Stock Price Prediction and Forecasting

SURYADEV SINGH RATHORE
197004457

Table of Contents

INTRODUCTION

Predicting the stock market price has always been very difficult. For the years analyst are trying to create a model that can predict the future price correctly. The analyst has used many different times of models to reach their goal, but the stock market price is correct prediction is still a mystery to most. Here I am trying to create the model to understand the machine learning concept to get one step closer to predicting the correct stock market price. Based on Closing stock market price for the Alphabet co. (GOOG). The prediction for future 30 days was performed. Deep learning model called stacked Long short term. The recent data was taken to achieve the best result.

PROBLEM DEFINATION

There are too many variables that influence stock prices to be taken into account. It is difficult to construct a detailed model based on all of these variables, and one of the key reasons is that most of the variables are not known beforehand: even though certain stock market events have occurred in the past, you never know what else is going to happen in the future.

TARGET AUDIENCE

The Targeted audience is the people who wants to invest their money in stock market and the people who has the interest in understanding of deep learning models. The people who invest their money in the stock market has to take their next steps very cautiously. In order to help them understand or to give them sneak for the probable future price this analysis was performed.

LIABRARIES USED FOR DATA PROCESSING

Lots of libraries were used while working on the project. Here are some libraries mentioned below which are used in this project: -

KERAS

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

TENSORFLOW

TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow is a symbolic math library based on dataflow and differentiable programming.

PANDAS

It presents a diverse range of utilities, ranging from parsing multiple file formats to converting an entire data table into a NumPy matrix array. This makes pandas a trusted ally in data science and machine learning.

PANDAS DATAREADER

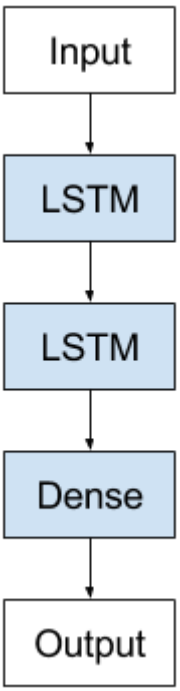
The Pandas datareader is a sub package that allows one to create a data frame from various internet data sources. For this project I am using **Tiingo** to collect Stock Market Data for performing the future prediction analysis.

STOCK COMPANY OVERVIEW

Alphabet Inc. provides online advertising services in the United States, Europe, the Middle East, Africa, the Asia-Pacific, Canada, and Latin America. It offers performance and brand advertising services. The company operates through Google and Other Bets segments. The Google segment offers products, such as Ads, Android, Chrome, Google Cloud, Google Maps, Google Play, Hardware, Search, and YouTube, as well as technical infrastructure. It also offers digital content, cloud services, hardware devices, and other miscellaneous products and services. The Other Bets segment includes businesses, including Access, Calico, CapitalG, GV, Verily, Waymo, and X, as well as Internet and television services. The company has an agreement with Sabre Corporation to develop an artificial intelligence-driven technology platform for travel. Alphabet Inc. was founded in 1998 and is headquartered in Mountain View, California.

STACKED LSTM MODEL EXPLAINATION

The Stacked LSTM is an extension to the LSTM model that has multiple hidden LSTM layers where each layer contains multiple memory cells. A Stacked LSTM architecture can be defined as an LSTM model comprised of multiple LSTM layers. A Stacked LSTM model provides a sequence output rather than a single value output to the LSTM model. Stacking LSTM hidden layers makes the model deeper and more accurate. We can easily create Stacked LSTM models in Keras Python deep learning library. Each LSTMs memory cell requires a 3D input. When an LSTM processes one input sequence of time steps, each memory cell will output a single value for the whole sequence as a 2D array.



COLLECTION OF STOCK DATA

Tiingo

Tiingo is a tracing platform that provides a data api with historical end-of-day prices on equities, mutual funds, and ETFs. I used Tiingo to collect the last 5 years stock market data for the company name **'GOOGLE'**.

