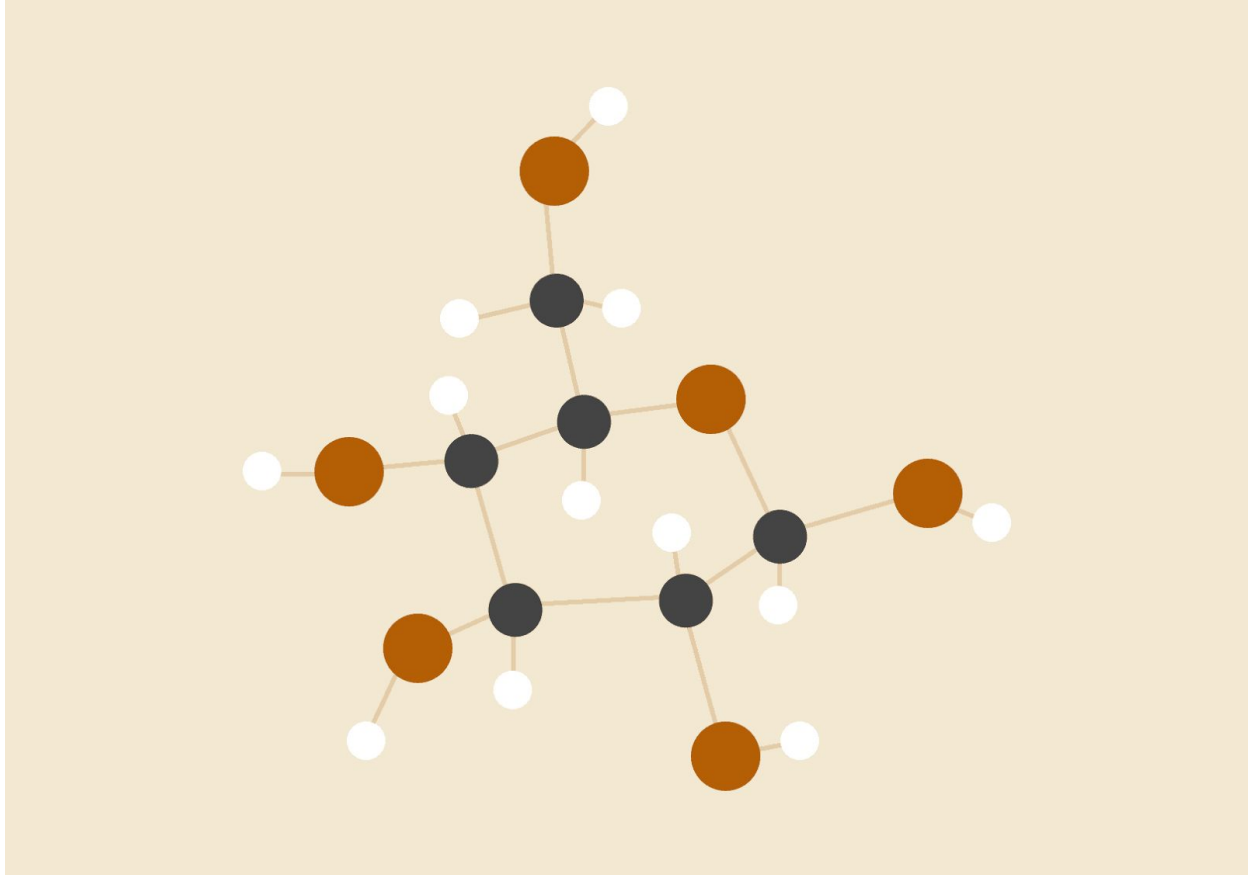


STOCK PRICE PREDICTOR

Using Long-Short Term Memory Networks



Rajat Dhyani

19.07.2017

Udacity Machine Learning Nanodegree

Table of Content

Table of Content	1
DEFINITION	2
Project Overview	2
Problem Statement	3
Metrics	3
ANALYSIS	4
Data Exploration	4
Exploratory Visualization	5
Algorithms and Techniques	6
Benchmark Model	7
METHODOLOGY	8
Data Preprocessing	8
Implementation	10
Refinement	11
RESULT	14
Model Evaluation and Validation	14
Justification	17
CONCLUSION	18
Free-Form Visualization	18
Reflection	19
Improvement	20

DEFINITION

Project Overview

Investment firms, hedge funds and even individuals have been using financial models to better understand market behavior and make profitable investments and trades. A wealth of information is available in the form of historical stock prices and company performance data, suitable for machine learning algorithms to process.

