Predicting stock prices using social network sentiment: An evaluative analysis of neural networks and time series models

Kevin Chen, David Chu, Carl Zhang, Jeanny Zhang

Abstract

Predicting stock prices is one of the most attractive problems in quantitative finance. We are interested in utilizing the predictive and computational power of neural networks in capturing the complexity and non-linearity of stock price data. Additionally, we believe that market sentiment correlates with how a specific stock performs. We extract that information by using deep-learning contextual language models trained on a large number of Tweets related to a certain stock. Combining past stock values and sentiment scores, we aim to make predictions on future stock prices using novel neural network architectures such as long short term memory and convolutional neural networks. Our modeling approach is shown to outperform traditional statistical time-series models such as the vector autoregressive model in predicting long-term stock prices. These results suggest that neural networks combined with sentiment data are more effective in forecasting trends in the stock market and can inform important decisions for financial institutions and investors.

1 Introduction

Sequence data is ubiquitous. We are naturally inclined to predict what comes next in the sequence using statistics and machine learning. Real-world uses of sequence modeling involve forecasting the next location of a ball, decoding audio signals, breaking down the text into characters or words, assessing medical readings, forecasting stock prices, and deciphering DNA sequences. We can use past stock prices and Twitter sentiments to predict the future movement of stock prices (See e.g. Wang and Yan (2021), Gandhi et al. (2021), Huang et al. (2020)). The presented research is not meant to be financial advice, but just a gentle exploration of modeling the stock price behavior using various time series and deep learning models that can account for market sentiments.

1.1 Vector Autoregressive Model

One of the goals we had, when we set out on this research, was to compare traditional statistical methods with newer, cutting-edge methods such as long short-term memory networks (Fischer and Krauss, 2018). The traditional statistical model we used is the vector autoregressive model (VAR) (See e.g. Ariyo et al. (2014), Hushani (2019)). Both of these methods can use past data to predict future stock prices and can also accommodate multiple variables to do so.

VAR is a linear regression model that uses multiple variables to predict the future value of a single stock. The VAR model is an extension of the autoregressive model (AR). An AR model is used to analyze the relationship between a variable and lagged versions of itself. More specifically, the output variable is a function of its previous values and an error term that accounts for the variation in the variable that cannot be explained by the model. Equation 1 below shows the general form of the model:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} \tag{1}$$

To use the model, we must specify the number of lagged values we want to include. This is known as the *order* of the model and is denoted p. After a lag has been chosen, the parameters of the model are then fit using maximum likelihood estimation (MLE). The VAR model generalizes the AR model to multiple variables, modeling each variable as a linear function of its own past values and the past values of the other variables in the model. As an example, Equation 2 below shows a VAR model of order 2 with 3 variables:

$$\begin{aligned}
y_{1,t} &= c_1 + \phi_{11,1} y_{1,t-1} + \phi_{12,1} y_{2,t-2} + \phi_{13,1} y_{3,t-1} + \phi_{11,2} y_{1,t-2} + \phi_{13,2} y_{3,t-2} + \phi_{13,2} y_{3,t-2} + \epsilon_{1,t} \\
y_{2,t} &= c_2 + \phi_{21,1} y_{1,t-1} + \phi_{22,1} y_{2,t-2} + \phi_{23,1} y_{3,t-1} + \phi_{21,2} y_{1,t-2} + \phi_{22,2} y_{3,t-2} + \phi_{23,2} y_{3,t-2} + \epsilon_{2,t} \\
y_{3,t} &= c_3 + \phi_{31,1} y_{1,t-1} + \phi_{32,1} y_{2,t-2} + \phi_{33,1} y_{3,t-1} + \phi_{31,2} y_{1,t-2} + \phi_{32,2} y_{3,t-2} + \phi_{33,2} y_{3,t-2} + \epsilon_{3,t}
\end{aligned} \tag{2}$$

Next, we conducted some preliminary exploratory analysis of the different variables to see if they met the necessary assumptions for the VAR model. First, we examined the stationarity of the variables. For each variable, we plotted the raw time series and the autocorrelation function (ACF). Since the variables Open, High, Low, and Close are highly correlated and follow the same pattern, we show just one of them in Figure 1:

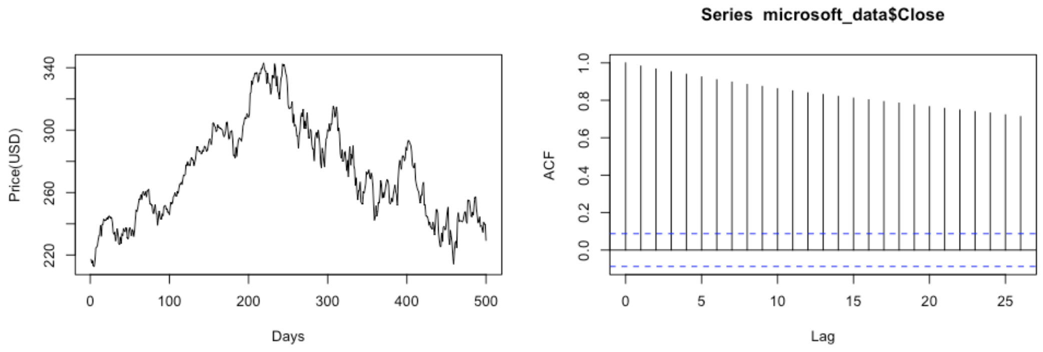

Figure 1: Plot of Microsoft closing price and ACF

Upon visual inspection, it's clear that the time series is not stationary: both the mean and the variance do not appear to be constant. And looking at the ACF, we can see that the correlation decreases extremely slowly as lag increases, which indicates non-stationarity. We then used the Augmented Dickey-Fuller (ADF) test on each of Open, High, Low, and Close to confirm that they are all non-stationary. To transform the variables to stationarity, we perform a first differencing. Below are plots showing one of the variables after the differencing in Figure 2:

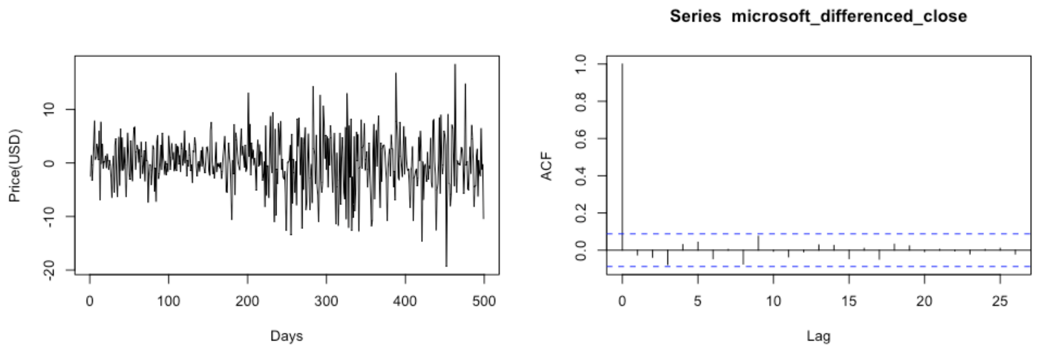

Figure 2: Plot of differenced Microsoft closing price and ACF

Upon visual inspection, the time series appears stationary: the mean and variance remain constant throughout. And examining the ACF, the correlation drops steeply after lag 0, a sign of stationarity. To confirm, we conducted the ADF test again, which confirmed that they were all stationary. Performing our analysis on the differenced data also allows us to avoid another potential assumption violation, which is that variables cannot be perfectly collinear. We checked the correlation between Open, High, Low, and Close, and we found that they are extremely correlated (correlation coefficients around 0.99). Although not perfectly correlated, this is high enough to where issues may arise. After differencing, there is no longer a concern, as the correlation coefficients became between 0.6 and 0.8. Observe the pairwise correlation plots of those differenced variables below in Figure 3:

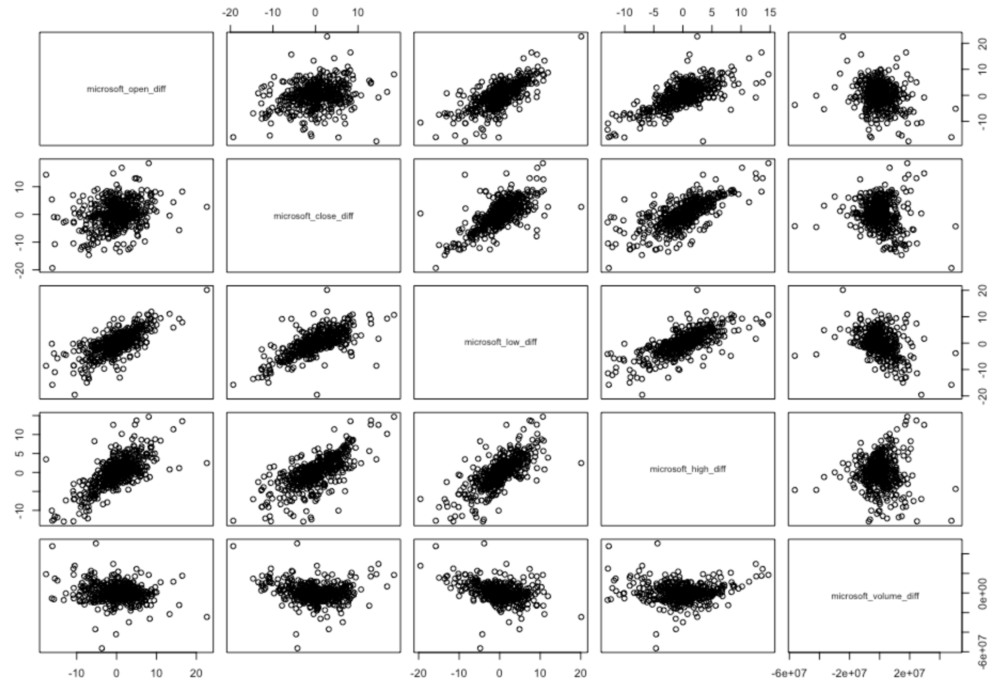

Figure 3: Correlation plots of differenced Microsoft variables

1.2 Neural Networks

A neural network is a type of machine learning algorithm inspired by the brain. Brains are made up of billions of cells called neurons, which are all connected to each other in a complex network. Similarly, neural networks are composed of layers of *neurons*, each of which receives information, processes it (performing some mathematical function on it), and transmits it to the next layer. There are several types of layers found in neural networks. The input layer is the layer that receives the input, and the number of neurons in the input layer is determined by the size of the input data. In our case, it's the stock variables. The output layer is the layer that contains the final output, and similarly, the number of neurons in the output layer depends on what the network is meant to predict. In our case, it's the stock variables. Finally, neural networks usually contain one or more *hidden layers*, which are responsible for learning the relationships between the input and the output.

We were interested in utilizing neural networks to capture trends in the Microsoft stock data due to the nonlinear nature of the data. As discussed in the section introducing VAR, each of the variables in the VAR model is modeled as a linear combination of its past values in addition to the past values of the other variables in the model. Many real-world phenomena are complex and don't follow a simple linear pattern; we might be able to make better predictions using neural networks.

1.3 Convolutional Neural Network

The main goal of the Convolutional Neural Network(CNN) model in sentiment analysis is to capture overall sentence sentiment through word groupings and context within a sentence. We were inspired to use CNN models based on some recent literature which utilized this kind of model to capture sentiment within tweets (See e.g. Liao et al. (2017)). To do this, there are convolutional kernels that slide over a list of inputted word embeddings and this list makes up the sentence being analyzed.

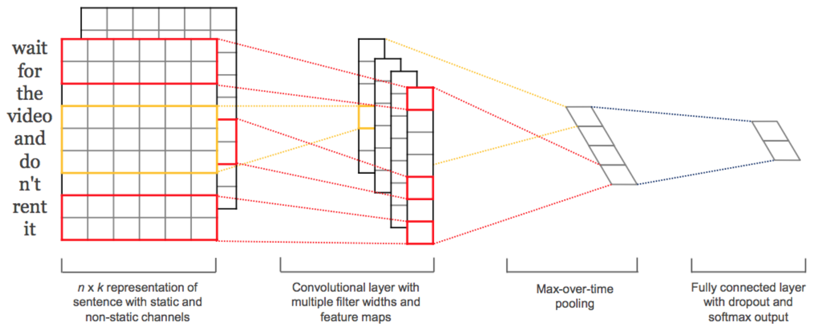
wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 4: Illustration of CNN Layer (Camacho, 2019)

Kernels of set heights(corresponding to the number of words being looked at) slide over the list and output features(values quantifying the sentiments of the words being looked at). For example, with a convolutional kernel of height two, the kernel will slide across the list of inputted word embeddings (a matrix of dimensions number of words in the sentence by features in each word embedding) looking at all of the two adjacent words in the sentence. In each instance that the kernel looks at two words, the calculations done between the kernel and the word embedding features is one similar to a matrix-wise product and the sum of weights in the kernel and values in the words. The corresponding entries of each matrix are multiplied together and added with the rest of the values.

After this process, the output feature vector has one column and a number of rows for a number of words in the sentence. The convolutional kernel essentially transforms the original input into a smaller matrix that captures the context of adjacent two words. This same process is carried out with kernels of multiple heights in order to understand the relationships between words that are not immediately adjacent to each other. The heights of these convolutional kernels have to be optimized or carefully chosen in the context of the application. The use of multiple kernels is permitted. Using height greater than 10, for example, the CNN could be analyzing words that are too far apart that are irrelevant to each other and this may make our model computationally complex.

Lastly, from this feature vector, we take the highest value to get the final sentence's sentiment score. This is known as the max-pooling layer, which is an abstract way to represent a sentence's sentiment by assuming the highest value is the most important value. There are many ways to pool from the feature vector, but max pooling tends to work best in image processing and text sentiment analysis in CNN.

1.4 Long Short Term Memory

A long-short-term memory (LSTM) is a special kind of recurrent neural network (RNN), which solves the vanishing gradient problems in neural networks by being capable of remembering previously learned information and making predictions based on that information. The *look-back* period in an LSTM refers to the number of previous time steps of a time series that are used as input to predict the next time step. Having a *look-back* period enables LSTMs to capture historical trends and patterns in the times series, making them particularly useful for tasks that involve sequential data, such as time series forecasting, language translation, and speech recognition (See e.g. Chu et al. (2022)).

