STOCK PREDICTION MODEL (using LSTM)

The art of predicting stocks has been a difficult task and even investors & researchers are highly interested int his areas development. For a good investment a future prediction of the market is necessary.

This is the model on predictions of Stock using ML. In this work we present a RNN and LSTM (long short term memory) approach to predict Stock Market. As they have proved to be one of the most successful in predicting values.

**STEP 1: RAW DATA**

Historical stock data is collected and this is used for prediction of future stock prices.

***Importing the libraries***

**import numpy as np** – numpy so that we can apply mathematical function and operations to our multidimensional arrays.

**import matplotlib.pyplot as plt** -for visualisation purposes.

**import pandas as pd** – a data analysis and manipulation tool.

**import datetime** – libraray to work wit data as data objects.

***Reading the dataset using.***

**dataset = pd.read\_csv('/content/Google\_Stock\_Price\_Train.csv',index\_col="Date",parse\_dates=True)**

***Display the dataset***. The head function gives us the top 5 rows present in the set. The tail gives us the last 5 rows present in the dataset.

**dataset.head()**

**dataset.isna().any()**

this function is used to check if any data is applicable or not. It detects any missing values or NA values.

**dataset.info()**

this basically prints the basic info. The 5 columns(head or tail assigned above) , number of non null values and data type and memory usage.

**dataset["Volume"] = dataset["Volume"].str.replace(',', '').astype(floa)**

this function has been written to homogenise the datatype of the column types as we see the data type is object while the others are float.

***The next function involves taking out the 7 DAY ROLLING MEAN.*** For every single stock prediction, we require to look 7 days back collect all transactions that fall in that range and find out the average.

**dataset. rolling(7). mean().head()**

***The next step would involve comparisons of the previous graphs with the rolling mean.***

***Lastly in this step we are creating our training dataset and reading using pandas.***

**STEP 2 : DATA PREPROCESSING**

It involves data discretization (reduce a part by keeping particular parts), Data transformation(normalise), Data cleaning (fill in missing values) and data integration. After it is divided into a clean data set it is divided into a training and testing sets to evaluate.

**from sklearn.preprocessing import MinMaxScaler**

**sc = MinMaxScaler(feature\_range = (0, 1))**

**training\_set\_scaled = sc.fit\_transform(training\_set)**

here we need to sacle the data using MinMax Scaler form sklearn ( a python library )in range 0 to 1.

***Next we require to create a data structure with 60 timesteps*** we are taking data from day 1 to day 60 and make predictiobs on 61st day and then follow by taking data from day 2 to day 61 and predict the 62nd day.

**X\_train = []**

**y\_train = []**

a loop from i=60 to the end and the we append the x train[]- starts from i-60 so i=61 then its 1 thus it starts from the 1st day and ends at 61. and y train [] – gives the prediction on the I th day (61st)

**for i in range(60, 1258):**

**X\_train.append(training\_set\_scaled[i-60:i, 0])**

**y\_train.append(training\_set\_scaled[i, 0])**

**X\_train, y\_train = np.array(X\_train), np.array(y\_train)**

***The next thing to be done is to reshape the data.***

**X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))**

**STEP 3: FEATURE EXTRACTION**

In this we are going to extract the features to be fed by the Neural Network from date, high, low, close and volume.

***We have to import Keras library and package to build the RNN. It is actually a Tensor Flow high level APP for building and training deep earning models.***

**from keras.models import Sequential**- linear stack of layers through which we create sequential models.

**from keras.layers import Dense**- this function is a regular deeply connected neural network layer commonly and frequently used to change dimensions of the output.

**from keras.layers import LSTM**

**from keras.layers import Dropout**

now ***We require to initialize the RNN*** for a time series problem we are using a regression model for that the first step is to read in the data which is a sequential data and assign it to regressor.

**regressor = Sequential()**

**STEP 4: TRAINING THE NEURAL NETWORK**

The data is fed to the Neural Network and trained for predictions. We are going to assign weights to the model. It is followed by a Sequential Input Layer followed by three LSTM layers and a dense layer with activation and a dense output layer with linear activation function.

We require to use dropout which is regularisation technique for reducing overfitting in NN. It drops units in NN as we require only one output so **units =1**

**regressor.add(LSTM(units = 50, return\_sequences = True, input\_shape = (X\_train.shape[1], 1)))**

**regressor.add(Dropout(0.2))**

# Adding a second LSTM layer and some Dropout regularisation

**regressor.add(LSTM(units = 50, return\_sequences = True))**

**regressor.add(Dropout(0.2))**

# Adding a third LSTM layer and some Dropout regularisation

**regressor.add(LSTM(units = 50, return\_sequences = True))**

**regressor.add(Dropout(0.2))**

# Adding a fourth LSTM layer and some Dropout regularisation

**regressor.add(LSTM(units = 50))**

**regressor.add(Dropout(0.2))**

# Adding the output layer

**regressor.add(Dense(units = 1))**

the next task would be to compile the RNN and use **optimizer(**one of the two arguments required to compile the Keras model). The type of optimizer can greatly affect how fast the algorithm converges the minimum value.

We have **used ADAM optimizer which combines the perks of ADAgrad and RMSprop.**

# Compiling the RNN

**regressor.compile(optimizer = 'adam', loss = 'mean\_squared\_error')**

while training we must be sure that the weights don’t get too large(focusing on one datapoint hence overfitting. We use Dropouts a new method for preventing overfitting.

**OUTPUT GENERATION**

On running and fitting the training set we have assigned value to **epoch** (a frame of time in ML) **batchsize** (refers the number of training examples)

# Fitting the RNN to the Training set

**regressor.fit(X\_train, y\_train, epochs = 100, batch\_size = 32)**

**STEP 5: Repeat the same for the test data and thus visualise the results.**

 Visualising the results

**plt.plot(real\_stock\_price, color = 'red', label = 'Real Google Stock Price')**

**plt.plot(predicted\_stock\_price, color = 'blue', label = 'Predicted Google Stock Price')**

**plt.title('Google Stock Price Prediction')**

**plt.xlabel('Time')**

**plt.ylabel('Google Stock Price')**

**plt.legend()**

**plt.show()**