Stock Price Prediction Using Machine Learning

John Langenderfer, Addison Morgan, Dawin Ye, Andrew Shen

1 Background

This project aimed to develop a machine learning model capable of accurately predicting stock prices over time, a task that has always been challenging due to the complex and dynamic nature of financial markets. The stock market is influenced by numerous factors such as global events, economic indicators, investor sentiment, market shocks, and idiosyncratic risks, which are unique to individual companies or industries.

There are two primary approaches to stock predictions: fundamental and technical analysis. Fundamental analysts focus on qualitative aspects, while technical analysts rely on firm, price, and volume data. The Efficient Market Hypothesis, a widely discussed theory, suggests that stock prices accurately reflect all available information, making it very difficult to try to beat the market. However, quantitative hedge funds like TwoSigma and Renaissance consistently outperform the market, indicating that a competitive advantage may be attainable.

To address the complexities of stock prediction, our team leveraged advanced techniques in data preprocessing, feature engineering, and model selection. We aimed to build a predictive model to assist investors in making informed decisions and improving their investing performance. To mitigate the impact of idiosyncratic risk, investors often diversify their portfolios using a "basket approach," spreading risk across multiple assets.

Each team member chose a different model to work on, discussing and comparing the results. Dawin focused on linear regression, Andrew on random forest, AJ on SVR, and John on LSTM. Accurate stock price predictions can provide valuable insights for investors and traders, help them make informed decisions, construct effective trading strategies, and potentially automate the investment process.

There have been numerous studies in the past that have attempted to predict closing stock prices for the next day using various machine learning models. Some of these studies have reported promising results, demonstrating the potential of machine learning techniques in this domain:

**** INSERT PREVIOUS WORKS ***

In this work, we aim to build on these previous studies and develop a model to predict the closing day stock prices for the next day and attempt to make an efficient investing strategy.

2 Approach

Technical analysts operate on the fundamental assumption that market patterns exist and that upward and downward trends can be identified. Ideally, the most effective models should capitalize on these large-scale trends and accurately predict when price shifts will occur. Our team gathered price and volume data for Apple from Yahoo Finance, dating back to various time periods, as the

model's performance varied based on the timeframe. The data included each day's Open, High, Low, and Close prices, as well as the volume traded.

To ensure that the model could produce accurate results and use those outcomes as input for subsequent day's predictions, we opted to focus on the Closing Price as the primary input. We experimented with varying the number of days of closing prices fed into the models. This approach enabled us to use the predicted closing price as the most recent day's closing price, allowing the model to make predictions for the following day, and so on.

By refining our methodology and emphasizing the Closing Price, we aimed to enhance the model's overall effectiveness and reliability, ultimately providing valuable insights for investors and traders to make informed decisions in the ever-changing stock market.

3 LSTM

The LSTM (Long Short-Term Memory) model is a type of recurrent neural network (RNN) that can handle long-term dependencies by selectively retaining or forgetting information. The model achieves this by using three gates - input, output, and forget - that are controlled by sigmoid functions to determine how much information to keep or discard at each time step. The loss function used is mean squared error, and weights are adjusted using backpropagation through time.

The initial LSTM model used for this project is a single-layered stacked LSTM model with 128 LSTM units in the first layer and 64 units in the second layer. The model was compiled using the Adam optimizer and mean squared error as the loss function.

