CS 5100: Foundations of AI

Stock Prediction Through the Use of AI via ARIMA and LSTM models

Mohammed J. Amer and John O'Connell

Northeastern University, 360 Huntington Ave, Boston, MA 02115
amer.mo@northeastern.edu, oconnell.j@northeastern.edu

Abstract

In this paper, we present our findings from studying different AI algorithms and assessing their ability to predict stock market data. We looked at two separate time series forecasting models and compared the forecasted results against each other. The two models studied were an Autoregressive Integrated Moving Average, or ARIMA, model and a Long Short-Term Memory, or LSTM, model. The data used to study these models was the closing price of Apple stock on the New York Stock Exchange (NYSE) over a 10-year period. Based on our study we can conclude that an LSTM model far outperforms an ARIMA model for this type and volume of time series forecasting.

Introduction

The concept of automating a program to better interpret stock market performance is nothing new. However, in recent years, particularly during COVID-19, we have seen a growth in the average person's interest in the stock market. This increased interest has been even further impacted through the ease-of-use apps such as Robinhood, TD Ameritrade, etc. During this time, the appeal of quick monetary gains through information from social media, news updates, and online chat forums further provoked previously uninterested individuals to invest. That being said, post COVID we can see these particular forms of investing becoming less and less feasible. With interest waning and income becoming sparse, we believe the use of AI algorithms can be leveraged to provide a reasonable prediction of stock market performance.

Using historical data, mathematical models, and machine learning we aim to predict short term fluctuations within the New York Stock Exchange (NYSE) on a given day, assuming no major anomalies which impact the performance of a stock. Stock analysis can be broken down into two categories: fundamental analysis and technical analysis. Fundamental analysis examines the intrinsic value of a company, whereas technical analysis focuses on past and present market activity to predict future price movements. Due to its short-term nature and the type of data it analyzes (stock prices, volume of trading, etc.), we will be using a technical analysis approach in our attempt to predict future stock prices.

As a result of our decision to use a technical stock analysis approach, we decided that representing the stock data as time-series data would yield the best results. Time series data is a sequence of data points obtained through repeated measurements over time. There are many different AI algorithms that can be used to forecast time-series data. For our project, we have chosen two different models to compare against one another: Autoregressive Integrated Moving Average (ARIMA) and Long Short-Term Memory (LSTM).

Background

In order to better comprehend our approach, it is necessary to understand how both ARIMA and LSTM models work.

Autoregressive Integrated Moving Average is one of a handful of classic time series models that 'explains a given time series based on its own past values, that is, its own lags and the lagged forecast errors' (Prabhakaran S 2022). An ARIMA model combines three techniques – autoregression, moving average, and integration. It uses the past values of the variable in question (autoregression) as well as the forecasted error of past values (moving average) to predict future values. Additionally, ARIMA uses integration to eliminate any trends or seasonality effects from the data. There are three key terms which correspond to these three techniques that must be chosen to program an ARIMA model. The AR term (p) is the number of lagged, or prior, observations used in the model. The I term (d) is the order of differencing required to make the data stationary. ARIMA models work under the assumption that the data provided is

stationary, meaning that the mean and variance do not vary with time. Finally, the MA term (q) is the number of lagged, or prior forecast errors used in the model.

Long Short Term Memory models can be formatted in several different ways however to grasp a foundational understanding of the intricacy of this particular model we need to generally express how it functions. There will be a more in-depth discussion on the functions used to derive the model itself within the approach portion of this paper, however for now let us discuss the essentials to understanding neural networks and recurrent neural networks. Neural networks are an artificial layered structure of connected neurons. They are not comprised of a single algorithm but rather multiple, and the combination of these various algorithms allow the alteration and manipulation of complex data. Recurrent Neural Networks (RNNs) are one specific type of Neural Network. The neurons within a RNN have cell state/memory and input is processed according to this internal state. LSTM is a special case of RNN which is capable of learning long term dependencies in data. The combination of four layers interacting with each other allow for this achievement.

