Jonathan Feng

DA 420

6/21/2021

KNOWLEDGED BASED ARTICLE:

Stock Price Prediction

Overview:

As common a project this may be, stock prediction has proven itself time and time again as one of the fundamental examples of machine learning and time series analysis. However, in this project I have managed to do not only the prediction with an LTSM model (Long Short-Term Memory Model), but also an integration of using real time data on the daily for making decisions on whether a stock is a buy or a sell for that given day's market opening.

This works with three main parts and will be explained further in this article: Our main program that handles a user's budget, a ticker portion that handles web scraping and data cleaning, and a ML prediction portion that handles the LTSM and outputting the desired results (buy or sell).

To preface, this is a little different from the project proposed as I did mention an auto-trading function, which should not be too hard to implement. But for the sake of brevity and purpose, we will not be including that portion, with the largest reason being that *I would have needed to open a brokerage account and deposit over 10,000 dollars to day trade without restrictions* **(Webull, 2021)**. I did also mention the option of paper trading in my proposal. Within a couple hours of digging around, I would have had to apply and wait for the API and the API keys to use a pre-existing system.

With all our little disclaimers out of the way, without further ado, we will get into the grit of the project.

[Portion 1] Ticker.py :

As mentioned before, I wanted to go into this knowing that this program could work wherever, whenever. What does that mean? Well, you guessed it, real time data. Without getting permissions and submitting forms and waiting for approval of APIs I opted for an easier more illegal solution, (*disclaimer: not for financial use or financial advice*), web scraping.

Now this is a topic I have explored on my own and done in some of my past classes, so it was familiar to me. I made a function as a part of this program to get our desired data off the yahoo finance site, in their category *trending tickers* **(Yahoo Trending Stocks, 2021)**.

```
In [41]: import pandas as pd
         import numpy as np
         from bs4 import BeautifulSoup
         import requests

         def trending_ticker():
             URL = "https://finance.yahoo.com/trending-tickers"
             dat = requests.get(URL)

             soup = BeautifulSoup(dat.text)
             top_ticker = []
             last_price = []
             ticker_name = []
             volume = []

             for listing in soup.find_all('tr', attrs={'class':'simpTblRow'}):

                 for individual in listing.find_all('td', attrs={'aria-label':'Symbol'}):
                   top_ticker.append(individual.text)

                 for individual in listing.find_all('td', attrs={'aria-label':'Name'}):
                   ticker_name.append(individual.text)

                 for individual in listing.find_all('td', attrs={'aria-label':'Last Price
                   last_price.append(individual.text)

                 for individual in listing.find_all('td', attrs={'aria-label':'Volume'}):
                   volume.append(individual.text)

             fin_dat = pd.DataFrame()

             fin_dat['symbol'] = top_ticker
             fin_dat['company_name'] = ticker_name
             fin_dat['last_price'] = last_price
             fin_dat['volume'] = volume
```

In the function created, trending_ticker(), we use beautiful soup , a data scraping library, to perform our data gathering. Some previous knowledge of how HTML is needed to physically look through the website for the relevant tags for identifying our data **(Real-Python, 2021)**. I found that everything relevant was under ‘aria-label’ and I proceeded to append the correct data to their specific arrays.

I used pandas in the very end to attach the data together which you will be able to see in *the figure below.*

```
    # pruning
    fin_dat["last_price"] = fin_dat["last_price"].str.replace(',','')
    fin_dat["last_price"] = pd.to_numeric(fin_dat["last_price"])
    fin_dat["volume"] = fin_dat["volume"].str.replace(',','')
    fin_dat["volume"] = fin_dat["volume"].str.replace('M','')
    fin_dat.loc[fin_dat['volume'].str.contains('B'), 'volume'] = '1000'

    fin_dat["volume"] = fin_dat["volume"].str.replace('B','')
    fin_dat["volume"] = pd.to_numeric(fin_dat["volume"])

    # REMOVE CRYPTO
    fin_dat = fin_dat[~fin_dat["symbol"].str.contains("USD")]
    fin_dat = fin_dat[~fin_dat["symbol"].str.contains("CAD")]

    # limit to volume to tickers above 100 million
    fin_dat = fin_dat[fin_dat['volume'] >= 100.0]

    return fin_dat

print(trending_ticker())
```

```
  symbol                     company_name  last_price   volume
0   WISH                ContextLogic Inc.       13.50  326.006
1   TRCH  Torchlight Energy Resources, Inc.        9.92  371.420
4   GSAT                  Globalstar, Inc.        1.51  155.832
```

I had to do some data cleaning at the very end since the web.DataReader() I was using later does not recognize crypto or tickers that are too recent. In addition to that, a removal of small volume trades was removed.

