Stock Market Prediction

using RNN Network(LSTM model)

Udacity Machine Learning Nanodegree
Capstone Project
Kundan Kumar
06/25/2018

Table of Content

DEFINITION

Project Overview

The stock market is a model which is used to understand the financial behavior of the markets as Economic data is significant to follow the company progress. It helps us to know whether the company will progress or not and also helpful in investing money in the companies' stocks. It has a significant impact on the economy. Investors lose or gain on the fall or rise of Share market price. Investors can make a rough guess by analyzing the stock data, study company history, industry trends, etc.

The stock market is highly volatile, and it is difficult to guess the prices of the stocks, but by the recent advancement in the technology, it can make a better prediction and gives a better idea how and where to invest so that we can make an optimal profit with minimal risk. Machine learning models are much efficient for analyzing and predicting the stock prices. The domain background of this project is used to create machine learning model which can predict the stock price accurately.

The model will understand the stock price of the company wisely and will able to predict the future value of the company's stock. This project uses deep learning for the stock price prediction like Recurrent neural networks (RNN) which is the dominant model for sequential data. A sliding window approach used for predicting future prices.

This project uses Long-Short Term Memory (LSTM) deep learning model to predict stock prices. Recurrent neural networks (RNNs) are useful for time series data. Also, this project uses Keras library for building an LSTM to predict stock prices using historical adjusted closing price.

Problem Statement

In this project, we are going to predict the future closing value of a given stock over a period (next day) in the future. This project uses LSTM model (Long Short Term Memory networks) to predict the adjusted closing price of the google stock based on the historical datasets. Here, we will be going to predict the any given day price of the “GOOGL” stock from New York Stock Exchange Data-Sets.

Major Goals of the Project.

a. Exploratory analysis on the prices of stocks.
b. Implement Linear regression model and find it’s model accuracy.
c. Implement LSTM model from Keras library.
d. Compare the accuracy results of models.

Metrics

For this project, we will use Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) for analyzing the performance of the model by calculating the difference between predicted and actual values of the target stock at an adjusted closing price. It is favorite evaluation metrics for regression as well as for LSTM model. It used to give the robust model and prevents canceling the positive and negative error values. It is mathematically calculated by

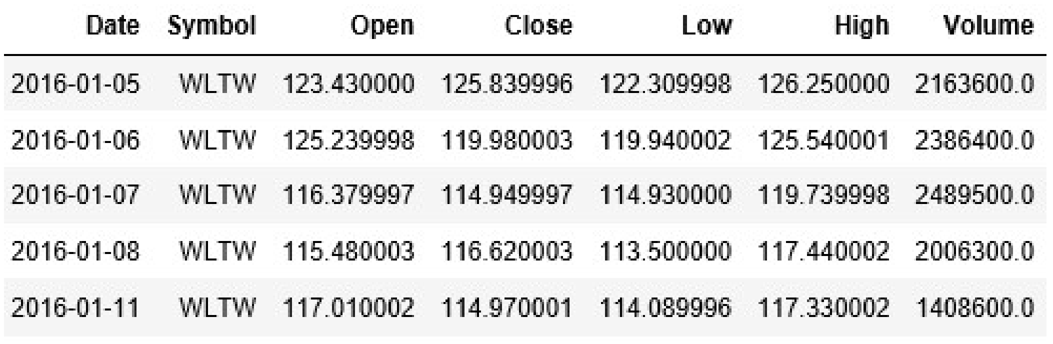
$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

ANALYSIS

Data Exploration

This project uses the New York Stock Exchange data from the Kaggle3 from January 4, 2010, to Dec 30, 2016. It is time series. The goal is to predict the adjusted closing price for any given date after training. There is four excel file, out of which price split adjusted data gives the Closing (Adjusted closing) price of the data. This file already adjusts the closing prices, so we need to predict for the "Close" price.
.

The sample datasets from the prices-split-adjusted are:



Date	Symbol	Open	Close	Low	High	Volume
2016-01-05	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0
2016-01-06	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0
2016-01-07	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0
2016-01-08	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0
2016-01-11	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0

Pic: The whole data are in project folder under datasets 'prices-split-adjusted.csv'

After checking, I didn't find any null or empty value in the datasets.

The mean, standard deviation, maximum and minimum of the data was found to be following:

Feature	Open	High	Low	Close	Volume
Mean	467.2965	471.0429	463.0375	467.0889	4096042.9057
Std	181.2923	182.5567	179.7161	181.1717	2883604.4816

Max	838.5	839.0	829.0399	835.7399	29619900.0
Min	219.3743	221.3613	217.0320	218.2532	520600.0

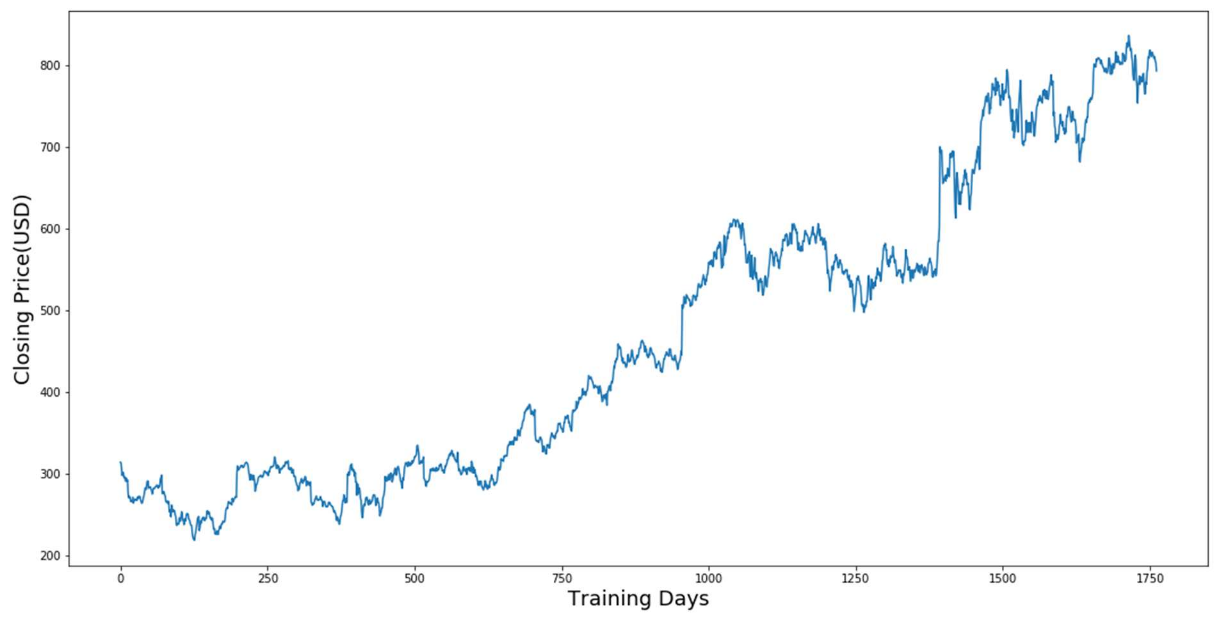
From the datasets, we can say that date, high and low values are not essential features of the data. It does not matter the highest prices of the stock or the lowest trading prices for a particular day. Only matters, here is the opening price of the stock and closing prices of the stock. If by the end of the day, if higher closing prices than the opening prices that it has profit else losses. The volume of share is a significant factor in the share market: If the volume is rising means the market will be rising. If decreasing volume and rising volume, show lack of interest which indicates a potential reversal. A price rise (or drop) on large volume is a stronger sign which indicates something has fundamentally changed in the stock.

