Capstone Proposal – Stock Price Prediction

I. Definition

Project Overview

Across the world, stocks are traded voluminously almost every day, making stock price prediction a highly popular topic. Apart from assessing a company based on its intrinsic value of stock based on its financials and other macroeconomic factors, there has been a growing trend towards harnessing historical stock price and volume data with the use of statistical and mathematical models. In quantitative trading, machine learning models are built with these historical data points as inputs to predict future stock prices and maximise returns. Previously, as these models required significant computational energy, they were only commonly used by large institutions. Gradually, machine learning technology has become more accessible. In this project, we will attempt to perform a simple case of predicting stock prices with the use of a recurrent neural network model.

Problem Statement

With increasing IPOs occurring and a greater demand for more capital, the quantity and volume traded is set to rise even further as more look to capitalise on potential returns. This in turn is likely to lead to increased volatility, making stock price prediction more challenging. In this project, we aim to predict the S&P 500 index stock price using historical price and volume data inputs with the use of the Long Short-Term Memory (LSTM) model, an artificial recurrent neural network. The S&P 500 index was selected as it measures the stock performance of the 500 largest companies listed in the United States, thus providing a comprehensive overview on the overall market performance over the period.

Evaluation metrics

As this is a regression problem, the chosen metric is the RMSE (root-mean-squared-error). RMSE measures the error of a model in terms of how far the predicted values are away from the actual values. Thus, a lower RMSE would indicate better performance. To measure the performance of the LSTM model, a linear regression will be used as a benchmark model.

II. Analysis

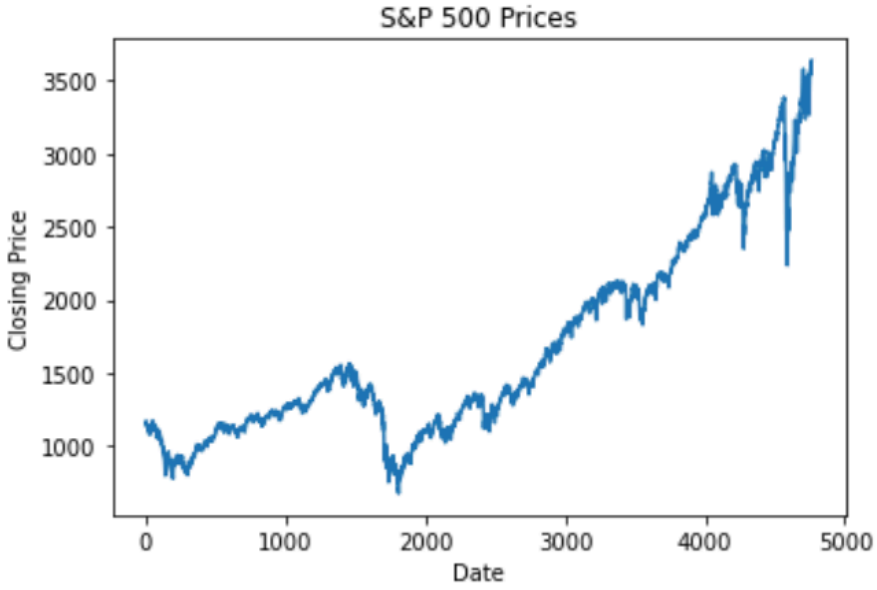
Data Exploration

To start off, the S&P 500 index (“GSPC” symbol) stock input from 1 Jan 2002 till 30 Nov 2020 were extracted from Yahoo! Finance. There are a total of 4762 rows at this initial stage and the extracted details include the Open, High, Low, Close and Adjusted Close price as well as the Volume traded. Here is a screenshot of the top 5 rows of the dataset.