Can we actually predict stock prices with machine learning? Investors make educated guesses by analyzing data. They'll read the news, study the company history, industry trends and other lots of data points that go into making a prediction. The prevailing theories is that stock prices are totally random and unpredictable but that raises the question why top firms like Morgan Stanley and Citigroup hire quantitative analysts to build predictive models. We have this idea of a trading floor being filled with adrenaline infused men with loose ties running around yelling something into a phone but these days they're more likely to see rows of machine learning experts quietly sitting in front of computer screens. In fact about 70% of all orders on Wall Street are now placed by software, we're now living in the age of the algorithm.

This project seeks to utilize Deep Learning models, Long-Short Term Memory (LSTM) Neural Network algorithm, to predict stock prices. For data with timeframes recurrent neural networks (RNNs) come in handy but recent researches have shown that LSTM, networks are the most popular and useful variants of RNNs.

I will use Keras to build a LSTM to predict stock prices using historical closing price and trading volume and visualize both the predicted price values over time and the optimal parameters for the model.

Problem Statement

The challenge of this project is to accurately predict the future closing value of a given stock across a given period of time in the future. For this project I will use a **Long Short**

Term Memory networks¹ – usually just called “LSTMs” to predict the closing price of the **S&P 500**² using a dataset of past prices

GOALS

1. Explore stock prices.
2. Implement basic model using linear regression.
3. Implement LSTM using keras library.
4. Compare the results and submit the report.

Metrics

For this project measure of performance will be using the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) calculated as the difference between predicted and actual values of the target stock at adjusted close price and the delta between the performance of the benchmark model (Linear Regression) and our primary model (Deep Learning).

¹ [Long Short Term Memory networks](#)

² [S&P 500 companies](#)

ANALYSIS

Data Exploration

The data used in this project is of the **Alphabet Inc**³ from **January 1, 2005 to June 20, 2017**, this is a series of data points indexed in time order or a time series. My goal was to predict the closing price for any given date after training. For ease of reproducibility and reusability, all data was pulled from the **Google Finance Python API**⁴.

The dataset is of following form :

Date	Open	High	Low	Close	Volume
30-Jun-17	943.99	945.00	929.61	929.68	2287662
29-Jun-17	951.35	951.66	929.60	937.82	3206674
28-Jun-17	950.66	963.24	936.16	961.01	2745568

The whole data can be found out in 'Google.csv' in the project root folder⁵

We can infer from this dataset that date, high and low values are not important features of the data. These can be removed in preprocessing step. The prediction has to be made for Closing (Adjusted closing) price of the data. Since Google Finance already **adjusts the closing prices for us**⁶, we just need to make prediction for "CLOSE" price.

³ [Alphabet Inc](#)

⁴ [Google Finance python api](#)