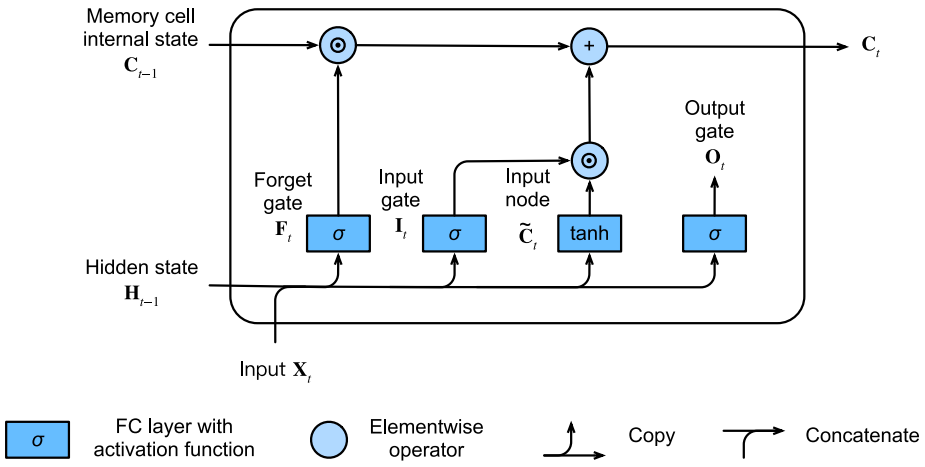

Figure 1: Single LSTM unit

Given a sequence of stock prices for three days, we want to use an LSTM to predict the stock price for the next day. The input vector x will have three elements (x_1, x_2, x_3) representing the stock prices for these three days.

In the case of our stock program, this vector would contain the 60 previous closing days, although to simplify the process, we will use 3 here.
The input vector, x_t:

$$x = \begin{bmatrix} 100 \\ 105 \\ 110 \end{bmatrix}$$

Assume we have a single LSTM layer with a hidden state size of 2. We will use made-up values for the weights and biases for illustration purposes.

$$W_f = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}, b_f = \begin{bmatrix} 0.5 \\ 0.6 \end{bmatrix}$$

$$W_i = \begin{bmatrix} 0.7 & 0.8 \\ 0.9 & 1.0 \end{bmatrix}, b_i = \begin{bmatrix} 1.1 \\ 1.2 \end{bmatrix}$$

$$W_o = \begin{bmatrix} 1.3 & 1.4 \\ 1.5 & 1.6 \end{bmatrix}, b_o = \begin{bmatrix} 1.7 \\ 1.8 \end{bmatrix}$$

$$W_C = \begin{bmatrix} 1.9 & 2.0 \\ 2.1 & 2.2 \end{bmatrix}, b_C = \begin{bmatrix} 2.3 \\ 2.4 \end{bmatrix}$$

The LSTM equations are given by:

$$\begin{aligned} f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\ o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ \tilde{C}_t &= \tanh(W_C[h_{t-1}, x_t] + b_C) \\ C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\ h_t &= o_t \odot \tanh(C_t) \end{aligned}$$

1. f_t: Forget gate - This controls how much information from the previous cell state (C_{t-1}) should be kept or forgotten. It is an element-wise multiplication, and its values are between 0 and 1, which determines how much of each component in the cell state should be retained.

2. i_t: Input gate - This controls how much of the new candidate cell state ($\tilde{C}_t$) should be added to the updated cell state (C_t). Like the forget gate, its values are between 0 and 1, which allows the LSTM to control the degree of contribution from the candidate cell state.

3. o_t: Output gate - This controls the proportion of information from the updated cell state (C_t) that should be passed on to the hidden state (h_t). The output gate values are also between 0 and 1, allowing the LSTM to modulate the information that is passed to the next layer or the next time step.

4. $\tilde{C}t$: Candidate cell state - This represents the new information that the LSTM cell can store in its cell state. It is a combination of the input vector (x_t) and the previous hidden state ($ht - 1$). The candidate cell state values can range between -1 and 1 due to the tanh activation function.

5. C_{t-1}: Previous cell state - This stores the information learned in the previous time step. The LSTM cell uses the forget gate (f_t) to decide how much of this information to keep or discard.

6. C_t: Updated cell state - This is the final cell state after combining the information from the previous cell state (C_{t-1}) and the candidate cell state ($\tilde{C}_t$). The input gate (i_t) controls how much of the candidate cell state contributes to the updated cell state.

7. h_{t-1}: Previous hidden state - This is the output from the previous time step that is fed back into the LSTM cell as input. It is used in conjunction with the input vector (x_t) to compute the candidate cell state ($\tilde{C}_t$) and the gate values (f_t, i_t, and o_t).

8. h_t: Updated hidden state - This is the output of the LSTM cell, which is passed to the next layer or the next time step. It is a combination of the updated cell state (C_t) and the output gate (o_t), allowing the LSTM to control the information passed on to the subsequent layer or time step.

Let's go through the LSTM cell step by step for our stock price vector x.