So, I have removed Date, High and low features from dataset at preprocessing step. The mean, standard deviation, maximum and minimum of the preprocessed data was found to be following:

	Mean	**Std**	**Max**	**Min**
Open	0.4004	0.2928	1.0	0.0
Close	0.4029	0.2934	1.0	0.0
Volume	0.1228	0.0990	1.0	0.0

Exploratory Visualization

Matplotlib libraries are used for data visualization. Data-point is plotted between Closing stock price vs. no of items (no of days) available.



X-axis: Represents Tradings Days

Y-axis: Represents Closing Price In USD

Through this data shows continuous growth in Google between the Financial year of 2010-2017 with minor hiccups.

Algorithms and Techniques

The problem includes time series prediction in which the sequence of data flows is essential, as the next data that goes after it is related to the prior data. Traditional neural networks can’t accomplish as they didn’t have memory or concept of persistency in maintaining the previous data that it has processed thus far [3]. Recurrent neural networks addressed this issue as it uses a loop within its network in order to store previous information.

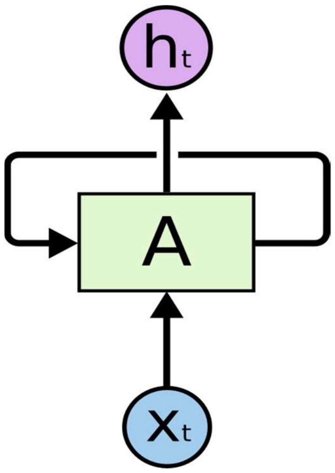


Fig: Loop within the Network

From a mathematical standpoint, the RNN takes each element of a sequence, multiply the element by a matrix and further sum the result with the previous output from the network. It can be summarized mathematically as.

$$h_t = \text{activation}(X_t W_X + W_h h_{t-1})$$

There are many types of RNN, but for this problem, I used Long Short-Term Memory (LSTM). LSTM is very popular and able to store memory or a concept from a sequence far back in the past to be applied in the present. Other types of RNN fall short of this due to a phenomenon called the vanishing gradient problem (which LSTM can overcome).

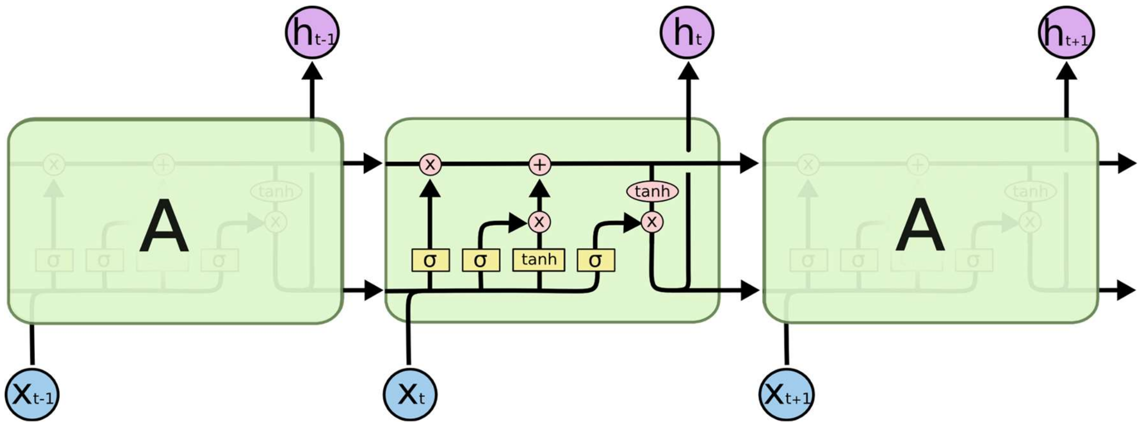


Fig: Repeating modules within an LSTM.

Above picture shows LSTM module. The LSTM itself has multiple variants of implementation. Each variant has been proven to work [4] better at some task and not in others. The variant that we will be using is called the Gated Recurrent Unit (GRU) [5].

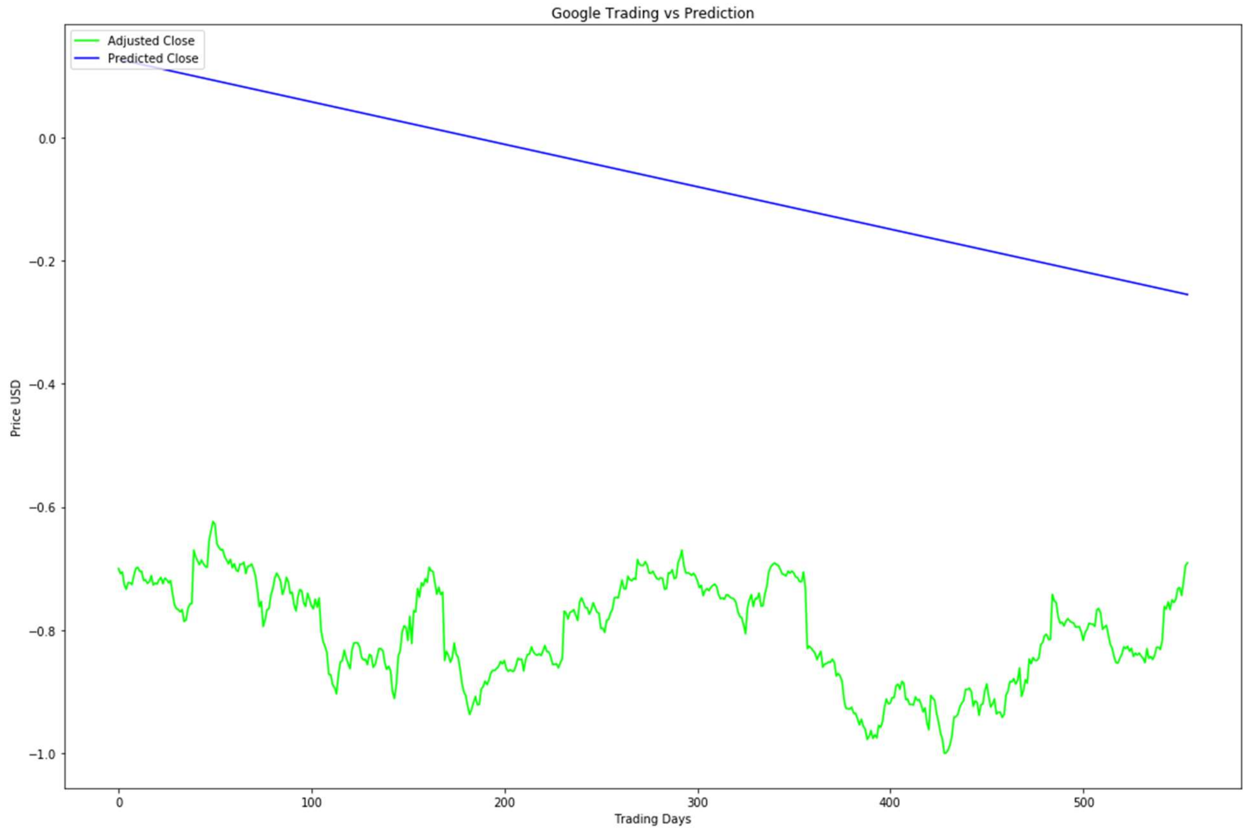
My approach in applying the algorithm and solve the problem is to use a simple 2-layer network RNN with default hyperparameters to establish our first basic LSTM model then we can improve the LSTM by tuning the parameters. Listed below the parameters and other details.