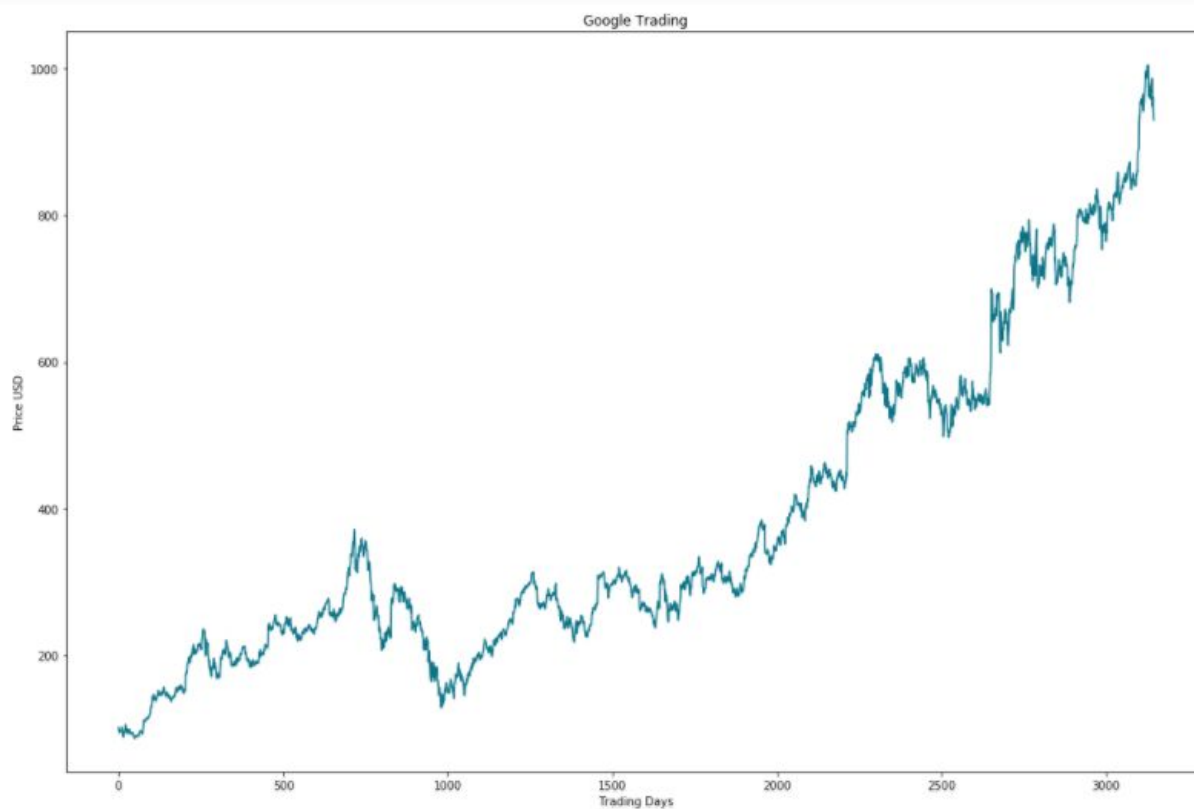
⁵ [Google.csv](#)

⁶ [adjusts the closing prices for us](#)

Exploratory Visualization

To visualize the data i have used **matplotlib**⁷ library. I have plotted Closing stock price of the data with the no of items(no of days) available.

Following is the snapshot of the plotted data :



*X-axis: Represents Tradings Days
Y-axis: Represents Closing Price In USD*

Through this data we can see a continuous growth in Alphabet Inc. The major fall in the prices between 600-1000 might be because of the Global Financial Crisis of 2008-2009.

⁷ [Matplotlib](#)

Algorithms and Techniques

The goal of this project was to study time-series data and explore as many options as possible to accurately predict the Stock Price. Through my research i came to know about **Recurrent Neural Nets (RNN)**⁸ which are used specifically for sequence and pattern learning. As they are networks with loops in them, allowing information to persist and thus ability to memorise the data accurately. But Recurrent Neural Nets have vanishing Gradient descent problem which does not allow it to learn from past data as was expected. The remedy of this problem was solved in **Long-Short Term Memory Networks**⁹, usually referred as LSTMs. These are a special kind of RNN, capable of learning long-term dependencies.

In addition to adjusting the architecture of the Neural Network, the following full set of parameters can be tuned to optimize the prediction model:

- Input Parameters
 - Preprocessing and Normalization (see Data Preprocessing Section)
- Neural Network Architecture
 - Number of Layers (how many layers of nodes in the model; used 3)
 - Number of Nodes (how many nodes per layer; tested 1,3,8, 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 (which function to optimize by minimizing error; used “Adam” throughout)
 - Epochs (how many times to run through the training process; kept at 1 for base

⁸ [Recurrent Neural Network](#)

⁹ [Long-Short Term Memory](#)

model and at 20 for improved LSTM)

Benchmark Model

For this project i have used a Linear Regression model as its primary benchmark. As one of my goals is to understand the relative performance and implementation differences of machine learning versus deep learning models. This Linear Regressor was based on the examples presented in Udacity's Machine Learning for Trading course and was used for error rate comparison MSE and RMSE utilizing the same dataset as the deep learning models.