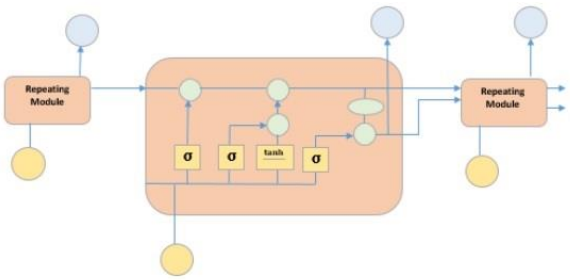

Fig. 1. The picture above depicts four neural networks (yellow boxes) point wise operators (green circles), input (yellow circles) and cell states (blue circles).

While this is a relatively naïve representation of an LSTM model the foundation of processing is easily represented. An LSTM model has a cell state and three gates which provides it with the ability to selectively learn, unlearn, and retain information. Each cell state provides the LSTM with information to flow through the units without alteration and containing few linear interactions. Each unit has an input, output, and a forget gate which can add or remove the information to the cell state. The forget gate uses a sigmoid function to decide which information from the previous cell state should be forgotten.

Related Work

With the allure of the stock market and potential gain of money, AI has been used frequently in an attempt to gain an edge in trying to project the market's performance. As we researched valid options it was clear that there are many applications which provide a service through utilization of these models. We concluded that ARIMA and LSTM would be the most interesting comparison. The traditional ARIMA model is widely considered to be a great stock prediction tool, and LSTM models can provide great predictive power. Although they are prone to not providing the greatest result, more recently deep learning methods have demonstrated better performance thanks to improved computational power and the ability of learning non-linear relationships enclosed in various financial features.

ARIMA falls into the category of classical time series models. This category includes Autoregressive (AR), Moving Average (MA), Autoregressive Moving Average (ARMA), Seasonal Autoregressive Integrated Moving Average (SARIMA), Seasonal Autoregressive Integrated Moving Average with Exogenous Regressors (SARIMAX), and of course ARIMA. After some research, we decided to go with the ARIMA model because it provides a good balance between complexity and accuracy in predicting time series data.

With respect to LSTM, Zhichao Zou and Zihao Qu, two professors from Stanford University, provide a detailed explanation of the differences between LSTM, Stacked-LSTM, and Attention-Based LSTM (Zhichao Zou and Zihao Qu, 2022). However, for our purposes a multi-layered LSTM model provided the most beneficial results. We determined that although there are a multitude of inputs that may potentially provide a more in-depth analysis or prediction, for the sake of comparison we focused only on the 'Closing' data per day of each stock. Given the size of the data, a multi-stacked LSTM model with 3 epoch runs provided a valid and sufficient model as opposed to a Stacked or Attention-Based LSTM model. Ultimately, we determined that a comparison between an ARIMA model and an RNN would provide the most interesting results.

Approach

For our comparison of the two models, we chose to look at 10 years' worth of Apple stock data. More specifically, we looked at the daily closing price of the stock during this time. We retrieved the data using the Python yfinance library, which offers a threaded and pythonic way of downloaded market data from Yahoo Finance. An overview of the data used can be seen in *figure* 2.

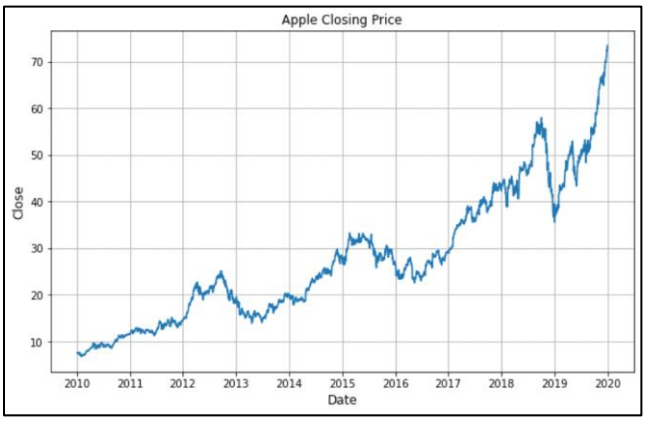

Fig. 2. Plot of Apple stock closing price over a 10-year period

Both models split the data into training and test sets, with 80% of the data used to train the models and 20% used to test their performance. This split can be seen in *figure* 3.