- Input Parameters
  - Preprocessing and Normalization (see Data Preprocessing Section)
- Neural Network Architecture
  - Number of Layers (how many layers of nodes in the model; used 2 for basic, 3 for improved)
  - Number of Nodes (how many nodes per layer; tested 16, 32, 64, 100,128)
- Training Parameters
  - Training / Test Split (how much of dataset to train versus test model on; kept constant at 82.95% and 17.05% for benchmarks and LSTM model)
  - Validation Sets (kept constant at 0.05% of training sets)
  - Batch Size (how many time steps to include during a single training step; kept at 1 for basic LSTM model and at 512 for improved LSTM model)
  - Optimizer Function (function to optimize by minimizing error; used “Adam” throughout)
  - Epochs (how many times to run through the training process; kept at 1 for base model and at 20 for improved LSTM)

Benchmark Model

The primary benchmark for this project is Linear Regression model. Primary goals is to understand the relative performance and implementation differences in machine learning versus deep learning models. This Linear Regressor is from the examples presented in Udacity's Machine Learning for Trading course and are used for error rate comparison MSE and RMSE utilizing the same dataset as the deep learning models.

The predicted results for my benchmark model :



X-axis: Represents Trading's Days

Y-axis: Represents Closing Price In USD

Green line: Adjusted Close price

Blue Line: Predicted Close price

Train Score: 0.9483 MSE (0.9738 RMSE)

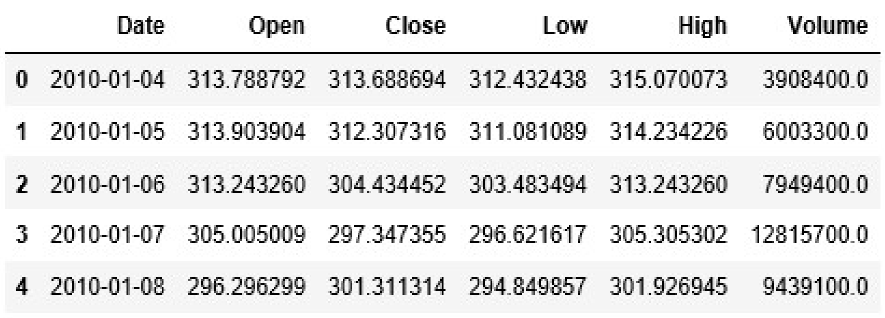
Test Score: 0.55686805 MSE (0.74623592 RMSE)

METHODOLOGY

Data Preprocessing

Followed the following Sequence for Getting and preprocessing the data.

a. Filtered the google stock data from the new York stock exchange data and saved as **google.csv** file in this format.

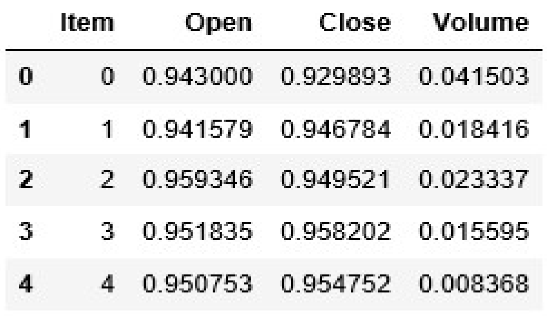


	Date	Open	Close	Low	High	Volume
0	2010-01-04	313.788792	313.688694	312.432438	315.070073	3908400.0
1	2010-01-05	313.903904	312.307316	311.081089	314.234226	6003300.0
2	2010-01-06	313.243260	304.434452	303.483494	313.243260	7949400.0
3	2010-01-07	305.005009	297.347355	296.621617	305.305302	12815700.0
4	2010-01-08	296.296299	301.311314	294.849857	301.926945	9439100.0

b. Remove unimportant features like date, high and low from the obtained data and reversed the order of data, i.e., from January 04, 2010 to Dec 30, 2010

Item	Open	Close	Volume
0	98.80	101.46	15860692
1	100.77	97.35	13762396
2	96.82	96.85	8239545
3	97.72	94.37	10389803

c. **MinMaxScaler** function from Scikit-Learn is used for Normalization of data.



	Item	Open	Close	Volume
0	0	0.943000	0.929893	0.041503
1	1	0.941579	0.946784	0.018416
2	2	0.959346	0.949521	0.023337
3	3	0.951835	0.958202	0.015595
4	4	0.950753	0.954752	0.008368

d. Stored normalized data in **preprocessed_data_google.csv** file for future reusability.

e. Dataset are splitted into the training (68.53%) and test (31.47%) datasets for linear regression model. The split was of following shape :

```
x_train (1207, 1)
y_train (1207, 1)
x_test (555, 1)
y_test (555, 1)
```

• Dataset are splitted into the training (82.95%) and test (17.05%) datasets for LSTM model. The Split was of following shape:

```
x_train (1226, 30, 3)
y_train (1226,)
x_test (466, 30, 3)
y_test (466,)
```

Implementation

Below the steps took to complete the projects:

LSTM:
Steps to perform google stock prediction using LSTM::

a. First downloaded CSV file from Kaggle and loaded the data with the help of Pandas.

b. Performed the exploratory analysis and dropped the unnecessary columns like date, high, low. Then remaining column left was open, close, volume column.

c. Datasets need to be normalized before applying to machine learning algorithm.

d. normalization is done using MinMaxScaler preprocessing class from the sci-kit-learn library.

e. Data modeling is important, and then model accuracy is important as it decides how well model performed on training and testing datasets. Training is must so that it can predict well on the unseen datasets.

f. The dataset is split in training and testing dataset in 80:20 ratio but tweaked the ratio for model accuracy.

g. Training and Testing datasets are numpy arrays consists of two lists, dataX and dataY.

h. Converted the training and testing dataset in the format t, t+1, where t tells the value of present day and t+1 tell the value of the next day.

i. A return_sequences parameter has been defined which tells about the number of days to look into in the future.

j. Before training the data, the parameters of the LSTM has been defined. e.g- model.add(LSTM(input_shape=(None, input_dim),units=output_dim, return_sequences=return_sequences))

k. In the LSTM cell, added a dense layer e.g- model.add(Dense(2))

l. Compiled the model before training. (loss='mean_squared_error', optimizer='adam')

m. Now the trainX and trainY components of the training dataset are fit into the model- model.fit(trainX, trainY, epochs=10, batch_size=1)

n. After training the model, the test dataset can be passed for testing the dataset in the trained model

o. Calculated “mean-square error” by finding the difference between the actual value and the predicted value.

Linear Regression:

Steps to perform google stock prediction using using Linear regression:

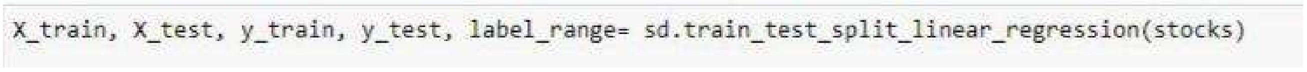
a. First downloaded CSV file from Kaggle and loaded the data with the help of Pandas.
b. Performed the exploratory analysis and dropped the unnecessary columns like date, high, low. Then remaining column left was open, close, volume column.
c. Datasets need to be normalized before applying to machine learning algorithm.
d. normalization is done using MinMaxScaler preprocessing class from the sci-kit-learn library.
e. Divided the dataset into training and testing in the ratio 80:20 using scikit learn library sklearn.model_selection.TimeSeriesSplit
f. built a linear regression model using sklearn.linear_model
g. Predicted the label for a given testing set.
h. Calculated “mean-square error” by finding the difference between the actual value and the predicted value.