To make a point, we are only focused on those stocks with *large quantities of volume* flowing in and out through the day, so I also filtered for stocks that only have volumes greater than 100 million to give our options a little more room for profit on percentage changes daily **(Mitchell, 2020).**

With all of that said, a final return was made to return the data set we wanted. And now, we can move onto the next portion main().

[Portion 2] main.py

Moving onto our second program we have a relatively short program that takes the input of the user in terms of an investment amount. This returns a further filtered data frame (with stock prices never exceeding the max amount of money the user inputs) of the stock symbols and information required for our ML model later.

In [1]:
```
# importing necessary packages

import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import sys
sys.path.insert(0, r'C:\Users\Taterthot\Desktop\420 final')
import ticker
#import algo

ticker_list = ticker.trending_ticker()
print(ticker_list)
```

```
  symbol                        company_name  last_price   volume
0   WISH                  ContextLogic Inc.       13.50  326.006
1   TRCH  Torchlight Energy Resources, Inc.        9.92  371.420
4   GSAT                   Globalstar, Inc.        1.51  155.832
  symbol                        company_name  last_price   volume
0   WISH                  ContextLogic Inc.       13.50  326.006
1   TRCH  Torchlight Energy Resources, Inc.        9.92  371.420
4   GSAT                   Globalstar, Inc.        1.51  155.832
```

In [2]:
```
def main():
    budget = float(input("Please input your investment amount: "))

    # stock symbols to look into within price range
    counter = 0
    ticker_list["last_price"] = pd.to_numeric(ticker_list["last_price"])
    trade_list = ticker_list[ticker_list['last_price'] <= budget]

    print(trade_list)

    return(trade_list)
main()
```

```
Please input your investment amount: 10000
  symbol                        company_name  last_price   volume
0   WISH                  ContextLogic Inc.       13.50  326.006
1   TRCH  Torchlight Energy Resources, Inc.        9.92  371.420
4   GSAT                   Globalstar, Inc.        1.51  155.832
```

Out[2]:

	symbol	company_name	last_price	volume
0	WISH	ContextLogic Inc.	13.50	326.006
1	TRCH	Torchlight Energy Resources, Inc.	9.92	371.420
4	GSAT	Globalstar, Inc.	1.51	155.832

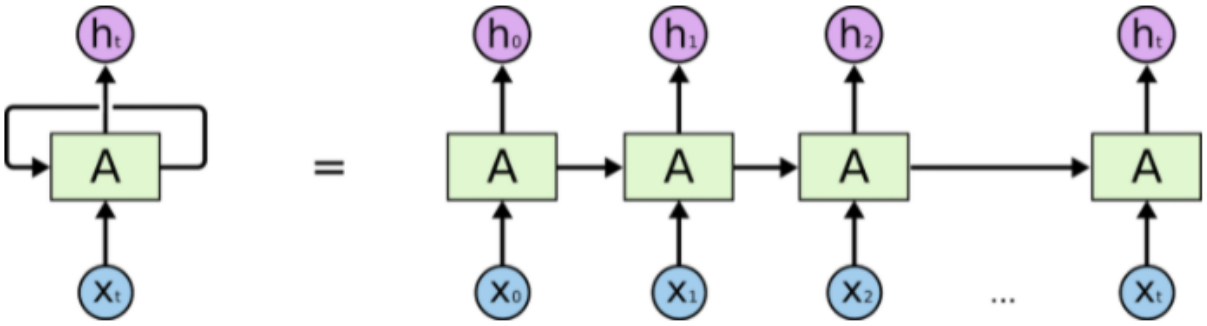
This was but a very short but necessary inclusion as I wanted to separate where the user would input their desired investments.

[Portion 3] algo.py

Whew! We got all that data cleaning out of the way, now what? The next and last part covers the main grit of the project where I will be explaining exactly what LTSM is, how it works in time series analysis, and how it is relevant to this stock prediction project.

Before we get to the actual predictions I will be explaining what LSTMs are. LSTM, Long-Short Term Memory networks/models, are designed for applications where the inputs are in ordered sequence where the information earlier may be important. While commonly used in language prediction and the auto-predict text on your phone where it predicts the next word you are going to type for sentence completion, concepts can be applied similarly in time series as the information from earlier is important **(Brownlee, 2020)**.

