Stock Market Prediction And Forecasting Using Stacked LSTM

Because financial capital markets are unpredictable and non-linear, projecting stock market returns may be difficult. Greater use of artificial intelligence and computers programmable techniques for forecasting market values have turned out to be more accurate in the new era that electricity has brought forth. An accurate forecast of a stock's future price will provide a sizable profit. To make prediction more dependable and straightforward, we have suggested a deep learning-based methodology. the application of the Long Short Term Memory algorithm (LSTM), a sophisticated recurrent neural network. We used multi-layer LSTM networks to estimate the future close prices of stock data and tested the accuracy of our model using stacked Long Short Term Memory. Following the trial, we were able to predict the closing price of the supplied stock for the next 10 days.

```
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import pandas as pd
import numpy as np
import math
warnings.filterwarnings('ignore')
```

Saved successfully!

```
ubusercontent.com/mwitiderrick/stockprice/master/NSE-TATA
```

```
Data.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	7162.3
1	2018-09-27	234.55	236.80	231.10	233.80	233.25	5082859	11859.9
2	2018-09-26	240.00	240.00	232.50	235.00	234.25	2240909	5248.6

```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2035 entries, 0 to 2034
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Date                  2035 non-null   object
 1   Open                  2035 non-null   float64
 2   High                  2035 non-null   float64
 3   Low                   2035 non-null   float64
 4   Last                  2035 non-null   float64
 5   Close                 2035 non-null   float64
 6   Total Trade Quantity  2035 non-null   int64
 7   Turnover (Lacs)       2035 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 127.3+ KB
```

```
Data.describe
```

Saved successfully!

```
<bound method NDFrame.describe of             Date    Open    High     Low    Last
Close  \
0     2018-09-28  234.05  235.95  230.20  233.50  233.75
1     2018-09-27  234.55  236.80  231.10  233.80  233.25
2     2018-09-26  240.00  240.00  232.50  235.00  234.25
3     2018-09-25  233.30  236.75  232.00  236.25  236.10
4     2018-09-24  233.55  239.20  230.75  234.00  233.30
                          .     ...     ...     ...
                          0  112.00  118.80  118.65
                          0  117.10  117.10  117.60
2032  2010-07-23  121.80  121.95  120.25  120.35  120.65
2033  2010-07-22  120.30  122.00  120.25  120.75  120.90
2034  2010-07-21  122.10  123.00  121.05  121.10  121.55

      Total Trade Quantity  Turnover (Lacs)
0                  3069914          7162.35
1                  5082859         11859.95
2                  2240909          5248.60
3                  2349368          5503.90
4                  3423509          7999.55
...                    ...              ...
2030                586100           694.98
2031                658440           780.01
2032                281312           340.31
2033                293312           355.17
2034                658666           803.56
```

```
[2035 rows x 8 columns]>
```

```
Data['Date'] = pd.to_datetime(d['Date'])
Data.dtypes
```

```
Date                      datetime64[ns]
Open                             float64
High                             float64
Low                              float64
Last                             float64
Close                            float64
Total Trade Quantity               int64
Turnover (Lacs)                  float64
dtype: object
```

```
Data = d.sort_values('Date')
```

```
Data.head()
```

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (La
2034	2010-07-21	122.1	123.00	121.05	121.10	121.55	658666	80:
2033	2010-07-22	120.3	122.00	120.25	120.75	120.90	293312	35!
2032	2010-07-23	121.8	121.95	120.25	120.35	120.65	281312	34(
2031	2010-07-26	120.1	121.00	117.10	117.10	117.60	658440	78(
2030	2010-07-27	117.6	119.50	112.00	118.80	118.65	586100	69.


```
plt.figure(figsize = (9,6))
plt.title('Tata Stocks Closing Price')
plt.plot(d['Close'],'g')
plt.xlabel('Date',fontsize=15)
plt.ylabel('Close',fontsize=15)
```

```
Text(0, 0.5, 'Close')
```


```
dcorr = d.corr()
top_corr_features = dcorr.index
plt.figure(figsize=(10,7))
sns.heatmap(d[top_corr_features].corr(), annot=True, cmap="YlGnBu")
```

Saved successfully!