Following is the predicted results that i got from my benchmark model :



*X-axis: Represents Tradings Days
Y-axis: Represents Closing Price In USD
Green line: Adjusted Close price
Blue Line: Predicted Close price*

Train Score: 0.1852 MSE (0.4303 RMSE)

Test Score: 0.08133781 MSE (0.28519784 RMSE)

METHODOLOGY

Data Preprocessing

Acquiring and preprocessing the data for this project occurs in following sequence, much of which has been modularized into the **preprocess.py** file for importing and use across all notebooks:

- Request the data from the Google Finance Python API and save it in **google.csv** file in the following format.

Date	Open	High	Low	Close	Volume
30-Jun-17	943.99	945.00	929.61	929.68	2287662
29-Jun-17	951.35	951.66	929.60	937.82	3206674
28-Jun-17	950.66	963.24	936.16	961.01	2745568

- Remove unimportant features(date, high and low) from the acquired data and reversed the order of data, i.e., from january 03, 2005 to june 30, 2005

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

- Normalised the data using **MinMaxScaler** helper function from Scikit-Learn.

Item	Open	Close	Volume
0	0.012051	0.015141	0.377248
1	0.014198	0.010658	0.325644
2	0.009894	0.010112	0.189820
3	0.010874	0.007407	0.242701

- Stored the normalised data in **google_preprocessed.csv** file for future reusability.
- Splitted the dataset into the training (68.53%) and test (31.47%) datasets for linear regression model. The split was of following shape :

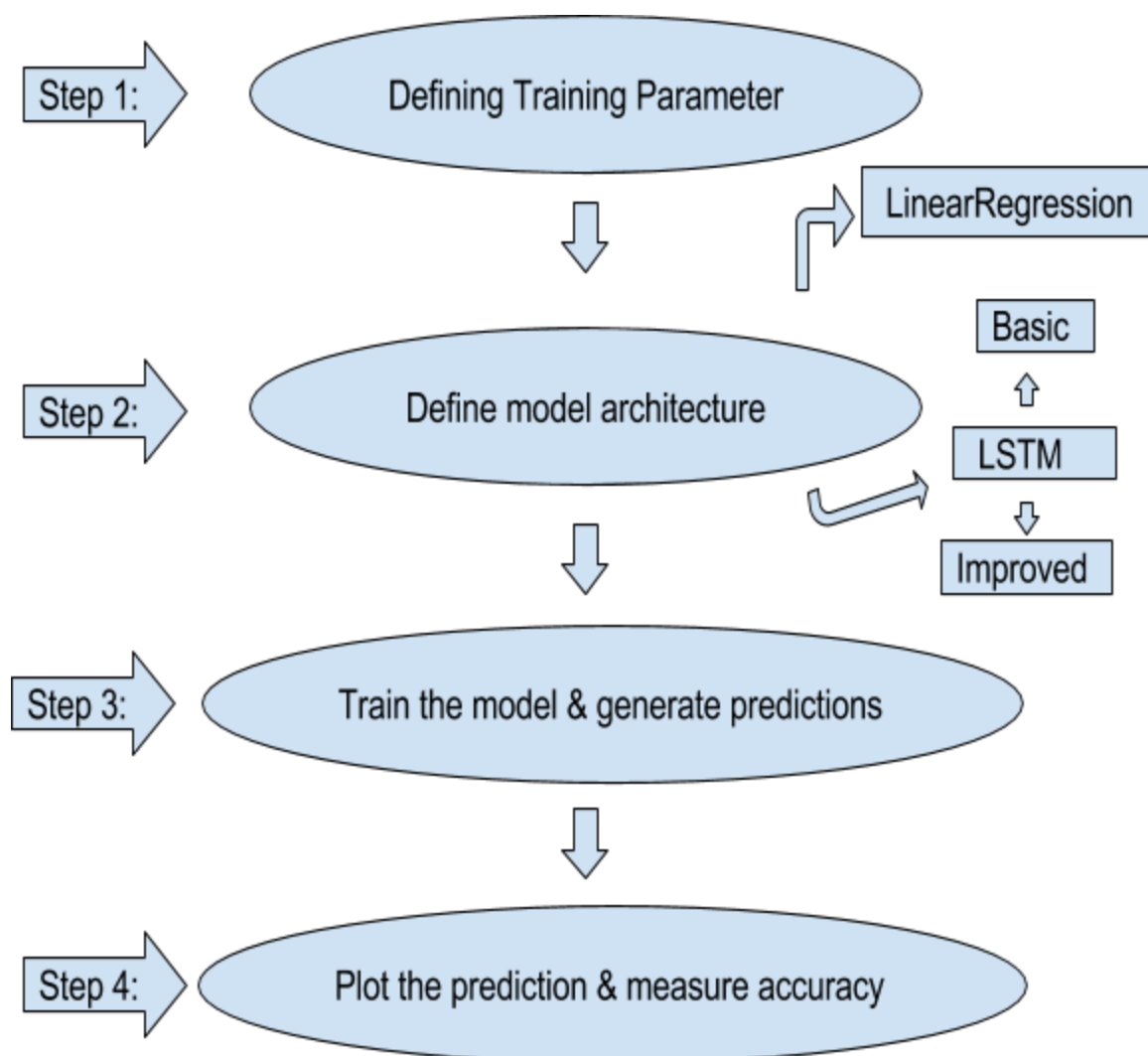
```
x_train (2155, 1)
y_train (2155, 1)
x_test (990, 1)
y_test (990, 1)
```

