Stock Price Prediction using LSTM

John Langenderfer
Boston College
`langenjo@bc.edu`

1. Background

Predicting stock prices has always been a challenging task due to the complex and dynamic nature of financial markets. One of the primary difficulties in predicting stock prices is the presence of idiosyncratic risk. Idiosyncratic risk refers to the risk associated with an individual stock, which is unique to that particular company or industry. Examples of idiosyncratic risk include company-specific news, earnings announcements, or changes in management.

To mitigate the impact of idiosyncratic risk, investors often diversify their portfolios by investing in a variety of assets, following a "basket approach." This approach involves holding a collection of different stocks, bonds, or other securities to reduce the overall risk of the portfolio. Diversification allows investors to spread their risk across multiple assets, reducing the potential impact of any single asset's poor performance.

The motivation for developing a machine learning project to predict closing day stock prices for the next day stems from the potential benefits of accurately predicting stock price movements. Accurate predictions can provide valuable insights for investors and traders, helping them make informed decisions regarding their portfolios. Furthermore, such predictions can be used to construct effective trading strategies or automate the investment process.

Long Short-Term Memory (LSTM) networks, a type of recurrent neural network, have shown promise in addressing this challenge. LSTM networks are designed to model complex temporal relationships in data, making them well-suited for time-series analysis, such as stock price prediction. By leveraging the capabilities of LSTM networks, researchers aim to improve the accuracy and reliability of stock price predictions.

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:

In this work, we aim to build on these previous studies and develop an LSTM-based model to predict the closing day stock prices for the next day.

2. Data

The data used for the LSTM and other models was obtained from Yahoo Finance using the `yfinance` library.

The specific metrics extracted from Yahoo Finance were initially only the closing prices. The data extracted covers the period since 2012.

To split the data into train and test sets, we used 89 percent of the data for training and testing over a period of one year, and 78 percent of the data for testing over a period of two years. The data was scaled using the `MinMaxScaler` from the `sklearn.preprocessing` library with the feature range of (0,1). The purpose of scaling the data is to normalize it so that it is more conducive for training and testing models.

Predictions were made using a sliding window of 60 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.

In future experiments, we will test the use of feature selection to include additional metrics such as volume, RSI, and MACD to improve the accuracy of the predictions.

3. Model

3.1. 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.

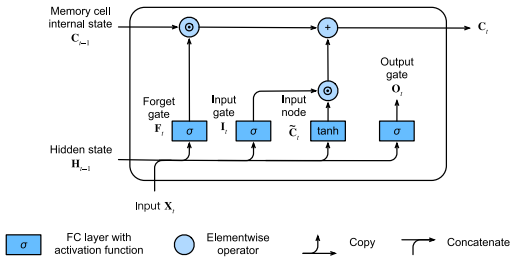

Figure 1. Single LSTM unit

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.

3.2. Bidirectional LSTM

In a bidirectional LSTM, there are two LSTMs that process the input sequence in opposite directions: one LSTM processes the sequence from the beginning to the end (forward LSTM), while the other processes the sequence from the end to the beginning (backward LSTM). The outputs of the two LSTMs are then concatenated to produce the final output.

The equations for a bidirectional LSTM can be derived by concatenating the forward and backward LSTM equations. Here are the equations for a bidirectional LSTM:

Forward LSTM equations:

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

Backward LSTM equations:

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

1. $f_t^{\rightarrow}$, $f_t^{\leftarrow}$: Forget gate values for the forward and backward LSTMs, respectively.
2. $i_t^{\rightarrow}$, $i_t^{\leftarrow}$: Input gate values for the forward and backward LSTMs, respectively.
3. $o_t^{\rightarrow}$, $o_t^{\leftarrow}$: Output gate values for the forward and backward LSTMs, respectively.
4. $\tilde{C}_t^{\rightarrow}$, $\tilde{C}_t^{\leftarrow}$: Candidate cell state values for the forward and backward LSTMs, respectively.
5. $C_t^{\rightarrow}$, $C_t^{\leftarrow}$: Updated cell state values for the forward and backward LSTMs, respectively.
6. $h_t^{\rightarrow}$, $h_t^{\leftarrow}$: Updated hidden state values for the forward and backward LSTMs, respectively.
7. $W_f^{\rightarrow}$, $W_f^{\leftarrow}$: Forget gate weights for the forward and backward LSTMs, respectively.
8. $W_i^{\rightarrow}$, $W_i^{\leftarrow}$: Input gate weights for the forward and backward LSTMs, respectively.
9. $W_o^{\rightarrow}$, $W_o^{\leftarrow}$: Output gate weights for the forward and backward LSTMs, respectively.
10. $W_C^{\rightarrow}$, $W_C^{\leftarrow}$: Cell state weights for the forward and backward LSTMs, respectively.
11. $b_f^{\rightarrow}$, $b_f^{\leftarrow}$: Forget gate biases for the forward and backward LSTMs, respectively.
12. $b_i^{\rightarrow}$, $b_i^{\leftarrow}$: Input gate biases for the forward and backward LSTMs, respectively.
13. $b_o^{\rightarrow}$, $b_o^{\leftarrow}$: Output gate biases for the forward and backward LSTMs, respectively.
14. $b_C^{\rightarrow}$, $b_C^{\leftarrow}$: Cell state biases for the forward and backward LSTMs, respectively.
15. x_t: Input vector at time step t.
16. $h_{t-1}^{\rightarrow}$, $h_{t+1}^{\leftarrow}$: Previous hidden states for the forward and backward LSTMs, respectively.
17. $C_{t-1}^{\rightarrow}$, $C_{t+1}^{\leftarrow}$: Previous cell states for the forward and backward LSTMs, respectively.

Compared to the equations for a standard LSTM, the equations for a bidirectional LSTM include additional terms that incorporate information from both the forward and backward directions. Specifically, the input and forget gates are computed based on both the previous output of the forward LSTM and the next output of the backward LSTM, and the candidate hidden state

3.3. Regularization

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.

3.4. Feature Selection

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. According to EMH, it is impossible to consistently achieve above-average returns by using news that is already publicly available. This means that news analysis, which involves analyzing news sources to predict market movements, would be ineffective in predicting the next day's closing price because the market would quickly adjust to any new information that is revealed. In other words, any new information that is revealed through news analysis would already be incorporated into the stock price, making it difficult to predict any further movements. Therefore, it is unlikely that news analysis would be an effective feature in predicting the next day's closing price.

3.5. Hyperparameter Tuning

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. A graph of the loss function over epochs is shown in Figure 2.

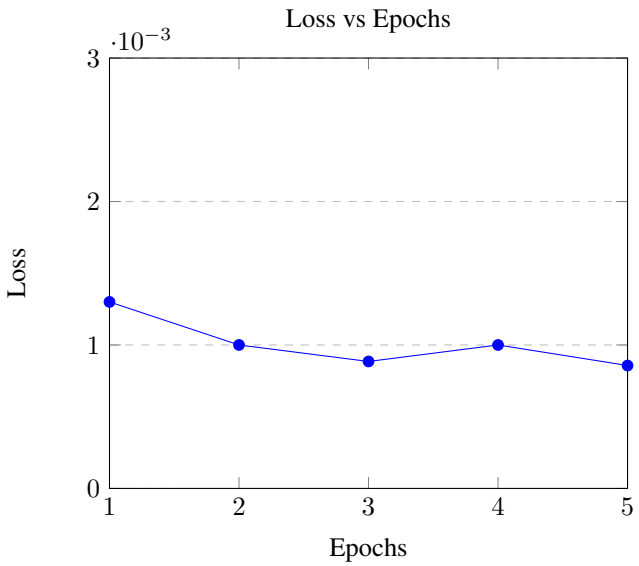

Figure 2. Loss values for each epoch

4. Results

4.1. Output of LSTM

The LSTM model takes an input vector of features including closing price, volume, MACD, and RSI, and produces an output vector of predicted prices. This model can be used to predict the closing price of the next day, which can then be used in real-world trading scenarios.

