CS725: Foundations of Machine Learning
Project Report
Stock Market Prediction

Prathamesh Karandikar (22M1155); Jay Sharma (22M1177);
Shounak Das (213020064); V Adarsh(21307r001)

Introduction:

Stock market prediction is a trivial problem which can be optimized to gain several benefits. Understanding the various stock market factors that may affect a stock's price and developing a model to predict the stock's price are both terms used to describe stock market prediction. This can assist individuals and institutions in making predictions about the direction of stock prices and in deciding whether to buy or sell stocks in order to make the most money.

Motivation:

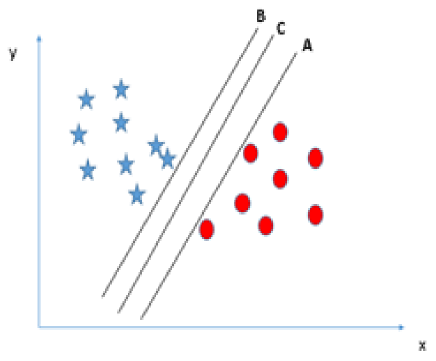
Predicting stock prices is a well-known and significant problem. We can learn about market behaviour over time and identify trends that might not have been seen without an effective stock prediction model. Machine learning will be a useful approach to solving this issue with the increased processing capacity of computers. However, many machine learning techniques can't be used with the public stock dataset because it is too small, and adding more features might cost thousands of dollars every day.In this project, Our aim is to create a model that can forecast the stock's stopping price and was trained using SVR and LSTM.

The SVR Model:

What is the Support Vector Machine?

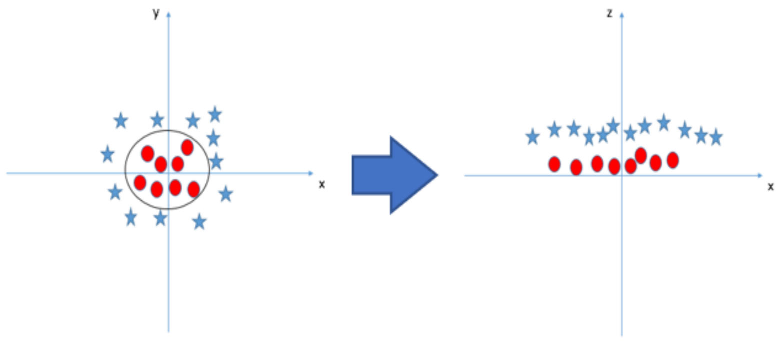
An algorithm known as "Support Vector Machine" (SVM) is a supervised machine learning technique that can be applied to classification or regression problems. It is primarily utilised, nevertheless, in classification issues. The SVM algorithm plots each data point as a point in n-dimensional space, where n is the number of features you have and each feature's value is a specific coordinate value. Then, classification is performed by identifying the hyper-plane that effectively distinguishes the two classes.

How does it work?



In this case, choosing the appropriate hyper-plane will be aided by maximising the distances between the nearest data point (of either class) and the hyper-plane. Margin is the name given to this distance. As you can see, hyper-plane C has a larger margin than both A and B. Therefore, we designate C as the proper hyper-plane. Robustness is another compelling argument in favour of choosing the hyper-plane with the bigger margin. There is a substantial possibility of mis-classification if a low margin hyper-plane is chosen.

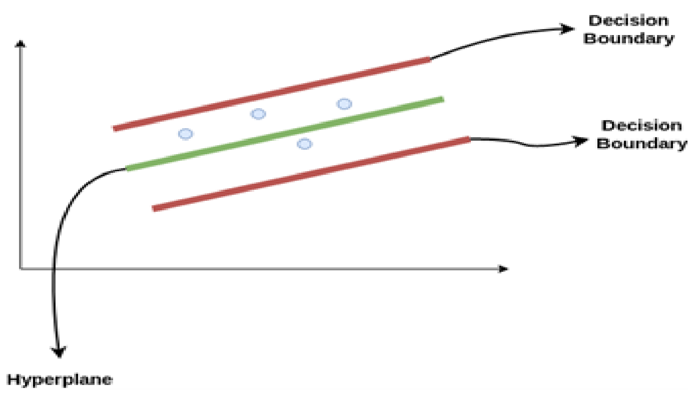
There are several situations when linear hyper-plane between the two classes cannot be used because the data is not linearly separable. We are unable to create a linear hyper-plane between the two classes when the data is not linearly separable in some situations.