- Splitted the dataset into the training (82.95%) and test (17.05%) datasets for LSTM model. The Split was of following shape:

```
x_train (2589, 50, 3)
y_train (2589,)
x_test (446, 50, 3)
y_test (446,)
```

Implementation

Once the data has been downloaded and preprocessed, the implementation process occurs consistently through all three models as follow:



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

Refinement

For this project i have worked on fine tuning parameters of LSTM to get better predictions. I did the improvement by testing and analysing each parameter and then selecting 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

Thus improved my mean squared error, for testing sets, from **0.01153170 MSE** to **0.00093063 MSE**.

The predicted plot difference can be seen as follows:



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

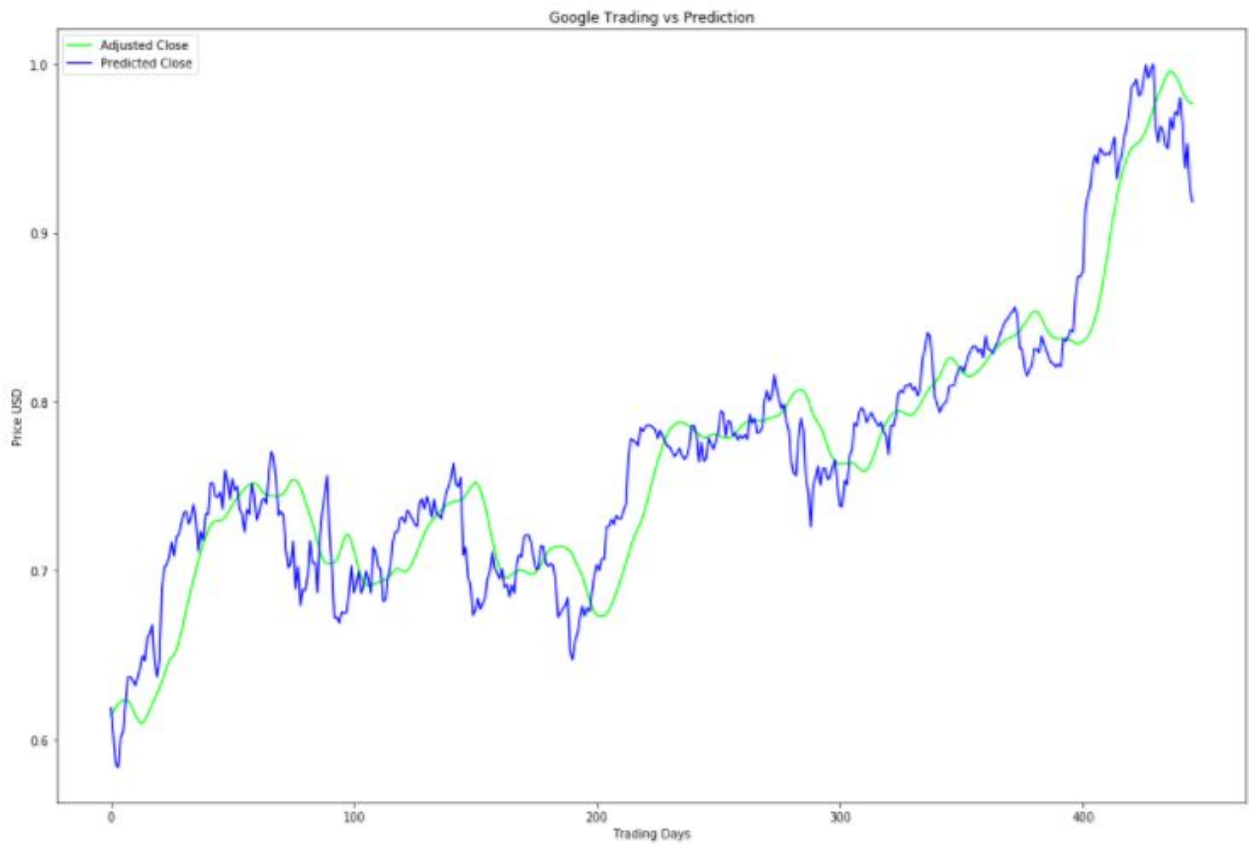


Fig : Plot For Adjusted Close and Predicted Close Prices for improved LSTM model

RESULT

Model Evaluation and Validation

With each model i have refined and fined tune my predictions and have reduced mean squared error significantly.

- For my first model using linear regression model:
 - **Train Score: 0.1852 MSE (0.4303 RMSE)**
 - **Test Score: 0.08133781 MSE (0.28519784 RMSE)**