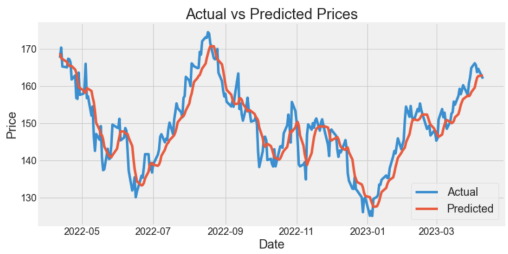

Figure 3. AAPL actual and predicted stock prices for the previous year

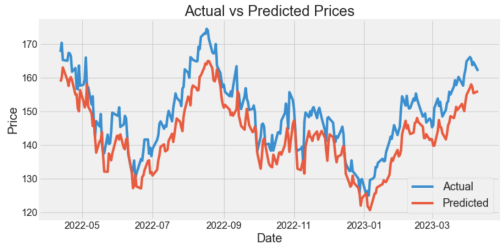

Figure 4. AAPL actual and predicted stock prices for the previous year, 5 epochs

During the hyperparameter tuning process, we halt the training at epoch 1 to prevent overfitting. It is important to note that when we extend the training to 5 epochs, the model appears to overfit the data. This observation can be particularly seen in the highlighted uptrend on the right of figure 4 compared to 3 As a result of overfitting, the model predicts slight negative trends within the overarching uptrend, which significantly impacts the performance of our investment strategy. Therefore, it is crucial to strike a balance in the training process to ensure accurate predictions and maintain the effectiveness of our investing approach.

4.2. Trading Algorithm

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.

Initially, we attempted to score the accuracy of the model by comparing the sign of the percent increase between the predicted price and the actual price for the next day. However, this method consistently resulted in an accuracy of around 50 percent. Therefore, we focused on analyzing the trends produced by the LSTM model to inform the 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.

Figure 5. AAPL buy and sell signals

4.3. Backtesting and Results

We tested the effectiveness of the LSTM model and trading algorithm using various techniques. The net profit/loss and percent gain/loss is produced by investing 100 dollars in each stock over the period of a year. There are 30 total stocks, which amounts to a total of 3000 dollars invested.

- Initial stacked LSTM model

- Bidirectional LSTM model

- Hyperparameter tuning

- Dropout and L2 regularization

- Additional features including MACD, RSI, and volume

- Combination of all the above techniques

Investing in stocks can be a challenging task, especially when it comes to predicting future prices. Stock prices are highly volatile, and predicting them accurately can be a difficult task. However, with the advent of machine learning algorithms, it is now possible to use past data to make predictions about future stock prices. The LSTM model used in this study is one such algorithm that can make use of past data to make predictions about future stock prices.

The results of this study showed that the stacked LSTM model performed the best, with an average gain of 9.63 percent and a maximum gain of 18.23 percent. The model was able to use the past data to make accurate predictions about future stock prices, and the results were impressive. The model was also able to handle the high volatility of the stock market and predict the prices accurately.

In contrast, the additional features that were added to the model did not perform as well as expected. This is because these features are lagging indicators and do not provide much information about the future prices. The study suggests that leading indicators, such as sentiment analysis, could be used in future studies to improve the predictions.

The dropout and L2 regularization model also produced interesting results. The model had a much lower average gain of 3.76 percent but had a higher minimum gain of 0.90 percent. This is because the model was more constrained, and there was less variance in the predictions. The model may be suitable for investors who are risk-averse and want to minimize their potential losses.

It is worth noting that predicting stock prices accurately is a difficult task, and even the best machine learning algorithms cannot guarantee accurate predictions. However, the results of this study show that machine learning algorithms, such as the LSTM model, can provide valuable insights into future stock prices.

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

Table 1. Summary of results for various models

5. 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.