	Date	Open	High	Low	Close	Adj Close	Volume
1	2001-12-31	1161.020020	1161.160034	1148.040039	1148.079956	1148.079956	943600000
2	2002-01-02	1148.079956	1154.670044	1136.229980	1154.670044	1154.670044	1171000000
3	2002-01-03	1154.670044	1165.270020	1154.010010	1165.270020	1165.270020	1398900000
4	2002-01-04	1165.270020	1176.550049	1163.420044	1172.510010	1172.510010	1513000000
5	2002-01-07	1172.510010	1176.969971	1163.550049	1164.890015	1164.890015	1308300000

Data Visualization

Before further processing was performed, a chart was plotted to see the overall trend. Below is the chart of of the closing price by date of this initial dataset, where we see an overall increasing trend over the 19 years period with increasingly frequent and drastic dips nearer the end of the period. Based on our knowledge, we know that this is contributed by the 2008 financial crisis as well as the current COVID-19 pandemic.



Algorithms and Techniques

As mentioned above, the LSTM model is used in this project to predict stock prices for the S&P 500 index. It is applicable here due to its capability to process data in a sequential order by storing past information. The LSTM model is a form of recurrent neural network (RNN) but a key characteristic that distinguishes LSTM from other RNNs is that it is able to learn long-term dependencies. More specifically, the LSTM model is a “recurrent network architecture in conjunction with an appropriate gradient-based learning algorithm” to overcome error back-flow problems (Hochreiter & Schmidhuber, 1997). This is particularly crucial here as the previous price of a stock is useful for predicting the future price patterns.

Benchmark model - Linear Regression

As mentioned, the benchmarking model is the linear regression model, a model that is often used for time series data. In this case, to predict the stock price, the 50-day simple moving day average ("SMA") was computed and used as a dependent variable. Essentially, the 50-day SMA is a commonly used simple computation in technical analysis to indicate the trend direction of the stock price based on the average price of recent historical data. This is in contrast to using just the prior day's price which may result in overfitting and is also highly susceptible to temporary noise and movement. The 50-day simple moving day average has been computed as such:

```
# Adding Simple Moving Average column for 50 days
data['SMA']=data.Close.rolling(50,win_type = 'triang').mean()
```

As SMA is computed based on the average of other days, certain days will have NaN data points under the SMA header, namely the first 50 days. Therefore, the dataset was narrowed down to include data from just the 51st row onwards. As such, the total number of rows became 4711. To verify that each row does indeed have a figure, a check is being performed to ensure that there are no NaN values.

```
# cleaning up the data such that each row has a data input for each header
clean_data = data[51:]
clean_data.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume	SMA
52	2002-03-15	1153.040039	1166.479980	1153.040039	1166.160034	1166.160034	1493900000	1114.914037
53	2002-03-18	1166.160034	1172.729980	1159.140015	1165.550049	1165.550049	1169500000	1114.874101
54	2002-03-19	1165.550049	1173.939941	1165.550049	1170.290039	1170.290039	1255000000	1115.040005
55	2002-03-20	1170.290039	1170.290039	1151.609985	1151.849976	1151.849976	1304900000	1115.382437
56	2002-03-21	1151.849976	1155.099976	1139.479980	1153.589966	1153.589966	1339200000	1115.868086

```
# checking the total number of rows of data

len(clean_data)
```

```
4711
```

For the linear regression used, two dependant variables will be used, namely the volume and the SMA to predict the closing price. Other inputs have been dropped to reduce overfitting. The linear regression model has been implemented using sklearn as seen below, and has yielded a RMSE of 135.59. Based on the below chart, we see that the predicted prices (in green) as compared to the actual prices (in orange) are generally in

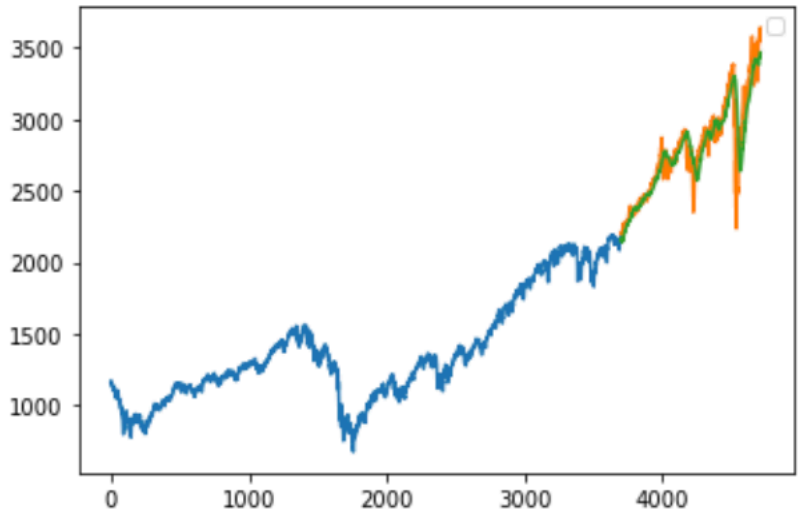
the right direction, albeit at different magnitudes. This RMSE will be compared against the LSTM model's RMSE to assess its effectiveness.