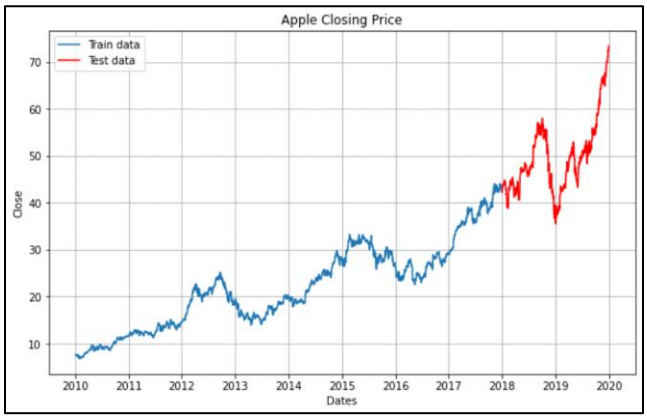

Fig. **3.** An 80-20 train to test split of the data

The first model we looked at was the ARIMA model. In order to program the ARIMA model, one must determine the value of the three terms (p, d, q) to be used. The first step in this process is to check the stationarity of the data, which will help us determine the required order of differencing. To get a sense of the trends and seasonality in the data, we can first plot the rolling mean as well as a seasonal decomposition of the data. These plots can be seen in *figure* 4 and *figure* 5, respectively.

