LetsGrowMore- Data Science Intern

Author: Soundarya G

TASK 2: Stock Market Prediction And Forecasting Using Stacked LSTM

Table of contents

INTRODUCTION

Stock values is very valuable but extremely hard to predict correctly for any human being on their own. This task seeks to solve the problem of Stock Prices Prediction by stacked Long-Short Term Memory (LSTM), to predict future stock values.

IMPORT THE NECESSARY LIBRARIES

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

LOADING DATASET

```
url = 'https://raw.githubusercontent.com/mwitiderrick/stockprice/master/NSE-TATAGLOBAL.csv'
dataset_train = pd.read_csv(url)
training_set = dataset_train.iloc[:, 1:2].values
```

```
dataset_train.head()
```

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	2018-09-28	234.05	235.95	230.20	233.50	233.75	3069914	7162.35
1	2018-09-27	234.55	236.80	231.10	233.80	233.25	5082859	11859.95
2	2018-09-26	240.00	240.00	232.50	235.00	234.25	2240909	5248.60
3	2018-09-25	233.30	236.75	232.00	236.25	236.10	2349368	5503.90
4	2018-09-24	233.55	239.20	230.75	234.00	233.30	3423509	7999.55

```
data_close = dataset_train['Close']
```

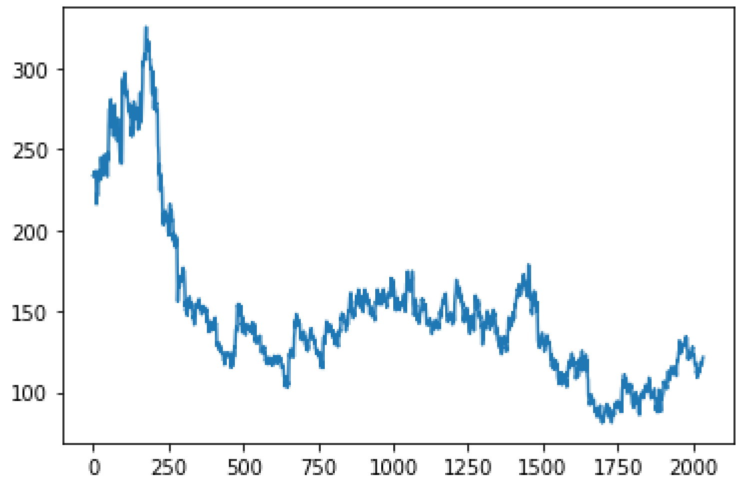
```
data_close
```

```
0       233.75
1       233.25
2       234.25
3       236.10
```

```
4       233.30
         ...
2030    118.65
2031    117.60
2032    120.65
2033    120.90
2034    121.55
Name: Close, Length: 2035, dtype: float64
```

```
plt.plot(data_close)
```

```
[<matplotlib.lines.Line2D at 0x7f145462f7d0>]
```



Since LSTM are sensitive to the scale of the data, so we apply MinMax Scaler to transform our values between 0 and 1

PRE-PROCESSING

Data Normalization

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

```
data_close.shape
```

```
(2035, 1)
```

```
print(data_close)
```

```
[[0.62418301]
 [0.62214052]
 [0.62622549]
 ...
 [0.1621732 ]
 [0.16319444]
 [0.16584967]]
```

Splitting the dataset into Train and Test sets

```
training_size = int(len(data_close) * 0.75)
test_size = len(data_close) - training_size
train_data, test_data = data_close[0:training_size,:], data_close[training_size:len(data_close),:1]
```

```
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]
        dataX.append(a)
        dataY.append(dataset[i+time_step, 0])
    return np.array(dataX), np.array(dataY)
```

```
time_step = 100
x_train, y_train = create_dataset(train_data, time_step)
x_test, y_test = create_dataset(test_data, time_step)
```

```
#Reshape the input to be [samples, time steps, features] which is the requirement of 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)
```

