# Predicting Stock Prices in the New York Stock Exchange

Are Support Vector Machine algorithms or Long Short Term Memory networks more effective in predicting stock market price trends in the New York Stock Exchange?

Fleming Yip

Table of Contents

## 1. Introduction

The stock market has long been an elusively intriguing phenomenon to professional investors, scholars and the common folk alike. The factors influencing the prices of the thousands of stocks listed just on the New York Stock Exchange are innumerable and varied, with each playing a different role. These include macroscopic events, like political events and business and investor outlook, and microscopic variables such as specific company statistics. As a result, it is almost impossible to accurately and consistently predict the trends of the stock market. Recently, advancements in machine learning have allowed humans to "train" computers with past data in predicting the stock market. A Support Vector Machine (SVM) is a type of classification algorithm that uses supervised learning, while the LSTM network is a recurrent neural network that is widely used in sequence prediction. This paper will seek to answer the question, **are Support Vector Machine algorithms or Long Short Term Memory networks more effective in predicting stock market price trends in the New York Stock Exchange?**

This investigation will use the sklearn library to apply the SVM model. The second part of the investigation will employ the TensorFlow library to create the neural network. It is hypothesized that the Long Short Term Memory network will be more effective in predicting stock prices, as it is capable of having a long-term “memory” that it can draw on to make predictions. In addition, the time period will have a positive correlation with the accuracy score of both models.

### 1.1 Historical Context

Investors and programmers have devised numerous tactics and programs in an attempt to automate stock trading. Due to the volatile nature of stock prices, researchers have, for example, looked towards using moving averages as a method to overlook the short-term fluctuations. As a

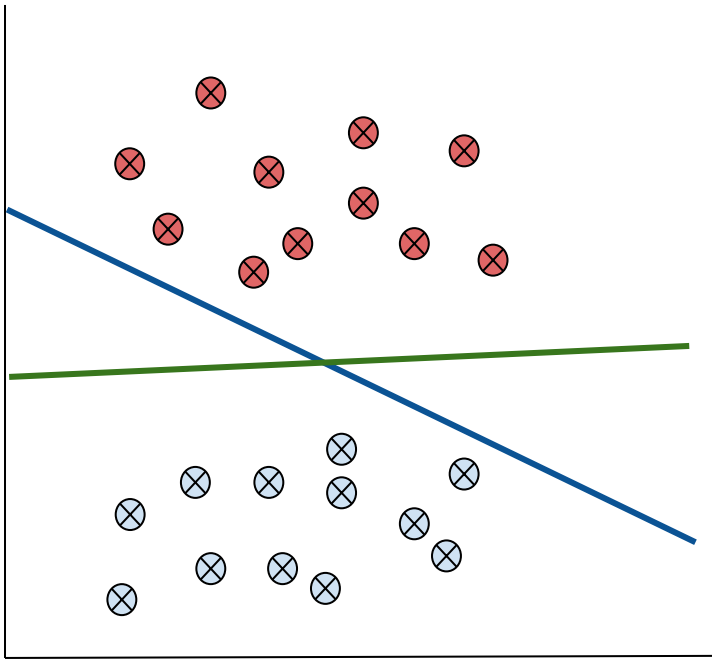
result, different types of moving average models have been created, such as the SMA (Simple Moving Average) and EMA (Exponential Moving Average) models. Both models calculate the moving average price of a stock in a given time period to investigate the trend, and use this to make predictions for future movements. However, such methods are hindered by many factors. For example, many investors and economists do not believe that moving averages have any significance, as they ignore the volatility which some believe is a crucial part of analysis. As a result, the trend that they produce is meaningless. Also, many investors argue that the history of stock prices does not indicate anything about the future (Kalen). Because of the many limitations of moving average models, researchers have turned to machine learning to predict stock prices.

## 2. Theory of Algorithms

### 2.1 Support Vector Machine

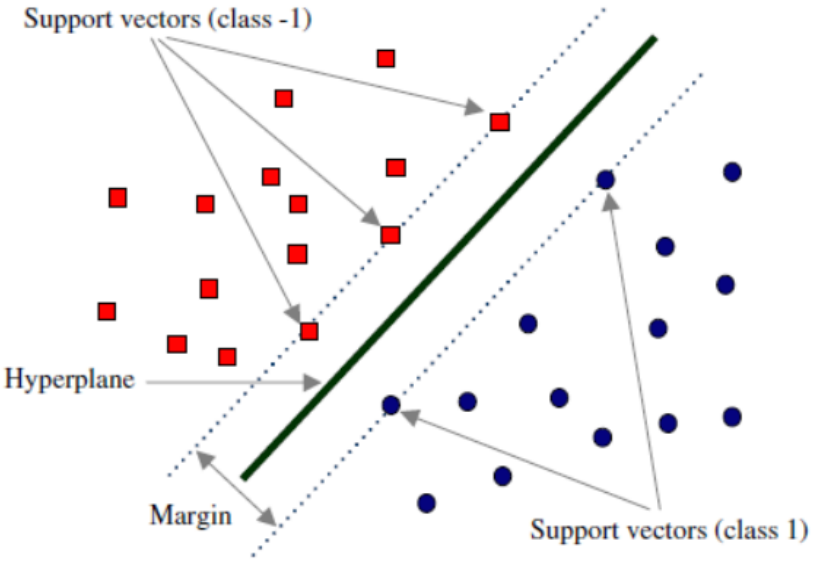
Support Vector Machine is a machine learning algorithm that uses supervised learning, which means that the input and output of the model will be labeled. It is conventionally used for regression and classification problems, with the latter being the current case. In essence, the algorithm seeks to find a hyperplane that best separates the two classes of data points, where a hyperplane is defined as a subspace of one dimension less than the given space. So, for example, if the data points are two-dimensional, the hyperplane will be a one-dimensional linear line. The two major types of SVM consist of linear and nonlinear. A linear SVM is used for when the data can be perfectly separated into two classes with a single straight line, which is not true in this case as the two data classes cannot. Hence, this investigation will use a nonlinear SVM, with a kernel function. A kernel function is a tool used to manipulate data points to a desired form. In this case, the data points will be mapped to a higher dimension in order to make them linearly separable. This is done through the vector dot product (“Major Kernel Functions”).