LSTMs are a type of recurrent network (RNN), which are networks that reuse part of the input from a previous step in the next step. As shown below, RNNs use other nodes in a particular sequence to influence outputs on later nodes (computational unit that takes inputs and outputs and connects to different nodes) **(Colah, 2015)**.



Recurrent Neural Network Structure – Image via colah.github.io

Taken From: https://lionbridge.ai/articles/difference-between-cnn-and-rnn/

Each of these A nodes are recurrent but they all have an internal state (memory space), which can be retrieved over many times, Both the input the output and the internal state are used in the main function ran in the node, leading to an updatable memory space **(Colah, 2015)**. In addition to all of this, the nodes also have gates that are mainly influenced based on an input performing a forgetting and recalling function for the previous or current node.

Similar gates are used to perform changes on the current state itself and the output, allowing for a myriad of different outputs depending on a given / past situation **(Ganegedara, 2020)**. Even though this may be complex to digest, it provides our network with the ability to learn complex patterns and interdependencies of the data.

Now that we have some knowledge on what LTSM is, we can jump straight into the program.

```
import matplotlib.pyplot as plt
import numpy as np
import scipy
import pandas as pd
import pandas_datareader as web
import datetime as dt


from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
```

First, we will import all necessary libraries for this program. We have some portions from sklearn, TensorFlow, and your classic pandas, NumPy, and matplotlib for graphing later.

Now before we get further into writing the model, we will have to gather some further data for training once more. We will use Yahoo again along with the data reader to grab time, dates, and closing prices to continue with further predictions.

```
predicted_values = []
def predict_stock(option):

        # Load Data\
        company = option
        start = dt.date(2012, 1, 1)
        end = dt.date(2021, 1, 1)
        data = web.DataReader(company, 'yahoo', start, end)
        # Prepare Data
        scaler = MinMaxScaler(feature_range=(0, 1))

        # predict closing price
        scaled_data = scaler.fit_transform(data['Close'].values.reshape(-1, 1))

        x_train = []
        y_train = []

        prediction_days = 60

        for x in range(prediction_days, len(scaled_data)):
            x_train.append(scaled_data[x-prediction_days:x, 0])
            y_train.append(scaled_data[x, 0])

        x_train, y_train = np.array(x_train), np.array(y_train)
        x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```

As you can see a call on MinMaxScaler (sklearn) and reshape (NumPy formatting for arrays) was used to restructure and put the data on a more manageable scale, for us between 0 and 1 **(TensorFlow, 2021)**. We will limit our prediction days to 60 even though we have data dating back all the way to 2012, and appending that to our two training sets x and y_train (binary classification, x being our inputs, y being

our expected outcomes) based off of the length of scaled data and our prediction days **(Brownlee, 2020)**.

```
# MODEL
model = Sequential()

model.add(LSTM(units=50, return_sequences=True, input_shape = (x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))
model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

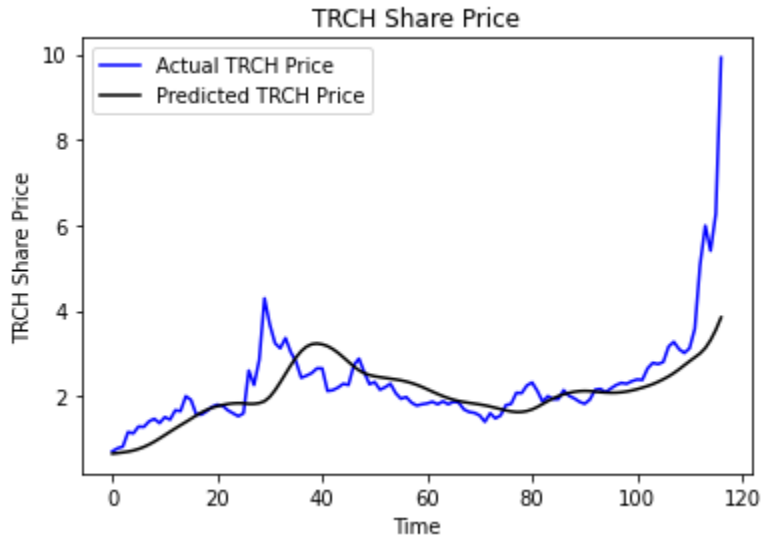
Now comes time to write our model. The model created above is based off a standard model and includes 50 LSTM blocks (hence units 50) and with 1 visible layer of input. We have a series of dropouts that drops out units of invisible and visible data, that addresses overfitting in a neural network. Our dense layer provides and feeds the layer of outputs to the individual nodes. Our compile method provides an optimizer of sorts in this case the commonly used 'adam' optimizer with a loss of mean squared error. There is no need to go into detail of why we use compile, all we need to know is that it is a necessary argument required to train **(Colah, 2015)**.

