STOCK PRICE PREDICTION

DATA MINING CSE3019

Embedded Project

WINTER SEMESTER 2018-19

G1 SLOT

SHIKHAR SINGH - 16BCE2316

Under the guidance

Of

Dr. BalaMurugan R

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCOPE, VIT VELLORE, TAMIL NADU- 632 014

ABSTRACT

This project addresses problem of predicting direction of movement of stock and stock price index for Indian stock markets. The study compares four prediction models, Gradient Boost Regression, Support Vector Machine (SVM), random forest regression and Reinforcement learning (Bagging Model) with two approaches for input to these models. The first approach for input data involves computation of ten technical parameters using stock trading data (open, high, low & close prices) while the second approach focuses on representing these technical parameters as trend deterministic data. Accuracy of each of the prediction models for each of the two input approaches is evaluated. Evaluation is carried out on specific years when stocks prices rise or fall. Accuracy of each of the 4 machine learning algorithm is calculated and printed with the correlation between stock opening price and oil price.

1. Introduction

Predicting stock price and oil price is difficult due to uncertainties involved. There are two types of analysis which investors perform before investing in a stock. First is the fundamental analysis. In this, investors looks at intrinsic value of stocks, performance of the industry and economy, political climate etc. to decide whether to invest or not. On the other hand, technical analysis is the evaluation of stocks by means of studying statistics generated by market activity, such as past prices and volumes. Technical analysts do not attempt to measure a security' s intrinsic value but instead use stock charts to identify patterns and trends that may suggest how a stock will behave in the future. Efficient market hypothesis by Malkiel and Fama (1970) states that prices of stocks are informationally efficient which means that it is possible to predict stock prices based on the trading data. This is quite logical as many uncertain factors like political scenario of country, public image of the company will start reflecting in the stock prices. So, if the information obtained from stock prices is pre-processed efficiently and appropriate algorithms are applied then trend of stock or stock price index may be predicted. Since years, many techniques have been developed to predict stock trends. Initially classical regression methods were used to predict stock trends. Since stock data can be categorized as non-stationary time series data, non-linear machine learning techniques have also been used. Gradient Boost Regression, Random Forest Regression Algorithm, Support Vector Machine (SVM) and reinforcement learning (Bagging Model) are four machine learning algorithms which we are using for predicting closing price from stock price. Each algorithm has its own way to learn patterns.

2. Literature Survey

Paper I:

Authors and Year: *Tina Ding, Vanessa Fang, Daniel Zuo* (2014)

Title: *Stock Market Prediction based on Time Series Data and* Market Sentiment

Concept: We will train the system on three models, SVM, logistic regression, and neural networks. Afterwards, we will compare the performances of these three learning models, with and without market sentiment, and determine which is the most effective. We will use N-fold cross validation on data in the timeframe of January 2008 and April 2010 to measure the performance of the system.

Methodology: We decided to use the Python NLTK (Natural Language Toolkit) for

our sentiment analysis . The NLTK is an open source suite that provides some useful tools and libraries for text processing. The NLTK package also includes a number of trainable classifiers, including a Naive Bayes classifier with built-in training and classifying methods.

Analysis: For time series data analysis, we directly imported the prices for S&P 500 from January 2008 to April 2010 from Yahoo! Finance into Excel spreadsheet.

Relevant Finding: We observed that the classification correctness improved the most in 10-day and 30-day timeframes. It can be explained by the fact that, in time series data analysis, the historical data from 10- day or 30-day timeframe was not quite relevant to current-day market movement, and thus the more relevant sentiment results helped to improve the performance more.

Limitations: SVM appeared to be the most accurate learning model for predicting market movement. But the statistical tests showed that SVM is not significantly better than logistic regression.

Across all three learning methods, 5-days of prior data achieved the highest percentage of correctly classified instances and are statistically better than other timeframes.

Paper II:

Authors and Year: *Mark Dunne* (2008)

Title: Stock Market Prediction

Concept: We take three different approaches at the problem: Fundamental analysis, Technical Analysis, and the application of Machine Learning. We find evidence in support of the weak form of the Efficient Market Hypothesis, that the his- toric price does not contain useful information but out of sample data may be predictive. We show that Fundamental Analysis and Machine Learning could be used to guide an investor's decisions.

Methodology: The random walk hypothesis sets out the bleakest view of the predictability of the stock market. The hypothesis says that the market price of a stock is essentially random. The hypothesis implies that any attempt to predict the stock market will inevitably fail.

It is important for the purpose of this project to confront the Random Walk Hypothesis. If the market is truly random, there is little point in continuing.

Analysis: For this project, we chose the Dow Jones Industrial Average and its components as a representative set of stocks. The Dow Jones is a large index traded on the New York stock exchange. It is a price-weighted index of 30 component companies. All companies in the index are large publicly traded companies, leaders in each of their own sectors. The index covers a diverse set of sectors featuring companies such as Microsoft, Visa, Boeing, and Walt Disney.

Relevant Finding: A large body of work was presented in this report.
Two of the most widely used methods, Fundamental Analysis and Technical Analysis showed little promise in the experiments carried out. Technical Anal- ysis specifically shows little to no potential of ever producing any statistically significant result when the correct methodology is applied.
Machine learning methods were then tested on a wide range of data sources. The result of some models looked hopeful, but ultimately failed when they were put through realistic trading simulations. This highlights that the stock market is prone to differences between theory and practice.
If there is anything that this report shows, it is that profitable stock market prediction is an extremely tough problem. Whether it is possible at all ultimately remains an open question.

Paper III:
Authors and Year: Kevin Lerman, Ari Gilder, Fernando Pereira (2007)

Title: Reading the Markets: Forecasting Public Opinion of Political Candidates by News Analysis

Concept: Media reporting shapes public opinion which can in turn influence events, particularly in political elections, in which candidates both respond to and shape public perception of their campaigns. We use computational linguistics to automatically predict the impact of news on public perception of political candidates. Our system uses daily newspaper articles to predict shifts in public opinion as reflected in prediction markets. We discuss various types of features designed for this problem. The news system improves market prediction over baseline market systems.

Methodology: Since both news and internal market information are important for modeling market behavior, each one cannot be evaluated in isolation. For example, a successful news system may learn to spot important events for a candidate, but

cannot explain the price movements of a slow news day. A combi- nation of the market history system and news features is needed to model the markets. **Relevant Finding**: The combined system proved an effective way of modeling the market with both information sources.
Limitations: We have presented a system for forecasting public opinion about political candidates using news media. Our results indicate that computational systems can process media reports and learn which events impact political candidates. Additionally, the system does better when the candidate appears more frequently and for negative events. A news source analysis could reveal which outlets most influence public opinion. A feature analysis could reveal which events trigger public reactions. While these results and analyses have significance for political analysis they could extend to other genres, such as financial markets.