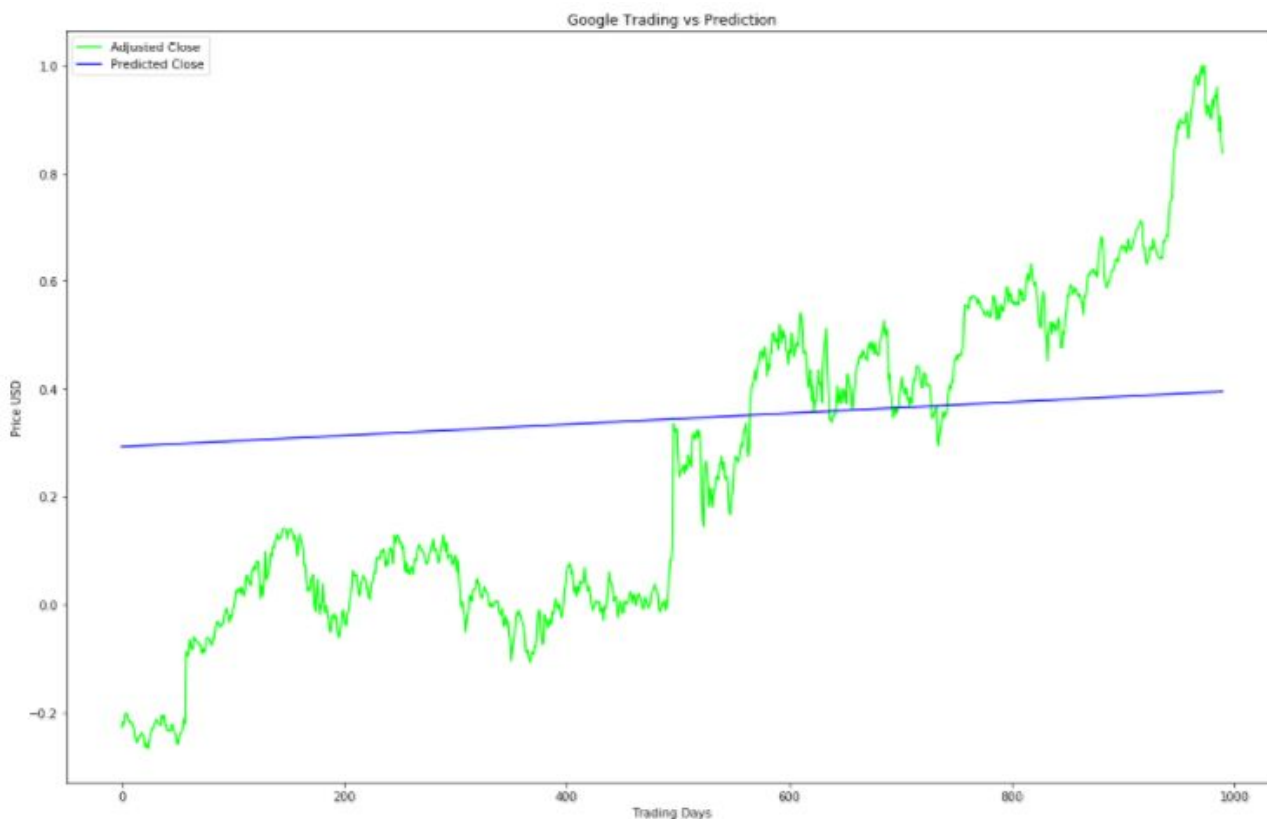


Fig: Plot of Linear Regression Model

- For my second model using basic Long-Short Term memory model:

STOCK PRICE PREDICTION

Using LSTM

RAJAT DHYANI

MLND Capstone Project

- Train Score: 0.00089497 MSE (0.02991610 RMSE)
- Test Score: 0.01153170 MSE (0.10738577 RMSE)



Fig: Plot of basic Long-Short Term Memory model

- For my third and final model, using improved Long-Short Term memory model:
 - **Train Score: 0.00032478 MSE (0.01802172 RMSE)**
 - **Test Score: 0.00093063 MSE (0.03050625 RMSE)**