MODEL BUILDING

```
#pip install keras
```

```
#Create the LSTM Model
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import Dense
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')
```

```
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
_________________________________________________________________
```


```
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
_________________________________________________________________
```


```
model.fit(x_train, y_train, validation_data = (x_test, y_test), epochs = 100, batch_size = 64, verbose = 1)
```

```
Epoch 1/100
23/23 [==============================] - 12s 257ms/step - loss: 0.0253 - val_loss: 0.0087
Epoch 2/100
23/23 [==============================] - 5s 204ms/step - loss: 0.0037 - val_loss: 9.4282e-04
Epoch 3/100
23/23 [==============================] - 5s 201ms/step - loss: 0.0016 - val_loss: 0.0013
Epoch 4/100
23/23 [==============================] - 5s 204ms/step - loss: 0.0015 - val_loss: 0.0014
Epoch 5/100
23/23 [==============================] - 5s 203ms/step - loss: 0.0014 - val_loss: 0.0010
Epoch 6/100
23/23 [==============================] - 5s 203ms/step - loss: 0.0013 - val_loss: 0.0011
Epoch 7/100
23/23 [==============================] - 5s 204ms/step - loss: 0.0012 - val_loss: 0.0011
Epoch 8/100
23/23 [==============================] - 5s 202ms/step - loss: 0.0012 - val_loss: 9.8614e-04
Epoch 9/100
23/23 [==============================] - 5s 204ms/step - loss: 0.0011 - val_loss: 0.0011
Epoch 10/100
23/23 [==============================] - 5s 205ms/step - loss: 0.0011 - val_loss: 8.5449e-04
Epoch 11/100
23/23 [==============================] - 5s 204ms/step - loss: 0.0012 - val_loss: 8.5015e-04
Epoch 12/100
23/23 [==============================] - 5s 201ms/step - loss: 0.0010 - val_loss: 0.0011
```

```
Epoch 13/100
23/23 [==============================] - 5s 206ms/step - loss: 9.8018e-04 - val_loss: 0.0011
Epoch 14/100
23/23 [==============================] - 5s 204ms/step - loss: 8.8726e-04 - val_loss: 8.1136e-04
Epoch 15/100
23/23 [==============================] - 5s 203ms/step - loss: 8.6354e-04 - val_loss: 9.2649e-04
Epoch 16/100
23/23 [==============================] - 5s 205ms/step - loss: 8.4866e-04 - val_loss: 7.4071e-04
Epoch 17/100
23/23 [==============================] - 5s 206ms/step - loss: 7.9973e-04 - val_loss: 7.2955e-04
Epoch 18/100
23/23 [==============================] - 5s 204ms/step - loss: 7.8569e-04 - val_loss: 8.8782e-04
Epoch 19/100
23/23 [==============================] - 5s 203ms/step - loss: 8.0511e-04 - val_loss: 0.0011
Epoch 20/100
23/23 [==============================] - 5s 206ms/step - loss: 9.5764e-04 - val_loss: 6.7732e-04
Epoch 21/100
23/23 [==============================] - 5s 208ms/step - loss: 8.3496e-04 - val_loss: 8.5265e-04
Epoch 22/100
23/23 [==============================] - 5s 207ms/step - loss: 7.5034e-04 - val_loss: 8.8869e-04
Epoch 23/100
23/23 [==============================] - 5s 205ms/step - loss: 7.0545e-04 - val_loss: 8.5740e-04
Epoch 24/100
23/23 [==============================] - 5s 204ms/step - loss: 7.3908e-04 - val_loss: 8.8193e-04
Epoch 25/100
23/23 [==============================] - 5s 204ms/step - loss: 6.7699e-04 - val_loss: 9.8909e-04
Epoch 26/100
23/23 [==============================] - 5s 205ms/step - loss: 6.5638e-04 - val_loss: 6.8022e-04
Epoch 27/100
23/23 [==============================] - 5s 205ms/step - loss: 6.5928e-04 - val_loss: 6.7229e-04
Epoch 28/100
23/23 [==============================] - 5s 204ms/step - loss: 6.4288e-04 - val_loss: 7.9469e-04
Epoch 29/100
23/23 [==============================] - 5s 205ms/step - loss: 6.0590e-04 - val_loss: 8.5250e-04
Epoch 30/100
```

```
#Lets predict and check performance metrics
train_predict = model.predict(x_train)
test_predict = model.predict(x_test)
```

```
#Transform back to original form
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
```

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

```
162.54456978514486
```

```
#Test Data RMSE
math.sqrt(mean_squared_error(y_test, test_predict))
```