Forget gate (f_t) calculation: We calculate the forget gate values based on the previous hidden state h_{t-1}, the input vector x_t, and the forget gate weights W_f and bias b_f. For the first step, we can assume h_{t-1} is initialized to zeros.

$$f_t = \sigma(W_f x_t + b_f) = \sigma \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{pmatrix} \begin{pmatrix} 100 \\ 105 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.6 \end{pmatrix} = \sigma \begin{pmatrix} 21 \\ 42 \end{pmatrix}$$

$$i_t = \sigma(W_i x_t + b_i) = \sigma \begin{pmatrix} 0.7 & 0.8 \\ 0.9 & 1.0 \end{pmatrix} \begin{pmatrix} 100 \\ 105 \end{pmatrix} + \begin{pmatrix} 1.1 \\ 1.2 \end{pmatrix} = \sigma \begin{pmatrix} 84.1 \\ 105.2 \end{pmatrix}$$

$$o_t = \sigma(W_o x_t + b_o) = \sigma \begin{pmatrix} 1.3 & 1.4 \\ 1.5 & 1.6 \end{pmatrix} \begin{pmatrix} 100 \\ 105 \end{pmatrix} + \begin{pmatrix} 1.7 \\ 1.8 \end{pmatrix} = \sigma \begin{pmatrix} 147.2 \\ 168.3 \end{pmatrix}$$

$$\begin{aligned} \tilde{C}_t &= \tanh(W_C x_t + b_C) \\ &= \tanh \begin{pmatrix} 1.9 & 2.0 \\ 2.1 & 2.2 \end{pmatrix} \begin{pmatrix} 100 \\ 105 \end{pmatrix} + \begin{pmatrix} 2.3 \\ 2.4 \end{pmatrix} \\ &= \tanh \begin{pmatrix} 210.3 \\ 231.4 \end{pmatrix} \end{aligned}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t = f_t \odot 0 + i_t \odot \tilde{C}_t = i_t \odot \tilde{C}_t$$

$$h_t = o_t \odot \tanh(C_t)$$

The sigmoid function is used in the forget, input, and output gates because its output ranges from 0 to 1, which allows the gates to control the flow of information in a proportional manner. When the sigmoid function's output is close to 0, it blocks information flow, and when it's close to 1, it allows information to flow.

The tanh function is used for the candidate cell state and the final hidden state update because its output ranges from -1 to 1, providing a normalized representation of the internal values.

Now, let's assume the actual stock price for the next day is $y = 112$. We can calculate the loss using the Mean Squared Error (MSE) between the predicted value ($\hat{y} = h_t$) and the actual value (y): In a typical sequence prediction problem using an LSTM, the input data is a sequence of length T, and the model generates a sequence of T predictions. In such a case, N in the MSE loss function would be equal to the number of predictions made by the LSTM, which is T.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

In our stock LSTM algorithm, we a predicting the closing price for the next day based off the previous 60 days We then calculate the loss using the output from that, and the actual closing price In our case, we are predicting a single number, so the equation would be

$$\text{MSE} = (y_i - \hat{y}_i)^2$$

To update the weights and biases, we compute the gradients and adjust the weights using stochastic gradient descent:

$$\frac{\partial \text{MSE}}{\partial W_f} = \text{some value}, \frac{\partial \text{MSE}}{\partial b_f} = \text{some value}$$

$$\frac{\partial \text{MSE}}{\partial W_i} = \text{some value}, \frac{\partial \text{MSE}}{\partial b_i} = \text{some value}$$

$$\frac{\partial \text{MSE}}{\partial W_o} = \text{some value}, \frac{\partial \text{MSE}}{\partial b_o} = \text{some value}$$

$$\frac{\partial \text{MSE}}{\partial W_C} = \text{some value}, \frac{\partial \text{MSE}}{\partial b_C} = \text{some value}$$

In practice, the gradient calculations involve more complex partial derivatives, and we would use a deep learning framework to compute them automatically using backpropagation through time.

Using the computed gradients, we update the weights and biases with stochastic gradient descent:

$$W_f = W_f - \alpha\frac{\partial \text{MSE}}{\partial W_f}, b_f = b_f - \alpha\frac{\partial \text{MSE}}{\partial b_f}$$

$$W_i = W_i - \alpha\frac{\partial \text{MSE}}{\partial W_i}, b_i = b_i - \alpha\frac{\partial \text{MSE}}{\partial b_i}$$

$$W_o = W_o - \alpha\frac{\partial \text{MSE}}{\partial W_o}, b_o = b_o - \alpha\frac{\partial \text{MSE}}{\partial b_o}$$

$$W_C = W_C - \alpha\frac{\partial \text{MSE}}{\partial W_C}, b_C = b_C - \alpha\frac{\partial \text{MSE}}{\partial b_C}$$

Where α is the learning rate.

