Stock Market Prediction

Using Machine Learning

Supervised By
Prof. L.P. Singh

Project By
Harsh Jain (20124021)
Paras Bajpai (20124033)

Branch: Mathematics and Computing

Introduction

Through this project we try to make stock market predictions. The process of determining the future value of a stock, or any other financial instrument traded on an exchange is known as stock market prediction. This done right can lead to significant amounts of profit. Many people trade on the stock exchanges on a daily basis, and many times, due to lack of knowledge tend to make wrong investment decision and hence face huge loses.

Traditionally finance experts have used several ratios and other techniques to judge future stock trends. With the development of technologies like Machine Learning, it was only a matter of time for them to be applied to make more efficient stock predictions.

Objective

The aim of the project is to successfully make predictions regarding future ‘Google’ stock prices. In the current emerging competitive market, predicting the stock returns as well as the company's financial status in advance will provide benefits for the investors in order to invest confidently. This prediction can be done by using the previous and current stock data and training them using different Machine Learning models such as LSTM etc.

Methodology

Stock Market Prediction is to be achieved by training a Recurrent Neural Network and then using Long Short Term Memory models. To train the model a list of opening and closing values of the stocks of Google has been provided. The model has been made by adding several layers of LSTM neural networks and has then been trained using the 'Adam' Algorithm. The trained model is then tested against a given test data. At both the stages the given raw data is first preprocessed. Data preprocessing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn; therefore, it is extremely important that we preprocess our data before feeding it into our model.

As we move through the slides we will explain Machine Learning, RNNs, LSTM, etc, along with the further explanation of the model.

We have used LSTMs because they are very powerful in sequence prediction problems as they're able to store past information.

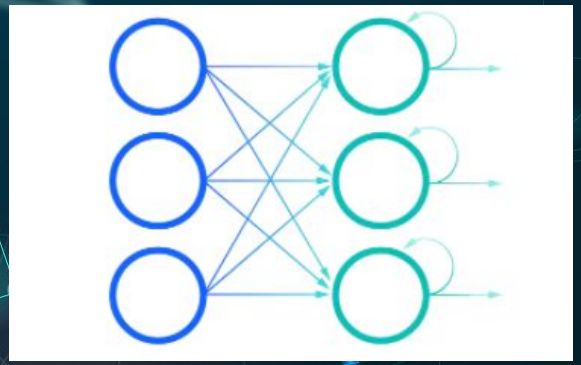
Defining Machine Learning

Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data. It is a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions. Machine learning algorithms are used in a wide variety of applications, such as in email filtering, speech or image recognition, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers; but not all machine learning is statistical learning. Some implementations of machine learning use data and neural networks in a way that mimics the working of a biological brain. In its application across business problems, machine learning is also referred to as predictive analytics.

What is RNN (Recurrent Neural Network)

A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, speech recognition, and image captioning; they are incorporated into popular applications such as Siri and Google Translate. Like feedforward and convolutional neural networks (CNNs), recurrent neural networks utilize training data to learn. They are distinguished by their “memory” as they take information from prior inputs to influence the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depend on the prior elements within the sequence.



Intro to LSTM

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning.n Ulike standard feedforward neural networks, LSTM has feedback connections. It can process not only single data points, but also entire sequences of data. For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic.

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is also an advantage of LSTM over RNNs.

Analysis

This work consists of three parts: data extraction and pre-processing of the stock prices of google, carrying out feature engineering, and stock price trend prediction model based on the long short-term memory (LSTM). We collected, cleaned-up, and structured 5 years of google stock market data. We reviewed different techniques often used by real-world investors, The system is customized with an LSTM prediction model, achieved high prediction accuracy. We also carried out a comprehensive evaluation of this work.

Our proposed solution is a LSTM model, we proposed a fine-tuned and customized deep learning prediction system with LSTM to perform prediction.

We have achieved a decent outcome from our proposed solution which makes predictions about future stock prices of Google with quite accuracy.

Results and discussions

We ran the code successfully multiple times and most optimum was to do 50 epochs (taking accuracy and time both as a factor) and found that training loss (calculated as mean squared error) dropped between 0.002 and 0.003 everytime and is thus quite accurate and we saw that the predicted prices of stocks matched to the actual one with quite good accuracy.