First, let us understand a linear SVM. In this case, the data can be separated into two classes by a single linear line. However, the question of which hyperplane to use arises as there are infinitely many hyperplanes that can separate the data into two groups. For instance, in the below graph, two classes of data are represented by the red and blue crosses, and both the green and blue hyperplanes separate the data perfectly into two groups.



Google Drawings

Here, the green hyperplane is the better choice. This is because the distance between it and the data points is larger than that of the blue hyperplane. Specifically, the distance between the 2 closest data points to the hyperplane is larger for the green hyperplane. The significance of this is that a larger distance would procure a more generalized hyperplane that would be better suited for unlabeled data (Pupale, 2018). To summarize, the ultimate aim of the SVM is to find a hyperplane that maximizes the distance to the support vectors, which are vectors formed by the points closest to the hyperplane (Anshul, 2021).



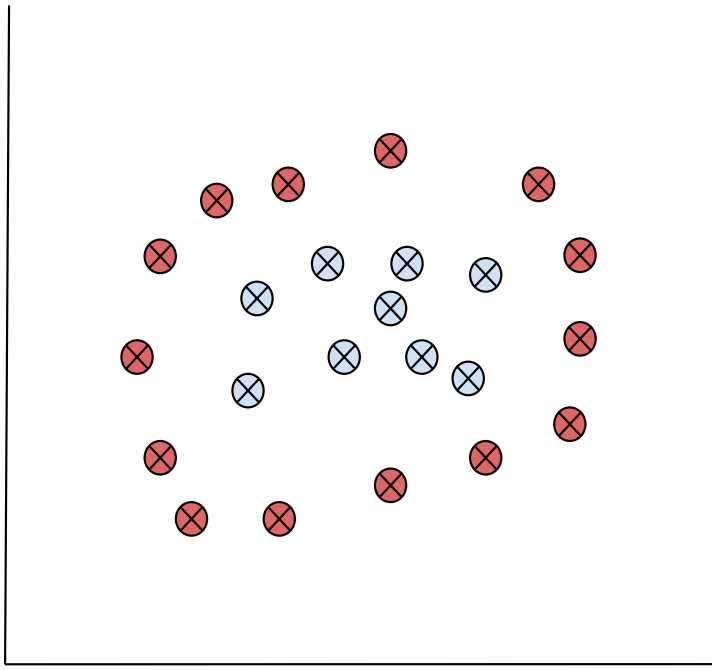

Kumar, 2020

The equation of the line can be written as y=mx+b. Or, in the case of a hyperplane:

$$H = w^T(x) + b,$$

where $w^T$ is the weight vector, b is the bias of the hyperplane, and x is the input vector (data points). The margins, which are two lines parallel to the hyperplane, represent the distance between the hyperplane and the support vectors. There are two types of margins: soft and hard margins. In the case of a linearly separable data set, a hard margin is used, meaning that all data points will fall outside of the margins. On the other hand, soft margins are commonly used when the data is non-linearly separable, allowing for some data points to be between the margin and hyperplane, which is known as misclassification (Karimi, 2021). With a soft margin, the risks of overfitting (when the model becomes too attuned and accurate for the training data, so that it is unable to work with new data) are reduced, as the hyperplane will not have to accommodate every data point. Furthermore, by permitting outliers to exist, the hyperplane can become a lot more generalized and hence accurate. The usage of a soft margin would require a loss function (a

mathematical function that calculates the difference between expected and actual output) as well, which should be minimized. When the data is not linearly separable, we must employ a kernel trick to transform the data so that it becomes linearly separable. For instance, let us assume the data set now looks like this:



Google Drawings

Clearly, this cannot be separated into two groups with a linear hyperplane. A kernel function manipulates the data points onto a higher dimension, with the goal of making them linearly separable. For the above data set, the red data points seem to be surrounding the blue data points in a circular shape. Recognizing this, we can attempt to plot these points onto a higher space with constraints of the formula of a circle:

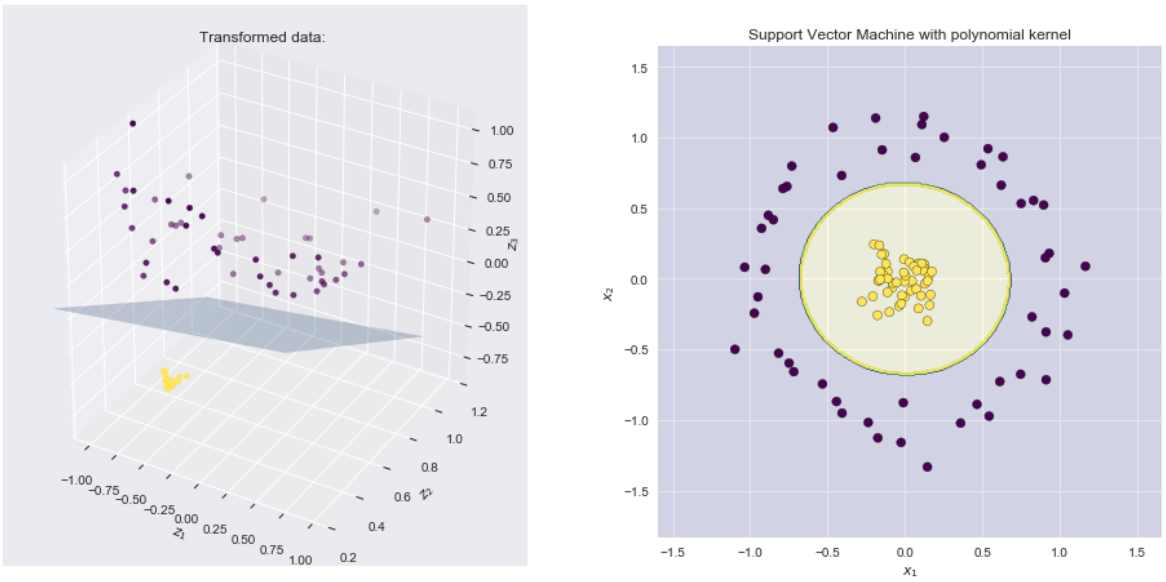
$$(x - h)^2 + (y - k)^2 = r^2.$$

Since the variables h and k represent horizontal and vertical shifts, they become insignificant when defining a plane. Here, r represents the new axis, which is also the distance between the

origin and a data point. We can also define $r^2$ as a new variable, z, which represents the squared distance between the origin and a data point (Pupale, 2018). The new equation would be:

$$z = x^2 + y^2.$$

By doing this, we can now place a hyperplane that separates the data points into two distinct groups. Here is a visual representation of the result:



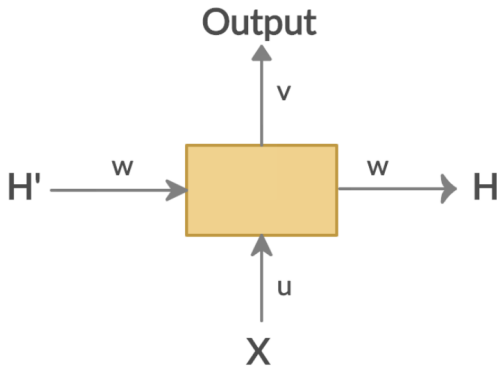

Sicotte, 2018

To do this, each vectorized data point must be transformed into the higher plane, which can be very computationally expensive (Stecanella, 2017). However, as SVMs only require the dot products instead of the actual vectors (as it does not require each individual vector to form a hyperplane, only the relationships between them), the only necessary calculation is finding the dot product in the new plane, which is exactly what the kernel function does. Once this is done, the optimal hyperplane, which is defined as the hyperplane that maximizes the distance to the support vectors, can be found. This is done by solving an optimization problem that seeks to maximize the distance between the hyperplane and support vectors, which can simply be

calculated using the formula for the distance between a point and line. When we apply the model to the data, two results can occur. Firstly, if the model predicts a positive value and the label is also positive, then a positive value will be produced. Likewise, if the model correctly predicts a negative value, a positive value will be produced as two negatives make a positive. On the other hand, if the model predicts a positive value when it is actually a negative value or vice versa, a negative value will be generated. This is how the model discerns between a correct classification and misclassification. If a misclassification occurs, the model is penalized and the loss function is utilized. Because non-linear SVMs can provide flexibility with regard to the types of relationships it can discern, they are especially well equipped for data sets with multiple features, such as financial data (McGregor, 2020). As a result, this data set would be suitable for a SVM model, which would hence improve the accuracy of the predictions.

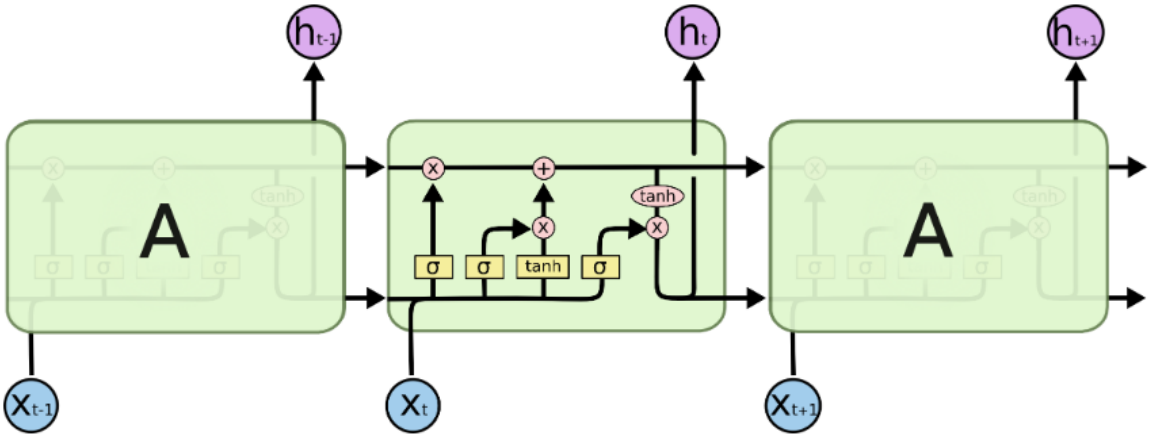
### 2.2 Long Short Term Memory Network

LSTM networks are a variant of Recurrent Neural Networks (RNNs), designed to solve a problem that RNNs could not overcome: vanishing/exploding gradient descent. RNNs are predominantly used for sequencing problems, where there is a dependency between the inputs. The structure of an RNN is as follows:



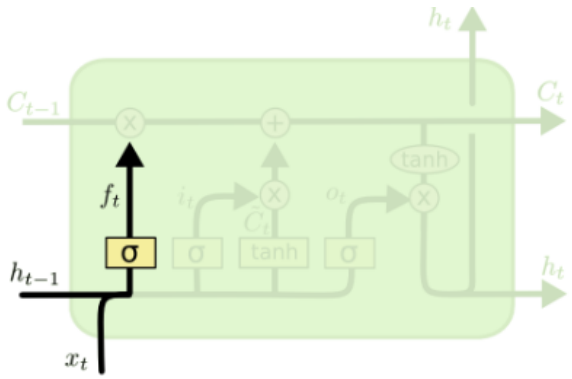

Vidisha Jitani, 2020

The input vector is X, with u, v, w being the weight vectors. H' is the previous state (hidden state), which is fed into the current input, and H is the current input being fed into the next state. Since the hidden layer will be continually multiplied by a constant weight vector, the input will either become infinitely large if w > 1 or tend towards 0 if w < 1. Thus, the vanishing/exploding gradient descent problem arises. For a more grounded example, consider the following sentence: "The color of grass is *green*". Here, RNNs would have no problem in predicting the word *green*. However, in a case such as "I like computer science, so next year in high school I will take *computer science*", the gap between relevant words is larger, which would cause problems for the network (Olah, 2015). To solve this issue, LSTM networks were created. Unlike RNNs, they are able to choose to forget certain information that is deemed irrelevant while keeping a memory of relevant information. The hidden layer of LSTMs is more complicated than that of basic RNNs, as it consists of 3 gates: forget gate, input gate and output gate. These gates also solve the vanishing and exploding gradient problem, as they prevent the values from decreasing to 0 and increasing to infinity.




Christopher Olah, 2015

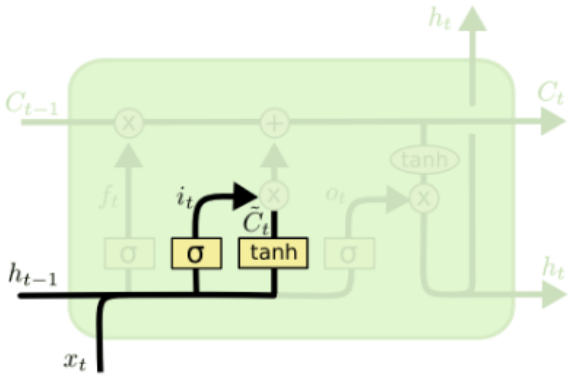
First, the forget layer, also known as the sigmoid layer, decides what information to dispose of. It outputs a number between 0 and 1, where 0 means a complete deletion, and 1 means to maintain all of the information. This is done in the first section of the diagram:




$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Christopher Olah, 2015

Next, the input layer decides on what new information to pass onto the cell state (the long term "memory" of the network). This process consists of two parts; the first of which is also a sigmoid layer, which determines which current values to update on a spectrum of 0 to 1. Afterward, a vector of new values is created by a tanh layer that will replace the current values. The two parts are then combined and the update is finalized.

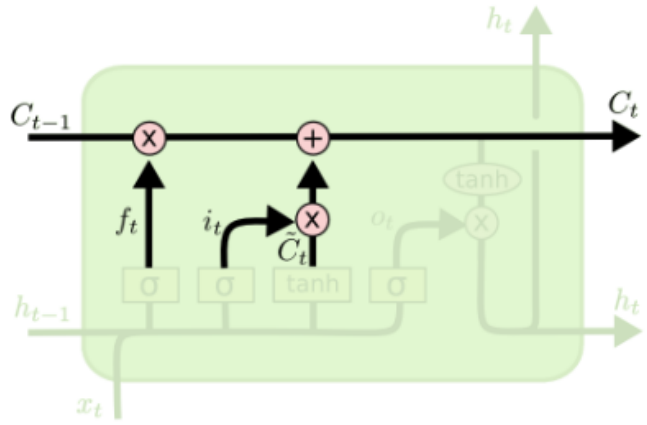



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Christopher Olah, 2015

To apply the updates to the model, the old state is multiplied by $f_t$, which forgets the necessary information established earlier. Subsequently, the product of $i_t$ and $\widehat{C}_t$ is added on,

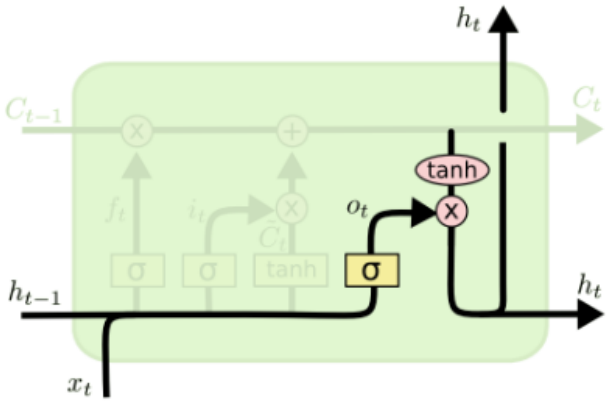
which is the product of the predetermined candidate values and their respective scaling quantities.