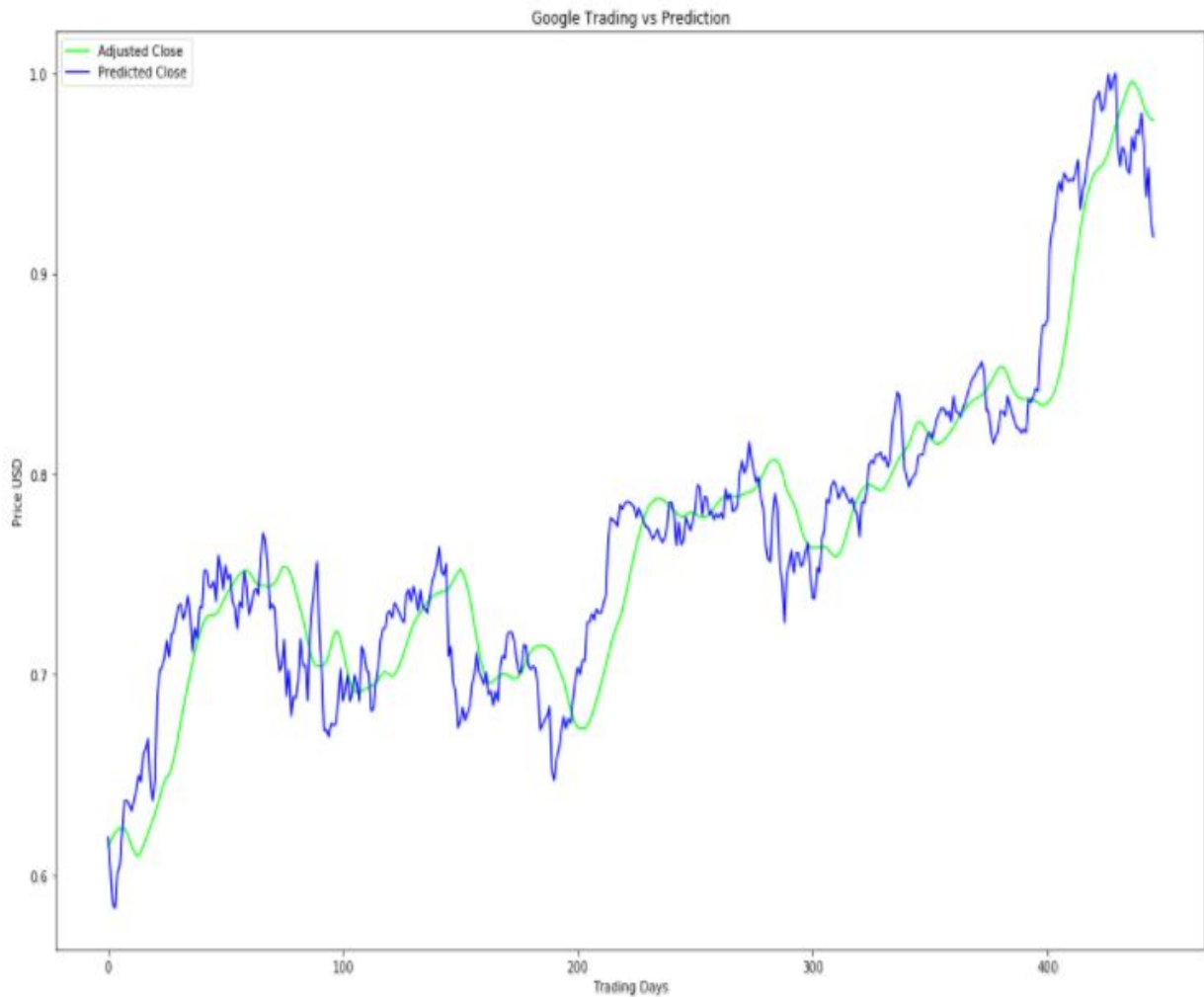


Fig: Plot of Improved Long-Short Term Memory Model

Justification

Comparing the benchmark model - Linear Regression to the final improved LSTM model, the Mean Squared Error improvement ranges from **0.08133781 MSE (0.28519784 RMSE)** [Linear Regression Model] to **0.00093063 MSE (0.03050625 RMSE)** [Improved LSTM]. This significant decrease in error rate clearly shows that my final model have surpassed the basic and benchmark model.

Also the Average Delta Price between actual and predicted Adjusted Closing Price values was:

Delta Price: 0.000931 - RMSE * Adjusted Close Range

Which is less than one cent :)

CONCLUSION

Free-Form Visualization

I have already discussed all the important features of the datasets and their visualisation in one of the above sections. But to conclude my report i would choose my final model visualization, which is improved version of LSTM by fine tuning parameters. As i was very impressed on seeing how close i have gotten to the actual data, with a mean square error of just 0.0009. It was an 'Aha!' moment for me as i had to poke around a lot (really ALOT !! :P). But it was fun working on this project.



Fig: Plot of Improved Long-Short Term Memory Model

Reflection

To recap, the process undertaken in this project:

- Set Up Infrastructure
 - iPython Notebook
 - Incorporate required Libraries (Keras, Tensor flow, Pandas, Matplotlib, Sklearn, Numpy)
 - Git project organization
- Prepare Dataset
 - Incorporate data of Alphabet Inc company
 - Process the requested data into Pandas Dataframe
 - Develop function for normalizing data
 - Dataset used with a 80/20 split on training and test data across all models
- Develop Benchmark Model
 - Set up basic Linear Regression model with Scikit-Learn
 - Calibrate parameters
- Develop Basic LSTM Model
 - Set up basic LSTM model with Keras utilizing parameters from Benchmark Model
- Improve LSTM Model
 - Develop, document, and compare results using additional labels for the LSMT model 5. Document and Visualize Results