Collection of Data

Imported Pandas datareader to create a data frame from Tiingo. Then using the method **`get_data_tiingo`** to call the required data and stored the data frame in a variable name **`df`**. After calling the data from Tiingo, the data was stored in CSV file format.

```
[ ]  import pandas_datareader as pdr
```

```
[ ]  df = pdr.get_data_tiingo('GOOG', api_key='8c20d3dcb50744d14b97326576601d1289f0e07c')
```

```
[ ]  df.to_csv('GOOG.csv')
```

Uploaded the Data set again from the CSV file to avoid using API keys again. And passed the command **`df.head and df.tail`** to check the data set how it looks like.

```
[ ]  df=pd.read_csv('GOOG.csv')
```

```
df.head()
```

	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume	divCash	splitFactor
0	GOOG	2015-12-21 00:00:00+00:00	747.77	750.00	740.00	746.13	1525703	747.77	750.00	740.00	746.13	1525703	0.0	1.0
1	GOOG	2015-12-22 00:00:00+00:00	750.00	754.85	745.53	751.65	1365520	750.00	754.85	745.53	751.65	1365520	0.0	1.0
2	GOOG	2015-12-23 00:00:00+00:00	750.31	754.21	744.00	753.47	1566726	750.31	754.21	744.00	753.47	1566726	0.0	1.0
3	GOOG	2015-12-24 00:00:00+00:00	748.40	751.35	746.62	749.55	527223	748.40	751.35	746.62	749.55	527223	0.0	1.0
4	GOOG	2015-12-28 00:00:00+00:00	762.51	762.99	749.52	752.92	1515716	762.51	762.99	749.52	752.92	1515716	0.0	1.0

```
df.tail()
```

	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume	divCash	splitFactor
1253	GOOG	2020-12-11 00:00:00+00:00	1781.77	1784.4500	1760.000	1763.06	1220651	1781.77	1784.4500	1760.000	1763.06	1220651	0.0	1.0
1254	GOOG	2020-12-14 00:00:00+00:00	1760.06	1797.3900	1757.215	1775.00	1600173	1760.06	1797.3900	1757.215	1775.00	1600173	0.0	1.0
1255	GOOG	2020-12-15 00:00:00+00:00	1767.77	1771.4200	1749.950	1764.42	1482261	1767.77	1771.4200	1749.950	1764.42	1482261	0.0	1.0
1256	GOOG	2020-12-16 00:00:00+00:00	1763.00	1773.0000	1756.080	1772.88	1513543	1763.00	1773.0000	1756.080	1772.88	1513543	0.0	1.0
1257	GOOG	2020-12-17 00:00:00+00:00	1747.90	1771.7836	1738.660	1768.51	1518074	1747.90	1771.7836	1738.660	1768.51	1518074	0.0	1.0

Here, the prediction is performed on the Closing Price of the stock. So, picking the **'close'** column for the processing. And the shape here tells that the data contains **1258** records.

```
[ ]  df1=df.reset_index()['close']
```

```
[ ]  df1.shape
```

```
(1258,)
```

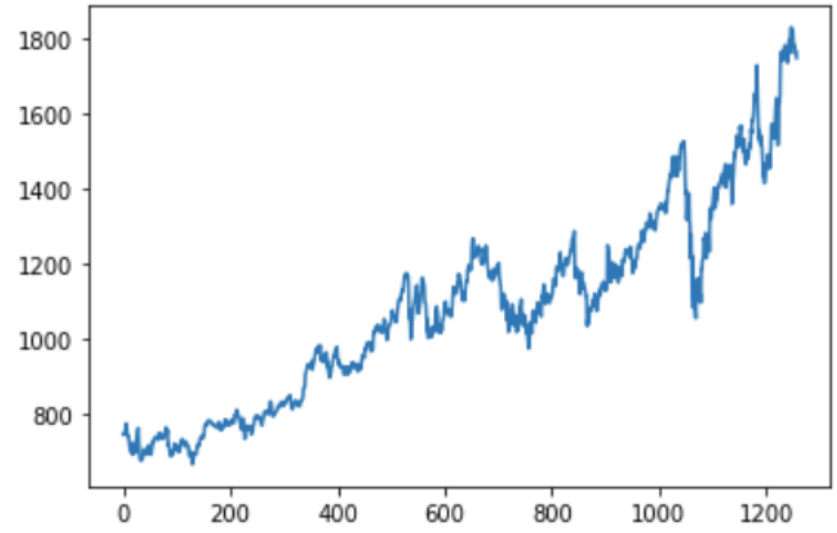
As shown on the data set above, the data set contains the data from the year 2015-12-21 to 2020-12-17.

Plotting the Data

The Closing Stock price for the Google Co. is plotted for last five years by importing **`matplotlib.pyplot.`**

```
import matplotlib.pyplot as plt
plt.plot(df1)
```

```
[<matplotlib.lines.Line2D at 0x7fef8726df28>]
```



This graph explains the closing stock price movement for the last 5 years.