```
<Axes: >
```


```
data_close = d.reset_index()['Close']
data_close.head()
scaler = MinMaxScaler(feature_range = (0, 1))
data_close = scaler.fit_transform(np.array(data_close).reshape(-1, 1))
```

```
train_size = int(len(data_close)*0.70)
test_size = len(data_close) - train_size
train, test = data_close[0 : train_size, :], data_close[train_size : len(data_close), :1]
```

```
def create_matrix(ds, time_step=1):
    dataX, dataY = [], []
    for i in range(len(ds)-time_step-1):
        a = ds[i:(i+time_step),0]
        dataX.append(a)
        dataY.append(ds[i+time_step,0])
    return np.array(dataX), np.array(dataY)
```

```
step=100
X_train, y_train = create_matrix(train, step)
X_test, y_test = create_matrix(test, step)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(1323, 100) (1323,)
(510, 100) (510,)
```

```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
[0], X_test.shape[1], 1)
```

Saved successfully!

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

```
m.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
_________________________________________________________________
```

```
history = m.fit(X_train, y_train, validation_split=0.1, epochs=77, batch_size=64, verbose=1,
```

Saved successfully!

```
19/19 [==============================] - 4s 209ms/step - loss: 3.7774e-04 - val_loss:
Epoch 70/77
19/19 [==============================] - 4s 204ms/step - loss: 5.0931e-04 - val_loss:
Epoch 71/77
19/19 [==============================] - 5s 289ms/step - loss: 4.2643e-04 - val_loss:
Epoch 72/77
19/19 [==============================] - 4s 209ms/step - loss: 3.2023e-04 - val_loss:
Epoch 73/77
19/19 [==============================] - 4s 211ms/step - loss: 3.1877e-04 - val_loss:
Epoch 74/77
19/19 [==============================] - 5s 288ms/step - loss: 3.5898e-04 - val_loss:
Epoch 75/77
19/19 [==============================] - 4s 204ms/step - loss: 3.5128e-04 - val_loss:
Epoch 76/77
19/19 [==============================] - 4s 208ms/step - loss: 3.4231e-04 - val_loss:
Epoch 77/77
19/19 [==============================] - 5s 286ms/step - loss: 2.9222e-04 - val_loss:
```

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

```
42/42 [==============================] - 3s 46ms/step
16/16 [==============================] - 1s 46ms/step
```

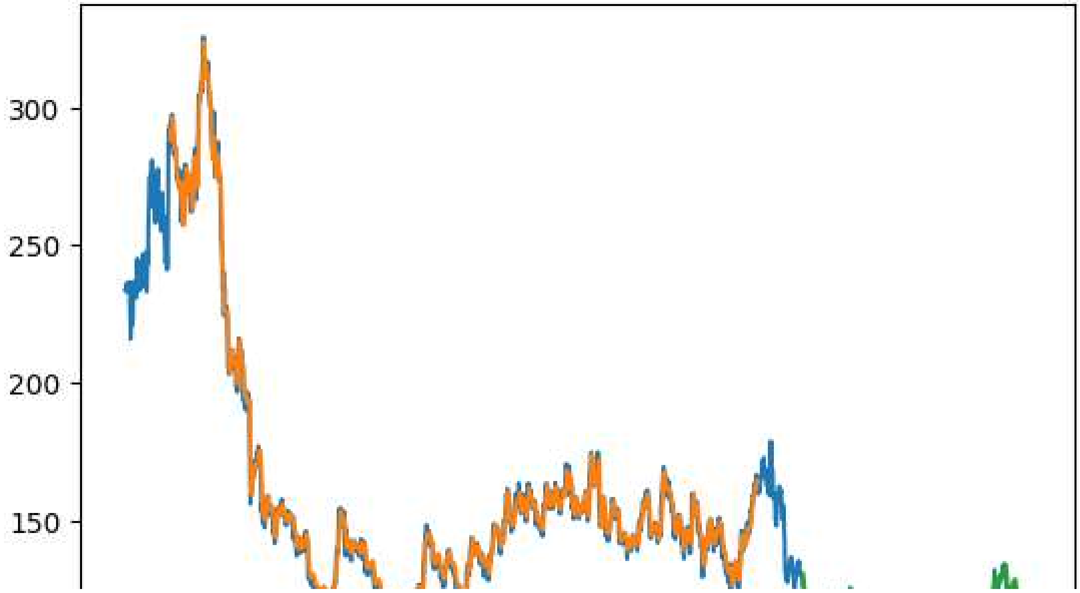
```
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
```