Paper IV:
Authors and Year: Jahidul Arafat , Mohammad Ahsan Habib and Rajib Hossain (2013)

Title: Analyzing Public Emotion and Predicting Stock Market Using Social Media

Concept: The focus of this research was to build a cloud based architecture to analyze the correlation between social media data and the financial markets. From analytical point of view this study refurbish the viability of models that treat public mode and emotion as a unitary phenomenon and suggest the needs to analyze thosein predicting the stock market status of the respective companies. With the aim to justify the correlation between social media and stock market prediction process our result reveals a proportional correspondence of pubic emotion over time with the company's market viabilities. The major significance of this research is the normalization and the conversion process that has utilized vector array list which thereby strengthen the conversion process and make the cloud storing easy. Furthermore, the experimental results demonstrate its improved performance over the factor of emotion analysis and synthesizing in the process of prediction to extract patterns in the way stock markets behave and respond to external stimuli and vice versa.

Methodology: Micro blogging is an increasingly popular form of communication on the web. It allows users to broadcast brief text updates to the public or to a limited group of contacts. In this micro blogging, stock market prediction has attracted much attention from academia as well as business. But can the stock market really be predicted? Early research on stock market prediction was based on random walk theory and the Efficient Market Hypothesis (EMH). According to the EMH stock market prices are largely driven by new information, i.e. news, rather

than present and past prices. Since news is unpredictable, stock market prices will follow a random walk pattern and cannot be predicted with more than 50% accuracy

Analysis: The Twitter Search API is a dedicated API for running searches against the real-time index of recent Tweets.

Relevant Finding: This research has a number of potential implications i.e. here public mood is measured from large-scale collection of tweets posted on twitter.com and correlated through this designed toolkit. The normalization and the conversion process has utilized vector array list which thereby strengthen the conversion process and make the cloud storing an easy.

Limitations: This work was a short term social-economic research and many extents of it could not be explored because of time limitation. Thereby this study suggests the following strategies for the further improvement: (a) Store the tweets or feed into Array. (b) Do lexical analysis on the stored feeds to retrieve the meaning of it (c) To aid to this analysis process all the English words associating the meaning to positive, negative or neutral are recommended to be gathered and stored into three different text file such as hate.txt, question.txt and positive.txt. (e) Read those files using java FileReader function or equivalent function and bring the associating words into Array. (f) Compare those words with the each words in the comment field. (g) Finally, categorize the feeds into associating field of emotion.

3. Methodology

After merging the entire data set we are applying the following 4 algorithms on the dataset to finally predict the accuracy of each model.

3.1 Random Forest Algorithm

Random Forest is a supervised learning algorithm. Like you can already see from it's name, it creates a forest and makes it somehow random. The „forest" it builds, is an ensemble of Decision Trees, most of the time trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Why Random Forest Algorithm?

- The same random forest algorithm or the random forest classifier can use for both classification and the regression task.
- Random forest classifier will handle the missing values.
- When we have more trees in the forest, random forest classifier won't overfit the model.
- Can model the random forest classifier for categorical values also.

Pseudo Code For The Algorithm

1. Randomly select "k" features from total "m" features.
    1. Where k << m
2. Among the "k" features, calculate the node "d" using the best split point.
3. Split the node into daughter nodes using the best split.
4. Repeat 1 to 3 steps until "l" number of nodes has been reached.

Build forest by repeating steps 1 to 4 for "n" number times to create "n" number of tree

3.2 Support Vector Machine

In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a nonprobabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

When data are not labeled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The support vector clustering algorithm created by Hava Siegelmann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications.

3.3 Gradient Boosting Algorithm:

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

The idea of gradient boosting originated in the observation by Leo Breiman that boosting can be interpreted as an optimization algorithm on a suitable cost function. Explicit regression gradient boosting algorithms were subsequently developed by Jerome H. Friedman simultaneously with the more general functional gradient boosting perspective of Llew Mason, Jonathan Baxter, Peter Bartlett and Marcus Frean. The latter two papers introduced the view of boosting algorithms as iterative *functional gradient descent* algorithms. That is, algorithms that optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction. This functional gradient view of boosting has led to the development of boosting algorithms in many areas of machine learning and statistics beyond regression and classification.

Gradient boosting involves three elements:

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

1. Loss Function

The loss function used depends on the type of problem being solved.

It must be differentiable, but many standard loss functions are supported and you can define your own.

For example, regression may use a squared error and classification may use logarithmic loss.

A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function that may want to be used, instead, it is a generic enough framework that any differentiable loss function can be used.

2. Weak Learner

Decision trees are used as the weak learner in gradient boosting.

Specifically regression trees are used that output real values for splits and whose output can be added together, allowing subsequent models outputs to be added and "correct" the residuals in the predictions.

Trees are constructed in a greedy manner, choosing the best split points based on purity scores like Gini or to minimize the loss.

Initially, such as in the case of AdaBoost, very short decision trees were used that only had a single split, called a decision stump. Larger trees can be used generally with 4-to-8 levels.

It is common to constrain the weak learners in specific ways, such as a maximum number of layers, nodes, splits or leaf nodes.

This is to ensure that the learners remain weak, but can still be constructed in a greedy manner.

3. Additive Model

Trees are added one at a time, and existing trees in the model are not changed.

A gradient descent procedure is used to minimize the loss when adding trees. Traditionally, gradient descent is used to minimize a set of parameters, such as the coefficients in a regression equation or weights in a neural network. After calculating error or loss, the weights are updated to minimize that error.
Instead of parameters, we have weak learner sub-models or more specifically decision trees. After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the loss (i.e. follow the gradient). We do this by parameterizing the tree, then modify the parameters of the tree and move in the right direction by (reducing the residual loss.

3.4 Reinforcement Learning:

Reinforcement learning (RL) is an area of machine learning inspired by behaviourist psychology concerned with how software agents ought to take *actions* in an *environment* so as to maximize some notion of cumulative *reward*. The problem, due to its generality, is studied in many other disciplines,

such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics and genetic algorithms. In the operations research and control literature, reinforcement learning is called *approximate dynamic programming,* or *neuro-dynamic programming.* The problems of interest in reinforcement learning have also been studied in the theory of optimal control, which is concerned mostly with the existence and characterization of optimal solutions, and algorithms for their exact computation, and less with learning or approximation, particularly in the absence of a mathematical model of the environment. In economics and game theory, reinforcement learning may be used to explain how equilibrium may arise under bounded rationality.