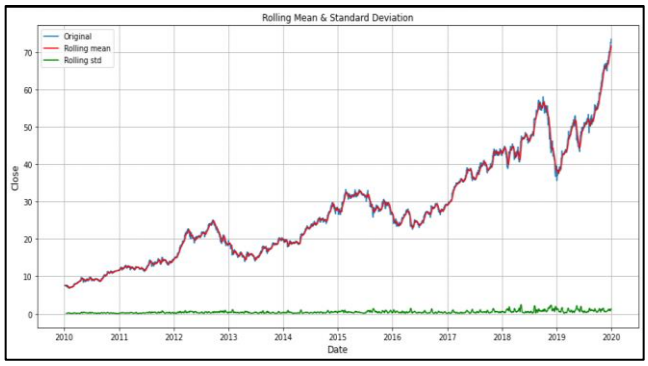

Fig. 4. Plot of rolling mean and standard deviation

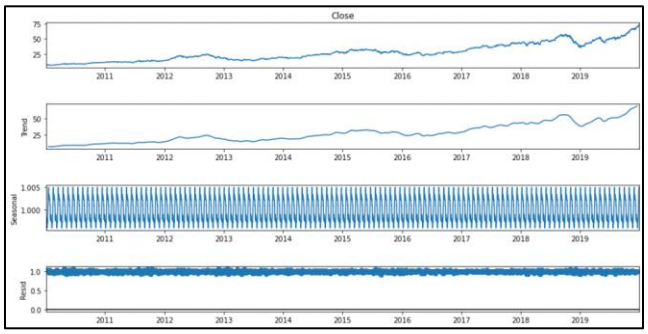

Fig. 5. Seasonal Decomposition Plot

From the rolling mean plot we can see that the mean is increasing over time. This is supported by the seasonal decomposition, where we can see an upward trend in the data as well as the presence of seasonality. In order to confirm these observations, an Augmented Dicky Fuller (ADF) test can be run on the data. An ADF test is a common statistical test used to determine whether a given time series is stationary or not by detecting the presence of a unit root in the data. It is a statistical significance test, meaning it gives results based on a null hypothesis. The null hypothesis of the ADF test is that the data is not stationary. When this is the case, the test will return a p-value greater than 0.5. On the other hand, if a p-value of less than 0.5 is returned we can reject the null hypothesis and the data is said to be stationary. The results of an ADF test on the raw data can be seen in *figure* 6 below.

```
Results of Dickey-Fuller Test:
Test Statistic                    0.929703
p-value                           0.993466
Lags Used                        30.000000
Number of Observations Used    3618.000000
Critical Value (1%)              -3.432159
Critical Value (5%)              -2.862339
Critical Value (10%)             -2.567195
dtype: float64
Cannot reject the Null Hypothesis: Data not stationary.
```

Fig. 6. Results of ADF Test on raw data

This test confirms our belief that the data is not stationary. In order to remove any trends or seasonality from the data we perform a first order differencing. This is done by computing the difference between consecutive observations in the data. For each observation y_t, we calculate $y_t = y_t - y_{t-1}$. The resulting data as well as the ADF test results for the differenced data can be seen in *figures* 7 and *figure* 8 respectively.

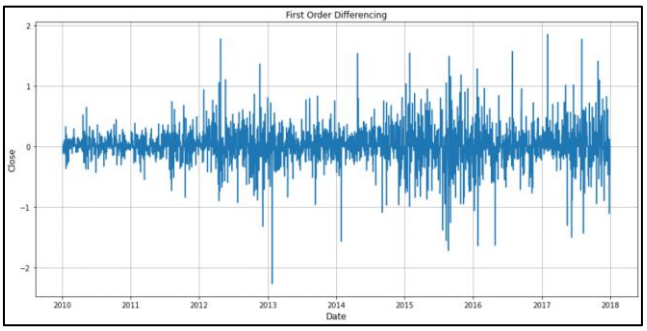

Fig. 7. Plot of data after first order differencing

```
Results of Dickey-Fuller Test:
Test Statistic                  -24.754714
p-value                           0.000000
Lags Used                         4.000000
Number of Observations Used    2913.000000
Critical Value (1%)              -3.432597
Critical Value (5%)              -2.862533
Critical Value (10%)             -2.567298
dtype: float64
Reject the Null Hypothesis: Data is stationary.
```

Fig. 8. Results of ADF Test on differenced data

The p-value returned from the ADF test on the differenced data is less than 0.5, so we can reject the null hypothesis and the data is said to be stationary. If first order differencing didn't produce stationary data, higher order differencing can be performed until the ADF test produces a p-value less than 0.5. The order of differencing necessary to detrend the data determines the d term of the ARIMA model. For our model, the d term will be set to 1.

Next, we must determine the AR (p) and MA (q) terms of the ARIMA model. This is done by plotting the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) of the differenced data. The ACF is the correlation between a time series and the lagged version of itself, while the PACF is the correlation between a time series and the lagged version of itself after excluding the contributions from intermediate lags. These "lollipop graphs" can be seen in *figure* 9 and *figure* 10.