How does the Model Work?

Now, we will explain our Model cell by cell:

1) Cell 1: Importing necessary Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense,LSTM,Dropout
```

2) Cell 2: Reading Train data into the object ‘data’, and then displaying its first five rows

```
[ ] data = pd.read_csv('Google_train_data.csv')
    data.head()
```

	Date	Open	High	Low	Close	Volume
0	1/3/2012	325.25	332.83	324.97	663.59	7,380,500
1	1/4/2012	331.27	333.87	329.08	666.45	5,749,400
2	1/5/2012	329.83	330.75	326.89	657.21	6,590,300
3	1/6/2012	328.34	328.77	323.68	648.24	5,405,900
4	1/9/2012	322.04	322.29	309.46	620.76	11,688,800

3) Cell 3: Displaying the data types of the different columns of the object ‘data’

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    1258 non-null   object
 1   Open    1258 non-null   float64
 2   High    1258 non-null   float64
 3   Low     1258 non-null   float64
 4   Close   1258 non-null   object
 5   Volume  1258 non-null   object
dtypes: float64(3), object(3)
memory usage: 59.1+ KB
```

4) Cell 4: This is where the data preprocessing begins. Firstly we convert the datatypes of the ‘Close’ column from object to float. Then we drop the rows which contain the empty cells (if any) from our dataset. Then we go on to create the object ‘trainData’, where we pass all rows of the 4th column of object ‘data’.

```
[ ] data["Close"]=pd.to_numeric(data.Close,errors='coerce')
    data = data.dropna()
    trainData = data.iloc[:,4:5].values
```

5) Cell 5: Here we display the front 5 rows of the object ‘data’, and we see that the data type of the ‘Close’ column has been changed to float.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1149 entries, 0 to 1257
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    1149 non-null   object 
 1   Open    1149 non-null   float64
 2   High    1149 non-null   float64
 3   Low     1149 non-null   float64
 4   Close   1149 non-null   float64
 5   Volume  1149 non-null   object 
dtypes: float64(4), object(2)
memory usage: 62.8+ KB
```

6) Cell 6: Here we normalise all the data in the object ‘trainData’ to the range (0,1) by creating the object ‘sc’ of the MinMaxScaler class. Then the size of the ‘trainData’ array is displayed, which has only 1 column and multiple rows.

```
sc = MinMaxScaler(feature_range=(0,1))
trainData = sc.fit_transform(trainData)
trainData.shape
```

```
(1149, 1)
```

7) Cell 7: The normalised ‘trainData’ array is displayed

```
[ ]  trainData
```

```
array([[0.40001392],
       [0.40665027],
       [0.38520976],
       ...,
       [0.68184982],
       [0.67660572],
       [0.65115092]])
```

8) Cell 8: We create lists ‘X_train’ and ‘Y_train’, and transfer data from ‘tainData’ to them with a timestep of 60.

```
[ ]  X_train = []
     y_train = []

     for i in range (60,1149):
         X_train.append(trainData[i-60:i,0])
         y_train.append(trainData[i,0])

     X_train,y_train = np.array(X_train),np.array(y_train)
     print(X_train)
```

```
[[0.40001392 0.40665027 0.38520976 ... 0.36279469 0.35745777 0.37766846]
 [0.40665027 0.38520976 0.36439577 ... 0.35745777 0.37766846 0.36065992]
 [0.38520976 0.36439577 0.30063115 ... 0.37766846 0.36065992 0.34406906]
 ...
 [0.65286802 0.66184797 0.66194078 ... 0.69625951 0.69312697 0.69693243]
 [0.66184797 0.66194078 0.66284574 ... 0.69312697 0.69693243 0.68184982]
 [0.66194078 0.66284574 0.65871543 ... 0.69693243 0.68184982 0.67660572]]
```

9) Cell 9: Now we add the batch_size axis to the ‘X_train’ object and then print its shape.

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

     (1089, 60, 1)
```

10) Cell 10: Now we define the structure of our model. First, we add 4 layers of LSTM neural networks along with dropout regularisation and then a dense layer of neural network. Then we go on to define the Algorithm to be used to train the model as “Adam” and that the loss is to calculated as the “Mean Squared Error”.