In machine learning, the environment is typically formulated as a Markov decision process (MDP), as many reinforcement learning algorithms for this context utilize dynamic programming techniques. The main difference between the classical dynamic programming methods and reinforcement learning algorithms is that the latter do not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become not feasible

Bootstrap Aggregation (Bagging)

Bootstrap Aggregation (or Bagging for short), is a simple and very powerful ensemble method.

An ensemble method is a technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model.
Bootstrap Aggregation is a general procedure that can be used to reduce the variance for those algorithm that have high variance. An algorithm that has high variance are decision trees, like classification and regression trees (CART).
Decision trees are sensitive to the specific data on which they are trained. If the training data is changed (e.g. a tree is trained on a subset of the training data) the resulting decision tree can be quite different and in turn the predictions can be quite different.

Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees.

Let's assume we have a sample dataset of 1000 instances (x) and we are using the CART algorithm. Bagging of the CART algorithm would work as follows.

1. Create many (e.g. 100) random sub-samples of our dataset with replacement.
2. Train a CART model on each sample.
3. Given a new dataset, calculate the average prediction from each model.

For example, if we had 5 bagged decision trees that made the following class predictions for a in input sample: blue, blue, red, blue and red, we would take the most frequent class and predict blue.

When bagging with decision trees, we are less concerned about individual trees overfitting the training data. For this reason and for efficiency, the individual decision trees are grown deep (e.g. few training samples at each leaf-node of the tree) and the trees are not pruned. These trees will have both high

variance and low bias. These are important characterize of sub-models when combining predictions using bagging.

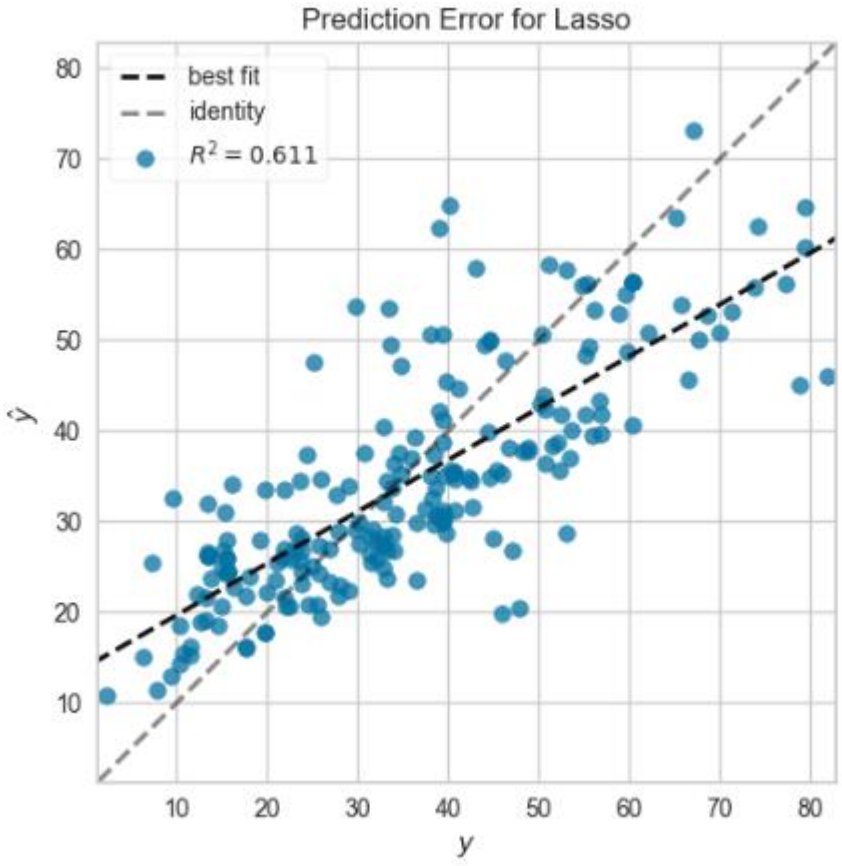
The only parameters when bagging decision trees is the number of samples and hence the number of trees to include. This can be chosen by increasing the number of trees on run after run until the accuracy begins to stop showing improvement (e.g. on a cross validation test harness). Very large numbers of models may take a long time to prepare, but will not overfit the training data.

Just like the decision trees themselves, Bagging can be used for classification and regression problems.

Visualization of Results:

Prediction Error plot:

A prediction error plot shows the actual targets from the dataset against the predicted values generated by our model. This allows us to see how much variance is in the model. Data scientists can diagnose regression models using this plot by comparing against the 45 degree line, where the prediction exactly matches the model.



Performance Measures or Metrics used for regression models:

MAE - Mean Absolute Error.

In statistics, **mean absolute error (MAE)** is a measure of difference between two continuous variables. Assume *X* and *Y* are variables of paired observations that express the same phenomenon. Examples of *Y* versus *X* include comparisons of predicted versus observed, subsequent time versus initial time, and one technique of measurement versus an alternative technique of measurement. Consider a scatter plot of *n* points, where point *i* has coordinates (x_i, y_i)... Mean Absolute Error (MAE) is the average vertical distance between each point and the identity line. MAE is also the average horizontal distance between each point and the identity line.

The Mean Absolute Error is given by:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}.$$

DataSets Used (Snippet Screenshots):

1. All Stock 5 years:

	A	B	C	D	E	F	G
1	date	open	high	low	close	volume	Name
2	2/8/2013	15.07	15.12	14.63	14.75	8407500	AAL
3	2/11/2013	14.89	15.01	14.26	14.46	8882000	AAL
4	2/12/2013	14.45	14.51	14.1	14.27	8126000	AAL
5	2/13/2013	14.3	14.94	14.25	14.66	10259500	AAL
6	2/14/2013	14.94	14.96	13.16	13.99	31879900	AAL
7	2/15/2013	13.93	14.61	13.93	14.5	15628000	AAL
8	2/19/2013	14.33	14.56	14.08	14.26	11354400	AAL
9	2/20/2013	14.17	14.26	13.15	13.33	14725200	AAL
10	2/21/2013	13.62	13.95	12.9	13.37	11922100	AAL
11	2/22/2013	13.57	13.6	13.21	13.57	6071400	AAL
12	2/25/2013	13.6	13.76	13	13.02	7186400	AAL
13	2/26/2013	13.14	13.42	12.7	13.26	9419000	AAL
14	2/27/2013	13.28	13.62	13.18	13.41	7390500	AAL
15	2/28/2013	13.49	13.63	13.39	13.43	6143600	AAL
16	3/1/2013	13.37	13.95	13.32	13.61	7376800	AAL
17	3/4/2013	13.5	14.07	13.47	13.9	8174800	AAL
18	3/5/2013	14.01	14.05	13.71	14.05	7676100	AAL
19	3/6/2013	14.52	14.68	14.25	14.57	13243200	AAL
20	3/7/2013	14.7	14.93	14.5	14.82	9125300	AAL
21	3/8/2013	14.99	15.2	14.84	14.92	10593700	AAL
22	3/11/2013	14.85	15.15	14.71	15.13	6961800	AAL
23	3/12/2013	15.14	15.6	14.95	15.5	8999100	AAL
24	3/13/2013	15.54	16.2	15.48	15.91	11380000	AAL