Long Short-Term Memory are very sensitive to the scale of the data. In this data the closing stock price value is not in the form of scale, so the value should be transformed in the form of scale. In this scenario we are going to take min max scale-up where we will be transforming our values between 0 to 1. For this transformation I am going to import **NumPy, MinMaxScaler** and from **sklearn.preprocessing.**

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))
```

We can now see that the data is transformed into the scale.

```
print(df1)
```

```
[[0.06855906]
 [0.07048192]
 [0.07074923]
 ...
 [0.94807412]
 [0.94396109]
 [0.93094082]]
```

PREPROCESS THE DATA

Splitting the data between Train and Test data

A significant aspect of evaluating data mining models is to split data into training and testing sets. Usually, much of the data is used for training when you split a data set into a training set and a testing set, and a smaller portion of the data is used for testing. Analysis Services sample the data randomly to help ensure that the sets of testing and training are similar.

You test the model by making predictions against the test set after a model has been processed by using the training set. Since the data already contains known values for the attribute you want to predict in the testing collection, it is easy to decide if the guesses of the model are accurate.

In the data set. There is total 1258 records. Splitting the total number of records in Training and Testing data.
Training Data = 65% of total data
Testing Data = 35% of total data

```
### Spliting Data set into Train and Test split
training_size=int(len(df1)*0.65)
test_size=len(df1)-training_size
train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]
```

Checking the training and testing size:
Training size - 817
Test size – 441

Preprocessing the data

While preprocessing the data here, the dataset was transformed from array values to dataset matrix and then it was reshaped to time step. It is always a good practice to have larger time step value. In this we have taken time step value = 100.

```
import numpy
#convert an array values into dataset matrix
def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step),0] ###i=0, 0,1,2,3
        dataX.append(a)
        dataY.append(dataset[i + time_step,0])
    return numpy.array(dataX), numpy.array(dataY)
```

```
###reshape into X=t,t+1, t+2, t+3 and Y=t+4
time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)
```

Here we can see how the training values looks like. We can observe here that while processing the data here the data value is shifted in the size of 100.

```
print(X_train)
```

```
[[0.06855906 0.07048192 0.07074923 ... 0.04055254 0.03884525 0.03670682]
 [0.07048192 0.07074923 0.06910229 ... 0.03884525 0.03670682 0.04158727]
 [0.07074923 0.06910229 0.08126892 ... 0.03670682 0.04158727 0.03274038]
 ...
 [0.36922387 0.37327654 0.37545808 ... 0.45274331 0.44604348 0.4451036 ]
 [0.37327654 0.37545808 0.32977503 ... 0.44604348 0.4451036  0.44493115]
 [0.37545808 0.32977503 0.36845645 ... 0.4451036  0.44493115 0.45751166]]
```

CREATING A STACKED LSTM MODEL

Long Short-term model required the input to be in three dimensions. So, it is required to reshape the input in three dimensions.

```
# Reshape input to be [samples, time steps, features] which is required for LSTM
X_train=X_train.reshape(X_train.shape[0],X_train.shape[1], 1)
X_test=X_test.reshape(X_test.shape[0],X_test.shape[1], 1)
```

Using the three libraries required to create LSTM model: Sequential, Dense and LSTM. After the importing libraries now, the model is created. As we can observe that this is the stacked LSTM model as one LSTM is used after another.

```
### Creating the LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

```
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

After executing the model, we can see the summary of the model.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100, 50)           10400
_________________________________________________________________
lstm_1 (LSTM)                (None, 100, 50)           20200
_________________________________________________________________
lstm_2 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
```

PREDICTING AND PLOTTING THE TEST DATA

Predicting the X train and x test data and checking performance metrics.

```
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

Transforming back to the original value for plotting the data

```
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

Calculating Root mean squared error performance metrics for train and test data.

Importing the math library, sklearn.metrics and mean squared error for performance metrics.