```
math.sqrt(mean_squared_error(y_train, train_predict))
math.sqrt(mean_squared_error(y_test,test_predict))
```

```
109.11172926936734
```

```
look_back = 100
                                    ike(data_close)
                                    an
train_num_pyredict_plot[look_back : len(train_predict) + look_back, :] = train_pred
test_predict_plot = np.empty_like(data_close)
test_predict_plot[:, :] = np.nan
test_predict_plot[len(train_predict) + (look_back * 2) + 1 : len(data_close) - 1, :
plt.plot(scaler.inverse_transform(data_close))
plt.plot(train_num_pyredict_plot)
plt.plot(test_predict_plot)
plt.show()
```

Saved successfully!

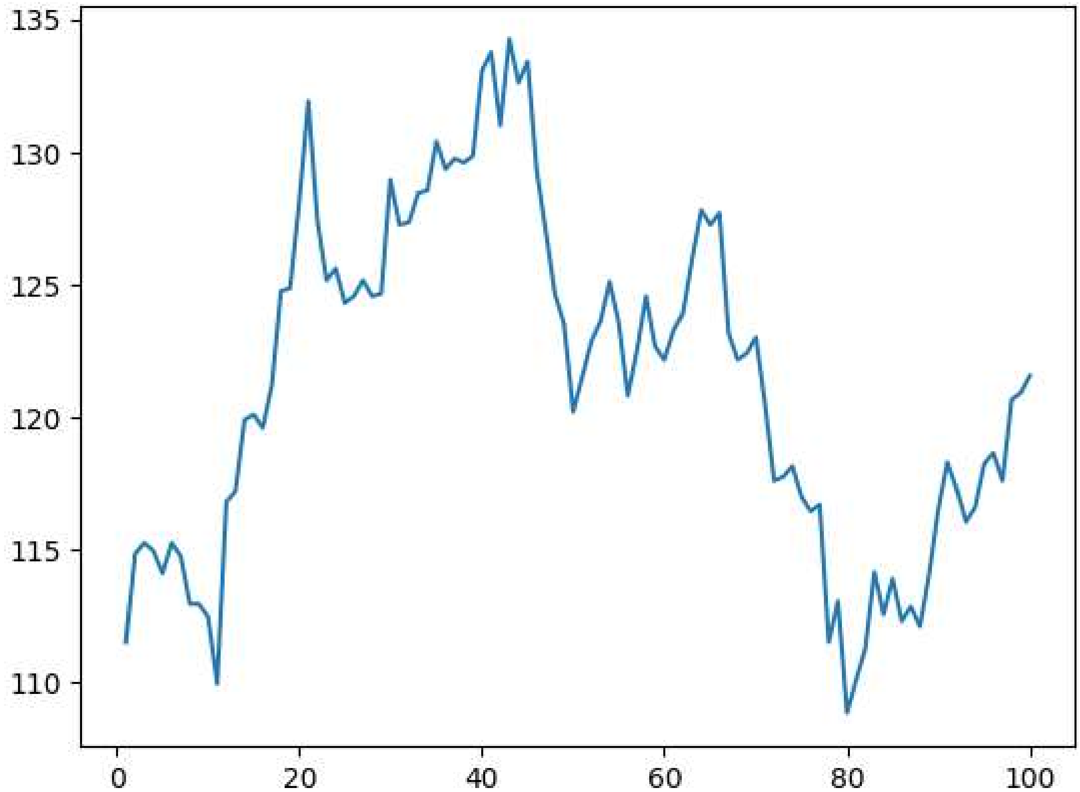



```
x_inum_pyut=test[307:].reshape(1, -1)
x_inum_pyut.shape
temp_inum_pyut = list(x_inum_pyut)
temp_inum_pyut = temp_inum_pyut[0].tolist()
temp_inum_pyut = list(x_inum_pyut)
temp_inum_pyut = temp_inum_pyut[0].tolist()


day_new = np.arange(1, 101)
day_pred = np.arange(101, 131)
plt.plot(day_new, scaler.inverse_transform(data_close[1935 : ]))
```

Saved successfully!

```
[<matplotlib.lines.Line2D at 0x7f03108cdb10>]
```



✓ 0s completed at 10:02 PM

Saved successfully!