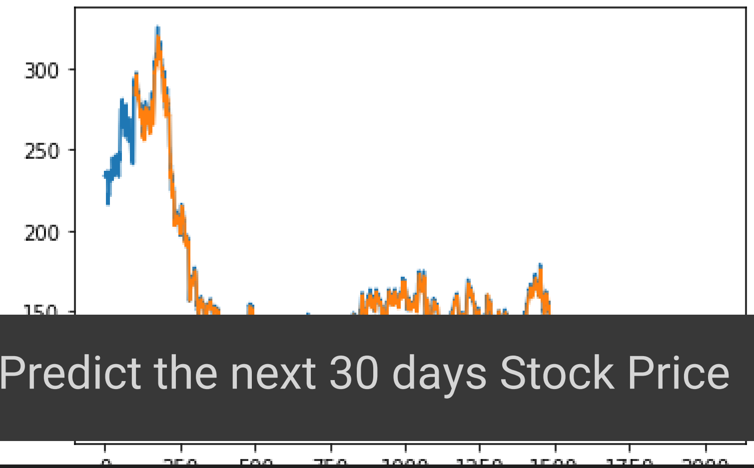
```
106.22748841248169
```

```
#Plotting

#Shift train prediction for plotting
look_back = 100
trainPredictPlot = np.empty_like(data_close)
trainPredictPlot[:,:] = np.nan
trainPredictPlot[look_back:len(train_predict) + look_back, :] = train_predict

#Shift test prediction for plotting
testPredictPlot = np.empty_like(data_close)
testPredictPlot[:,:] = np.nan
testPredictPlot[len(train_predict) + (look_back * 2)+1:len(data_close) - 1, :] = test_predict

#Plot baseline and predictions
plt.plot(scaler.inverse_transform(data_close))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



Predict the next 30 days Stock Price

```
len(test_data), x_test.shape
```

```
(509, (408, 100, 1))
```

```
x_input = test_data[409:].reshape(1,-1)
x_input.shape
```

```
(1, 100)
```

```
temp_input = list(x_input)
temp_input = temp_input[0].tolist()
```

```
lst_output=[]
n_steps=100
nextNumberOfDays = 30
i=0

while(i<nextNumberOfDays):

    if(len(temp_input)>100):
        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))
        yhat = model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        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)
```

```
[0.16615]
101
1 day input [0.13848039 0.14011438 0.13888889 0.13541667 0.14011438 0.1380719
 0.13071895 0.13071895 0.12867647 0.11846405 0.14644608 0.14808007
 0.15910948 0.15992647 0.15788399 0.16441993 0.17892157 0.17933007
 0.19260621 0.20812908 0.18974673 0.18055556 0.18239379 0.17708333
 0.17810458 0.18055556 0.17810458 0.17851307 0.19607843 0.18913399
 0.18954248 0.19403595 0.19444444 0.20200163 0.19771242 0.19934641
 0.19873366 0.1997549  0.2128268  0.21568627 0.20445261 0.21772876
 0.21098856 0.21425654 0.19750817 0.18811275 0.17851307 0.17381536
 0.16033497 0.16564542 0.17116013 0.17422386 0.18035131 0.17401961
 0.16278595 0.16973039 0.17810458 0.17034314 0.16830065 0.17279412
 0.17544935 0.18382353 0.19138072 0.18913399 0.19097222 0.17238562
 0.16830065 0.1693219  0.17177288 0.16156046 0.14971405 0.1503268
 0.15196078 0.14726307 0.14501634 0.14603758 0.12479575 0.13112745
 0.11397059 0.1190768  0.12377451 0.13562092 0.12908497 0.13459967
 0.12806373 0.13031046 0.12724673 0.13521242 0.14522059 0.15257353
 0.14848856 0.14338235 0.14562908 0.15236928 0.15400327 0.14971405
 0.1621732  0.16319444 0.16584967 0.16615   ]
1 day output [[0.1682977]]
2 day input [0.14011438 0.13888889 0.13541667 0.14011438 0.1380719  0.13071895
 0.13071895 0.12867647 0.11846405 0.14644608 0.14808007 0.15910948
 0.15992647 0.15788399 0.16441993 0.17892157 0.17933007 0.19260621
 0.20812908 0.18974673 0.18055556 0.18239379 0.17708333 0.17810458
 0.18055556 0.17810458 0.17851307 0.19607843 0.18913399 0.18954248
```