After updating the weights and biases, the LSTM would go through the sequence again, and the process would be repeated multiple times (epochs) until the loss converges to a minimum value.

3.1 Data Collection

The stock data was collected using the Yahoo Finance API in Python. The dataset includes the daily open, close, high, low, and volume data for the selected stocks. The dataset also includes additional features such as moving averages, MACD, and RSI. We did feature selection, although found just the closing price to work the best. For the LSTM model, it was trained on data for each of 30 different stocks since 2014, and then tested on the last year.

3.2 Training and Validation

The data was split into training and a validation set. The training set consisted of 89 percent of the data (2014-2022), while the validation set and test set each contained 11 percent (the last year). The model was trained on the training set, and the performance was evaluated using the validation set. Predictions were made using a sliding window of 60 days (SVR will be 20 days), meaning that each prediction was based on the previous 60 days of data. This approach was chosen to capture temporal dependencies in the data and improve the accuracy of the predictions.

3.3 Hyperparameter Tuning

Hyperparameter tuning was performed to optimize the performance of the models. Increasing the number of epochs was found to lead to overfitting, as the loss function quickly converged. A graph of the loss function over epochs is shown in Figure 2.

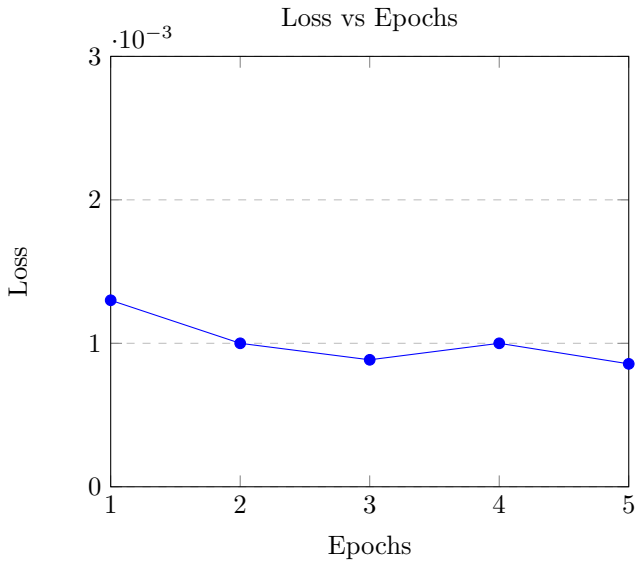

Figure 2: Loss values for each epoch

3.4 Data Preprocessing

The data was preprocessed to remove any missing values and to normalize the data. Normalization was performed to ensure that each feature was on the same scale and to prevent any one feature from dominating the others. The dataset was then split into training and testing sets. The training set consisted of the first 80 percent of the data, while the testing set consisted of the remaining 20 percent.

3.5 Model Architecture

Two types of LSTM models were implemented, namely, the normal LSTM and the bidirectional LSTM. Both models consisted of two hidden layers with 128 neurons each, followed by a dense output layer with a single neuron. The normal LSTM model takes in a sequence of data points and processes them sequentially, while the bidirectional LSTM processes the sequence in both directions.

Regularization techniques were used to prevent overfitting of the model. Dropout and L2 regularization were applied to both the normal and bidirectional LSTM models. Dropout randomly drops out a certain percentage of neurons during training, while L2 regularization adds a penalty term to the loss function to encourage smaller weights.

Additional features were added to the model to improve its accuracy. These features included the MACD (Moving Average Convergence Divergence) and RSI (Relative Strength Index). The RSI measures the strength of a stock's price action, while the MACD measures the difference between two exponential moving averages. These features were calculated using the `ta` (Technical Analysis) library in Python. The models were tested with the addition of these features, both with the initial LSTM architecture and with the bidirectional LSTM architecture with regularization. Some additional features were considered, although not implemented such as sentiment analysis from news sources, due to EMH.

Hyperparameter tuning was performed to optimize the performance of the models. Changing the number of epochs was found to lead to overfitting, as the loss function quickly converged. Dropout and L2 regularization were added to prevent overfitting. The number of hidden layers and neurons per layer was also experimented with.