2. Federal funds:

	A	B	C
1	DATE	FEDFUNDS	
2	2/1/2013	0.15	
3	3/1/2013	0.14	
4	4/1/2013	0.15	
5	5/1/2013	0.11	
6	6/1/2013	0.09	
7	7/1/2013	0.09	
8	8/1/2013	0.08	
9	9/1/2013	0.08	
10	10/1/2013	0.09	
11	11/1/2013	0.08	
12	12/1/2013	0.09	
13	1/1/2014	0.07	
14	2/1/2014	0.07	
15	3/1/2014	0.08	
16	4/1/2014	0.09	
17	5/1/2014	0.09	
18	6/1/2014	0.1	
19	7/1/2014	0.09	
20	8/1/2014	0.09	

3. Oil Prices:

	A	B
1	date	value
2	2/8/2013	95.71
3	2/11/2013	97.01
4	2/12/2013	97.48
5	2/13/2013	97.03
6	2/14/2013	97.3
7	2/15/2013	95.95
8	2/19/2013	96.69
9	2/20/2013	94.92
10	2/21/2013	92.79
11	2/22/2013	93.12
12	2/25/2013	92.74
13	2/26/2013	92.63
14	2/27/2013	92.84
15	2/28/2013	92.03
16	3/1/2013	90.71
17	3/4/2013	90.13
18	3/5/2013	90.88
19	3/6/2013	90.47
20	3/7/2013	91.53
21	3/8/2013	92.01
22	3/11/2013	92.07
23	3/12/2013	92.44

4. CPI_Index:

	A	B	C	D	E	F	G
1	LOCATION	INDICATOR	SUBJECT	MEASURE	FREQUENCY	TIME	Value
2	USA	CPI	TOT	AGRWTH	M	2013-02	1.977924
3	USA	CPI	TOT	AGRWTH	M	2013-03	1.473896
4	USA	CPI	TOT	AGRWTH	M	2013-04	1.063085
5	USA	CPI	TOT	AGRWTH	M	2013-05	1.361965
6	USA	CPI	TOT	AGRWTH	M	2013-06	1.754417
7	USA	CPI	TOT	AGRWTH	M	2013-07	1.960682
8	USA	CPI	TOT	AGRWTH	M	2013-08	1.518368
9	USA	CPI	TOT	AGRWTH	M	2013-09	1.184925
10	USA	CPI	TOT	AGRWTH	M	2013-10	0.9636127
11	USA	CPI	TOT	AGRWTH	M	2013-11	1.237072
12	USA	CPI	TOT	AGRWTH	M	2013-12	1.501736
13	USA	CPI	TOT	AGRWTH	M	2014-01	1.578947
14	USA	CPI	TOT	AGRWTH	M	2014-02	1.126349
15	USA	CPI	TOT	AGRWTH	M	2014-03	1.512203
16	USA	CPI	TOT	AGRWTH	M	2014-04	1.952858
17	USA	CPI	TOT	AGRWTH	M	2014-05	2.127111
18	USA	CPI	TOT	AGRWTH	M	2014-06	2.072341
19	USA	CPI	TOT	AGRWTH	M	2014-07	1.992329
20	USA	CPI	TOT	AGRWTH	M	2014-08	1.699611
21	USA	CPI	TOT	AGRWTH	M	2014-09	1.657919
22	USA	CPI	TOT	AGRWTH	M	2014-10	1.66434
23	USA	CPI	TOT	AGRWTH	M	2014-11	1.322355

5. Exchange Rate:

	A	B
1	DATE	DEXUSEU
2	2/4/2013	1.3527
3	2/5/2013	1.3569
4	2/6/2013	1.3528
5	2/7/2013	1.3382
6	2/8/2013	1.3366
7	2/11/2013	1.3414
8	2/12/2013	1.345
9	2/13/2013	1.3448
10	2/14/2013	1.3334
11	2/15/2013	1.3362
12	2/18/2013	.
13	2/19/2013	1.3387
14	2/20/2013	1.335
15	2/21/2013	1.3204
16	2/22/2013	1.3166
17	2/25/2013	1.3172
18	2/26/2013	1.3054
19	2/27/2013	1.3104
20	2/28/2013	1.3079
21	3/1/2013	1.2988

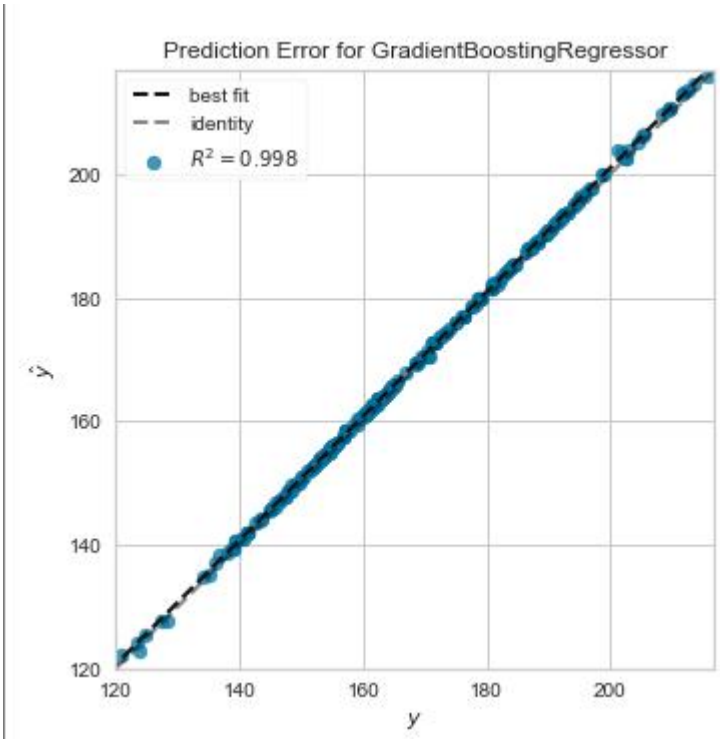
4. Results

```
In [1]: runfile('C:/Users/aarav/Desktop/dmfinal.py', wdir='C:/Users/aarav/Desktop')
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:58: DeprecationWarning:
and will be removed in 0.22. Import impute.SimpleImputer from sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
Enter the company name -
```