An LSTM model has a chain-like structure consisting of layers of nodes. Unlike an RNN model that has one activation function in each node, an LSTM model has four interacting layers in each node as shown in Figure 5. The four interacting layers can be thought of as gates that control the flow of information within the node such as which information to keep and which information to forget. In addition, there is a cell state, which is a vector that stores information from the previous cell, and a hidden state, which is a vector that stores relevant information from all previous cells. The cell state and the hidden state are updated at each time step based on the input and the output of the gates. The four interacting layers work together as follows:

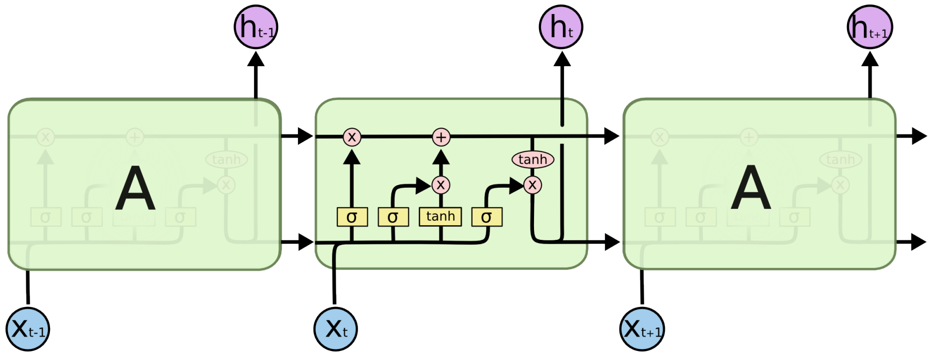

Figure 5: Illustration of LSTM Cell (Colah, 2015)

The first layer is called the forget gate, which uses a sigmoid activation function to determine which pieces of the current cell state should be forgotten. Values come out between 0 and 1; the closer to 0 means to forget, and the closer to 1 means to remember. The second layer is called the input gate which uses another sigmoid activation function that decides which pieces of the current input should be added to the cell state. The third layer is another input gate. It uses a tanh activation function that outputs values between -1 and 1 to help regulate the network. The output from the forget gate is multiplied by the previous cell state and the output from the second and third layers are added together which updates the cell state to new values. The fourth layer is called the output gate which uses a sigmoid activation function and a tanh activation function to decide what information the hidden state should carry. The new cell state and the new hidden state are then carried over to the next time step. The output of the LSTM node is a combination of the updated cell state and the output produced by the output gate. This output is fed back into the node as input for the next time step, allowing the LSTM to retain information from the past and use it to make predictions about the future.

Univariate LSTM

The most basic type of LSTM is the univariate LSTM, which is a type of model when the data being modeled has a single input variable. Based on the predetermined *look-back* period, the model utilizes previous times steps of the times series as input to predict the next time step. The univariate LSTM is able to capture the complex nonlinear patterns and long-term dependencies in the time series (See e.g. Brownlee (2018)). It is also computationally efficient and requires fewer parameters than a multivariate LSTM model.

However, there are some limitations to the univariate LSTM model. One major limitation is that it can only take into account a single variable, which may be insufficient to thoroughly capture the complex relationships and interactions that may exist between multiple variables in the stock market. This could lead to overfitting, which eventually leads to large bias and variance in out-of-sample prediction.

Furthermore, the univariate LSTM model also assumes that the underlying statistical properties of the time series remain constant over time, which may not always be the case. This can result in the model failing to adapt to sudden changes or unexpected events that may impact stock prices in subsequent time steps of the prediction. Therefore, a more sophisticated model is called for that can take into account the complex interactions and interdependencies among multiple variables in the stock market.

Multi-Step Multivariate LSTM

The multi-step multivariate LSTM model is a variant of the LSTM architecture that is specifically designed to handle data with multiple input variables and make predictions for multiple future time steps. To make multi-step predictions, the output of the model at each time step is fed back into the model as input for the next time step. This enables the model to use its own predictions as input for future predictions.

Unlike the traditional LSTM model that only considers a *look-back* period, the multi-step multivariate LSTM model also incorporates a *look-out* period into its prediction process. The model first takes in a pre-defined number of past time steps for all variables in the time series and utilizes the LSTM architecture to capture the temporal dependencies between these variables. The model then generates a set of predictions for a predetermined number of future time steps for all variables in the time series (See e.g. Brownlee (2018)). The *look-back* and *look-out* periods are important hyperparameters that can significantly impact the performance of the multi-step multivariate LSTM model.

Similar to a standard multivariate LSTM model, multi-step multivariate models can capture complex dependencies between multiple variables in the time series. Moreover, the multi-step multivariate LSTM models have the advantage of producing more accurate predictions for multiple future time steps, thereby providing a more comprehensive view of the time series. However, there are some potential drawbacks to using multi-step multivariate LSTM models. Firstly, they can be computationally expensive to train, especially if the dataset is large and complex. Secondly, since they are designed to predict multiple future time steps, they may be less accurate at short-term predictions than simpler models such as univariate LSTMs. Thirdly, there are more hyperparameters in multi-step multivariate LSTMs that require careful tuning in order to make the models effective.