The SVM algorithm uses a method known as the kernel trick in such circumstances. The SVM kernel is a function that transforms not separable problems into separable problems by taking a low-dimensional input space and raising its dimension. It is most helpful in problems with non-linear separation. Simply put, it performs a number of extremely complex data transformations before determining how to separate the data in accordance with the labels or outputs you've specified.

Support Vector Regression:

Similar to SVM, Support Vector Regression (SVR) tackles regression issues. Finding a function that roughly maps an input domain to actual numbers using a training sample is the goal of regression analysis.



Think of the green line as the hyperplane, and the two red lines as the decision boundaries. When using SVR, our goal is to essentially take into account the points that are inside the decision boundary line. The hyperplane with the most points serves as our best fit line. The decision boundary (the hazardous red line above!) is the first concept that we will comprehend. Think of these lines as being at any distance from the hyperplane, let's say 'a'. The lines that we draw at '+a' and '-a' distances from the hyperplane are thus as follows. The text basically refers to this 'a' as epsilon. Assuming that the hyperplane's equation:

$$Y = wx + b$$

Then the equations of decision boundary become:

$$wx + b = +a$$

$$wx + b = -a$$

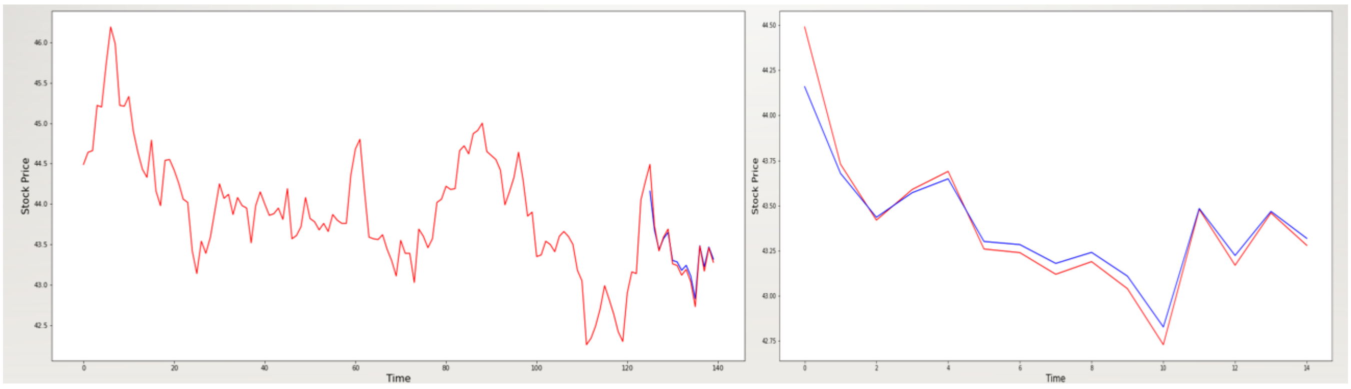
Thus, any hyperplane that satisfies our SVR should satisfy:

$$-a < (Y - wx + b) < +a$$

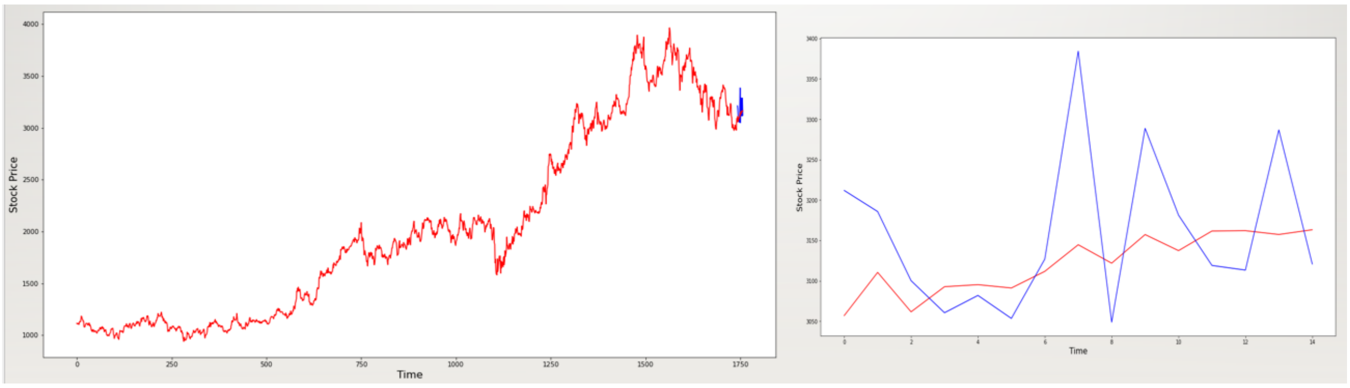
The primary goal of this exercise is to select a decision boundary at a distance from the initial hyperplane such that the data points that are closest to the hyperplane or the support vectors fall within the boundary. As a result, we will only include points that fall within the decision border, have the lowest mistake rate, or fall within the margin of tolerance. As a result, we get a model that fits better.