Input: IBM

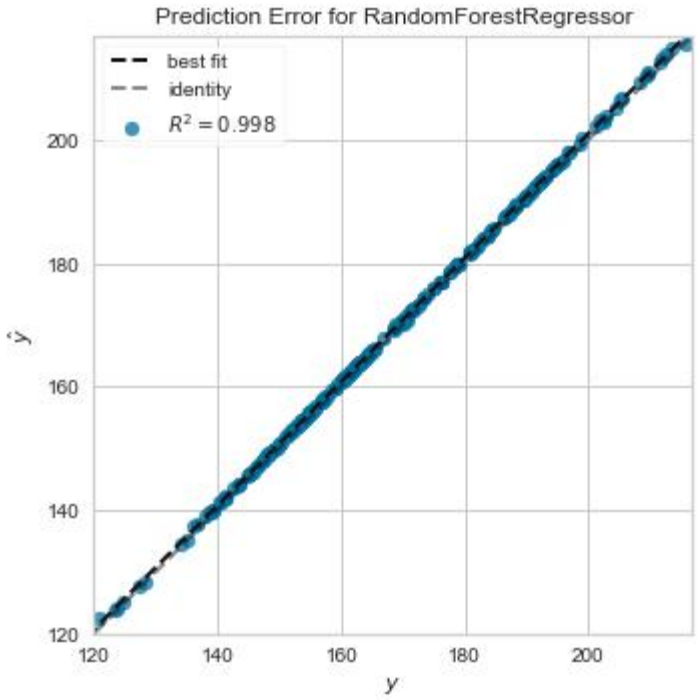
Gradient Boost Regression:

```
  y = column_or_1d(y, warn=True)
Accuracy of GradientBoostRegression is
[99.93935483]
The mean absolute error of GradientBoost
0.8582967228685184
```



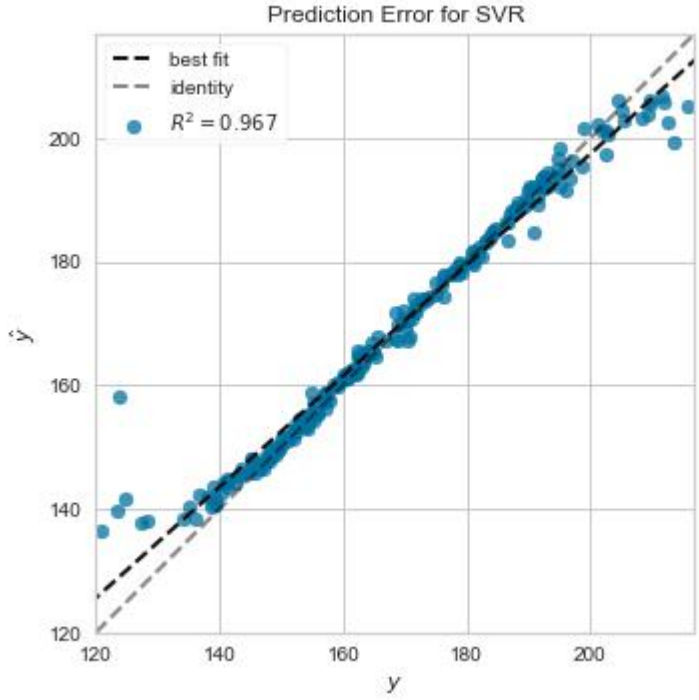
Random Forrest Regression:

```
Accuracy of random forest regression -  [99.4921895]
The mean absolute error of RandomforrestRegressor
0.8493351150793608
```



Support Vector Regression:

```
Accuracy of support vector machine -  [98.85120424]
The mean absolute error of SVMregressor
1.8085478414497285
```



Reinforcement Learning:

```
[344, 346, 316]
Accuracy of reinforcement learning -  [0.99368185]
```

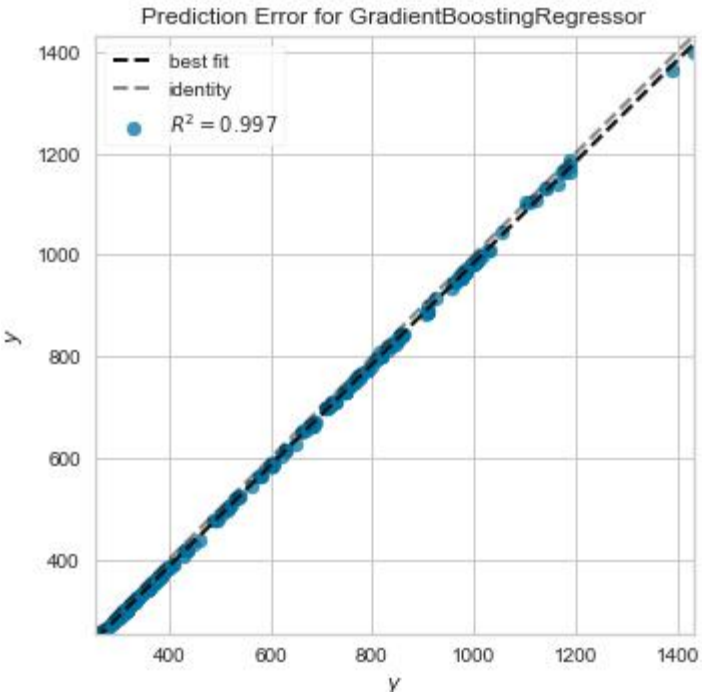
Correlation:

```
Correlation of stock opening prices and oil prices -
0.8659224540360999
```