Furthermore, we finally train by using model fit, which takes our x_train, y_train (inputs, expected outputs) and specify some standardizations in ML. A batch-size (number of samples ran before the model update) is 32 as standard, and our epochs (number of complete passthroughs of our training set) 5 (*usually 25)* is used for the brevity and speed of the code **(TensorFlow, 2021)**.

```
# Test model accuracy
''' In this section we test accuracy against the newer data from 1/1/2020 to the current date''

# Load data
test_start = dt.date(2021, 1, 1)
test_end = dt.datetime.now()
test_data = web.DataReader(company, 'yahoo', test_start, test_end)

actual_prices = test_data['Close'].values

total_dataset = pd.concat((data['Close'], test_data['Close']), axis = 0)
model_inputs = total_dataset[len(total_dataset) - len(test_data) - prediction_days:].values
model_inputs = model_inputs.reshape(-1, 1)
model_inputs = scaler.transform(model_inputs)

# Make Predictions
x_test = []

for x in range(prediction_days, len(model_inputs)):
    x_test.append(model_inputs[x-prediction_days:x, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

predicted_prices = model.predict(x_test)
predicted_prices = scaler.inverse_transform(predicted_prices)
```

A similar process is run for testing the model accuracy against our close values from the original dataset we obtained from Yahoo **(Ganegedara, 2020)**. At the very end we use more concurrent data, from (1/1/2021) to our current date for the actual prediction putting it into the same model from before.

```
# plot

plt.plot(actual_prices, color="blue", label=f"Actual {company} Price")
plt.plot(predicted_prices, color="black", label=f"Predicted {company} Price")
plt.title(f"{company} Share Price")
plt.xlabel('Time')
plt.ylabel(f'{company} Share Price')
plt.legend()
plt.show()
```



Here is a simple plot we can write that shows our predicted vs actual prices for the models above.

With that done now we can use the following code based on our model to project ourselves one day in the future by adding in the necessary inputs and adding 1 to the last day predicted on.

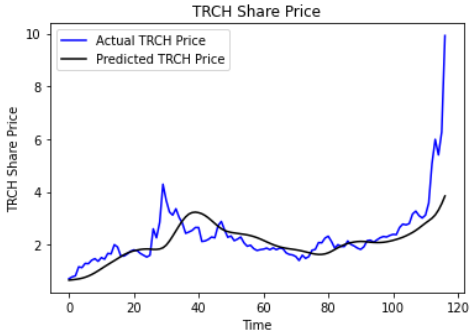
```
# predicting the next move
real_data = [model_inputs[len(model_inputs) + 1 - prediction_days:len(model_inputs+1), 0]]
real_data = np.array(real_data)
real_data = np.reshape(real_data, (real_data.shape[0], real_data.shape[1], 1))

prediction = model.predict(real_data)
prediction = scaler.inverse_transform(prediction)
predicted_values.append(prediction)
```

All that is now left to do is gather the data together for a reasonable looking output and prediction. What is the real question? Well but of course it is whether we buy or sell the stock. I put some simple Boolean logic code below and some print statements to decipher our outputs findings.

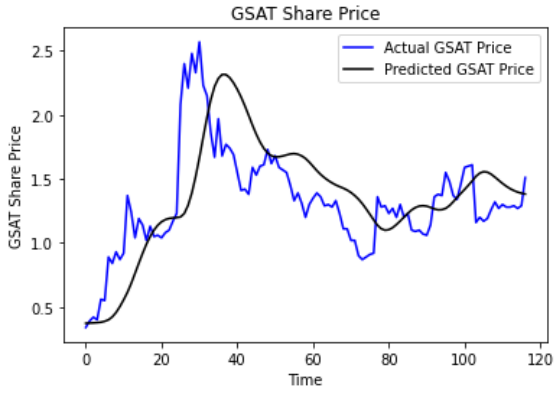
```
import main
ticker_list = main.main()

# remove WISH because not enough data
ticker_list = ticker_list[ticker_list['symbol'] != 'WISH']
print(ticker_list)

for i in ticker_list['symbol']:
    predict_stock(i)

predicted_price = pd.DataFrame(predicted_price)
ticker_list = pd.DataFrame(ticker_list)
counter = 0

for predicted_price in predicted_values:

    print(ticker_list.iloc[counter][0] + ':')
    print(predicted_price[0][0])
    print(ticker_list.iloc[counter][2])
    counter +=1

counter_2 = 0
for predicted_price in predicted_values:
    temp_price = predicted_price[0][0]
    temp_symbol = ticker_list.iloc[counter_2][0]

    if(temp_price > ticker_list.iloc[counter_2][2]):
        print(temp_symbol + " Action: BUY")
    else:
        print(temp_symbol + " Action: SELL")
    counter_2 +=1
```