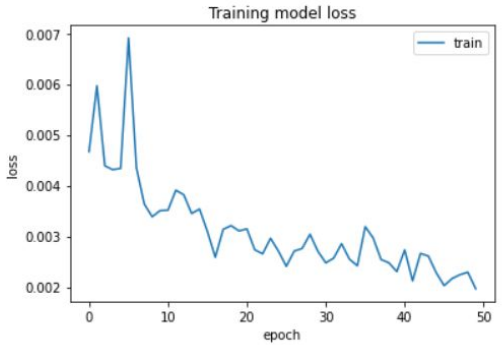
```
[ ]  model = Sequential()
     model.add(LSTM(units=100, return_sequences = True, input_shape =(X_train.shape[1],1)))
     model.add(Dropout(0.2))

     model.add(LSTM(units=100, return_sequences = True))
     model.add(Dropout(0.2))

     model.add(LSTM(units=100, return_sequences = True))
     model.add(Dropout(0.2))

     model.add(LSTM(units=100, return_sequences = False))
     model.add(Dropout(0.2))

     model.add(Dense(units =1))
     model.compile(optimizer='adam',loss="mean_squared_error")
```

11) Cell 11: Now we finally train the model. A total of 50 epochs have to be performed. The dataset has also been divided into 32 batches for each iteration.

```
[ ]  hist = model.fit(X_train, y_train, epochs = 50, batch_size = 32, verbose=2)
```

12) Cell 12: Now we plot the loss encountered in each iteration against the epoch number. We see that the loss decreases with increasing number of iterations.

```
[ ]  plt.plot(hist.history['loss'])
     plt.title('Training model loss')
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(['train'], loc='upper right')
     plt.show()
```



13) Cell 13: Now we take the first step to test our trained model. That is we do the preprocessing of the Test data.

```
testData = pd.read_csv('Google_test_data.csv')
testData["Close"]=pd.to_numeric(testData.Close,errors='coerce')
testData = testData.dropna()
testData = testData.iloc[:,4:5]
y_test = testData.iloc[60:,0:].values

inputClosing = testData.iloc[:,0:].values
inputClosing_scaled = sc.transform(inputClosing)
inputClosing_scaled.shape
X_test = []
length = len(testData)
timestep = 60
for i in range(timestep,length):
    X_test.append(inputClosing_scaled[i-timestep:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))
X_test.shape
```

```
(192, 60, 1)
```

14) Cell 14: Now the normalised predictions are made and the results are stored in the ‘y_pred’ object.

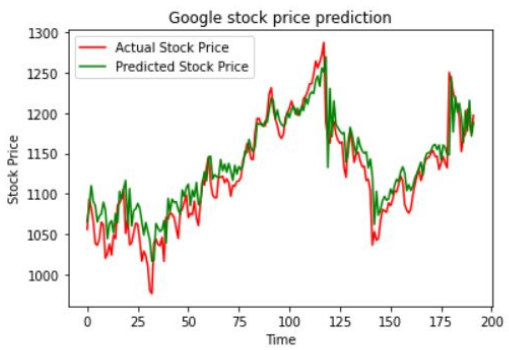
```
[ ] y_pred = model.predict(X_test)
    y_pred
```

15) Cell 15: Now we do the inverse transformation of the object ‘y_pred’ by using the ‘sc’ object created earlier, to scale them to their original values. These are then stored in the ‘predicted_price’ object.

```
[ ]  predicted_price = sc.inverse_transform(y_pred)
```

16) Cell 16: Now we finally plot our predicted price against the actual future prices and we see that the model is quite accurate.

```
plt.plot(y_test, color = 'red', label = 'Actual Stock Price')
plt.plot(predicted_price, color = 'green', label = 'Predicted Stock Price')
plt.title('Google stock price prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```



References

- Geeksforgeeks
- Wikipedia
- TowardsDataScience
- Course by Andrew Ng

Data Source

The dataset is present here.

Thank You!