Input: Amazon

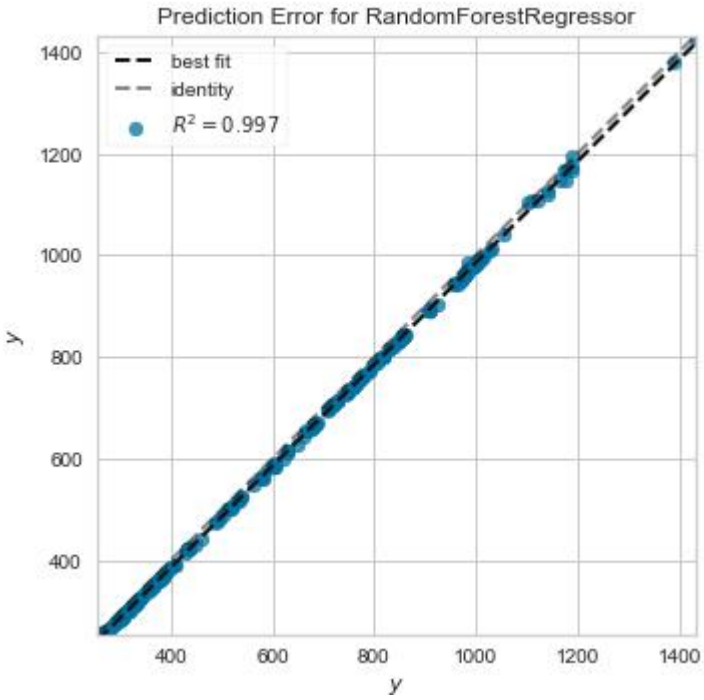
Gradient Boost Regression:

```
C:\ProgramData\Anaconda3\lib\site-packages\sklear
change the shape of y to (n_samples, ), for examp
  y = column_or_1d(y, warn=True)
Accuracy of GradientBoostRegression is
[99.80847954]
The mean absolute error of GradientBoost
14.286677446597022
```



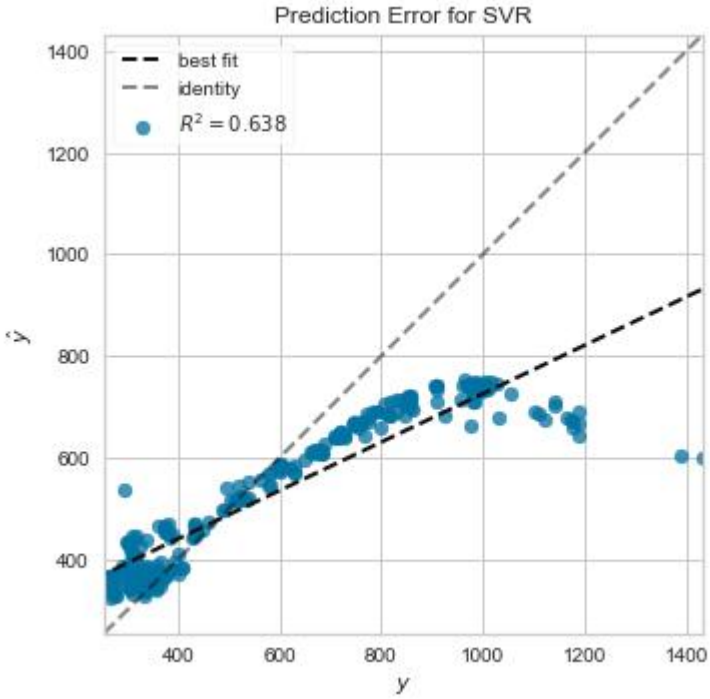
Random Forrest Regression:

```
Accuracy of random forest regression -  [97.07133819]
The mean absolute error of RandomforrestRegressor
14.245030742063527
```



Support Vector Regression:

```
Accuracy of support vector machine -  [82.3935809]
The mean absolute error of SVMregressor
113.41973239068024
```



Reinforcement Learning:

```
[449, 456, 101]
Accuracy of reinforcement learning -  [0.92192787]
```

Correlation:

```
Correlation of stock opening prices and oil prices -
-0.6183165583546534
```

5. Conclusion And Future Work

Stock market plays a very important role in fast economic growth of the developing country like India. So our country and other developing nations' growth may depend on performance of stock market. If stock market rises, then countries economic growth would be high. If stock market falls, then countries economic growth would be down.

Based on the results shown and experiments performed, it is evident that input data plays an important role in prediction along with machine learning techniques.

Most importantly, the above experiment not only helped us in predicting the accuracy of the machine learning techniques used for outcome but also gave us valuable insights about the nature of data, which can be used in future to train our classifiers in a much better way. The project can be expanded further by improvising feature list and with different classifier. Future work include the use of unsupervised preprocessor along with the supervise classifier. We could extend our work include inclusion of additional data sources. We expect a diminishing return with the inclusion of new data sources, since we expect some redundancy in the information captured from online data sources. That being said, it would be interesting to rank the value obtained from the different online data sources (for different stocks and indices). A third direction can be to consider the stochastic nature of the prediction.

References

[1] J. Bollen and H. Mao. Twitter mood as a stock market predictor. IEEE Computer, 44(10):91–94. http://arxiv.org/pdf/1010.3003.pdf.

[2] V. H. Shah. 2007. Machine Learning Techniques for Stock Prediction. Foundations of Machine Learning, New York University. http://www.vatsals.com/Essays/MachineLearningTechniques forStockPrediction.pdf.

[3] T. B. Trafalis and H. Ince. Support vector machine for regression and applications to financial forecasting. IJCNN2000, 348-353. http://www.svms.org/regression/TrIn00.pdf

[4] H. Yang, L. Chan, and I. King. Support vector machine regression for volatile stock market prediction. Proceedings of the Third International Conference on Intelligent Data Engineering and Automated Learning, 2002. http://www.cse.cuhk.edu.hk/~lwchan/papers/ideal2002.pdf

[5] M. Cohen, P. Damiani, S. Durandeu, R. Navas, H. Merlino, E. Fernandez. Sentiment analysis in microblogging: a practical implementation. Red de Universidades con Carreras en Informática (RedUNCI), P. 191-200. http://sedici.unlp.edu.ar/bitstream/handle/10915/18642/Documento_completo.pdf? sequence=1

[6] G Garner. Prediction of Closing Stock Prices. Course project for Engineering Data Analysis and Modeling at Portland State University, Fall term, 2004. http://web.cecs.pdx.edu/~edam/Reports/2004/Garner.pdf

[7] S. Bird, E Loper, and E Klein. Natural Language Processing with Python. O'Reilly Media Inc., 2009. http://ntlk.org/

[8] http://www.infochimps.com/

APPENDIX

CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


dataset_all_stocks=pd.read_csv('C:/Users/aarav/.spyder-py3/dm/all_stocks_5yr.csv')
X=dataset_all_stocks.iloc[:,:].values
j1=0
k=0
dict_X={}
l=[]

#extracting companies that have complete weekday data
for i in range(0,X.shape[0]):
    if X[k][6]==X[i][6]:
        j1=j1+1
    else:
        if (j1==1258):
            dict_X[k]=j1
        j1=0
        k=i
if (j1==1258):
    dict_X[k]=j1

#seperating company wise data
dict_dataset_comp={}
X_dataset_comp=[]
for i in list(dict_X.keys()):
    for j in range(0,dict_X[i]):
        l.append(X[j+i])
    X_dataset_comp.append(l)
    dict_dataset_comp[len(X_dataset_comp)-1]=X[i][X.shape[1]-1]
    l=[]