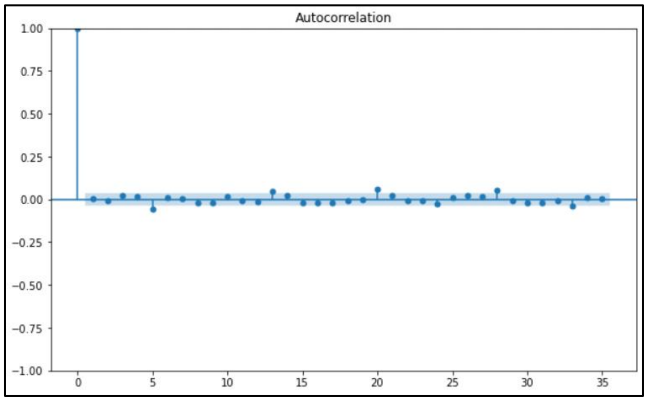

Fig. 9. ACF Plot

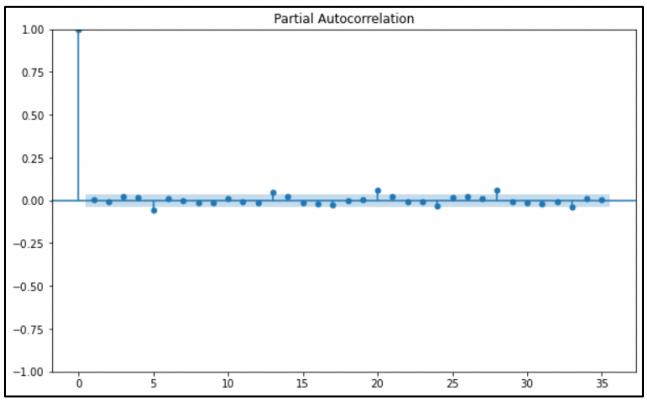

Fig. 10. PACF Plot

The blue shaded area in the graphs depicts the 95% confidence interval and is an indicator of the significance threshold. That means 'anything within the blue area is statistically close to zero and anything outside the blue area is statistically non-zero' (Monigatti L 2022). Typically, one or both of these plots follow a geometric decaying pattern which can help in determining the order of the AR and MA terms in the model. Unfortunately, our ACF and PACF are a bit ambiguous. We can however see that both models suggest a correlation at the 5th lag. We therefore decide to set an upper limit of 5 for both the AR and MA terms in the model.

The next step in creating the ARIMA model was to feed our parameters into the auto_arima function, which is part of the pmdarima package in Python. The auto_arima function takes in the d term, as well as min and max parameters for the p and q terms, and automatically finds the best fit ARIMA model for the data. It does so using the Akaike Information Criterion (AIC), a mathematical method for evaluating how well a model fits the data it was generated from. The results of the auto_arima call can be seen in *figure* 11 and *figure* 12.

```
#Fit ARIMA model to data
model_arima= auto_arima(df_train['Close'],trace=True,
                        error_action='ignore',
                        start_p=1,
                        start_q=1,
                        max_p=5,
                        max_q=5,
                        suppress_warnings=True,
                        stepwise=False,
                        seasonal=False)
model_arima.fit(df_train['Close'])
```

Fig. 11. Python call to auto_arima

```
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1022.487, Time=0.18 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=1024.462, Time=0.12 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=1026.317, Time=0.27 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=1026.714, Time=0.35 sec
 ARIMA(0,1,4)(0,0,0)[0] intercept   : AIC=1027.500, Time=0.48 sec
 ARIMA(0,1,5)(0,0,0)[0] intercept   : AIC=1021.288, Time=0.62 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=1024.463, Time=0.34 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=1026.463, Time=0.22 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=1028.439, Time=0.42 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=1028.499, Time=1.30 sec
 ARIMA(1,1,4)(0,0,0)[0] intercept   : AIC=1025.087, Time=1.39 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=1026.318, Time=0.15 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=1028.129, Time=1.35 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=1017.769, Time=1.86 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=1018.501, Time=2.05 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=1026.582, Time=0.21 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=1028.373, Time=0.67 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=1018.518, Time=2.26 sec
 ARIMA(4,1,0)(0,0,0)[0] intercept   : AIC=1027.469, Time=0.24 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=1024.962, Time=2.24 sec
 ARIMA(5,1,0)(0,0,0)[0] intercept   : AIC=1020.720, Time=0.28 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0] intercept
Total fit time: 17.000 seconds
```

Fig. 12. Results of auto_arima call

The best fit returned by the call to auto_arima was a ARIMA (2, 1, 2) model, meaning the AR (p) term is equal to 2, the I (d) term is equal to 1, and the MA (q) term is equal to 2. We now have a fully parameterized ARIMA model that we can use to predict future data.