3.6 Trading Algorithm

Our evaluation of the models was difficult to do. MSE did not provide much insight, except for which model to choose for the trading algorithm. Based off the nature of the LSTM, and as you will see with other models, it is better at predicting overall trends, so using MSE or seeing if they both moved in the same direction the next day turned out with poor results. Thus, we came up with this trading algorithm based off the next day predictions, and only implemented it with LSTM as it had the best trend predictions.

To test the effectiveness of the LSTM model, we created a trading algorithm that buys and sells stocks based on the predicted prices for the next day. The algorithm was tested on a list of 30 stocks from different industries over the past year, with the aim of producing a profitable trading algorithm.

The trading algorithm follows these rules:

- If the predicted price for the next day is greater than the current price, buy the stock.
- If the predicted price for the next day is less than the current price, sell the stock.

- If you sell and the next day's price is less, hold.
- If you buy and the next day's price is more, hold.

The output of the performance is given by the total profit or loss across the 30 stocks, assuming an equal distribution of 100 dollars per stock in the portfolio. Below is an example for TSLA, although this was done for 30 stocks over the last year and results shown below in the results section.

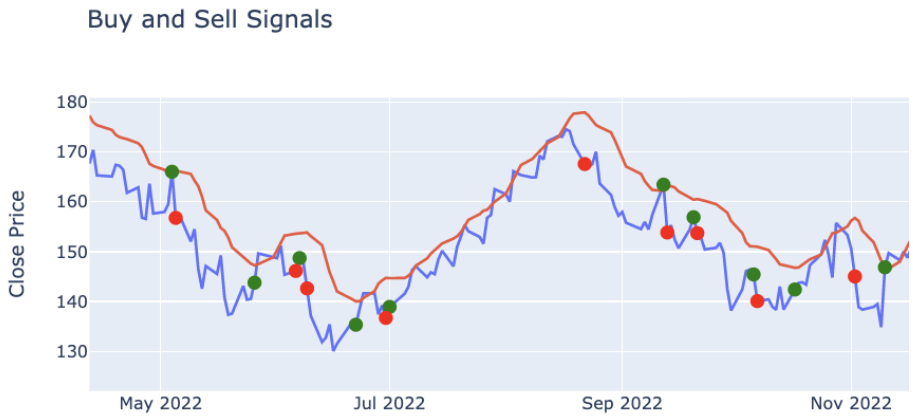

Figure 3: Example of trading algorithm for TSLA

4 SVR

For the SVR model, a different approach was needed than for the LSTM model, as Support Vector Machines can do better with less data given. Instead of its input data being 60 days, it worked better down to 20 days or less. It was also only using about two years or less of data in training and testing, unlike the years of data in the LSTM approach. This was to deal with overfitting, as using larger amounts of data caused it to heavily rely on the previous day's closing price for the prediction. On paper, that does not sound too bad, but the main problem was the output graph for the test data was almost the exact same as the real data, just shifted over a day, meaning the model was always a day late on the prices. Another way to deal with overfitting was to use Grid Search and find a good value for the C regularization parameter, as well as doing a grid search on the kernel and gamma parameters. These tended to be a linear kernel and the "scale" gamma parameter, which uses a gamma of $1/(num_features * Var(X))$. The C value changed the most, depending on the length and starting date, while the kernel and gamma stayed the same for the most part. As compared to a linear regression model given the same amount of data, the SVR model outperformed it up to about this length of data given, which was expected due to Support Vector Machines being a more useful model given less data.