```
# implement linear regression
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train,y_train)
```

```
LinearRegression()
```

```
import numpy as np
#make predictions and find the rmse
preds = model.predict(x_test)
rms=np.sqrt(np.mean(np.power((np.array(y_test)-np.array(preds)),2)))
rms
```

```
135.58947263877997
```



III. Methodology

Data Extraction & Preprocessing

To start off, the ticker symbol, start and end dates were entered and converted to the specified format to generate the Yahoo Finance link. The extracted details in the csv file include the Open, High, Low, Close and Adjusted Close price as well as the Volume traded for the S&P 500 Index.

The final dataset after the pre-processing used for the LSTM model includes just the headers Date & Closing Price with a total of 4711 rows, the same as that used in the Linear Regression benchmark model.

```
new_data.head()
```

	Close
Date	
2002-03-15	1166.16
2002-03-18	1165.55
2002-03-19	1170.29
2002-03-20	1151.85
2002-03-21	1153.59

Train-test split

The general train test split recommended ranges from 70-90% but ultimately this is also dependant on the overall size of the dataset and the project itself. In this project, an 80% test train split was used for both the linear regression benchmark model and the LSTM model, which would approximate the first 3700 dates (out of 4711). As the data set is time-series based, the sequential order will matter as well and hence the split was performed with that in mind.

```
#creating train and test sets using a train-test split of approx 80%
dataset = new_data.values

train = dataset[0:3700,:]
test = dataset[3700:,:]
```

Implementation

LSTM was implemented using the Keras module with Tensorflow backend. The necessary libraries were first imported.

```
!pip install keras
!pip install tensorflow
```

```
#importing required libraries
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
```

Along with that was scaling using the MinMaxScaler function from the sklearn library. Though this project only covers the prices of one stock, best practice would involve scaling so as to enable neural networks to converge more quickly. Sequence generation was also performed to account for the chronological order of the stock prices.

```
#converting dataset into x_train and y_train
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)

x_train, y_train = [], []
for i in range(60,len(train)):
    x_train.append(scaled_data[i-60:i,0])
    y_train.append(scaled_data[i,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))
```

These are the details of the final LSTM model built. For each iteration, the RMSE was noted and compared to assess which would be the best model.

- 2 LSTM layers with 100 and 75 dimensions of hidden units used
- Drop out of 0.2 used per layer
- Final layer is the output dense layer with 1 neuron
- Batch sized used of 128
- Validation proportion of 0.1
- 25 epochs

Hyperparameters as used in building the model:

```
# create and fit the LSTM network
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=75))
model.add(Dropout(0.2))
model.add(Dense(1))
```

```
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_train, y_train, epochs=25, batch_size=128, validation_split = 0.1, verbose=2)

#predicting using past data from the train data
inputs = new_data[len(new_data) - len(test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs  = scaler.transform(inputs)

X_test = []
for i in range(60,inputs.shape[0]):
    X_test.append(inputs[i-60:i,0])
X_test = np.array(X_test)