We have already trained the data set on training dataset so based on that we can say that output is good enough. Now, by comparing trained data performance metrics to test data performance metrics, we can see that the difference between them is not large enough. It states that the LSTM model we built is working well.

```
## Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

```
991.3536824855819
```

```
## Test data RMSE
math.sqrt(mean_squared_error(ytest,test_predict))
```

```
1421.1300848848146
```

Plotting

The data which we got from Train and Test predicting are plotted over the original data.

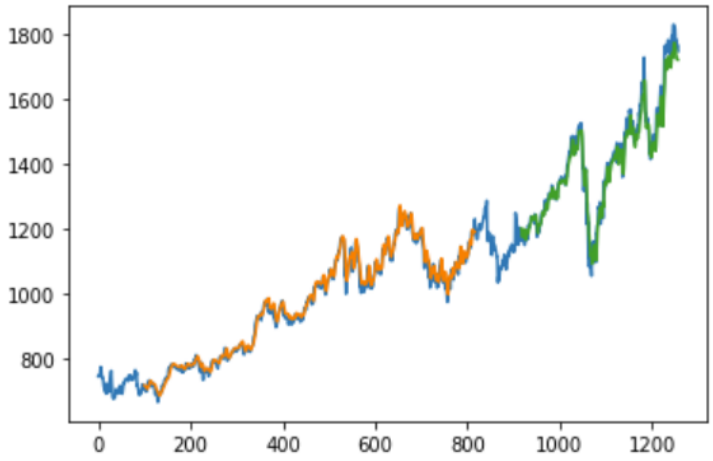
The **Blue** color output is complete data set

The **Yellow** Color output is Training data set

The **Green** color output is Test data set

```
## Plotting
# shift train predictions for plotting
look_back=100
trainPredictPlot = numpy.empty_like(df1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
#shift test predictions for plotting
testPredictPlot = numpy.empty_like(df1)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1, :] = test_predict
#plot baseline and prediction
plt.plot(scaler.inverse_transform(df1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show
```

```
<function matplotlib.pyplot.show>
```



PREDICTING THE FUTURE DATA AND PLOTTING THE OUTPUT

Reshaping the previous hundred data for the prediction

```
x_input=test_data[341:].reshape(1,-1)
x_input.shape
```

```
(1, 100)
```

Prediction for next 30 days

Passing the result data to the model to perform prediction. Before performing the predictions steps, we reshape the data set. Given below is the logic used for the prediction and reshaping the dataset.

```
from numpy import array

lst_output=[]
n_steps=100
i=0
while(i<30):

    if (len(temp_input)>100):
        #print(temp_input)
        x_input=np.array(temp_input[1:])
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        #print(x_input)
        yhat = model.predict(x_input, verbose=0)
        print("{} day input {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print temp_input
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps, 1))
        yhat = model.predict(x_input, verbose=0)
        print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        print(len(temp_input))
        lst_output.extend(yhat.tolist())
        i=i+1

print(lst_output)
```

Plotting the Predicted Data

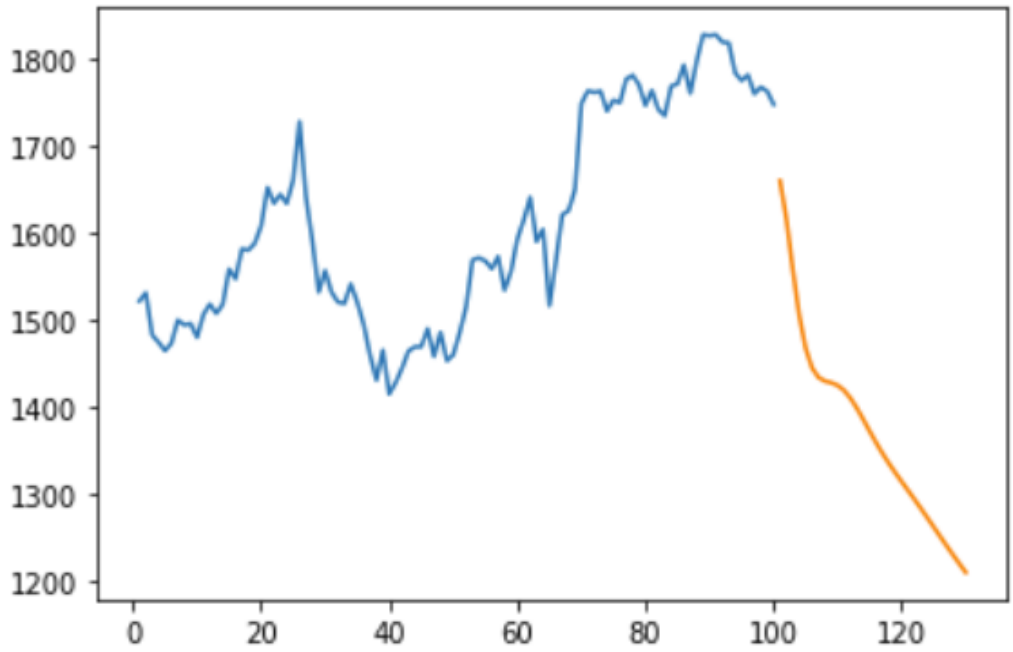
Putting the 100 indexes inside the **day_new**, and putting 30 indexes inside the **day_pred** for the future predict.