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

Christopher Olah, 2015

Lastly, the model determines what information it needs to output, which will be based on the cell state. A sigmoid layer is applied to the cell state to choose which parts are needed for the output, and a tanh layer manipulates the values of the cell states so that they are between -1 and 1. Then, the two parts are multiplied together and the output is decided (Olah, 2015).




$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Christopher Olah, 2015

A potential problem that could arise for neural networks such as LSTMs is overfitting, which is when the network becomes too acquainted with the training data. This could result in very low error values, but on the other hand very high error values for unseen data. Several

factors could contribute to such an issue, for instance, an overly complex model with an excessive amount of parameters or training the model too many times on one data set (Brownlee, 2018).

The network is trained through learning by back propagation, which is the process of tracking backward from the output layer to the input layer in order to adjust the weights and biases, with the goal of minimizing the loss function (Seth, 2021). The weights in neural networks control the strength of connectivity between layers, while the biases increase or decrease the output of individual neurons (Rose, 2021). In this case, the loss function used will be the mean squared error, which calculates the difference between the predicted output and actual output. These differences are squared in order to account for negative values, and finally summed up. The network will attempt to minimize this sum by iterating through and modifying the weights and biases.

Finally, revisiting the example of "I like computer science, so next year in high school I will take *computer science*", we can now see how a LSTM network would work. Assuming the network has already been trained with similar sentences (such as "I like history, so next year in high school I will take *history"*), it will choose to remember the word *history* in the cell state because the word after "like" has commonly been repeated. For every other time step between the occurrences of the word *history*, the model will have no trouble predicting the words as they have never been changed. This means that once it sees, for example, "next" in the previous step, it will predict "year" in the current step. However, as the model has never predicted a recurrence of these words before, it will choose to forget them. As it reaches the word "take" or "like", it recognizes that a word must be repeated. Thus, it looks toward the cell state and predicts the word stored there, which is *history*.

## 3. Investigation

### 3.1 Data Collection

The data used in the experiments are retrieved in real time using the “yfinance” library, which extracts relevant data from the official website of Yahoo Finance. The features of each stock that are taken into account for the SVM model include the opening and closing prices, as well as the highest and lowest prices within a trading cycle. The LSTM model will only consider the closing prices.

### 3.2 Methodology

Two different stocks were selected randomly from each of the 11 stock sectors: healthcare, materials, real estate, consumer staples, consumer discretionary, utilities, energy, industrials, consumer services, financials, and technology. Each individual stock is then fed to both the SVM model and LSTM model, where the historical data period ranges from 1 year, 5 years and 10 years. These values were chosen in order to conduct a holistic investigation that incorporates time periods. In this case, efficiency will be defined as the accuracy with which each model can predict the future price direction of different stocks.

### 3.2 Implementation

Google Colaboratory, which is a cloud-based version of Jupyter Notebook, was used to implement both models. The SVM model was imported from the sklearn library, while the LSTM model was created using the TensorFlow library. For both models, the data set was split into a training set and test set, resulting in an 80% and 20% split respectively. The LSTM neural network was defined as a sequential model, with two LSTM layers of 100 network nodes each, and two Dense layers (layers where each neuron of the previous layer is connected to the neurons

of the current layer). In order to avoid overfitting, the network will only run for two epochs (iterations) on each data set. The SVM model returned binary prediction values 0 or 1, where 0 signals a decrease in closing price and 1 an increase. Since the LSTM model predicts specific stock prices, a method that determines whether the model predicted an increase or decrease in closing price had to be added in order to match the SVM results.

### 3.3 Results

**Table 1. Short-term (1/1/2021 - 1/1/2022)**

| Sector | Stock Symbol | SVM | LSTM |
|---|---|---|---|
| | | Accuracy Score (%) | Accuracy Score (%) |
| Technology | ASX | 54.90 | 48.00 |
| | TWTR | 50.98 | 64.00 |
| Telecommunications | USM | 58.86 | 38.00 |
| | IDT | 39.22 | 64.00 |
| Healthcare | THC | 50.98 | 52.00 |
| | ITGR | 45.10 | 42.00 |
| Financials | MS | 52.94 | 38.00 |
| | BTZ | 45.10 | 30.00 |
| Real Estate | KW | 49.02 | 42.00 |
| | ORC | 56.86 | 50.00 |
| Consumer Discretionary | MCD | 54.90 | 62.00 |

| | | | |
|---|---|---|---|
| | GME | 47.06 | 48.00 |
| Consumer Staples | KR | 47.06 | 46.00 |
| | FLO | 52.94 | 38.00 |
| Industrials | ETN | 49.02 | 58.00 |
| | TEX | 47.06 | 48.00 |
| Basic Materials | SUZ | 49.02 | 52.00 |
| | BVN | 56.86 | 56.00 |
| Energy | SHEL | 35.29 | 38.00 |
| | DVN | 49.02 | 36.00 |
| Utilities | PEG | 56.86 | 52.00 |
| | AWK | 64.71 | 52.00 |
| Average Score (%) | / | 50.63 | 47.91 |
| Standard Deviation (%) | / | 6.56 | 9.47 |

