Stock Market Prediction Using Deep Learning(CNN-LSTM Model)

Aditya Agrawal

Roll no. - 200100015

200100015@iitb.ac.in

Mentored By:
Agam, Aastha

Jan 2023

Contents

1. Introduction

The stock market is an essential marketplace where people may transact and purchase stock in companies they trust. It is the sole source of income for some people while serving as an extra source for others. One of the biggest issues in the economic world has always been the trend change in stock market prices. The conventional approach to analysis uses fundamental and technical analysis and is based on economics and finance.

This is a time-consuming practice that has a lot of risks when used for a living. Machine learning has gained enormous popularity in recent years, and one of its subcategories, Deep Learning, is widely used in this industry. Convolutional Neural Network (CNN) - Long and Short-term is what we use in this project.

2. Dataset

2.1 Data Collection

The mentors of the project provided Stock Market data. The data used was TTM.csv (Tata Motors)

2.2 Data Preprocessing

The data did not have any missing values when checked, also all the data was of numerical type(int or float). hence no changes were required. The model was trained and predicted using only the day-to-day closing values, hence all other column values like 'open',' high',' low', etc were dropped. As there is a large gap in the input data, the closing values were standardized as follows:-

$$y_i = \frac{x_i - \overline{x}}{s},$$

$$x_i = y_i * s + \overline{x},$$

where yi = standardized value

xi = input value

s = standard deviation of the input data

xbar = average of the input data

Code snippet to carry out the standardization:-

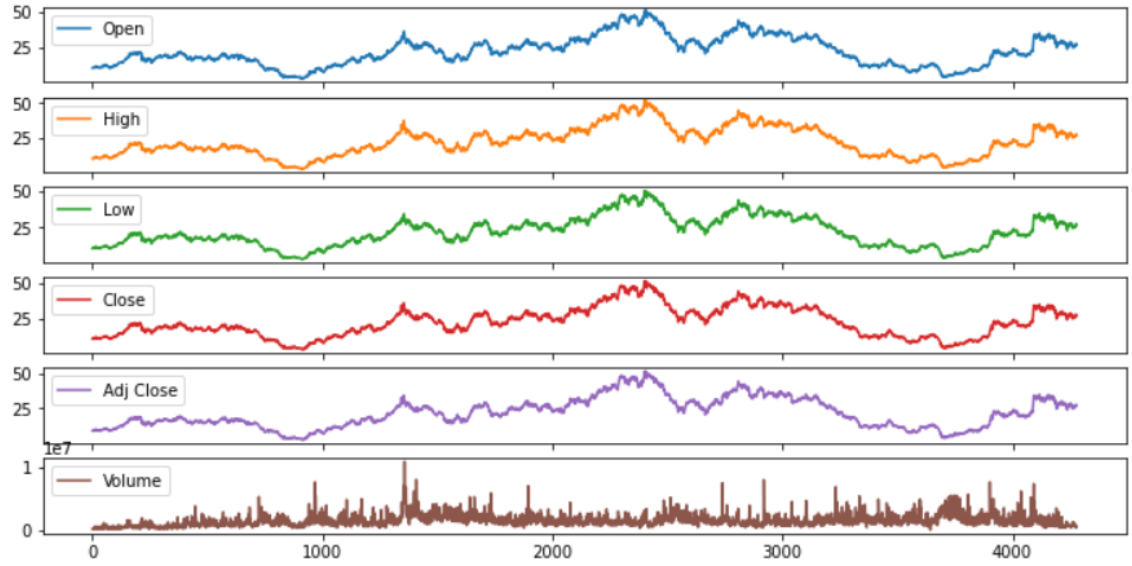
```
mean = close2['Close'].mean()
s = close2['Close'].std()

close2['mean'] = mean
close2['standardized_value'] = (-close2['mean'] + close2['Close'])/s
```

As the index was simple numbers, I set the dates as the index for the data frame, and then the input data was split into training and test datasets with 3536 entries for training data and 541 for test data

2.3 Data Visualization

the data can be visualized as follows:



3. Understanding LSTM - CNN model

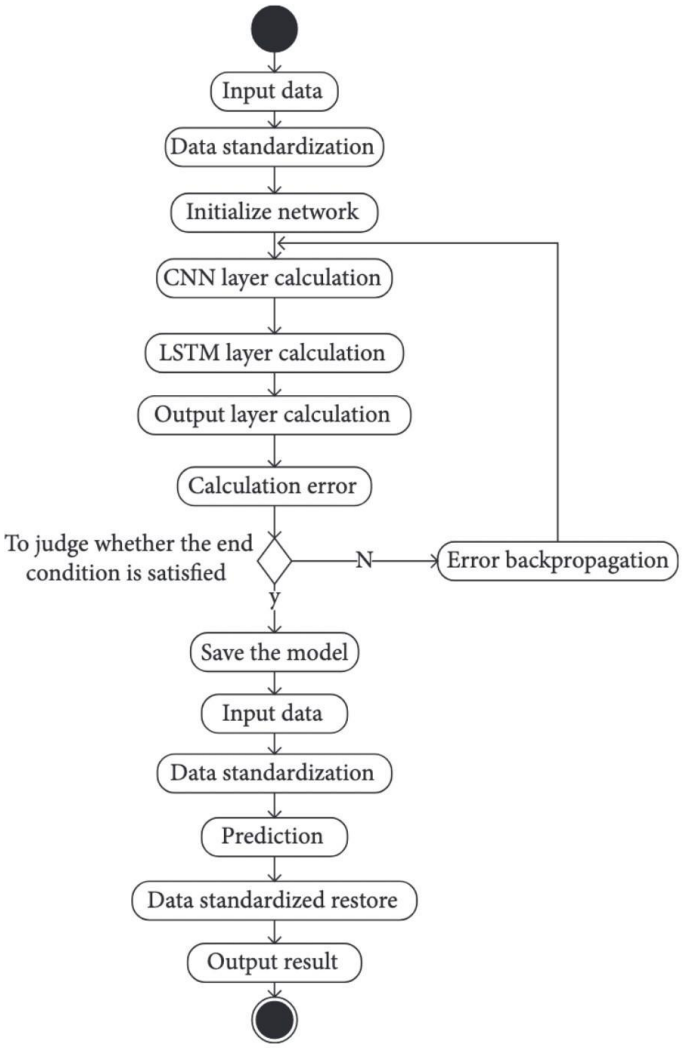
CNN(*Convolutional Neural Network*) helps in understanding and *finding patterns and features* using the convolution layer and pooling layer.

LSTM(*Long Short-Term Memory*) networks are well-suited to classifying, processing, and making predictions based on time series data.
According to the characteristics of CNN and LSTM, a stock forecasting model based on CNN-LSTM is established. The model structure includes an input layer, a one-dimensional convolution layer, a pooling layer, LSTM hidden layer, and a full connection layer.

CNN is mainly composed of two parts: the convolution layer and the pooling layer. Each convolution layer contains a plurality of convolution kernels. After the convolution operation of the convolution layer, the features of the data are extracted, but the extracted feature dimensions are very high, so in order to solve this problem and reduce the cost of training the network, a pooling layer is added after the convolution layer to reduce the feature dimension.

LSTM is a network model designed to solve the longstanding problems of gradient explosion and gradient disappearance in RNN. The LSTM memory cell consists of three parts: the forget gate, the input gate, and the output gate.

The CNN-LSTM process of training and prediction is shown in Figure 3.



4. Evaluation Metrics

In order to evaluate the forecasting effect of CNN-LSTM, *the mean absolute error (MAE)*, *root mean square error (RMSE)*, and *R-square (R2)* are used as the evaluation criteria for the model.

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|\widehat{y}_i - t y_i\right|,$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\widehat{y}_i - y_i\right)^2},$$

$$R^2 = 1 - \frac{\left(\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2\right)/n}{\left(\sum_{i=1}^{n}\left(\bar{y}_i - t\hat{y}_i\right)^2\right)/n,}$$

yi is the true value

yi(cap) is the predictive value

yi(bar) is the average value

The smaller the MAE, and RMSE, the better the forecasting while the closer the R^2 is to 1, the better the forecasting.