Results:

1. Input Stock Data: GOLDBEES (Less Volatile)



2. Input Stock Data: TCS (More Volatile)



As we can see, SVR works well in low volatility stocks like GOLDBEES but performs poorly in high volatility stocks like TCS because of a lack of memory component. So, we must switch to neural networks.

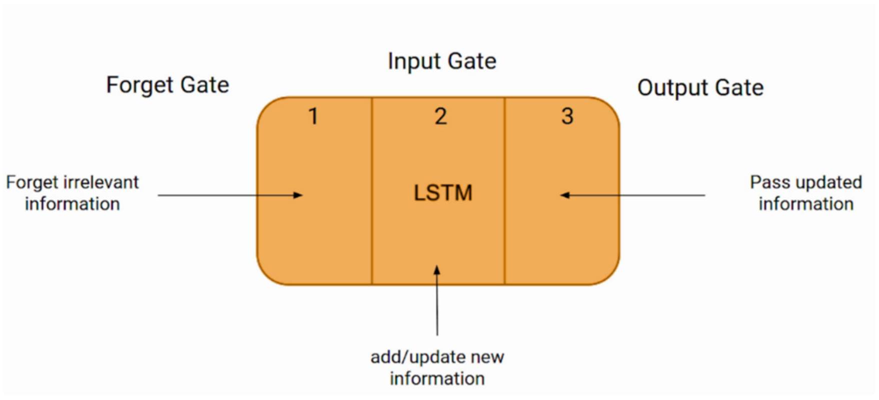
Because SVR was an optimising method and deep learning is necessary to perform better on unknown datasets, we switched from SVR to LSTM. LSTM is a remarkable deep learning model that improves on recurrent neural networks.

Long Short-Term Memory (LSTM):

The Long Short Term Memory Network is a sophisticated RNN, a sequential network, that permits information persistence. It has the ability to solve the RNN's vanishing gradient issue. Persistent memory is implemented using a recurrent neural network, or RNN. Imagine that when reading a book you are aware of what happened in the previous chapter or that while watching a film you remember the prior scene. Similar to how RNNs function, they keep track of prior knowledge and use it to process the data at hand. Due of their inability to remember long-term dependencies, RNNs have this drawback. Long-term dependency problems are specifically avoided with LSTMs.

LSTM Architecture:

LSTM functions on a high level very similarly to an RNN cell. The LSTM network's internal operation is seen below. As seen in the image below, the LSTM is composed of three sections, each of which has a distinct function.



The first section determines whether the information from the preceding timestamp needs to be remembered or can be ignored. The cell attempts to learn new information from the input to this cell in the second section. The cell finally transmits the revised data from the current timestamp to the next timestamp in the third section. Gates refer to these three LSTM cell components. The Forget gate, Input gate, and Output gate are the names of the three components, respectively.

An LSTM has a hidden state, just like a straightforward RNN, with H(t-1) standing for the hidden state of the prior timestamp and Ht for the hidden state of the present timestamp. Additionally, LSTMs have a cell state that is denoted by the timestamps C(t-1) and C(t), which stand for the prior and current timestamps, respectively. In this case, the cell state is referred to as the long-term memory and the hidden state as the short-term memory. See the illustration below. It is interesting to note that the cell state carries the information along with all the timestamps.

Forget Gate:

The first step in an LSTM network cell is to choose whether to keep or discard the data from the previous timestamp. The forget gate equation is given below.

$$f_t = (X_t * U_f + H_{t-1} * W_f)$$

A sigmoid function is then put over it later. As a result, ft will become a number between 0 and 1. As seen below, this ft is later multiplied by the cell state of the preceding timestamp.

$$C_{t-1} * f_t = 0; \qquad \textit{if } f_t \;=\; \textit{0} \ldots \textit{(Forget Everything)}$$

$$Ct - 1 * ft = Ct - 1; \qquad \textit{if ft} \;=\; \textit{1} \ldots \textit{(Forget Nothing)}$$

Input Gate:

The value of the fresh information carried by the input is measured by the input gate. The input gate's equation is shown below.

$$i_t = (X_t * U_i + H_{t-1} * W_i)$$

We once more applied the sigmoid function to it. As a result, we will have a value between 0 and 1 at timestamp t.