The second model, LSTM, applies a multi-layer input sequence. For our model we utilized a sequential model which consists of linear stack of layers. This sequential model was imported from the *keras* library of *tensorflow*. An LSTM network computes a mapping from an input sequence x = (x_1, ..., x_T) to an output sequence y = (y_1, … y_r) by calculating the network unit activations of the following equations from t=1 to T:

$$it = \sigma(Wiixt + bii + Whiht - 1 + bhi)$$
$$ft = \sigma(Wifxt + bif + Whfht - 1 + bhf)$$
$$gt = \tanh(Wigxt + big + Whght - 1 + bhg)$$
$$ot = \sigma(Wioxt + bio + Whoht - 1 + bho)$$
$$ct = ft \odot ct - 1 + it \odot gtht = ot \odot \tanh(ct)$$
$$ht = ot \odot \tanh(ct)$$

"Where the W terms denote weight matrices (e.g. Wix is the matrix of weights from the input gate to the input), the b terms denote bias vectors (bi is the input gate bias vector), σ is the logistic sigmoid function, and i, f, o and c are respectively the input gate, forget gate, output gate and cell activation vectors, all of which are the same size as the cell output activation vector m". As mentioned previously, we separated the data obtained from the *yfinance* library into a training set and a test set. For the LSTM model, we performed the separation using a MinMaxScaler to normalize the data into a range from 0 to 1. We extract 80% of the closing price data from our acquired stock data to use as our training set, and use the remaining 20% as the test set. The train set is created via a 20 % window feature and reformat the feature data into a *Numpy array*, since that is the data format required by the Tensorflow.

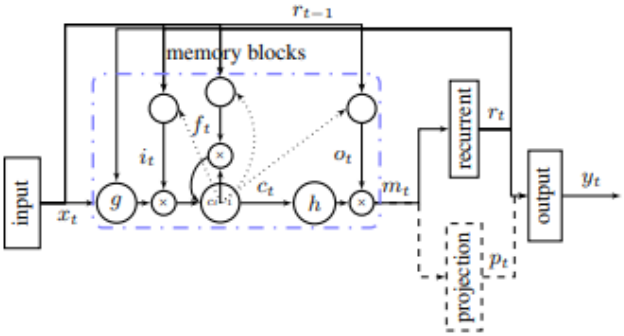

Fig. 13. LSTM based RNN architectures with a recurrent projection layer and an optional non-recurrent projection layer. A single memory block is shown for clarity.

Experiments and Results

We compared our two models by both graphing the results and looking at different forecast KPIs for the data.

The results from the ARIMA model as well as the relevant KPIs can be seen in the *figure 14 and 15*.

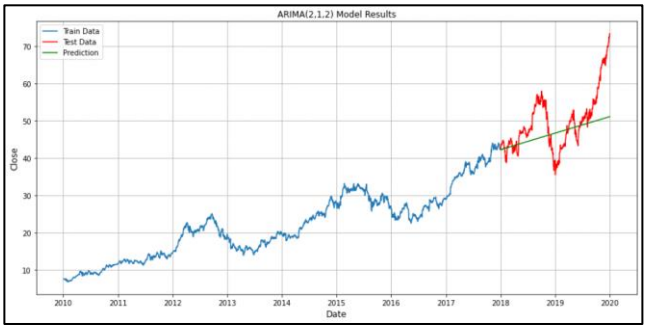

Fig. 14. Results of ARIMA model prediction

```
Mean Square Error ARIMA:  48.072328826429015
Mean Absoulute Percentage Error ARIMA:  0.09684883829128955
```