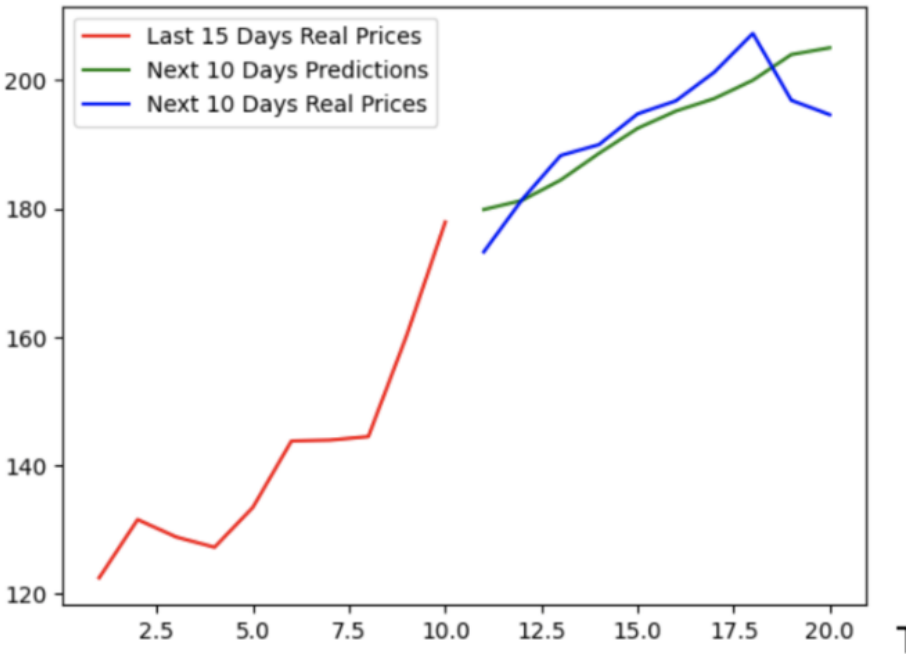

Figure 4:

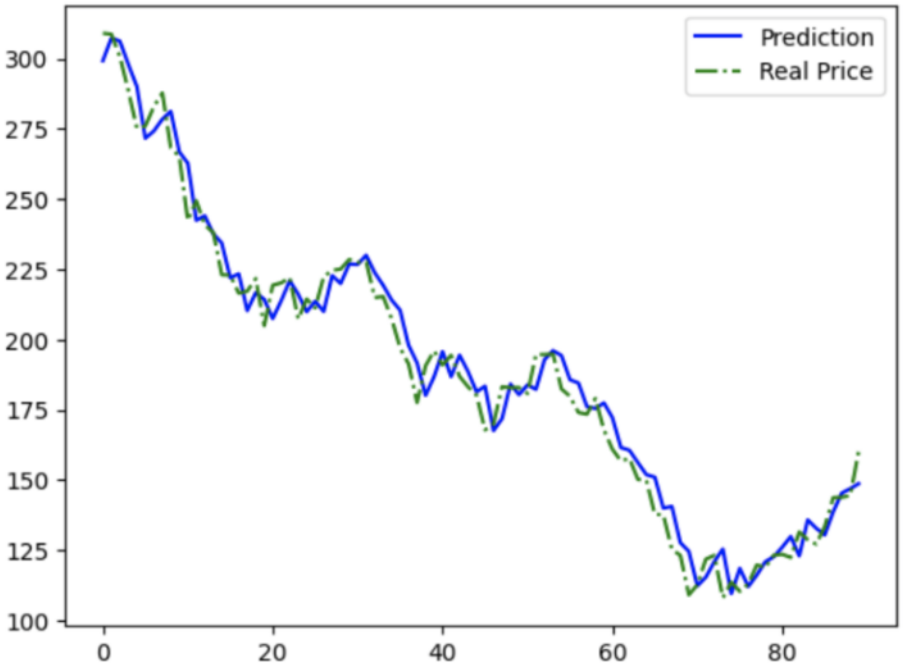

Figure 5:

It was able to follow along with the test data for the most part accurately, and when it came time to test the model feeding back into itself, it stayed close to the true data.

The model over these predicted days does follow along with the true prices, and loses its accuracy towards the end. This is due to the weight the model puts on the "day before" price that is its own prediction, so the further the prediction is from the last true date, the less the model is able to accurately predict. With this same data, the Linear Regression model had about the same test score, but its future predictions strayed further away from the true future prices.

5 Regressions

Here we employed two types of regression models for analysis and prediction, Ridge Regression and Linear Regression, which are implemented to predict future stock prices based on historical data and selected features. The choice of these models allows for a comparative analysis of their effectiveness and accuracy in predicting stock prices. The code acquires historical stock prices for FAANG companies as input data, including various features such as opening price, highest price, lowest price, closing price, trading volume, dividends, and other pertinent financial information. Ridge Regression, one of the models employed in the code, incorporates a regularization term to mitigate overfitting. The regularization term is a penalty applied to the coefficients' magnitude, which helps to shrink them and produce a simpler and more generalized model. By including this regularization term, Ridge Regression can perform better when dealing with multicollinearity or noisy data compared to the standard Linear Regression.