2 Methodology

2.1 Framework

Our research is concerned with forecasting stock prices utilizing social network sentiment and aims to provide an evaluative analysis of neural network and time series models. To achieve this objective, our research was divided into three distinct parts. The entire workflow described above can be found in Figure 6 below.

The first part of the research consisted of constructing a CNN-LSTM model for the purpose of generating sentiment scores based on Twitter data. We first scraped Microsoft-related tweets from Twitter. We used the *sentiment.ai* R package (Institute, 2021), which is equipped with a pre-trained machine learning model for sentiment analysis, to generate sentiment scores for the Microsoft tweets. The Microsoft tweets and their sentiment scores were then fed into the CNN-LSTM model as training and testing data, with the aim of enabling the model to specialize in analyzing the contexts of tweets related to Microsoft, the company and the stock. Subsequently, the trained CNN-LSTM model was deployed to generate sentiment scores for tweets pertaining to Microsoft. To evaluate our sentiment scores, we manually examined a subset of our tweets with their predicted scores to assess how close they are to our intuition.

The second part of the research aimed to construct a multi-step Multivariate LSTM by combining sentiment scores from Twitter with stock price data from Yahoo Finance. This model was designed to leverage the strengths of both social network sentiment and stock price data to generate more accurate stock price predictions. In the third part of our research, we developed two distinct models, namely a VAR model and a univariate LSTM, both of which exclusively employed stock price data. These models were to serve as benchmark models to provide comparisons with the multi-step multivariate LSTM. Through this comparative analysis, we aimed to evaluate the effectiveness of combining social network sentiment with stock price data in predicting stock prices.

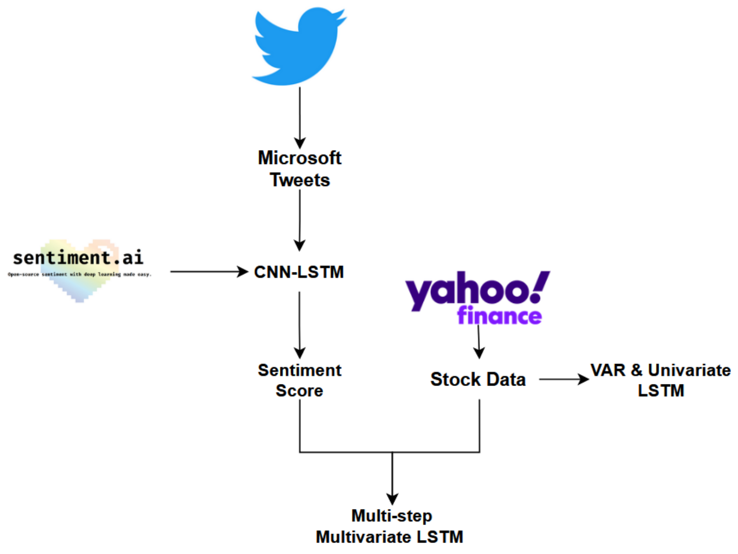

Figure 6: The workflow of our research

2.2 Data

Tweets

Using the Twitter API, we queried all Tweets from January 11th, 2021 to January 5th, 2022 that mentioned the keywords “MSFT” or “Microsoft”. Our Tweets were extracted with the following configurations: English, from verified users, and do not contain retweets. The scraped data were sanitized using regular expressions to remove: non-ASCII characters, URLs, HTML characters, and emojis. Our final data contained 451,306 tweets.

Stock Data

The stock data are collected from the Python package, *yfinance* (Aroussi, 2017), using the ticker symbol, *MSFT*. The following variables are tracked daily for each stock and are shown in Table 1:

Variables	Description
Open	Stock price when trading began that day.
Close	Stock price when trading closed that day.
High	Maximum stock price that day.
Low	Minimum stock price that day.
Volume	The number of shares traded on that day.

Table 1: Variables and their description.

One of our primary goals was to be able to make accurate predictions for the price of a stock. Therefore, the stock we chose must be relatively stable and follow typical market trends. Being an established, large company, Microsoft fits these criteria. In addition, one of our main goals was to incorporate sentiment scores into our predictions. Given that a large portion of Microsoft’s products are directly customer-facing, there is likely a lot of public discourse about the company. Therefore, the sentiment would likely be an insignificant factor in the price of the stock. Compare this to a company such as the defense contractor Lockheed Martin, about which very few ordinary people talk about.

Selecting Microsoft as our stock also allowed us to extract our sentiment features more easily; we needed to scrape Tweets to gather public sentiment on the stock and the company as a whole. Microsoft was a good choice for this, as it allowed us to avoid problems that we encountered with other companies. For instance, when we tried to use Amazon, many of the tweets we encountered were not related to the discussion of the company; many were related to selling a product. Similarly, when we tried to use Meta, many tweets were not actually discussing the company and were instead using the word for discussing other topics. When we scraped Microsoft tweets, however, most of the tweets were actually indicative of public sentiment toward the company.