#extracting oil prices
oil_prices_dataset=pd.read_csv('C:/Users/aarav/.spyder-py3/dm/oil_prices.csv')
Y=oil_prices_dataset.iloc[:,:].values

date_X=list(dataset_all_stocks.iloc[0:len(X_dataset_comp[0]),0].values)
date_X=set(date_X)
date_X=list(date_X)
date_Y=list(oil_prices_dataset.iloc[:,0].values)
date_X.sort()
date_Y.sort()
date_unequal_index=[]

for i in range(0,len(date_X)):
    if (date_X[i]!=date_Y[i]):
        date_unequal_index.append(date_Y.index(date_Y[i]))
```

```
        date_Y.remove(date_Y[i])

date_Y.remove(date_Y[len(date_Y)-1])
list1_oil_price_add=[]
for i in range(0,len(date_X)):
    pos=-1
    for j in range(0,Y.shape[0]):
        if (Y[j][0]==date_Y[i]):
            pos=j
            break
    a=[]
    a.append(Y[j][1])
    list1_oil_price_add.append(a)

#extracting exchange rate
exchange_dataset=pd.read_csv('C:/Users/aarav/.spyder-py3/dm/exchange_rate.csv')
e=exchange_dataset.iloc[:,:].values
j=0
e=np.delete(e,0,axis=0)
e=np.delete(e,0,axis=0)
e=np.delete(e,0,axis=0)
e=np.delete(e,0,axis=0)
rows=e.shape[0]
for i in range(0,rows):
    if i==e.shape[0]:
        break
    f=0
    for j in range(0,len(date_X)):
        if (date_X[j]==e[i][0]):
            f=1
            break
    if f==0:
        e=np.delete(e,i,axis=0)
avg=0
x=0
list1_exchange_values=[]
for i in e:
    a=[]
    if (i[1]!='.'):
        avg=(avg*x+float(i[1]))/(x+1)
    else:
        i[1]=avg
    a.append(float(i[1]))
    list1_exchange_values.append(a)

#merging company wise oil prices and exchange rates
for i in range(0,len(X_dataset_comp)):
    X_dataset_comp[i]=np.hstack((X_dataset_comp[i],list1_oil_price_add))
    X_dataset_comp[i]=np.hstack((X_dataset_comp[i],list1_exchange_values))

#convert date to year,month,day
a_year=[]
a_month=[]
a_day=[]

for i in range(0,len(date_X)):
```

```
    date_c=date_X[i].split("-")
    for j in range(0,3):
        date_c[j]=float(date_c[j])
    a=[]
    a.append(date_c[0])
    a_year.append(a)
    a=[]
    a.append(date_c[1])
    a_month.append(a)
    a=[]
    a.append(date_c[2])
    a_day.append(a)


for i in range(0,len(X_dataset_comp)):
    X_dataset_comp[i]=np.hstack((X_dataset_comp[i],a_year))
    X_dataset_comp[i]=np.hstack((X_dataset_comp[i],a_month))
    X_dataset_comp[i]=np.hstack((X_dataset_comp[i],a_day))

#merging monthly datasets indexing
cpi_d=pd.read_csv('C:/Users/aarav/.spyder-py3/dm/CPI_Index.csv')
cpi=cpi_d.iloc[:,:].values
l_cpi_index=[]
k=0
for i in range(0,6):
    l_year=[]
    for j in range(0,12):
        if (i==5 and j==2):
            break
        if (i==0 and j==0):
            l_year.append(0)
        else:
            l_year.append(cpi[k][6])
            k=k+1
    l_cpi_index.append(l_year)

fedfund_d=pd.read_csv('C:/Users/aarav/.spyder-py3/dm/FEDFUNDS.csv')
fedfund=fedfund_d.iloc[:,:].values
l_fedfund_index=[]
k=0
for i in range(0,6):
    l_year=[]
    for j in range(0,12):
        if (i==5 and j==1):
            break
        if (i==0 and j==0):
            l_year.append(0)
        else:
            l_year.append(fedfund[k][1])
            k=k+1
    l_fedfund_index.append(l_year)
l_fedfund_index[5].append(1.42)

#merging CPI
l_cpi_values=[]
for i in range(0,len(X_dataset_comp)):
```

```
    for j in range(0,len(X_dataset_comp[i])):
        r=int(X_dataset_comp[i][j][9])-2013
        c=int(X_dataset_comp[i][j][10])-1
        a=[]
        a.append(l_cpi_index[r][c])
        l_cpi_values.append(a)
    X_dataset_comp[i]=np.hstack((X_dataset_comp[i],l_cpi_values))
    l_cpi_values=[]

#merging federal fund rates
l_fedfund_values=[]
for i in range(0,len(X_dataset_comp)):
    for j in range(0,len(X_dataset_comp[i])):
        r=int(X_dataset_comp[i][j][9])-2013
        c=int(X_dataset_comp[i][j][10])-1
        a=[]
        a.append(l_fedfund_index[r][c])
        l_fedfund_values.append(a)
    X_dataset_comp[i]=np.hstack((X_dataset_comp[i],l_fedfund_values))
    l_fedfund_values=[]

#data preprocessing
from sklearn.preprocessing import StandardScaler,Imputer
im=Imputer(strategy='mean',axis=1)
ss=StandardScaler()

dict_comp={}
Y_dataset_comp=[]
for i in range(0,len(X_dataset_comp)):
    dict_comp[X_dataset_comp[i][0][6]]=i
    X_dataset_comp[i]=np.delete(X_dataset_comp[i],6,1)
    X_dataset_comp[i]=np.delete(X_dataset_comp[i],0,1)
    Y=X_dataset_comp[i][:,1]
    Y_dataset_comp.append(Y)

print('Enter the company name - ')
comp=input()
index=dict_comp[comp]

X=X_dataset_comp[index]
Y=Y_dataset_comp[index]
X=im.fit_transform(X)
Y=Y.reshape(-1,1)
Y=im.fit_transform(Y)

#splitting data into training and test data
from sklearn.linear_model import Lasso, LassoCV, Ridge, RidgeCV
from sklearn.model_selection import train_test_split
from yellowbrick.regressor import AlphaSelection, PredictionError, ResidualsPlot
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=0)