I have thoroughly specified all the steps to build, train and test model and its predictions in the notebook itself.

The major challenges faced to find the performance of LSTM model, it required lots of combination of parameters to get the desired results. The only way to overcome is to play around the parameters and check the optimality of the model.

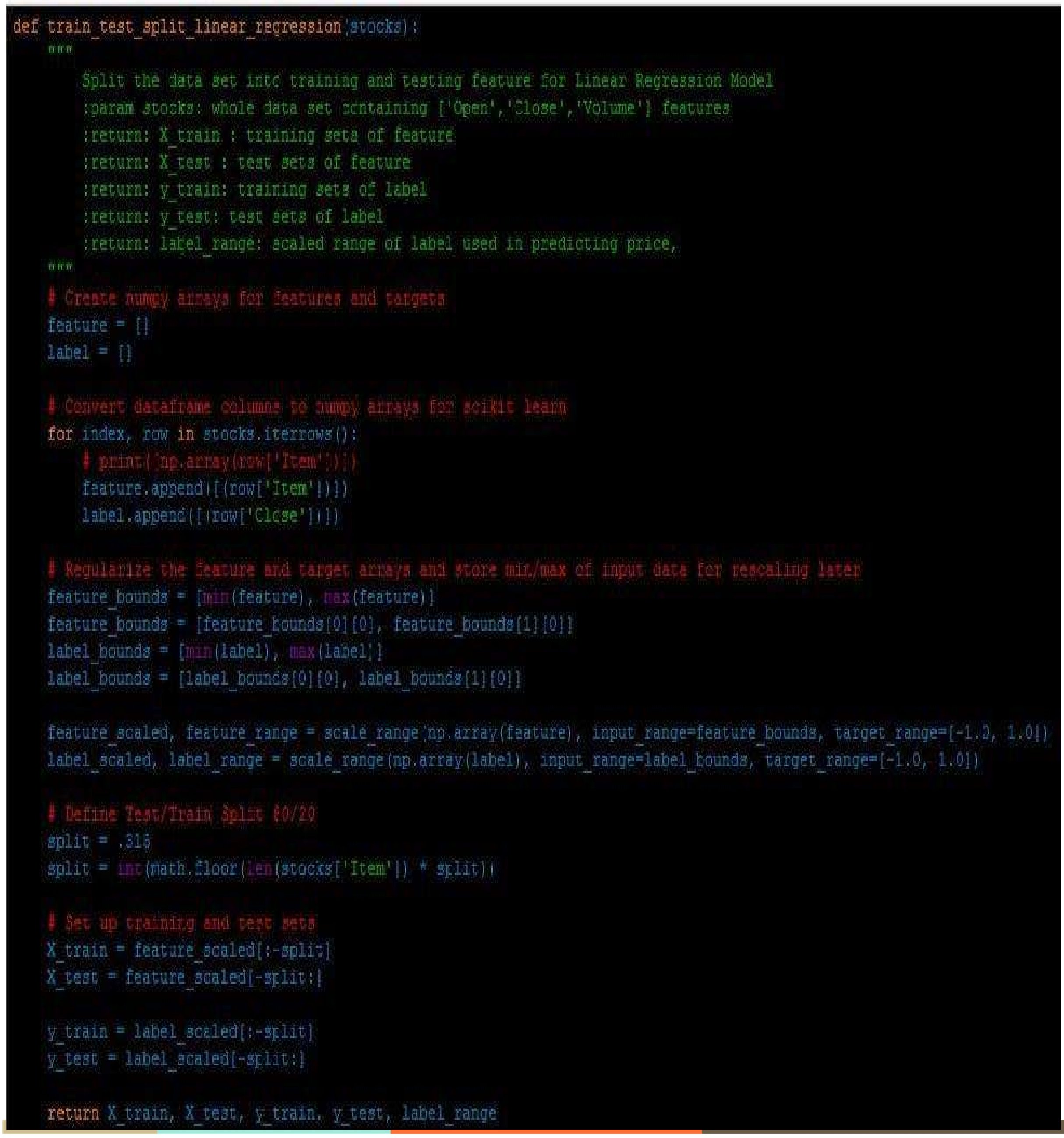
Some code implementation insight:

Benchmark model :



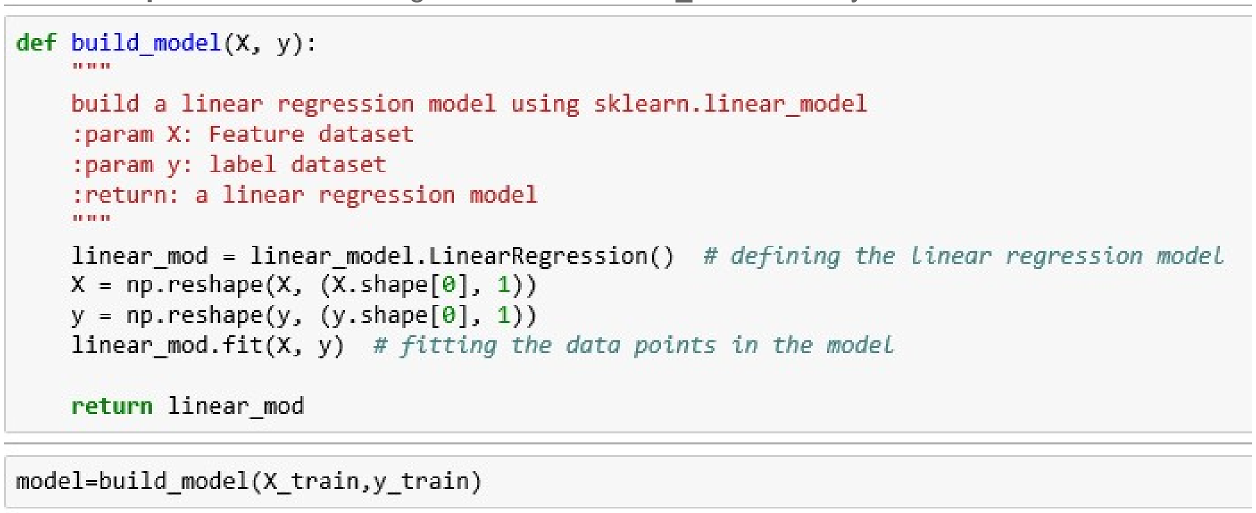
```
X_train, X_test, y_train, y_test, label_range= sd.train_test_split_linear_regression(stocks)
```

Step 1 : Split into train and test model :A function called from **‘stock_data.py’**which splits the data for linear regression model. The function is as follows:



```
def train_test_split_linear_regression(stocks):
    """
        Split the data set into training and testing feature for Linear Regression Model
        :param stocks: whole data set containing ['Open','Close','Volume'] features
        :return: X_train : training sets of feature
        :return: X_test : test sets of feature
        :return: y_train: training sets of label
        :return: y_test: test sets of label
        :return: label_range: scaled range of label used in predicting price,
    """
    # Create numpy arrays for features and targets
    feature = []
    label = []

    # Convert dataframe columns to numpy arrays for scikit learn
    for index, row in stocks.iterrows():
        # print([np.array(row['Item'])])
        feature.append([(row['Item'])])
        label.append([(row['Close'])])

    # Regularize the feature and target arrays and store min/max of input data for rescaling later
    feature_bounds = [min(feature), max(feature)]
    feature_bounds = [feature_bounds[0][0], feature_bounds[1][0]]
    label_bounds = [min(label), max(label)]
    label_bounds = [label_bounds[0][0], label_bounds[1][0]]

    feature_scaled, feature_range = scale_range(np.array(feature), input_range=feature_bounds, target_range=[-1.0, 1.0])
    label_scaled, label_range = scale_range(np.array(label), input_range=label_bounds, target_range=[-1.0, 1.0])

    # Define Test/Train Split 80/20
    split = .315
    split = int(math.floor(len(stocks['Item']) * split))

    # Set up training and test sets
    X_train = feature_scaled[:-split]
    X_test = feature_scaled[-split:]

    y_train = label_scaled[:-split]
    y_test = label_scaled[-split:]

    return X_train, X_test, y_train, y_test, label_range
```

Step 2: Model built using **scikit-learn linear_model** library.