The code selects essential features for predicting stock prices, such as opening and closing prices, trading volume, and other financial indicators like moving averages, relative strength index (RSI), or historical volatility. These features are used as input variables for the regression models, enabling them to predict the target variable - future stock prices. The process of feature selection may involve techniques like correlation analysis, stepwise selection, or recursive feature elimination to identify the most relevant and useful predictors. The Ridge Regression model has a hyperparameter called 'alpha' that controls the strength of the regularization term. The code may include a process to find the optimal value of alpha to achieve the best model performance. This process can involve using cross-validation, a grid search, or a random search to test various alpha values and evaluate their performance based on predefined evaluation metrics. The optimal alpha value is chosen based on its ability to minimize the error or maximize the accuracy of the model's predictions.

The output generated by the code is a set of predicted stock prices for the FAANG companies using both the Ridge Regression and Linear Regression models. These predictions are derived from the input historical data and the selected features, and they provide insights into the expected future performance of the stocks. The code presents the results, which include the predicted stock prices and performance metrics such as Mean Squared Error (MSE), R-squared, Mean Absolute Error (MAE), and others, to evaluate and compare the Ridge Regression and Linear Regression models' effectiveness. By analyzing these performance metrics, users can understand the accuracy of the models and make informed decisions regarding investments in the FAANG companies. Additionally, the results may be visualized using graphs and charts to better understand trends, patterns, and the overall performance of the models in predicting stock prices.

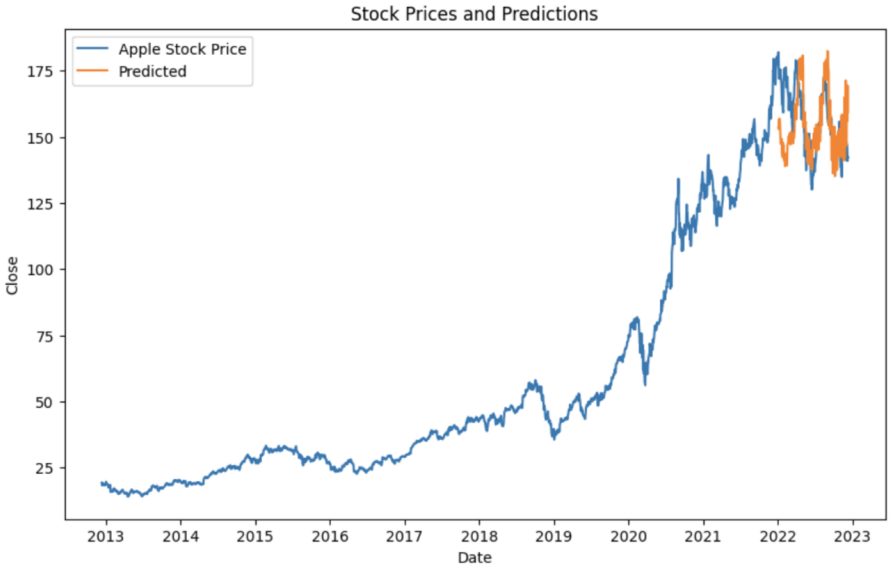

Figure 6: Regression Prediction.

6 Random Forest

The Random Forest model took another approach, trying to take advantage of the cyclical nature of stock price movement. The Random Forest model was run two different times using a different set of features for each. The first used historical data of previous days Open, High, Low and Close Price to predict the next day's closing price. Historical data allows the model to gain insight on trends and momentum in the market and stock, and future prices can recognize these movements in making its predictions. Our model takes the average Open, High, Low, etc price of the previous 10 days, but this as well can be adjusted to another metric. The biggest difficulty in training a Random Forest regression model was overfitting on the training data, as the decision trees should not get too hyper specific on the data it is being learned on. After careful experimentation, the model performed best at a depth of 4 with 160 estimators.

The model did not perform greatly on the test data, with a coefficient of determination of 75 percent and a mean-squared error above 29. Looking at the graph, the model fails to predict daily prices at an accurate rate, but it does match the longer term trends. The more nuanced shifts are not captured by this model. In predicting future prices, the model also did not perform very well, as the price predicted by the model differed rather drastically than the actual prices for the day. One issue with the prediction of this model was that it had a very limited forecast horizon. The model only outputs a closing price, but uses other features such as volume traded in the previous days when making its prediction. Therefore, if we tried to predict the closing price two weeks in the future, the model does not have information about the volume traded in the days prior to the prediction date, forcing it to use outdated information. Another model was therefore built using only the previous 10 days' closing price as a feature. This model actually performed much better, with a MSE of only 10 and a test score of over 90 percent. Moreover, the prediction horizon

Figure 7: Caption of the first image.