Fig. 15. MSE and MAPE Values of ARIMA model predictions

The forecasted data produced a Mean Squared Error of 48.07, which translates to a Root Mean Squared Error of about 7. This can be interpreted to show that model differed from the actual values by an average of 7 for each observation. Additionally, the Mean Absolute Percentage Error was about 9.68% meaning the model was only 90% accurate in its predictions. *Figure 14* shows that the model produced a linear prediction. While the prediction is somewhat accurate, there is room for improvement. We believe the observed result may be due to two different factors. First, we believe the data may have been over normalized when performing first order differencing. The extremely low p-value as well as the lack of any patterns in the ACF and PACF plots indicate the differenced data became similar to white noise, which is randomized data that is extremely difficult to predict. In order to test this hypothesis, we can look at different methods of removing trends from the data aside from differencing and see if another method produces more accurate predictions. Another potential reason for the linear prediction could be that there is still some seasonality to the data, even after differencing. In order to test this hypothesis, we can try to fit the data with a SARIMAX model, which is similar to an ARIMA model but also accounts for seasonality and exogenous factors.

With regards to the LSTM model, we utilized an LSTM layer with 100 network units and set the *return_sequence* to true so that the output layer will be of the same length. Next, we added another layer with the same network units and set this to false, which returned only the last output in the sequence. We found that having two densely connected neural networks, one with 25 network units and the other with 1, formatted the data properly.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60, 100)           40800

 lstm_1 (LSTM)               (None, 100)               80400

 dense (Dense)               (None, 25)                2525

 dense_1 (Dense)             (None, 1)                 26

=================================================================
Total params: 123,751
Trainable params: 123,751
Non-trainable params: 0
```

Fig. 16. Represents the summary of our LSTM network architecture.