New information:

$$N_t = tanh(X_t * U_c + H_{t-1} * W_c); \qquad \ldots \textit{(New Information)}$$

The new data that had to be sent to the cell state now depends on a concealed state at timestamp t-1 in the past and input x at timestamp t. Tanh is the activation function in this case. The tanh function causes the value of fresh information to range from -1 to 1. The information is deducted from the cell state if the value of N_t is negative, and added to the cell state at the current timestamp if the value is positive. The N_t won't, however, be added to the cell state immediately. This is the revised equation.

$$C_{t-1} = f_t * C_{t-1} + i_t * N_t \qquad \ldots \textit{(Updating cell state)}$$

Output Gate:

Here is the equation of the Output gate, which is pretty similar to the two previous gates.

$$ot = (Xt * Uo + Ht - 1 * Wo)$$

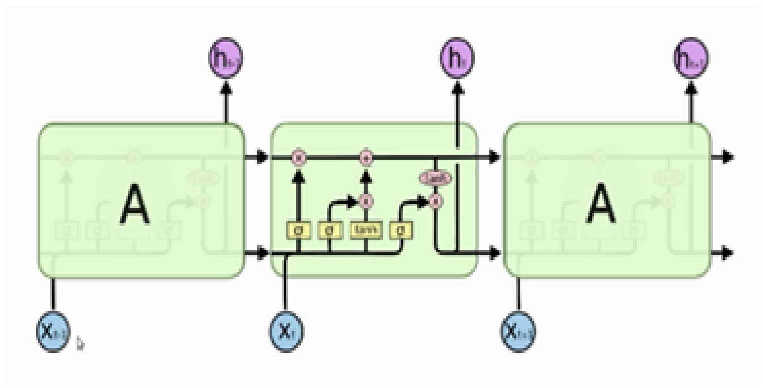
Due to this sigmoid function, it will also have a value between 0 and 1. We will now use O_t and tanh of the updated cell state to determine the current hidden state. as displayed below.

$$H_t = o_t * tanh(C_f)$$

It turns out that the concealed state depends on both the present output and long-term memory (Ct). Simply activate SoftMax on hidden state Ht if you need to take the output of the current timestamp. Prediction is the token in this case having the highest score in the output.

$$Output = softmax(Ht)$$

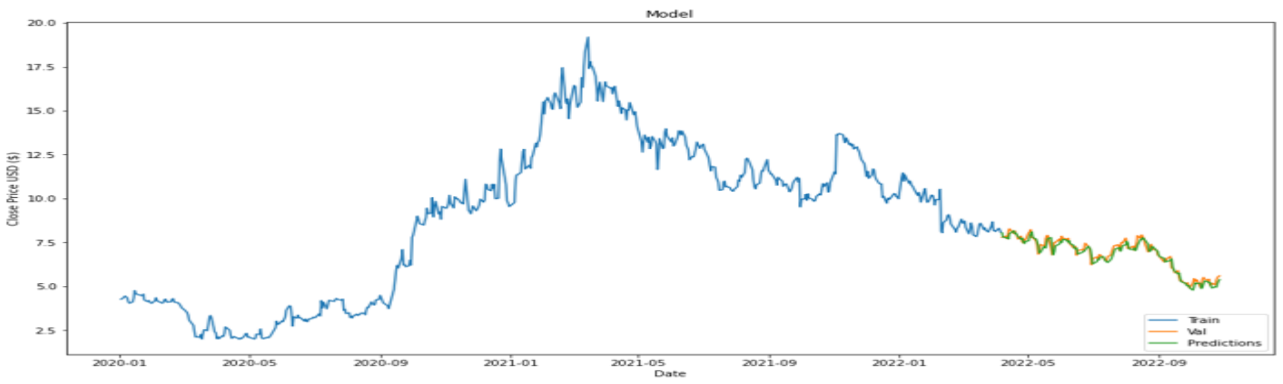
This is the LSTM network's more understandable diagram:

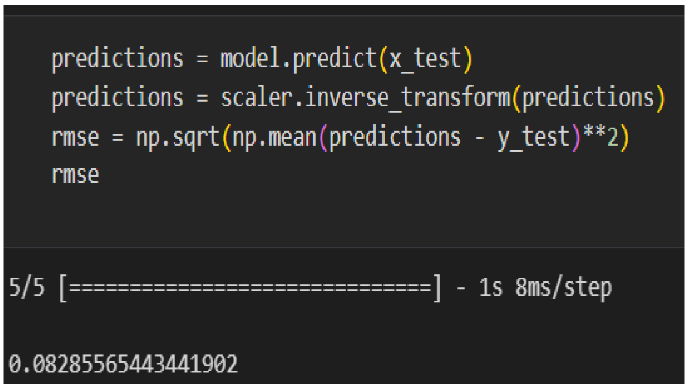


Results:

We scaled a single parameter from Model 1 to Model 2's four characteristics. Better depth of the Stock Market can be realised as a result. From graph 1 to 2 we can see that model 2 is better able to predict Highly volatile data.

1. The LSTM Model-1



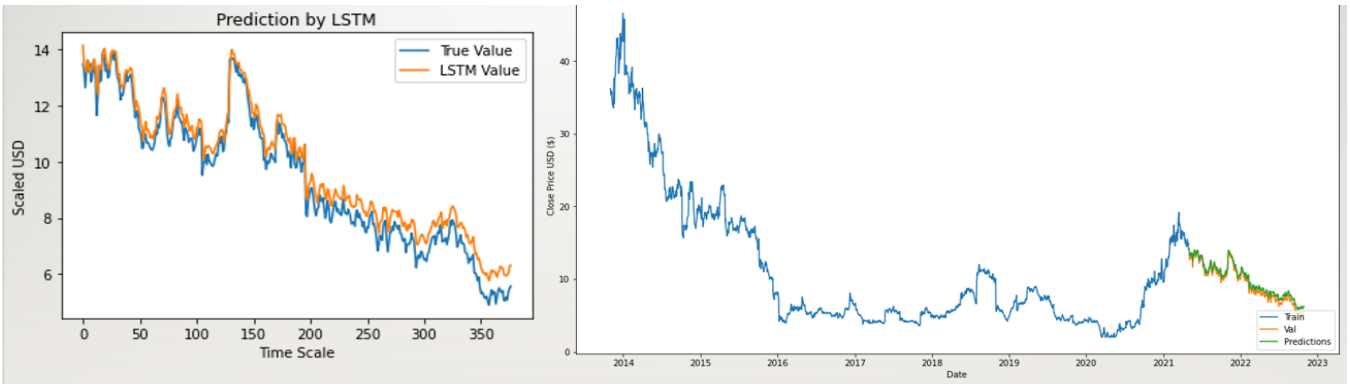


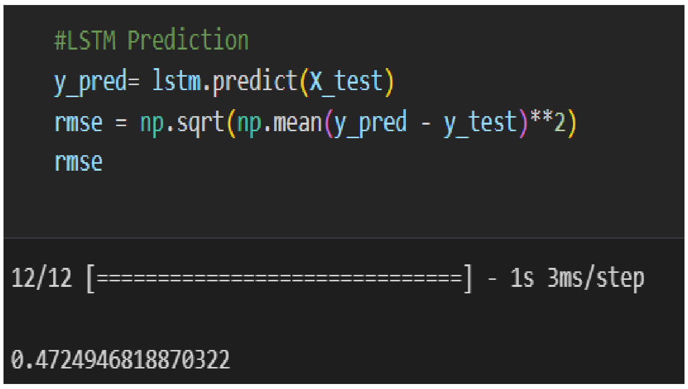
```
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
rmse = np.sqrt(np.mean(predictions - y_test)**2)
rmse
```

```
5/5 [==============================] - 1s 8ms/step

0.08285565443441902
```

2. The LSTM Model-2





```
#LSTM Prediction
y_pred= lstm.predict(X_test)
rmse = np.sqrt(np.mean(y_pred - y_test)**2)
rmse
```

```
12/12 [==============================] - 1s 3ms/step

0.4724946818870322
```

Conclusion:

Hence through several of these models we can conclude that Support Vector Machine are good to fit data but usually non-volatile data and is an optimizing algorithm. Moving to Deep learning Neural Network gives us more understanding of data and hence this can be seen in Long Short-Term Memory Model. From LSTM model 1 to model 2 it can also be observed that even more variation in data is predicted correctly.