X_train=ss.fit_transform(X_train)
X_test=ss.fit_transform(X_test)
```

```
#classifier part
from sklearn.ensemble import GradientBoostingRegressor
gbregressor=GradientBoostingRegressor(n_estimators=100,random_state=0)
modelgb = gbregressor.fit(X_train,Y_train)
y_pred_gbr=gbregressor.predict(X_test)
y_pred_train_gbr=gbregressor.predict(X_train)
y_pred_train_gbr=y_pred_train_gbr.tolist()



acc_gbr=[]
for i in range(0,len(y_pred_gbr)):
    acc_gbr.append(abs(y_pred_gbr[i]-Y_test[i])/Y_test[i])
final_s_gbr=sum(acc_gbr)/len(acc_gbr)

acc_train_gbr=[]
for i in range(0,len(y_pred_train_gbr)):
    acc_train_gbr.append(abs(y_pred_train_gbr[i]-Y_train[i])/Y_train[i])
final_s_train_gbr=sum(acc_train_gbr)/len(acc_train_gbr)
final_acc_gbr = (1 - final_s_train_gbr)*100
print("Accuracy of GradientBoostRegression is")
print(final_acc_gbr)
print("The mean absolute error of GradientBoost ")
mae_gbr = mean_absolute_error(Y_test, y_pred_gbr)
print(mae_gbr)
model = Lasso()
visualizer1 = PredictionError(modelgb)
visualizer1.fit(X_train,Y_train)  # Fit the training data to the visualizer
visualizer1.score(X_test, Y_test)  # Evaluate the model on the test data
g = visualizer1.poof()

from sklearn.ensemble import RandomForestRegressor
rfregressor=RandomForestRegressor(n_estimators=100,random_state=0)
modelrfr = rfregressor.fit(X_train,Y_train)
y_pred_rfr=rfregressor.predict(X_test)
y_pred_train_rfr=rfregressor.predict(X_train)
y_pred_train_rfr=y_pred_train_rfr.tolist()

acc_rfr=[]
for i in range(0,len(y_pred_rfr)):
    acc_rfr.append(abs(y_pred_rfr[i]-Y_test[i])/Y_test[i])
final_s_rfr=sum(acc_rfr)/len(acc_rfr)

acc_train_rfr=[]
for i in range(0,len(y_pred_train_rfr)):
    acc_train_rfr.append(abs(y_pred_train_rfr[i]-Y_train[i])/Y_train[i])
final_s_train_rfr=sum(acc_train_rfr)/len(acc_train_rfr)
final_acc_rfr = (1 - final_s_rfr)*100
print("Accuracy of random forest regression - ",final_acc_rfr)
print("The mean absolute error of RandomforrestRegressor ")
mae_rfr = mean_absolute_error(Y_test, y_pred_rfr)
print(mae_rfr)
model = Lasso()
visualizer2 = PredictionError(modelrfr)
visualizer2.fit(X_train,Y_train)  # Fit the training data to the visualizer
visualizer2.score(X_test, Y_test)  # Evaluate the model on the test data
```

```
g = visualizer2.poof()


from sklearn.svm import SVR
svmregressor=SVR(kernel='rbf')
modelsvm = svmregressor.fit(X_train,Y_train)
y_pred_svm=svmregressor.predict(X_test)
y_pred_train_svm=svmregressor.predict(X_train)
y_pred_train_svm=y_pred_train_svm.tolist()

acc_svm=[]
for i in range(0,len(y_pred_svm)):
        acc_svm.append(abs(y_pred_svm[i]-Y_test[i])/Y_test[i])
final_s_svm=sum(acc_svm)/len(acc_svm)
final_acc_svm = (1- final_s_svm)*100
print("Accuracy of support vector machine - ",final_acc_svm)
print("The mean absolute error of SVMregressor ")
mae_svr = mean_absolute_error(Y_test, y_pred_svm)
print(mae_svr)
model = Lasso()
visualizer3 = PredictionError(modelsvm)
visualizer3.fit(X_train,Y_train)  # Fit the training data to the visualizer
visualizer3.score(X_test, Y_test)  # Evaluate the model on the test data  # Evaluate the model on the test data
g = visualizer3.poof()


acc_train_svm=[]
for i in range(0,len(y_pred_train_svm)):
        acc_train_svm.append(abs(y_pred_train_svm[i]-Y_train[i])/Y_train[i])
final_s_train_svm=sum(acc_train_svm)/len(acc_train_svm)

#dataset of deviation of training data
l=[]
for i in range(0,len(acc_train_gbr)):
    l1=[]
    l1.append((-1)*acc_train_gbr[i])
    l1.append((-1)*acc_train_rfr[i])
    l1.append((-1)*acc_train_svm[i])
    l.append(l1)

#reinforcement learning
import math
N=1006
d=3
t=0
no_of_selections=[0]*d
score_arr=[0]*d
for i in range(0,N):
    max_bound=0
    ad=0
    for j in range(0,d):
        if no_of_selections[j]>0:
            a=score_arr[j]/no_of_selections[j]
            delta_j=math.sqrt(3/2*(math.log(i+1)/no_of_selections[j]))
            a=a+delta_j
        else:
```

```
            a=1e400
        if a>max_bound:
            max_bound=a
            ad=j
    no_of_selections[ad]=no_of_selections[ad]+1
    reward=l[i][ad]
    score_arr[ad]=score_arr[ad]+reward

    #  ads_selected.append(ad)
print(no_of_selections)
w1=no_of_selections[0]/sum(no_of_selections)
w2=no_of_selections[1]/sum(no_of_selections)
w3=no_of_selections[2]/sum(no_of_selections)

final_pred=w1*rfregressor.predict(X_test)+w2*svmregressor.predict(X_test)+w3*gbregressor.predict(X_test)

acc_test=[]
for i in range(0,len(final_pred)):
    acc_test.append(abs(final_pred[i]-Y_test[i])/Y_test[i])
final_s_test_pred=sum(acc_test)/len(acc_test)
final_acc_ri = (1-final_s_test_pred)
print("Accuracy of reinforcement learning - ",final_acc_ri)

Y_final=Y

print("Correlation of stock opening prices and oil prices - ")
m1=np.mean(Y_final)*1258
m2=np.mean(list1_oil_price_add)*1258
pr=0
for i in range(0,1258):
    n2=Y_final[i].tolist()[0]
    n3=list1_oil_price_add[i][0]
    pr=pr+n2*n3
num=pr*1258-m1*m2
varx=0
vary=0
for i in range(0,1258):
    varx=varx+Y_final[i].tolist()[0]*Y_final[i].tolist()[0]
    vary=vary+list1_oil_price_add[i][0]*list1_oil_price_add[i][0]
varx=varx*(1258)
vary=vary*(1258)
varx=varx-m1*m1
vary=vary-m2*m2
x_dev=math.sqrt(varx)
y_dev=math.sqrt(vary)
num=num/(x_dev*y_dev)
print(num)
```