**Table 2. Medium-term (1/1/2012 - 1/1/2022)**

| Sector | Stock Symbol | SVM | LSTM |
|---|---|---|---|
| | | Accuracy Score (%) | Accuracy Score (%) |
| Technology | ASX | 48.92 | 50.60 |
| | TWTR | 44.34 | 49.22 |
| Telecommunications | USM | 50.22 | 44.62 |

| | | | |
|---|---|---|---|
| | IDT | 50.25 | 46.91 |
| Healthcare | THC | 42.45 | 50.02 |
| | ITGR | 52.03 | 45.25 |
| Financials | MS | 51.87 | 44.48 |
| | BTZ | 52.00 | 38.53 |
| Real Estate | KW | 49.88 | 45.83 |
| | ORC | 49.28 | 48.24 |
| Consumer Discretionary | MCD | 51.10 | 43.43 |
| | GME | 52.51 | 45.82 |
| Consumer Staples | KR | 47.45 | 51.00 |
| | FLO | 51.99 | 50.77 |
| Industrials | ETN | 56.29 | 51.79 |
| | TEX | 45.67 | 51.52 |
| Basic Materials | SUZ | 54.25 | 46.59 |
| | BVN | 44.12 | 48.78 |
| Energy | SHEL | 48.82 | 44.21 |
| | DVN | 53.23 | 48.34 |
| Utilities | PEG | 57.90 | 36.75 |
| | AWK | 55.34 | 47.36 |

| | | | |
|---|---|---|---|
| Average Score (%) | / | 50.46 | 46.82 |
| Standard Deviation (%) | / | 3.97 | 3.91 |

**Table 3. Long-term (1/1/2017 - 1/1/2022)**

| Sector | Stock Symbol | SVM | LSTM |
|---|---|---|---|
| | | Accuracy Score (%) | Accuracy Score (%) |
| Technology | ASX | 49.84 | 51.99 |
| | TWTR | 43.57 | 57.21 |
| Telecommunications | USM | 48.59 | 44.78 |
| | IDT | 50.78 | 48.76 |
| Healthcare | THC | 45.45 | 49.25 |
| | ITGR | 51.72 | 51.24 |
| Financials | MS | 51.10 | 48.26 |
| | BTZ | 53.29 | 49.76 |
| Real Estate | KW | 50.16 | 50.25 |
| | ORC | 49.22 | 48.77 |
| Consumer Discretionary | MCD | 50.78 | 47.76 |
| | GME | 52.66 | 49.75 |
| Consumer Staples | KR | 48.59 | 48.76 |
| | FLO | 53.61 | 52.24 |

| | | | |
|---|---|---|---|
| Industrials | ETN | 56.43 | 51.74 |
| | TEX | 49.84 | 56.22 |
| Basic Materials | SUZ | 52.04 | 47.26 |
| | BVN | 43.26 | 56.72 |
| Energy | SHEL | 49.37 | 43.28 |
| | DVN | 53.29 | 46.27 |
| Utilities | PEG | 58.31 | 42.79 |
| | AWK | 55.17 | 50.75 |
| Average Score (%) | / | 50.78 | 49.72 |
| Standard Deviation (%) | / | 3.70 | 3.83 |

**Table 4. Summary**

| Time (years) | SVM Average Accuracy Score (%) | LSTM Average Accuracy Score (%) |
|---|---|---|
| 1 | 50.63 | 47.91 |
| 5 | 50.46 | 46.82 |
| 10 | 50.78 | 49.72 |
| Average | 50.62 | 48.15 |

### 3.4 Analysis

From the short-term dataset, it is evident that the SVM model has a higher average accuracy score than the LSTM model when predicting an increase or decrease in closing price for the chosen stocks by 1.06%. The standard deviation is also smaller for the SVM model by 0.13%. This result implies that the SVM model is the more accurate, consistent model in predicting whether a stock price will increase or decrease for the given data set in the short-term.

Similarly, for a medium-term prediction, the SVM model also proves to be the more accurate one. The standard deviation (3.97%) is a lot smaller than for the short-term model, which implies that this model is a lot more consistent. This hints at the fact that the increased amount of training data has made the hyperplane more generalized for each individual stock. As a result, a more optimal hyperplane is created and thus the margins are larger. These factors contribute to reducing the presence of lower accuracy scores. Nevertheless, with an average accuracy percentage of 50.46%, it is also surprisingly less accurate than the short-term model even with more data points. This was unexpected as having more labeled training data should aid the accuracy of the model by allowing it to be more generalized. Moreover, since the data set is smaller for the short-term model, it is more likely that randomness played a role. This implies that for some stocks, the price trend coincidentally aligned correctly with the hyperplane, and, as the data set is quite small, these coincidental predictions were not ironed out with predictions that would’ve occurred with a larger data set. In comparison, the LSTM model also fell short in terms of accuracy of the short-term model by 1.09%, while the standard deviation similarly decreased significantly, almost three-fold. Thus, it can be discerned that the randomness effect has also taken place, only more dramatically. This is corroborated by the fact that LSTM models depend heavily on the abundance of past data, as the output of each layer depends not only on the

previous output, but also the overarching “memory” of the network, which is influenced by all previous output and governed by the cell state. The lack thereof eliminates a lot of its tools, thereby resulting in a reduced model that relies on chance.

As expected, the long-term LSTM model with 10 years worth of data improved the accuracy by 2.9%, which is significantly better. Also, the standard deviation decreased by 0.08%, demonstrating that the model is (slightly) more consistent. Nevertheless, the SVM model still has a higher accuracy score by 1.06%, and a lower standard deviation by 0.13%.