2.3 VAR Prediction

As described in the introductory section for VAR, we transformed the variables to stationarity by performing a first differencing. After doing so, we split the data into two sets: *train* (490 days) and *test* (30 days). For a model we are considering, we fit the model on the *train* data, perform a forecast on the “test” set, and then compare that forecast with the true data for those 30 days to check the accuracy. We use the Bayesian Information Criterion (BIC) as the metric for model selection. We decided to use the BIC over other metrics such as the Akaike Information Criterion (AIC) after repeating the following procedure using AIC and found that the models deemed optimal by the BIC performed better with respect to forecast accuracy. To select the order of the model, we fit VAR models up to a large number of lags(12) and then select the model that minimizes the BIC.

In our case, the optimal lag was 2. After fitting the model, we perform a 30-day ahead forecast, inverse transform the first difference, and compare our results with the real data. We evaluate the predictions of the closing price using the Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) as metrics.

2.4 Sentiment Score Prediction

In order to enhance the prediction of stock prices besides using previous stock values, we utilize sentiment analysis to extract the public sentiment regarding a particular company. We hypothesize the sentiment derived from social media forums like Twitter is highly predictive of how the stock market changes. We get the pseudo-labels for each Tweet which are sentiment scores from fitting a sentiment.ai model - an acceptably efficient, pre-trained sentiment analysis model. These scores were used in the initial iteration of the modeling process.

After tokenizing the Tweets, the first layer of our model converts the word tokens into word vectors. A word vector is a numerical representation of a word that captures its meaning and relationships with other words. Using word vectors allows our model to perform arithmetic operations efficiently and in a meaningful manner. We used a word embedding layer to convert the text tokens into unique word vectors. Using a customized word embedding fine-tunes our algorithm to fit the task and the data that we worked with.

Our neural network model incorporates convolutional neural network layers and Long Short Term Memory layers. This combination is effective in capturing the linguistic and semantic context of natural language and can account for the non-linear nature of text data. The CNN layer extracts important features from the feature space. The LSTM layer learns the context of faraway words which decides which to forget and remember. Finally, the dense layer condenses the dimension of the output and captures the non-linearity in our model to output sentiment scores. We split our data into a training set and a test set (60:40). We used the test set to tune the hyperparameters to optimize the performance of the model. We also performed several human evaluations on a subset of the data to ensure the sentiment scores agreed with our tuition. Our implementation of this model is based on the *keras* package in R.

Figure 7 shows the workflow described above.

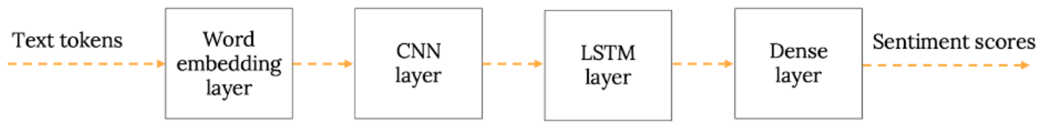

Figure 7: The architecture of the sentiment analysis model

2.5 Stock Price Prediction

To conduct a comprehensive evaluation of the utility of social network sentiment in predicting stock prices, we developed four distinct LSTM models. These models included a univariate LSTM, a multi-step multivariate LSTM without sentiment scores, a multi-step multivariate LSTM with mean sentiment scores, and a multi-step multivariate LSTM with median sentiment scores. In order to generate the mean and median sentiment scores, we first collated all sentiment scores for each day and then calculated the average and median values. The four LSTM models, along with their respective input data columns, are presented in Table 2.

LSTM Model	**Input Data Columns**
Univariate	*Close*
Multi-Step Multivariate 1	*Open*, *High*, *Low*, *Close*, *Volume*
Multi-Step Multivariate 2	*Open*, *High*, *Low*, *Close*, *Volume*, *Mean Sentiment Score*
Multi-Step Multivariate 3	*Open*, *High*, *Low*, *Close*, *Volume*, *Median Sentiment Score*

Table 2: Stock Price Prediction LSTMs and Their Input Data

We employed the use of several Python packages including *Keras* (Chollet et al., 2015), *scikit-learn* (Pedregosa et al., 2011), and *TensorFlow* (Abadi et al., 2015) to develop the aforementioned models. The pre-processing stage involved scaling the data to a range between 0 and 1 via the implementation of a min-max scaler. In order to divide the dataset into train and test datasets for both the x and y variables, we implemented the training-testing-split rule of 75/25, with a *look-back* period of 5 and a *look-out* period of 10. Following the optimization of the hyperparameters of the models, we proceeded to predict the stock prices for the subsequent 30 time steps, or days. The results of these predictions were then compared to the actual values to calculate the respective prediction errors.

2.6 Hyperparameter Optimization

The optimization and the selection of hyperparameters play an important role in the performance of an LSTM. The following part gives an overview of how we chose the tuning parameters. Hyperparameters are preset variables before the training process that determine the behavior and performance of the machine learning model. Some main hyperparameters include epochs, batch size, activation function, optimizer, the number of hidden layers and units, and dropout rate as detailed in Table 3.