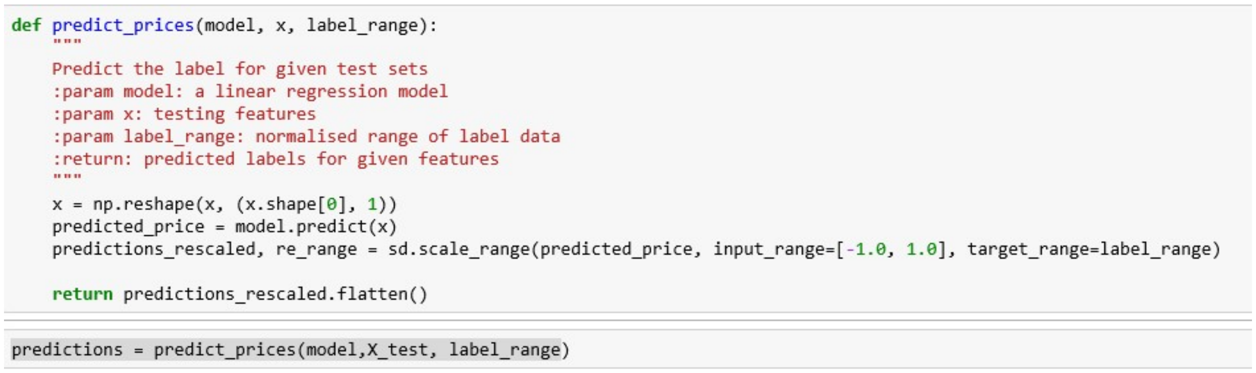


```
def build_model(X, y):
    """
    build a linear regression model using sklearn.linear_model
    :param X: Feature dataset
    :param y: label dataset
    :return: a linear regression model
    """
    linear_mod = linear_model.LinearRegression()  # defining the linear regression model
    X = np.reshape(X, (X.shape[0], 1))
    y = np.reshape(y, (y.shape[0], 1))
    linear_mod.fit(X, y)  # fitting the data points in the model

    return linear_mod
```

```
model=build_model(X_train,y_train)
```

Model build for this project is attached above.

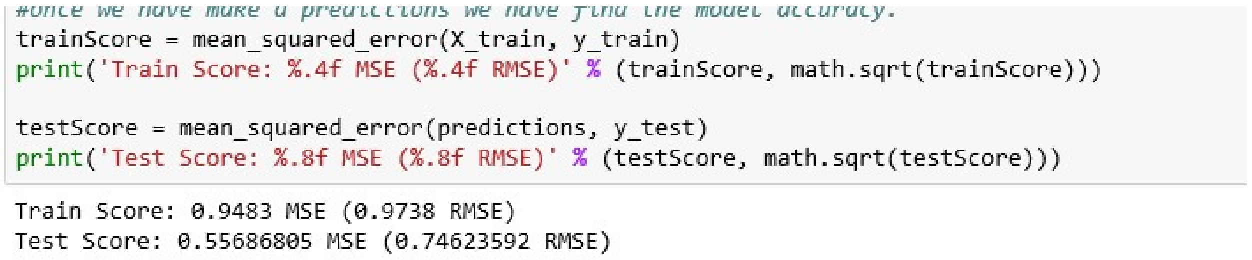
Step 3: To predict the prices for given test datasets. Screenshot for the function below.



```
def predict_prices(model, x, label_range):
    """
    Predict the label for given test sets
    :param model: a linear regression model
    :param x: testing features
    :param label_range: normalised range of label data
    :return: predicted labels for given features
    """
    x = np.reshape(x, (x.shape[0], 1))
    predicted_price = model.predict(x)
    predictions_rescaled, re_range = sd.scale_range(predicted_price, input_range=[-1.0, 1.0], target_range=label_range)

    return predictions_rescaled.flatten()
```

```
predictions = predict_prices(model,X_test, label_range)
```

Step 4: Finally calculate the test score and plot the results of benchmark model

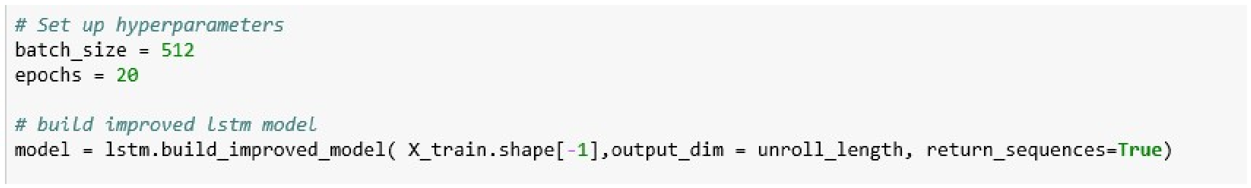


```
#once we have make a predictions we have find the model accuracy.
trainScore = mean_squared_error(X_train, y_train)
print('Train Score: %.4f MSE (%.4f RMSE)' % (trainScore, math.sqrt(trainScore)))

testScore = mean_squared_error(predictions, y_test)
print('Test Score: %.8f MSE (%.8f RMSE)' % (testScore, math.sqrt(testScore)))

Train Score: 0.9483 MSE (0.9738 RMSE)
Test Score: 0.55686805 MSE (0.74623592 RMSE)
```

Improved LSTM model :

Step 1 : Split dataset into training and testing data ,it is same for Basic LSTM.

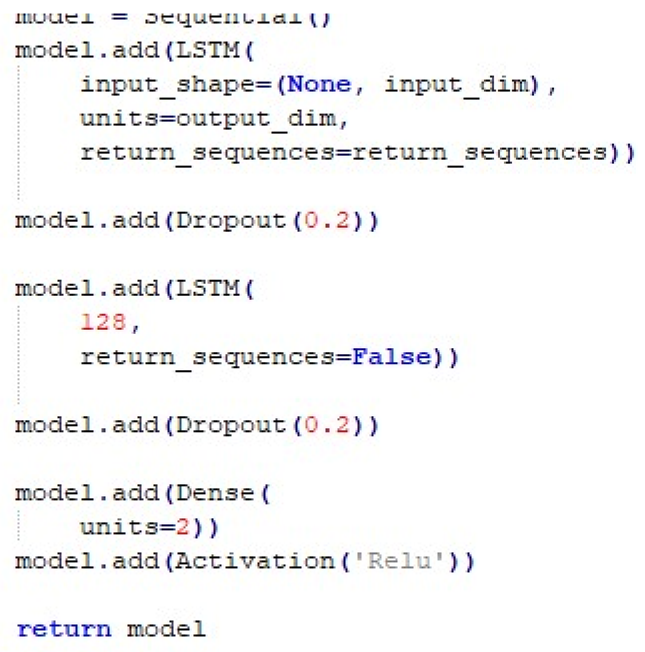
Step 2 : Build an improved LSTM model :



```
# Set up hyperparameters
batch_size = 512
epochs = 20

# build improved lstm model
model = lstm.build_improved_model( X_train.shape[-1],output_dim = unroll_length, return_sequences=True)
```