Overall, it appears that the SVM model is a more accurate, thus more effective, indicator of stock price directions than the LSTM model, with a higher accuracy and a mostly lower standard deviation.

## 4. Conclusion

### 4.1 Evaluation of Results

Ultimately, the result of the experiment firmly states that the SVM model is more effective than the LSTM model in predicting the general direction of stock prices by an average of 2.47%. The experiment tested both models against 22 different stocks, selected from the 11 sectors of finance. Given the aim of the SVM model is to find an optimal hyperplane by solving an optimization problem, it does not require a large dataset. In fact, an excessively large dataset would be inequitable for it as the training time would become infeasibly long. This is because the memory of the kernel matrix has a quadratic relationship with the number of data points, and the training time increases in a superlinear manner (Pedregosa, 2011). Thus, the results were consistent even with the changes in data set size. Fluctuations were minimal, except for the short-term model with a standard deviation of 6.56%. Clearly, it still requires a substantial amount of data points to make intelligent predictions. This is due to the fact that with a small

amount of training data points, the hyperplane would not be generalized enough to predict unseen data.

Conversely, the LSTM model relies heavily on the sufficiency of data. Since the model has a lot more trainable parameters (weight and biases) than the SVM model, it inherently requires more data. Additionally, because it specializes in time series data and predicts future values based on past data and a dynamic "memory", it requires a significant amount of data to train.

In conclusion, due to the nature of the SVM model, it performed better as it is fundamentally a classification model while the LSTM model specializes in predicting specific values. To this end, the LSTM model would be more useful to most investors that want a specific forecast. However, this does not imply that it is the better model; the SVM model had the more accurate predictions. When trying to determine the general price trend of a stock or industry, it would be more beneficial to use the SVM model. On the contrary, in the case of attempting to predict distinct prices, the LSTM model would prove more useful. This result contradicts the initial hypothesis in that the SVM model had a better average accuracy score, but mostly agrees with the statement that an increase in time period would cause an increase in accuracy for both models.

### 4.1 Evaluation of Methodology

The methodology used in this experiment could be improved. Its major flaw is that it did not take into account the specialization of the models, therefore resulting in a biased result, where one model outperformed the other based on perhaps inherent qualities. However, a method was written to convert the results of the LSTM model into the same measurable type as those of the SVM model, which allowed a comparison to be done. Although this did not eliminate the

aforementioned flaw completely, it still provided the means to conduct a meaningful investigation. Furthermore, the investigation was comprehensive in that it explored a wide variety of stocks from each sector, providing a holistic representation. Additionally, by having 3 sets of different time periods, it was possible to take into account and analyze the impact of time on the accuracies of the models.

### 4.1 Further Research

Improvements or further studies could investigate additional relevant variables. For example, adding parameters such as moving averages, volatility or other financial statistics of individual stocks could improve the models as these factors (and numerous others) all play a role in determining the price of the stocks. Moreover, the prediction window could be manipulated by observing, for instance, the accuracy of predicting whether the price will increase or decrease over a 1 week period. This would reduce some of the short-term volatilities and better ascertain as to which model would be more useful, as most investors do not trade on a daily basis. Furthermore, a larger amount of stocks could be tested, which would generate a more holistic analysis of the accuracy. Other suitable machine learning models such as random forest and naive bayes could also be compared.

Works Cited

Bora, Leena. "Simplified Math behind Complex LSTM equations." *Medium*, medium.com/@leenabora1/simplified-math-behind-complex-lstm-equations-66cff0d52d78. Accessed 13 Sept. 2022.

Brownlee, Jason. "How to Avoid Overfitting in Deep Learning Neural Networks." *Machine Learning Mastery*, 17 Dec. 2018, machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/. Accessed 13 Sept. 2022.

Jitani, Vidisha. "Recurrent Neural Network: Maths in 5 Minutes." *Medium*, 22 Sept. 2020, vidishajitani.medium.com/recurrent-neural-network-maths-69214e4d69e1. Accessed 13 Sept. 2022.

Karimi, Akabr. "Using a Hard Margin vs. Soft Margin in SVM." *Baeldung*, 13 Oct. 2021, www.baeldung.com/cs/svm-hard-margin-vs-soft-margin. Accessed 9 Oct. 2022.

Kumar, Ajitesh. "SVM Algorithm as Maximum Margin Classifier." *Data Analytics*, 7 July 2020, vitalflux.com/svm-algorithm-maximum-margin-classifier/. Accessed 9 Oct. 2022.

Li, Katherine Yi. "Predicting Stock Prices Using Machine Learning." *Neptune.ai*, neptune.ai/blog/predicting-stock-prices-using-machine-learning. Accessed 13 Sept. 2022.

"Major Kernel Functions in Support Vector Machine (SVM)." *GeeksforGeeks*, 7 Feb. 2022, www.geeksforgeeks.org/major-kernel-functions-in-support-vector-machine-svm/. Accessed 9 Oct. 2022.

McGregor, Milecia. "SVM Machine Learning Tutorial – What is the Support Vector Machine Algorithm, Explained with Code Examples." *freeCodeCamp*, 1 July 2020,

www.freecodecamp.org/news/svm-machine-learning-tutorial-what-is-the-support-vector-machine-algorithm-explained-with-code-examples/. Accessed 8 Oct. 2022.

Olah, Christopher. "Understanding LSTM Networks." *colah's blog*, 27 Aug. 2015, colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed 13 Sept. 2022.