```
Epoch 1/3
3968/3968 [==============================] - 49s 12ms/step - loss: 9.8598e-05
Epoch 2/3
3968/3968 [==============================] - 45s 11ms/step - loss: 4.7033e-05
Epoch 3/3
3968/3968 [==============================] - 45s 11ms/step - loss: 2.7788e-05
32/32 [==============================] - 1s 10ms/step
```

Fig. 17. Shows the Epoch values containing the loss values and time step.

Through the use of the Tensorflow library, we utilized the 'adam' optimizer and a loss representation of 'mean_squared_error'. *Figure 17* represents the loss rate when squared, producing the root mean squared error (RMSE). RMSE 'is a metric that tells us how far apart our predicted values are from our observed values in a model, on average' (Zach, 2020). For instance, as represented in *figure 17* the first Epoch produces a loss value of 9.859e-05 and a RMSE value of 0.0099. 'The number of epochs is the number of complete passes through the training dataset' (Brownlee J, 2022), meaning just one pass through already produces promising results. By Epoch 3 we get a RSME value of 0.005, which is well between 0 and 1 and shows that this is a valid and stable model.

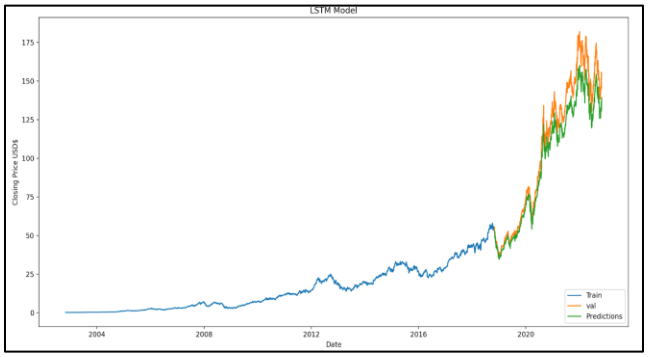

Fig. 18. LSTM Graph showing the model prediction data. The training data (blue) is set for 80% of the data set, the actual value of data (orange) and the prediction value of the LSTM model (green).

As represented in the *figure 18* graph above, LSTM performed remarkably well. With the RMSE deviation value of less than 1, we can see that the green projection line and the actual value of the data in orange are almost identical. There were some areas where the prediction model had a difference from the actual test data but those were instances of great volatility.

Conclusion

This project showcases the differences between ARIMA and LSTM models. Clearly, the LSTM model far outperformed the ARIMA model in this case. The ARIMA model forecasts produced an RSME value of close to 7, whereas the LSTM model's RSME was less than 1. The poor performance of the ARIMA model may be explained by the aforementioned potential over differencing or seasonality of the data, as well as the fact that ARIMA models are typically best at short term predictions. Given we used a timeseries containing 10 years of data and attempted to predict 2 years' worth of said data, LSTM had the opportunity to shine.

There were minimal instances of overfitting or overcomplication of the LSTM model. Unlike ARIMA, LSTM models do not rely on specific assumptions about the data such as time series stationarity. However, there is one common disadvantage of LSTM models. Because they are RNN based, difficulty can arise when interpreting and attempting to gain intuition into their behavior. That being said, if we were to apply these models to a smaller time series, we may expect the results to be reversed and the ARIMA model to perform better.

Finally, in terms of the breakdown of work our group decided to separate this project into 4 parts - creating and editing the data set/time-series information, implementing ARIMA algorithm, implementing the LSTM algorithm, and a final comparison of the algorithms. The initial pulling of data from yfinance, as well as the cleaning and formatting of the data was performed as a group. From there, each group member took responsibility for the implementation of one of the two algorithms, ARIMA (John) and LSTM (Mohammed). Once the implementation of the algorithms was complete, the group came together again and compared the two algorithms as a team, each member sharing their findings from their model analysis.

References

Bhandaria HN, Rimalb B, Pokhrelc NR, et al (2022) Predicting stock market index using LSTM. In: Machine Learning with Applications. https://www.sciencedirect.com/science/article/pii/S2666827022000378. Accessed 15 Dec 2022

Brownlee J (2022) Difference between a batch and an epoch in a neural network. In: MachineLearningMastery.com. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/#:~:text=batches%20and%20epochs.-,What%20Is%20the%20Difference%20Between%20Batch%20and%20Epoch%3F,passes%20through%20the%20training%20dataset. Accessed 14 Dec 2022

Fernandez J (2022) Creating an Arima Model for Time Series Forecasting. In: Towards Data Science. https://towardsdatascience.com/creating-an-arima-model-for-time-series-forecasting-ff3b619b848d. Accessed 14 Dec 2022

Hyndman, R.J., & Athanasopoulos, G. (2018) Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia. OTexts.com/fpp2. Accessed on 14 Dec 2022

Korstanje J (2022) How to select a model for Your time series prediction task [guide]. In: neptune.ai. https://neptune.ai/blog/select-model-for-time-series-prediction-task. Accessed 15 Dec 2022

Monigatti L (2022) Interpreting ACF and PACF plots for time series forecasting. In: Towards Data Science. https://towardsdatascience.com/interpreting-acf-and-pacf-plots-for-time-series-forecasting-af0d6db4061c. Accessed 14 Dec 2022

Nissa NK (2020) Stock Price Prediction Using Auto-ARIMA. In: Medium. https://nzlul.medium.com/stock-price-prediction-using-auto-arima-5569fcceae59. Accessed 14 Dec 2022

Prabhakaran S (2022) ARIMA Model - Complete Guide to Time Series Forecasting in Python. In: Machine Learning Plus. https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/. Accessed 14 Dec 2022

Sak H, Senior A, Beaufays F (2014) In: LONG SHORT-TERM MEMORY BASED RECURRENT NEURAL NETWORK ARCHITECTURES FOR LARGE VOCABULARY SPEECH RECOGNITION. https://arxiv.org/pdf/1402.1128.pdf.

Zou Z, Qu Z CS230 deep learning. In: CS230 Deep Learning. https://cs230.stanford.edu/. Accessed 15 Dec 2022

Zach (2020) How to calculate RMSE in python. In: Statology. https://www.statology.org/rmse-python/. Accessed 14 Dec 2022

Link to Code: https://github.com/John-OConnell/CS5100_Project