The function calls from lstm.py and it uses **keras Long short term memory** library for LSTM model.

a. Batch_Size increased form 1 to 512.
b. Epochs increased from 1 to 20 for improved LSTM model
c. In the function, added increased hidden layer from 100 to 120 and drop out to 0.2 to all the layers.



```
model = Sequential()
model.add(LSTM(
    input_shape=(None, input_dim),
    units=output_dim,
    return_sequences=return_sequences))

model.add(Dropout(0.2))

model.add(LSTM(
    128,
    return_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(
    units=2))
model.add(Activation('Relu'))

return model
```

Step 3: We now need to train our model: built in library function to train the model.

```
model.fit(X_train,
          y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=2,
          validation_split=0.05
         )
```

Step 4: Predict the prices for given test datasets: built-in function to predict the outcomes of the model.

```
# Generate predictions
predictions = model.predict(X_test, batch_size=batch_size)
```

Step 5: Calculate the test score and plot the results of improved LSTM model.

```
trainScore = model.evaluate(X_train, y_train, verbose=0)
print('Train Score: %.8f MSE (%.8f RMSE)' % (trainScore, math.sqrt(trainScore)))

testScore = model.evaluate(X_test, y_test, verbose=0)
print('Test Score: %.8f MSE (%.8f RMSE)' % (testScore, math.sqrt(testScore)))
```

```
Train Score: 0.00140906 MSE (0.03753746 RMSE)
Test Score: 0.00077077 MSE (0.02776274 RMSE)
```

Refinement

This project I fine tune the parameters of LSTM to get better predictions. By testing and analyzing each parameter, model improved a lot, and this selects the final value for each of them.

To improve LSTM,I have done following:

- Increased the number of hidden node from 100 to 128.
- Added Dropout of 0.2 at each layer of LSTM
- Increased batch size from 1 to 512
- Increased epochs from 1 to 20
- Added verbose = 2

- Made prediction with the batch size

The mean square error has improved from 0.001698 to 0.00077 for testing set.

The predicted plot difference be follows:

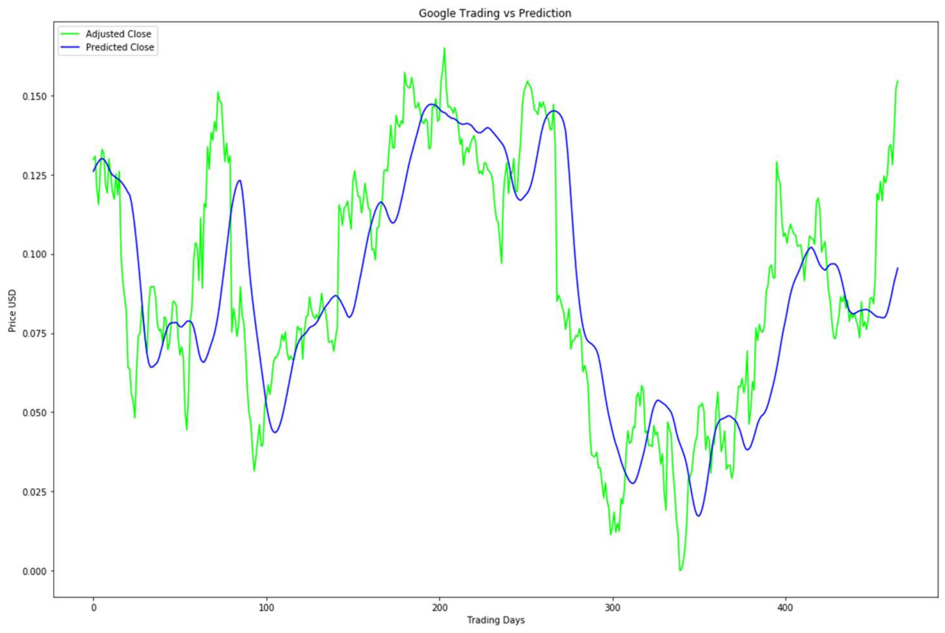


Fig : Plot For Adjusted Close and Predicted Close Prices for improved LSTM model

RESULT

Model Evaluation and Validation

For each model, it was fined tune my predictions and hence reduced mean squared error significantly.

- For first model using linear regression model:
    - **Train Score: 0.9483 MSE (0.9738 RMSE)**
    - **Test Score: 0.55686805 MSE (0.74623592 RMSE)**

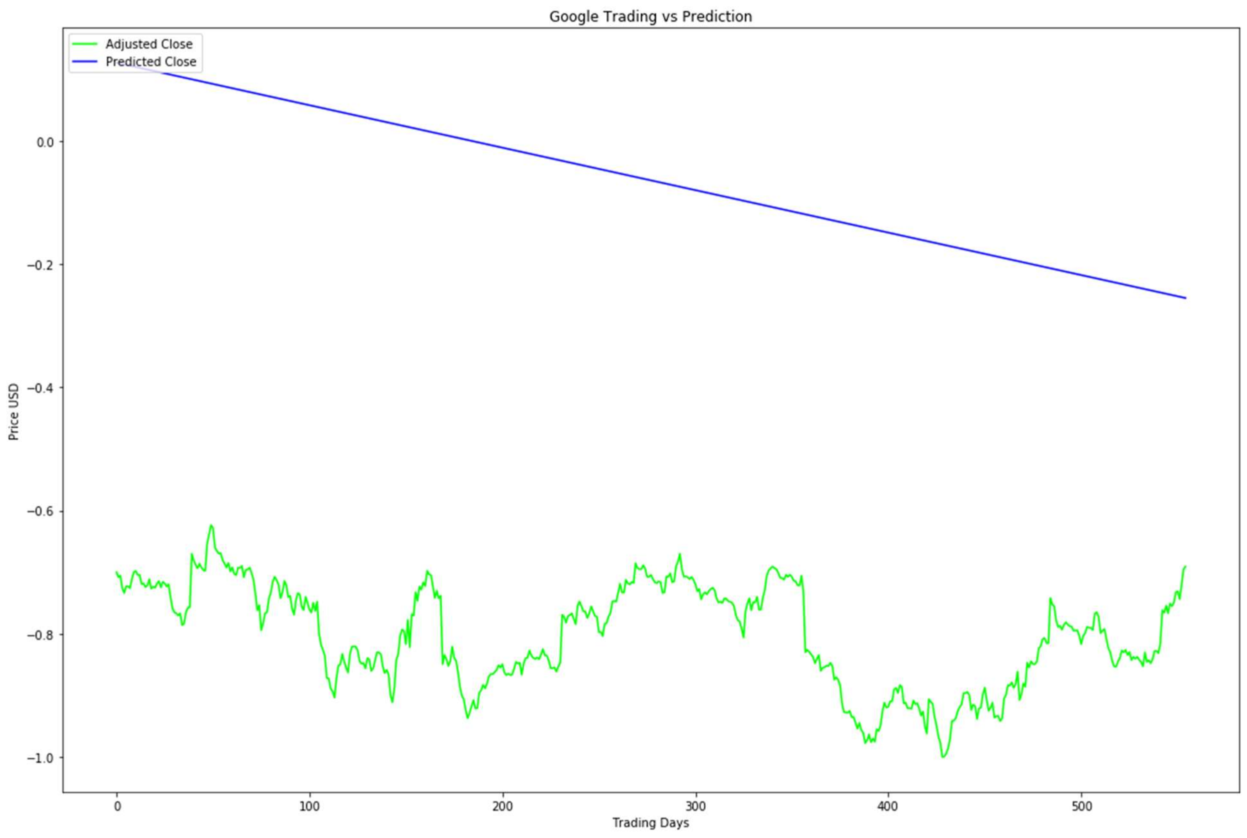


Fig: Linear Regression Model Plot

- For second model using basic Long-Short Term memory model:
    - **Train Score: 0.00262154 MSE (0.05120101 RMSE)**
    - **Test Score: 0.00169895 MSE (0.04121831 RMSE)**