Continuing below is an output of everything that this program outputs.

```
Please input your investment amount: 10000
  symbol                       company_name  last_price   volume
0   WISH                 ContextLogic Inc.       13.50  326.006
1   TRCH  Torchlight Energy Resources, Inc.        9.92  371.420
4   GSAT                  Globalstar, Inc.        1.51  155.832
  symbol                       company_name  last_price   volume
1   TRCH  Torchlight Energy Resources, Inc.        9.92  371.420
4   GSAT                  Globalstar, Inc.        1.51  155.832
Epoch 1/5
69/69 [==============================] - 10s 44ms/step - loss: 0.0083
Epoch 2/5
69/69 [==============================] - 3s 43ms/step - loss: 0.0027: 0s - loss: 0
Epoch 3/5
69/69 [==============================] - 3s 41ms/step - loss: 0.0023
Epoch 4/5
69/69 [==============================] - 3s 41ms/step - loss: 0.0025
Epoch 5/5
69/69 [==============================] - 3s 49ms/step - loss: 0.0021
```




```
Epoch 1/5
69/69 [==============================] - 8s 40ms/step - loss: 0.0093
Epoch 2/5
69/69 [==============================] - 3s 44ms/step - loss: 0.0039
Epoch 3/5
69/69 [==============================] - 3s 46ms/step - loss: 0.0031
Epoch 4/5
69/69 [==============================] - 3s 44ms/step - loss: 0.0031
Epoch 5/5
69/69 [==============================] - 3s 43ms/step - loss: 0.0029
```



```
TRCH:
4.2498217
9.92
GSAT:
1.3834957
1.51
TRCH Action: SELL
GSAT Action: SELL
```

As you can see, we have our desired graphical outputs along with the stock predictions. The simple print statements at the end tell us whether we should buy or sell, if the predicted price is lower than the actual price sell it, if the price predicted is higher buy it. A very simplistic way of determining whether we buy a stock. Of course, we could go into more detail and make this a whole lot more complex with financial knowledge, identifying day-trading patterns (bullish-pennant, inverse neck, and shoulder, etc.) that would be way too complex to implement for this project (**Mitchell, 2020)**.

Initially I was going to add an NLP (Natural Language Processing) Portion as well where I would take data from the Twitter API to determine the sentiment of the stock, but that would influence the run time of this project significantly, so I decided to opt out, not to mention that Twitter never got the credentials back to me for accessing the API.

Conclusion

I was able to take data off from the web, organize, and put it into a LSTM Neural Network to get a graphical representation of predictions on a given stock within a certain time frame and the prediction of the price for the next day. This allowed the program to output whether we should buy or sell a given security based on whether the prediction was higher or lower than the actual value. I have learned a lot about Machine Learning and Neural Networks through the process of this course. Although I am not proficient by any means, this is a good takeaway into the applications I can use in the future in tandem with data analysis.

Works Cited

Brownlee, J. (2020, August 27). *Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras*. Machine Learning Mastery. https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/.

Colah. (2015, August 27). *Understanding LSTM Networks*. Understanding LSTM Networks -- colah's blog. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Ganegedara , T. (2020, January 1). *(Tutorial) LSTM in Python: Stock Market Predictions*. DataCamp Community. https://www.datacamp.com/community/tutorials/lstm-python-stock-market.

Mitchell, C. (2020, September 12). *Triangle Chart Patterns and Day Trading Strategies*. The Balance. https://www.thebalance.com/triangle-chart-patterns-and-day-trading-strategies-4111224.

Real Python. (2021, March 6). *Beautiful Soup: Build a Web Scraper With Python*. Real Python. https://realpython.com/beautiful-soup-web-scraper-python/.

tf.keras.layers.LSTM : TensorFlow Core v2.5.0. TensorFlow. (n.d.). https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM.

What is day trading? Webull. (n.d.). https://www.webull.com/hc/categories/fq366-What-is-day-trading.

Yahoo! (n.d.). *Trending Stocks Today*. Yahoo! Finance. https://finance.yahoo.com/trending-tickers.