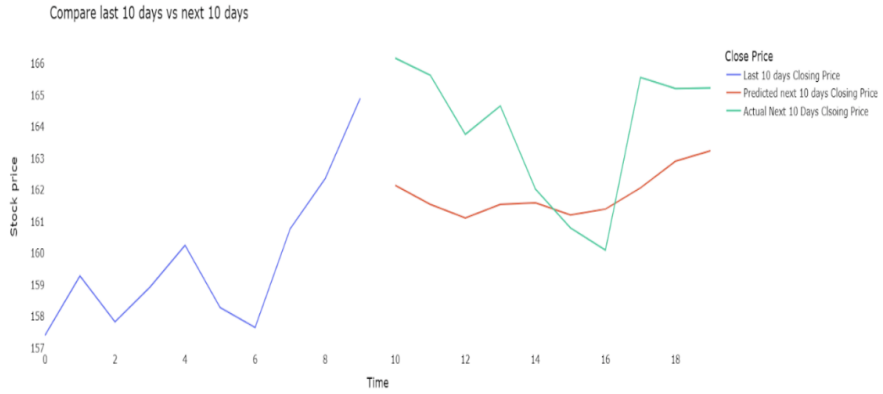

Figure 8: Caption of the second image.

could be expanded indefinitely, as it would generate new close prices that it would then use for prediction. The predictions for this model are still not extremely accurate, but they do follow the same movement pattern as the actual stock, which can help with buy/hold/sell decisions that investors make daily.

7 Conclusion

In conclusion, the stacked LSTM model was the best performing algorithm in this study, and investors may want to consider using it to make predictions about future stock prices. However, as with all investments, there is no guarantee of success, and investors should use caution when making investment decisions. Further studies could explore the use of leading indicators, such as sentiment analysis, to improve the accuracy of the predictions. Overall, this study demonstrates the potential of machine learning algorithms to provide valuable insights into future stock prices.

Figure 9: Caption of the first image.

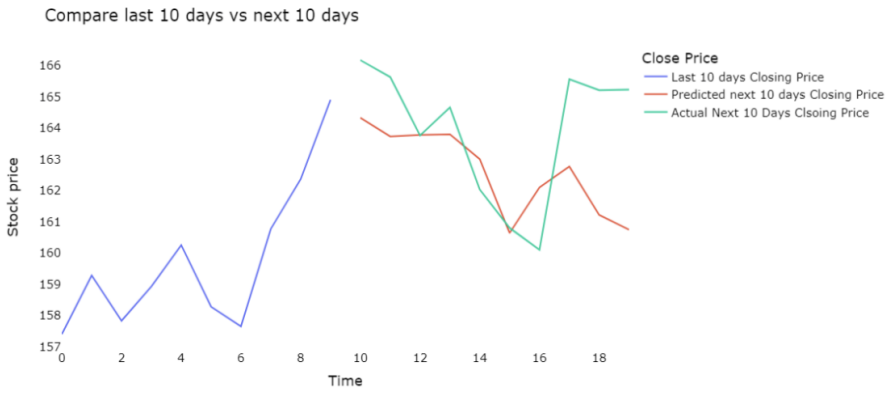

Figure 10: Caption of the second image.

	Predicted		Actual	
	Profit/Loss	% Gain/Loss	Profit/Loss	% Gain/Loss
Stacked				
Average	288.96	9.63%		
Max	546.95	18.23%		
Min	-6.7	-0.22%		
Bidirectional				
Average	185.16	6.17%		
Max	400.95	13.37%		
Min	8.58	0.29%		
Dropout + L2				
Average	112.67	3.76%		
Max	224.48	7.48%		
Min	26.95	0.90%		
Additional Features, MACD, RSI				
Average	178.86	5.96%		
Max	409.4	13.65%		
Min	-96.32	-3.21%		
Additional Feature Volume				
Average	159.27	5.31%		
Max	435.97	14.53%		
Min	-101.99	-3.40%		
Actual	-146.11	-4.87%	-146.11	-4.87%

Table 1: Summary of results for various models