and deep learning for social media chatter) would provide a more accurate predictive model than what I produced in here.

References:

[1] https://en.wikipedia.org/wiki/Candlestick_chart

[2] http://www.investopedia.com/terms/f/fundamentalanalysis.asp

[3] https://en.wikipedia.org/wiki/Mean_squared_error

[4]]http://www.statisticshowto.com/rmse/

[5]http://www.statisticshowto.com/what-is-a-coefficient-of-determination/

[6]Raschka, Sebastian. Python Machine Learning. Birmingham, Packt Publishing Ltd, 2015.

[7]http://scikit-learn.org/stable/modules/sgd.html

[8]https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781783553112/7/ch07lvl1sec55/what-is-regression%253f

[9]]Raschka, Sebastian. Python Machine Learning. Birmingham, Packt Publishing Ltd, 2015.

[10]http://scikit-learn.org/stable/modules/svm.html

[11]https://en.wikipedia.org/wiki/Forecasting

[12]https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/

[13]https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-   and-how-to-get-good-at-it/

[14]http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html