Pedregosa, Fabian. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, 2011, jmlr.csail.mit.edu/papers/v12/pedregosa11a.html. Accessed 8 Oct. 2022.

"Predicting Stock Price Direction using Support Vector Machines." *GeeksforGeeks*, 23 Sept. 2021, www.geeksforgeeks.org/predicting-stock-price-direction-using-support-vector-machines/. Accessed 9 Oct. 2022.

Pupale, Rushikesh. "Support Vector Machines(SVM) — An Overview." *Towards Data Science*, 16 July 2018, towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989. Accessed 8 Oct. 2022.

Radhika. "The Mathematics Behind Support Vector Machine Algorithm (SVM)." *Analytics Vidhya*, www.analyticsvidhya.com/blog/2020/10/the-mathematics-behind-svm/#:~:text=The%20goal%20of%20the%20algorithm%20involved%20behind%20SVM%3A&text=Finding%20a%20hyperplane%20with%20the,(known%20as%20support%20vectors).&text=wT(%CE%A6(x))%20%2B%20b%20%3C%200. Accessed 13 Sept. 2022.

Rose, Doug. "Functions, Weights, and Bias in Artificial Neural Networks." *Linkedin*, 9 Aug. 2021,

www.linkedin.com/pulse/functions-weights-bias-artificial-neural-networks-doug-rose. Accessed 9 Oct. 2022.

Saini, Anshul. "Support Vector Machine(SVM): A Complete guide for beginners." *Analytics Vidhya*, 12 Oct. 2021, www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/. Accessed 7 June 2022.

Seth, Neha. "How does Backward Propagation Work in Neural Networks?" *Analytics Vidhya*, 1 June 2021, www.analyticsvidhya.com/blog/2021/06/how-does-backward-propagation-work-in-neural-networks/. Accessed 9 Oct. 2022.

Sicotte, Xavier Bourret. "How to intuitively explain what a kernel is?" *Stack Exchange*, 8 July 2018, stats.stackexchange.com/questions/152897/how-to-intuitively-explain-what-a-kernel-is. Accessed 9 Oct. 2022.

Smith, Kalen. "The 7 Pitfalls of Moving Averages." *Investopedia*, www.investopedia.com/articles/trading/11/pitfalls-moving-averages.asp. Accessed 13 Sept. 2022.

Stecanella, Bruno. "Support Vector Machines (SVM) Algorithm Explained." *MonkeyLearn*, 22 June 2017, monkeylearn.com/blog/introduction-to-support-vector-machines-svm/. Accessed 9 Oct. 2022.

Teo, Bee Guan. "Stock Prices Prediction Using Long Short-Term Memory (LSTM) Model in Python." *Medium*,

medium.com/the-handbook-of-coding-in-finance/stock-prices-prediction-using-long-short-term-memory-lstm-model-in-python-734dd1ed6827. Accessed 13 Sept. 2022.

"What is Hyperplane." *IGI Global*, www.igi-global.com/dictionary/machine-learning-for-industrial-iot-systems/34033. Accessed 7 June 2022.

## Appendix:

SVM model code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
!pip install yfinance
import yfinance as yf
stock = ''
#https://www.geeksforgeeks.org/predicting-stock-price-direction-using-support-vector-machines/
data = yf.download(stock, "2021-01-01", "2022-01-01", auto_adjust = True)
data['Open-Close'] = data.Open - data.Close
data['High-Low'] = data.High - data.Low
X = data[['Open-Close', 'High-Low']]
y = np.where(data['Close'].shift(-1) > data['Close'], 1, 0)
split_percentage = 0.8
split = int(split_percentage*len(data))
X_train = X[:split]
y_train = y[:split]
X_test = X[split:]
y_test = y[split:]
cls = SVC().fit(X_train, y_train)
data['Predicted_Signal'] = cls.predict(X)
accuracy_test = accuracy_score(y_test, cls.predict(X_test))
print('Test Accuracy:{: .2f}%'.format(accuracy_test*100))
```

LSTM model code:

```
import math
!pip install yfinance
import yfinance as yf
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import pandas as pd
```

```
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
#medium.com/the-handbook-of-coding-in-finance/stock-prices-prediction-usin
g-long-short-term-memory-lstm-model-in-python-734dd1ed6827
stock_data = yf.download('', start='2021-01-01', end='2022-01-01')
stock_data.head()
close_prices = stock_data['Close']
values = close_prices.values
training_data_len = math.ceil(len(values)* 0.8)
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(values.reshape(-1,1))
train_data = scaled_data[0: training_data_len, :]
x_train = []
y_train = []
for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])
x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
test_data = scaled_data[training_data_len-60: , : ]
x_test = []
y_test = values[training_data_len:]
for i in range(60, len(test_data)):
  x_test.append(test_data[i-60:i, 0])
x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
model = keras.Sequential()
model.add(layers.LSTM(100, return_sequences=True,
input_shape=(x_train.shape[1], 1)))
model.add(layers.LSTM(100, return_sequences=False))
model.add(layers.Dense(25))
model.add(layers.Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, batch_size= 1, epochs=2)
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
data = stock_data
data = data[-math.floor(len(data)*20/100):]
y = np.where(data['Close'].shift(-1) > data['Close'], 1, 0)
from sklearn.metrics import accuracy_score
```

```
test_data = scaler.inverse_transform(test_data)
df = pd.DataFrame(predictions)
yy = np.where(df.shift(-1) > df, 1, 0)

acc = accuracy_score(np.array(yy), np.array(y))
acc = acc*100
print(acc)
```