```
 0.19403595 0.19444444 0.20200163 0.19771242 0.19934641 0.19873366
 0.1997549  0.2128268  0.21568627 0.20445261 0.21772876 0.21098856
 0.21425654 0.19750817 0.18811275 0.17851307 0.17381536 0.16033497
 0.16564542 0.17116013 0.17422386 0.18035131 0.17401961 0.16278595
 0.16973039 0.17810458 0.17034314 0.16830065 0.17279412 0.17544935
 0.18382353 0.19138072 0.18913399 0.19097222 0.17238562 0.16830065
 0.1693219  0.17177288 0.16156046 0.14971405 0.1503268  0.15196078
 0.14726307 0.14501634 0.14603758 0.12479575 0.13112745 0.11397059
 0.1190768  0.12377451 0.13562092 0.12908497 0.13459967 0.12806373
 0.13031046 0.12724673 0.13521242 0.14522059 0.15257353 0.14848856
 0.14338235 0.14562908 0.15236928 0.15400327 0.14971405 0.1621732
 0.16319444 0.16584967 0.16615    0.16829769]
2 day output [[0.16965233]]
3 day input [0.13888889 0.13541667 0.14011438 0.1380719  0.13071895 0.13071895
 0.12867647 0.11846405 0.14644608 0.14808007 0.15910948 0.15992647
 0.15788399 0.16441993 0.17892157 0.17933007 0.19260621 0.20812908
 0.18974673 0.18055556 0.18239379 0.17708333 0.17810458 0.18055556
 0.17810458 0.17851307 0.19607843 0.18913399 0.18954248 0.19403595
 0.19444444 0.20200163 0.19771242 0.19934641 0.19873366 0.1997549
 0.2128268  0.21568627 0.20445261 0.21772876 0.21098856 0.21425654
 0.19750817 0.18811275 0.17851307 0.17381536 0.16033497 0.16564542
 0.17116013 0.17422386 0.18035131 0.17401961 0.16278595 0.16973039
 0.17810458 0.17034314 0.16830065 0.17279412 0.17544935 0.18382353
 0.19138072 0.18913399 0.19097222 0.17238562 0.16830065 0.1693219
 0.17177288 0.16156046 0.14971405 0.1503268  0.15196078 0.14726307
 0.14501634 0.14603758 0.12479575 0.13112745 0.11397059 0.1190768
 0.12377451 0.13562092 0.12908497 0.13459967 0.12806373 0.13031046
 0.12724673 0.13521242 0.14522059 0.15257353 0.14848856 0.14338235
 0.14562908 0.15236928 0.15400327 0.14971405 0.1621732  0.16319444
 0.16584967 0.16615    0.16829769 0.16965233]
3 day output [[0.17067352]]
4 day input [0.13541667 0.14011438 0.1380719  0.13071895 0.13071895 0.12867647
 0.11846405 0.14644608 0.14808007 0.15910948 0.15992647 0.15788399
```

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

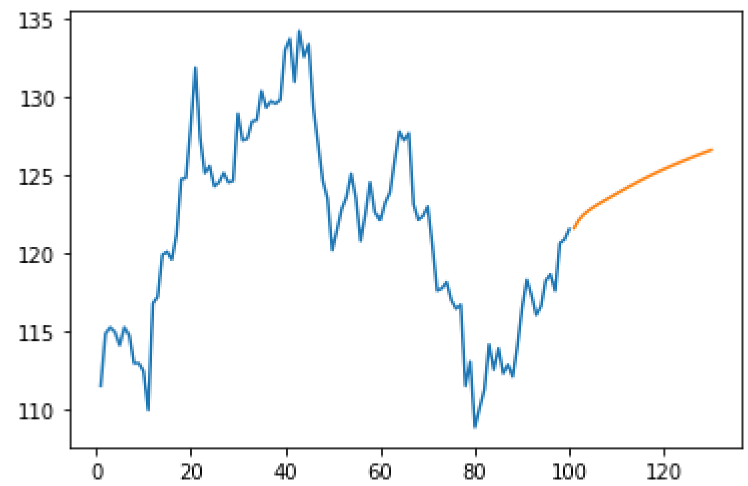
```
df3 = data_close.tolist()
df3.extend(lst_output)
```

```
len(data_close)
```

```
2035
```

```
plt.plot(day_new, scaler.inverse_transform(data_close[1935:]))
plt.plot(day_pred, scaler.inverse_transform(lst_output))
```

```
[<matplotlib.lines.Line2D at 0x7f14454a1850>]
```



THANK YOU