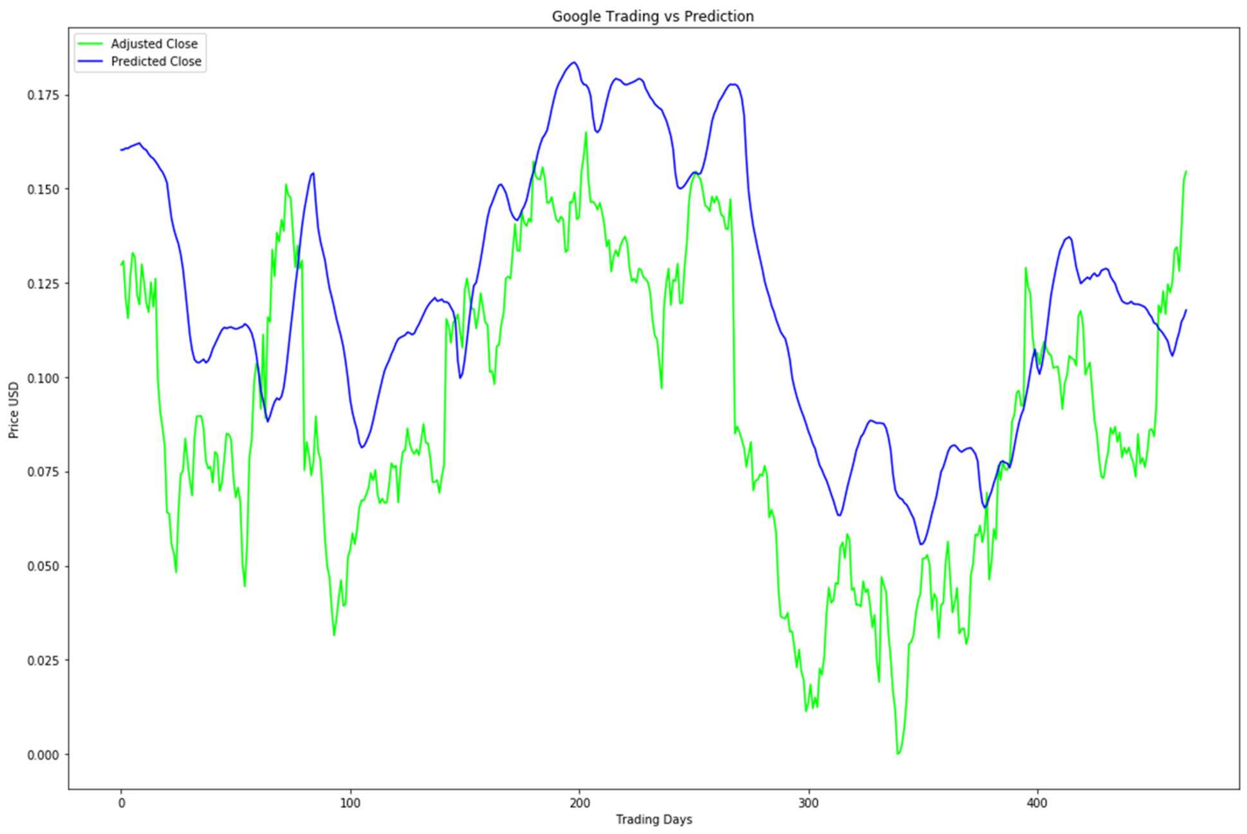


Fig: Basic Long-Short Term Memory model Plot

- For my third and final model, using improved Long-Short Term memory model:
    - **Train Score: 0.00140906 MSE (0.03753746 RMSE)**
    - **Test Score: 0.00077077 MSE (0.02776274 RMSE)**

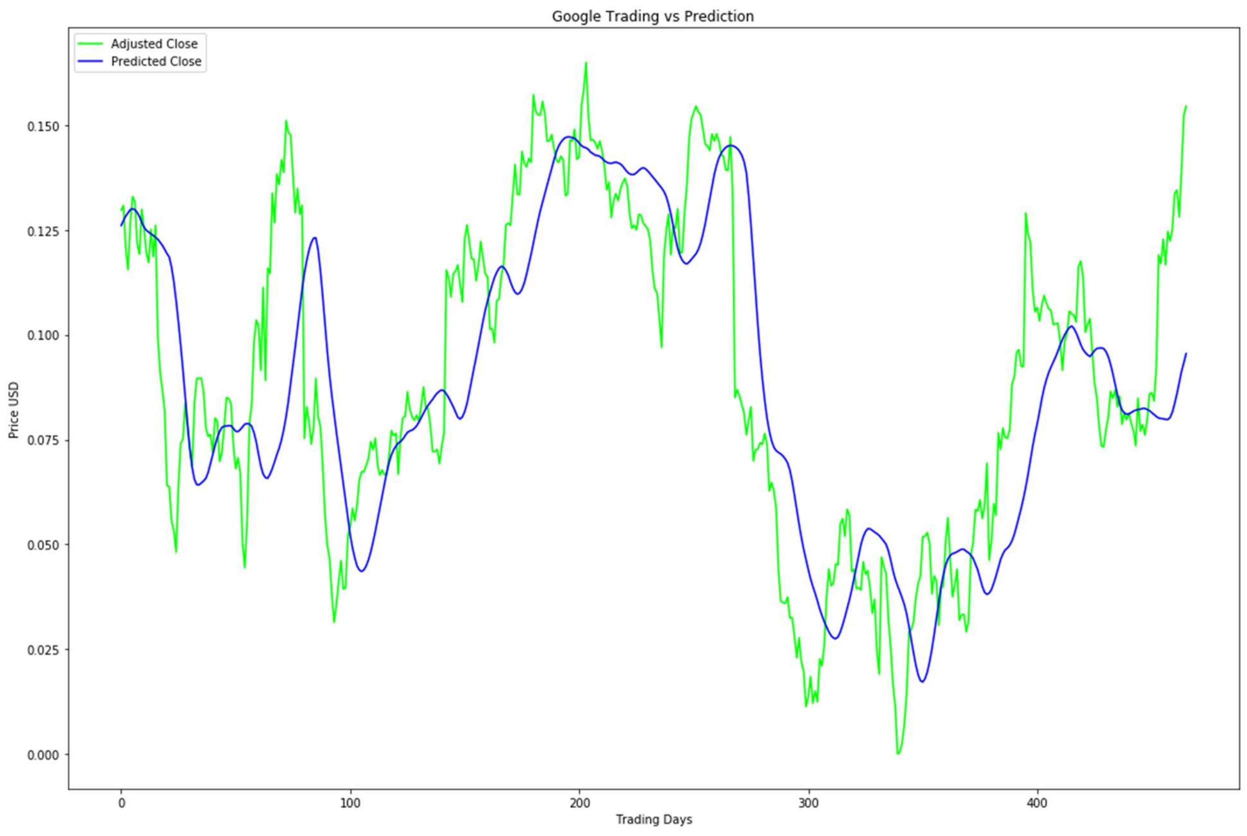


Fig: Improved Long-Short Term Memory Model Plot

Robustness:

For robustness checking of my final model, I used an unseen data of Google from July 1, 2017, to July 20, 2017. I have a decent result for unseen data. The results are as:

Test Score: 0.3738 MSE (0.6114 RMSE)

Justification

Comparing the benchmark model from Linear Regression to the improved LSTM model, the Comparing the benchmark model from Linear Regression to the improved LSTM model, the Mean Squared Error improves from 0.55686805 MSE (0.74623592 RMSE) [Linear Regression Model] to 0.00077077 MSE (0.02776274 RMSE)[Improved LSTM].The significant decrease in error rate clearly shows that final model has better performance than the primary and benchmark model.
The Average Delta Price between actual and predicted Adjusted Closing Price values was:

Delta Price: 0.000931 - RMSE * Adjusted Close Range

CONCLUSION

Free-Form Visualization

Already discussed above features of the datasets and their visualization. To conclude report, I have chosen my final model visualization, which is improved version of LSTM by fine-tuning parameters. I am impressed with the last model which has a mean square error of just 0.00077077. It is nice working on the project.

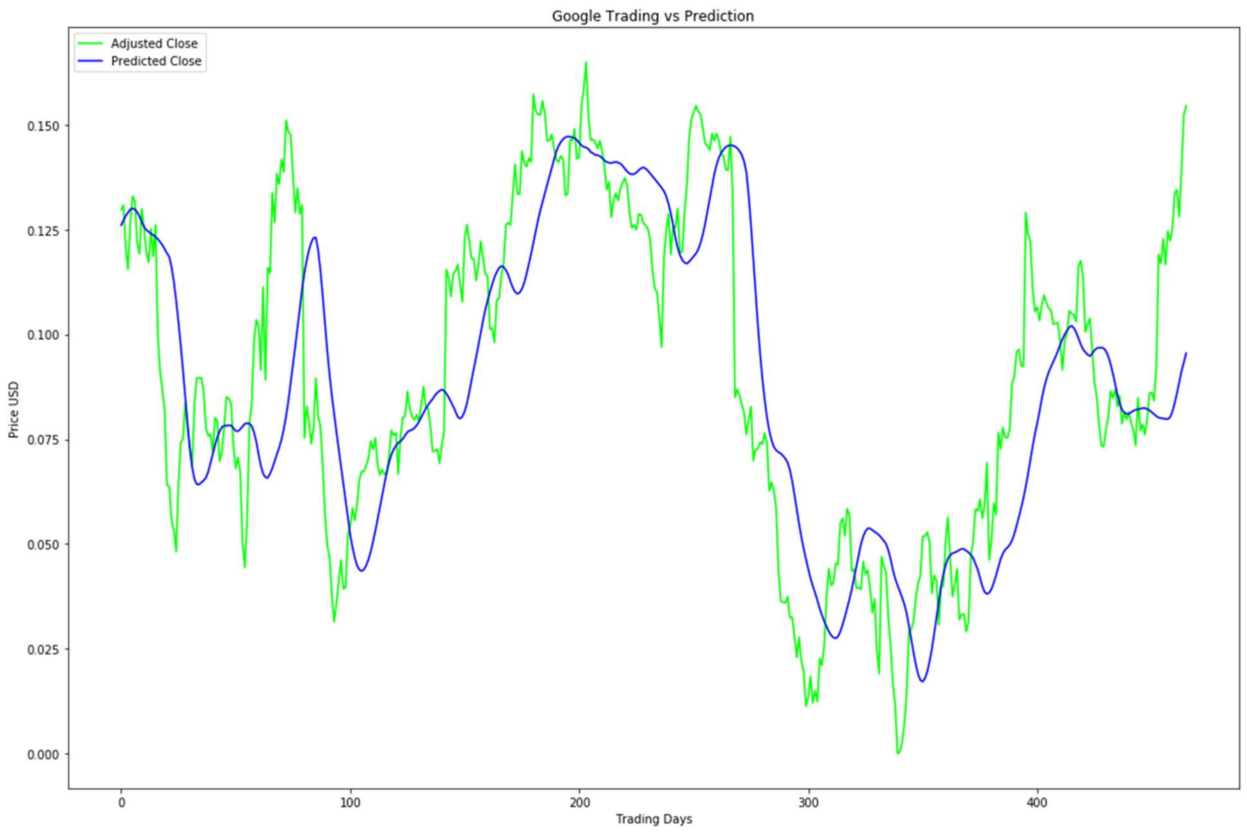


Fig: Improved Long-Short Term Memory Model Plot

Reflection

To recap the entire project:

- Set Up Infrastructure
    - Juypter Notebook
    - Incorporate required Libraries (Keras, Tensor flow, Pandas, Matplotlib, Sklearn, Numpy)
    - GitHub project
- Data Processing
    - Incorporate adjusted data of New York Exchange.
    - Filtered the data based on google company
    - Generated the new csv file which conatins only google stocks
    - Processed Dataset using Pandas Dataframe
    - Normalized the dataset
    - 80/20 dataset split on training and test data across all models
- Develop Benchmark Model
    - Linear Regression model with Scikit-Learn was set up
    - Calibrated the parameters
- Develop Basic LSTM Model
    - Basic LSTM model with Keras utilizing parameters from Benchmark Model was set up
- Improve LSTM Model
    - Developed, Visualized, and compare the results for the LSMT model
- Plot Actual, Benchmark Predicted Values, and LSTM Predicted Values per time series
- Analyze and describe results for report.

The main motto of this project to explore time series data sets and learn new machine learning algorithm, i.e., Long-Short Term Memory (LSTM). I am satisfied with the results as final model meets my expectation and also performs well.

The main problem I had when I was exploring the data. It was the most onerous task to convert the dataset to preprocess data and then split into training and test data. I worked for two models, i.e., Linear Regression and Long-Short Term Memory, as both of them have different inputs sizes which was quite challenging. I read several article and research papers to reach this final model, and I think it worth it.

Improvement:

There is a scope of improvement needed with this project. Though it has a minimum Mean Squared error for predicting closing prices of the Google stocks, still projects require a bit of improvement.

- The New York Stock Exchange datasets (from Kaggle) contain google stock from 2010 to 2016 only. If we collect more data (from 2005 to 2017) and train it, chances will be higher to get a model with better performance. Moreover, there is no information about the public release and narratives which impact stock prices. These types of data are helpful to analyze prices (rise or fall of stock) and can give better results. I will try to associate these data in the future with current datasets and find out how public release and narratives can impact stock prices.
- There is no user interface .UI will be added so that user can check the value of futures dates
- Entire stock exchange data will be trained so that user can check the stock values of any stock

I would like to add above improvement to this project in future.

References:

[1]. Kaggle,"New York Stock Exchange": https://www.kaggle.com/dgawlik/nyse

[2]. https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

[3]. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[4]. http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf

[5]. http://arxiv.org/pdf/1406.1078v3.pdf

[6] https://www.ijsr.net/archive/v6i4/ART20172755.pdf

[7] https://medium.com/mlreview/a-simple-deep-learning-model-for-stock-price-prediction-using-tensorflow-30505541d877