```
day_new=np.arange(1,101)
day_pred=np.arange(101,131)
```

Displaying the previous 100 days data and then displaying the predicted 30 days data.

```
plt.plot(day_new,scaler.inverse_transform(df1[1158:]))
plt.plot(day_pred,scaler.inverse_transform(lst_output))
```

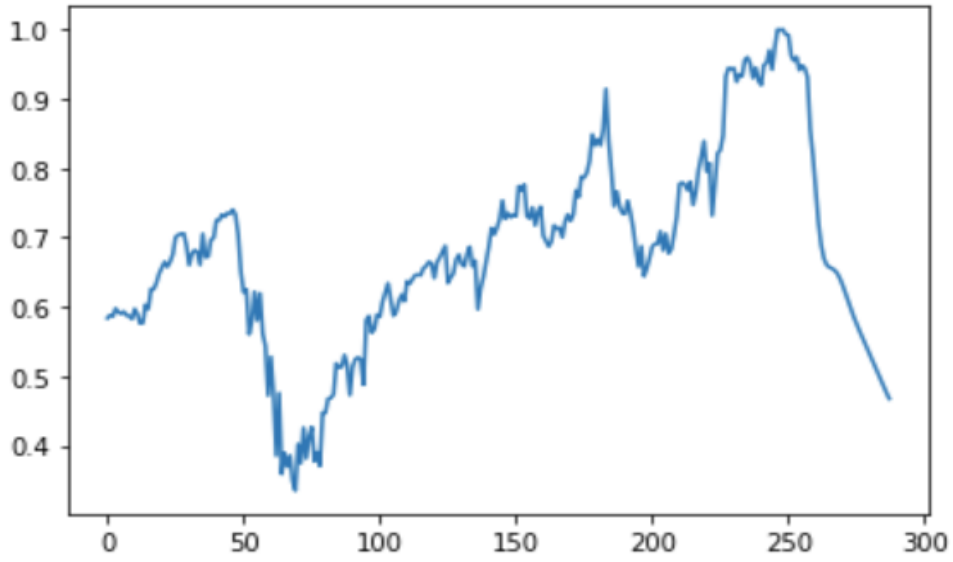
```
[<matplotlib.lines.Line2D at 0x7f2e4f637240>]
```



Complete Output plot

```
df3=df1.tolist()
df3.extend(lst_output)
plt.plot(df3[1000:])
```

```
[<matplotlib.lines.Line2D at 0x7f2e4f5e9c50>]
```



CONCLUSION

The Collection of Data and preprocessing the data went well. Preprocessing the data gave root mean squared error, which was very close between Training and Testing data. This implies that the Long Short-Term Model created is working well. The LSTM model gave good prediction for 30 days. We can observe in the plot graph that the prediction line is steeped downwards. We can understand from this that predicting the stock market price with only one stacked LSTM model is not a good idea as it as limitations. While working on time series data or predicting the stock market price there are lots of other factors and parameters to consider before predicting the future. For instance, Stock Market price variation depends on many factors such as stock issuing company changing its policies, Political climates, demand and supply, fluctuation in the economy. etc.

References: –

https://en.wikipedia.org/wiki/Keras

https://en.wikipedia.org/wiki/TensorFlow

https://riptutorial.com/pandas/topic/1912/pandas-datareader

https://www.educative.io/edpresso/what-is-pandas-in-python

https://docs.microsoft.com/en-us/analysis-services/data-mining/training-and-testing-data-sets?view=asallproducts-allversions

https://finance.yahoo.com/quote/GOOG/profile?p=GOOG

https://www.youtube.com/watch?v=H6du_pfuznE&list=WL&index=10&t=1877s&ab_channel=KrishNaik

https://www.quora.com/Why-is-the-stock-market-so-difficult-to-predict

https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.thebalance.com%2Fwhy-do-stock-prices-fluctuate-356347&psig=AOvVaw1EFO-nGP4rwLJmDG_l31pX&ust=1608402909045000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCKD35b-W2O0CFQAAAAAdAAAAABAK