5. Model Implementation

5.1 Importing crucial libraries

For the project, I imported some useful libraries like Pandas, Numpy, Matplotlib, Seaborn, Tensorflow, Etc.

```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
from datetime import datetime
from pandas.plotting import autocorrelation_plot
%matplotlib inline
```

```
import tensorflow as tf
from tensorflow.keras.layers import Conv1D, Dense, Dropout

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

5.2 Defining Model Parameters

The snippet below shows the 1D convolutional layers involved, with values set for kernel size, number of filters, activation function as Relu, and max pooling being used.

Similarly the layer for LSTM model are used with dropout at 0.5 being implemented to avoid overitting on the input data.

```
from tensorflow.keras.layers import MaxPooling1D, Flatten

model=Sequential()
model.add(Conv1D(64,kernel_size=3,activation='relu', input_shape=(100,1)))
model.add(Conv1D(64,kernel_size=3,activation='relu'))
model.add(MaxPooling1D(2))
# model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dropout(0.5))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

5.3 Fitting the model

The model is then fitted on the training data, with batch size of 32, and number of epochs set to 30(so as to reduce the training time)

```
model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=30,batch_size=32,verbose=1)
```

```
Epoch 1/30
111/111 [==============================] - 22s 133ms/step - loss: 0.1350 - val_loss: 0.0396
Epoch 2/30
111/111 [==============================] - 7s 59ms/step - loss: 0.0521 - val_loss: 0.0326
Epoch 3/30
111/111 [==============================] - 6s 59ms/step - loss: 0.0477 - val_loss: 0.0217
Epoch 4/30
111/111 [==============================] - 7s 59ms/step - loss: 0.0426 - val_loss: 0.0164
Epoch 5/30
111/111 [==============================] - 7s 60ms/step - loss: 0.0378 - val_loss: 0.0188
Epoch 6/30
111/111 [==============================] - 7s 60ms/step - loss: 0.0352 - val_loss: 0.0159
```

6. Model Performance Evaluation

6.1 Values for different Evaluation Metrics

Importing the MSE, RMSE, Rsquare from sklearn.metrics, we got the corresponding values for the training and test data.

```
from sklearn.metrics import mean_squared_error,mean_absolute_error
train_rmse=math.sqrt(mean_squared_error(y_train,train_predict))
test_rmse=math.sqrt(mean_squared_error(y_test,test_predict))
print(f"train_rmse={train_rmse} and test_rmse={test_rmse}")
```

```
train_rmse=0.07787162403246937 and test_rmse=0.08202100731409348
```

```
train_mae=mean_absolute_error(y_train,train_predict)
test_mae=mean_absolute_error(y_test,test_predict)
print(f"train_mae={train_mae} and test_mae={test_mae}")
```

```
train_mae=0.059880355705784545 and test_mae=0.06151900145427015
```

```
from sklearn.metrics import r2_score
train_r2square=r2_score(y_train,train_predict)
test_r2square=r2_score(y_test,test_predict)
print(f"train_r2square={train_r2square} and test_r2square={test_r2square}")
```

```
train_r2square=0.9939274836135729 and test_r2square=0.9906824297559187
```

6.2 Prediction graphs and plots

Here is the plot for actual training data vs the predicted training data from the model.

Here is the plot for actual test data vs the predicted test data from the model.

7. Conclusion

Thus we were successfully able to create a CNN LSTM model to predict the stock market prices, using the dropout, it helped us with overfitting because we can see that the results for test data are nearly as good as training data.

For both the training and test evaluation, the Rsquared score is more than 0.99 and thus the model has done good work fitting the data.

Also the model accuracy and predictions can be further improved by changing the model parameters like layer size, kernel_size, number of filters, number of training epochs, batch size, activation function used, etc.

Although, one must remember that predicting stock market prices is not easy, as they are subjected to various external factors and do not always follow a particular pattern. They are subjected to news, political and governmental decisions, economic impacts, etc. Hence, one must be very careful while relying on these models for their investments.

Thus a successful model has been created and a great thanks to the mentors for the project: Agam, Aastha for helping me out.

THANK YOU