- Plot Actual, Benchmark Predicted Values, and LSTM Predicted Values per time series
- Analyze and describe results for report.

I started this project with the hope to learn a completely new algorithm, i.e, Long-Short Term Memory and also to explore a real time series data sets. The final model really exceeded my expectation and have worked remarkably well. I am greatly satisfied with these results.

The major problem i faced during the implementation of project was exploring the data. It was toughest task. To convert data from raw format to preprocess data and then to split them training and test data. All of this requires a great deal of patience and very precise approach. Also i had to work around a lot to successfully use the data for 2 models, i.e, Linear Regression and Long-Short Term Memory, as both of them have

different inputs sizes. I read many research papers to get this final model right and i think it was all worth it :)

Improvement

Before starting my journey as Machine Learning Nanodegree Graduate i had no prior experience in python. In the beginning of this course everything i had to with python, i google it. But now i am not only comfortable with python, i have explored many libraries along the ways and can use them like a pro. This is all because of highly interactive videos and forum provided by Udacity. I am really happy and satisfied taking up this course.

And as there is scope of improvement in each individual so is the case with this project. This project though predicts closing prices with very minimum Mean Squared Error, still there are many things that are lagging in this project. Two of most important things are :

- There is no user interaction or interface provided in this project. A UI can be provided where user can check the value for future dates.
- The stocks used for this project are only of Apple Inc, we can surely add more S&P 500 in the list so as to make this project more comprehensive.

I would definitely like to add these improvement to this project in future.