Hyperparameter	**Definition**
Epoch	Number of times the entire training dataset is passed through the network during training
Batch size	Number of training examples utilized in one forward/backward pass of the network during training
Activation function	Mathematical function applied to the output of a neuron or group of neurons to introduce non-linearity and enable the network to learn complex patterns in the input data
Optimizer	Algorithm used to adjust the weights and biases of the network during training to minimize the error or loss function and improve the accuracy of the predictions
Hidden layers	Layers of artificial neurons between the input and output layers
Units	Artificial neurons in a given layer
Dropout rate	Probability of randomly dropping out or ”deactivating” a given neuron during training, which helps to prevent overfitting

Table 3: Hyperparameter Definitions

Setting appropriate hyperparameters is often a process of trial and error that requires a significant amount of experimentation to determine the optimal values. Search algorithms are often used to systematically search for the best hyperparameters for a machine learning model such as grid search and random search. Due to the constraint of computing power and time, we were not able to tune every hyperparameter. Instead, we arbitrarily select some hyperparameters and use search algorithms to tune the rest of the hyperparameters.

To optimize the CNN-LSTM model for the sentiment analysis process, we utilized grid search. Because of our large text data, we can only optimize 2 more prioritized hyperparameters that we believe are fundamental to how the model performs: the number of dense units and the dropout rate. Because of the time and computational constraints, we used grid search on a short range of parameter space in hope of hitting the optimized set of hyperparameters. As shown in Table 4, the parameter space for the number of dense units is a sequence of 10 elements from 16 to 512, and the parameter space for the dropout rate is a sequence of 10 elements from 0 to 0.1.

Hyperparameters	**Value Range**
Dense Layer Units	[16.00, 71.11, 126.22, 181.33, 236.44, 291.56, 346.67, 401.78, 456.89, 512.00]
Dropout rate	[0.00, 0.011, 0.022, 0.033, 0.044, 0.055, 0.066, 0.077, 0.088, 0.100]

Table 4: CNN-LSTM Hyperparameters

To optimize the LSTM models for stock price prediction, we decided to use a random search algorithm to systematically search for the best values for a number of selected hyperparameters shown in Table 5. Instead of exhaustively searching through all possible combinations, the random search algorithm randomly selects values for the hyperparameters from a predefined range for a fixed number of iterations. This makes the optimization process faster and the resulting model architecture more promising (Thormann et al., 2021).

Hyperparameters	**Value Range**
Number of Dense Layer	[0, 1, 2, 3]
Dense Layer Activation Function	[”relu”, ”tanh”, ”sigmoid”]
Number of Dropout Layer	[1, 2, 3, 4]
Hyperparameters	**(Min Value, Max Value, Step)**
1st LSTM Layer Units	(32, 512, 16)
2nd LSTM Layer Units	(32, 256, 16)
Dense Layer Units	(32, 512, 16)
Dropout Layer Rate	(0.05, 0.2, 0.05)

Table 5: Multi-Step Multivariate LSTM Hyperparameters

We used a random search with 10 maximal trials and 50 executions per trial. The predetermined hyperparameters include 500 epochs, a batch size of 16, and a validation split of 0.25. The optimizer utilized in the models is Adam, while the loss function used is a mean absolute error (MAE). In the univariate LSTM model, a single LSTM layer is followed by an optimized number of dropout and dense layers. In contrast, the multi-step multivariate LSTM models contain an LSTM layer followed by an optimized number of dropout and dense layers, and an additional LSTM layer.

3 Results and Discussion

Now, we present the final results we obtained during this project. First, we show the final set of optimal hyperparameters we discovered for our CNN-LSTM model and for our multi-step multivariate LSTM. Then, we show and discuss the forecast accuracy of each of the models we tested.

3.1 Optimal Hyperparameters

Table 6 below shows the optimal set of hyperparameters we obtained for the CNN-LSTM model using grid search.

Hyperparameters	**Value**
Dense Layer Units	16
Dropout Rate	0.011

Table 6: Optimal CNN-LSTM Architecture

Next, we show the optimal set of hyperparameters we obtained for the multi-step multivariate LSTMs using random search in Table 7. They are ordered in the same way as they appear in the model.

Hyperparameters	**Value**
1st LSTM Layer Units	336
1st Dropout Rate	0.1
Dense Layer Units	256
Dense Layer Activation Function	Tanh
2nd Dropout Rate	0.05
2nd LSTM Layer Units	80

Table 7: Optimal Multi-Step Multivariate LSTM Architecture

3.2 Prediction Results

Table 8 below shows the average 30-day-out prediction performance over five runs for each of our models.