X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
closing_price = model.predict(X_test)
closing_price = scaler.inverse_transform(closing_price)
```

Refinement ideas

i) More systematic approach of hyperparameter tuning

As weights are randomly initialised per run, different runs would show different results. Hence, one way to improve comparability across the different runs performing hyperparameter tuning would be to set a seed number for the initialised weights. This would help to reduce the randomness variability.

Another would be to look into automated hyperparameter optimization rather than doing it manually. This could be done with the use of the Talos library and would involve hyperparameter scanning with the use of different visualisations. By first having an overview of the correlations between the various hyperparameters and the performance metrics, further narrowing down and tuning can be done more efficiently and optimally.

ii) Inclusion of additional variables and inputs

Additional variables and features could be added to reduce overfitting as well as to enhance the model. This could include using the SMA as a feature of the LSTM model as well. Other macroeconomic variables that have a large impact on the overall economy (Bao, Yue & Rao, 2017) could also be included into the model.

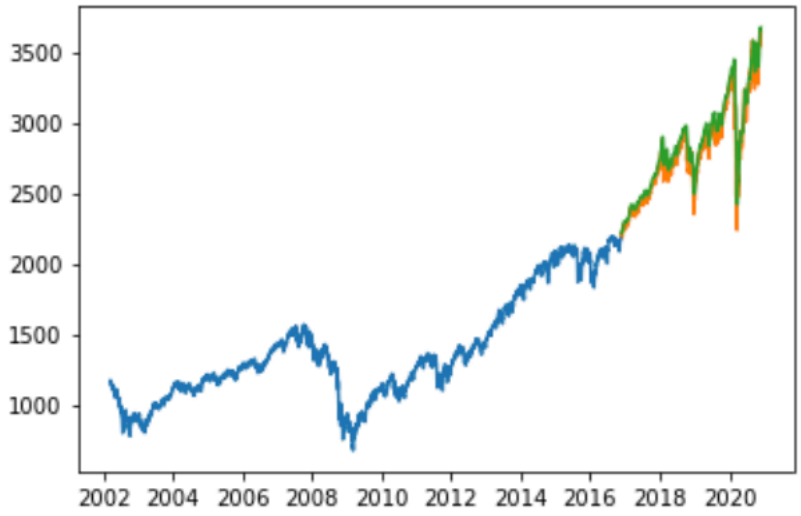
IV. Results

Model Evaluation and Validation

Based on the above hyperparameters for the LSTM model, an evaluation metric of 71.48 was yielded. Below is also a chart of the prediction results plotted against the actual ground truth prices. From the graph, the orange prices represent the actual prices while the green represent the predicted prices. Similarly, we note that the model has managed to predict the general direction of the prices relatively well.

```
rms=np.sqrt(np.mean(np.power((test-closing_price),2)))
rms
```

```
71.48414079543073
```



Justification

As compared to the linear regression benchmark model, the RMSE yielded by the LSTM model is lower, indicating that the LSTM is better able to predict stock prices. With further model enhancements to the LSTM model, there is potential for it to yield an even lower RMSE to become an even better model for stock price prediction.

V. Conclusion

In this project, an introductory attempt was made to predict the S&P 500 index prices using the LSTM model as well as the linear regression benchmark model. Though the scope was limited to just one index, there were considerable lessons learnt.

Initially, while first implementing the linear regression model, the variables used were just the data points extracted from Yahoo! Finance and this led to the overfitting of the model. With that in mind, the simple moving average was added on to enhance the model.

As for the LSTM model, further enhancements and extensions could be developed to make the model more robust. This includes implementing (PCA) which could help alleviate overfitting as well as other regularisation techniques. A larger training data set could also help to improve the model, expanding from just the S&P 500 index to including the component stocks of the index too. As mentioned, a more systematic way of performing hyperparameter tuning could also be incorporated to find the best LSTM model.

References

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780. doi:10.1162/neco.1997.9.8.1735

Bao Wei, Jun Yue, and Yulei Rao. “A deep learning framework for financial time series using stacked autoencoders and long-short term memory.” PloS one 12.7 (2017): e0180944.