Model	**MAE**	**MAPE**
VAR	21.29	8.7
Univariate LSTM	6.86	2.85
Multi-Step Multivariate LSTM w/o Sentiment	6.63	2.80
Multi-Step Multivariate LSTM w/ Mean Sentiment	5.98	2.51
Multi-Step Multivariate LSTM w/ Median Sentiment	6.11	2.67

Table 8: Model Performances Using MAE and MAPE (in percent) Averaged over Five Runs

Comparing the performance of our LSTM models to the performance of our VAR model, we can see that our LSTM models outperform our VAR model by a significant margin. If one observes the difference between row 1 and any of the rows pertaining to a LSTM model, the accuracy from the VAR forecast is significantly less accurate. The VAR model requires a lot of assumptions: the relationship between variables is linear, and the variables themselves are stationary. In predicting stock prices, whose path is highly non-linear and driven by a variety of macroeconomic factors, some of these assumptions might not be satisfied. Thus, we expected the LSTM model to perform considerably better than the VAR model.

Comparing the performance of our LSTM model with sentiment vs. without sentiment, we can see that our model with sentiment outperforms the model without sentiment by a small margin. If one observes rows 3-5 in Table 8, we can see that the accuracy of the Multivariate LSTM prediction does improve when sentiment is added as a feature, but it's not a very large difference. This difference could be more pronounced with a longer period of out-of-sample prediction and smaller time steps of data measurements (hourly instead of daily stock data). We conclude that more investigation must be done to test the predictive power of Twitter sentiment data on the stock price prediction.

4 Limitations and Future Work

There are several notable limitations in our research. First, we planned to evaluate the effect of sentiment data on multiple companies' stock prices. However, due to time constraints, we were only able to look at Microsoft. Looking at multiple stocks would help us cross-validate our results across multiple companies which would give more reliable results. Secondly, the process we used to clean the tweets was not perfect and resulted in some Tweets containing irrelevant information or noisy data. This might result in inaccurate sentiment scores, which would negatively affect the accuracy of our predictions.

The performance of our model could have also been improved by implementing more granular data into our model and predictions. If we used hourly data instead, perhaps we could have captured short-term trends with our prediction more effectively. Likewise, we could have more to analyze the difference between sentiment and lack of sentiment in our model. The differences have the possibility to be more pronounced with data with smaller time steps.

As for future work, we plan to explore the use of more advanced sentiment analysis techniques. In particular, the Transformer-based language model, a kind of neural network architecture used for natural language processing tasks, is one of the more cutting-edge big language models. They are referred to as "transformer" models since they process input sequences via self-attention processes as opposed to recurrent connections. Because of its parallelizability, increased training effectiveness, and superior performance on a number of benchmark datasets, the Transformer architecture has recently taken over as the preferred method for many applications. The BERT (Bidirectional Encoder Representations from Transformers) language model, created by Google, is the most well-known transformer-based language model. We think this new approach will enhance our capacity to predict market sentiment because of its complexity and sophisticated capabilities.

References

Abadi, M. et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Ariyo, A. A., Adewumi, A. O., and Ayo, C. K. (2014). Stock price prediction using the arima model. In *2014 UKSim-AMSS 16th international conference on computer modelling and simulation*, pages 106–112. IEEE.

Aroussi, R. (2017). Download market data from yahoo! finance's api. Package available from https://github.com/ranaroussi/yfinance.

Brownlee, J. (2018). *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery.

Camacho, C. (2019). Cnns for text classification.

Chollet, F. et al. (2015). Keras.

Chu, N., Dao, B., Pham, N., Nguyen, H., and Tran, H. (2022). Predicting performances of mutual funds using deep learning and ensemble techniques. *arXiv preprint arXiv:2209.09649*.

Colah (2015). Understanding lstm networks – colah's blog. Website available from https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Fischer, T. and Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European journal of operational research*, 270(2):654–669.

Gandhi, U. D., Malarvizhi Kumar, P., Chandra Babu, G., and Karthick, G. (2021). Sentiment analysis on twitter data by using convolutional neural network (cnn) and long short term memory (lstm). *Wireless Personal Communications*, pages 1–10.

Huang, J., Chai, J., and Cho, S. (2020). Deep learning in finance and banking: A literature review and classification. *Frontiers of Business Research in China*, 14(1):1–24.

Hushani, P. (2019). Using autoregressive modelling and machine learning for stock market prediction and trading. In *Third International Congress on Information and Communication Technology: ICICT 2018, London*, pages 767–774. Springer.

Institute, K. F. (2021). sentiment.ai: Open source ai-based sentiment analysis. Package available from https://github.com/BenWiseman/sentiment.ai.

Liao, S., Wang, J., Yu, R., Sato, K., and Cheng, Z. (2017). Cnn for situations understanding based on sentiment analysis of twitter data. *Procedia computer science*, 111:376–381.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Thormann, M.-L., Farchmin, J., Weisser, C., Kruse, R.-M., Säfken, B., and Silbersdorff, A. (2021). Stock price predictions with lstm neural networks and twitter sentiment. *Statistics, Optimization & Information Computing*, 9(2):268–287.

Wang, Y. and Yan, G. (2021). Survey on the application of deep learning in algorithmic trading. *Data Science in Finance and Economics*, 1(4):345–361.