Stock Price Prediction

Ojas Mehta
Computer Science Department
University of Massachusetts, Lowell
Lowell, Massachusetts, USA
Ojas_Mehta@student.uml.edu

Prachi Patel
Computer Science Department
University of Massachusetts, Lowell
Lowell, Massachusetts, USA
Prachi_Patel@student.uml.edu

Ryan Vavruska
Computer Science Department
University of Massachusetts, Lowell
Lowell, Massachusetts, USA
Ryan_Vavruska@student.uml.edu

***Abstract*—Making reliable stock predictions is a challenging task because of the complexity, uncertainties and regularly changing nature of stocks. Therefore, machine learning has often been used to predict these variations in the stock market to increase profits, and maximize gains. The objective of this paper is to test the validity of these machine learning models, compare their performances to each other by training them with a publicly available dataset from Kaggle, and provide meaningful insight into the strengths and weakness of different Machine Leaning approaches.**

***Index Terms*—Deep Learning, RNN, LSTM, KNN**

I. Introduction

Predicting the stock market is one of the most profitable businesses if done correctly. There are many factors that contribute to the price of stocks, which makes it highly volatile. However, the price is not completely random, it is rather chaotic in nature, and therefore can be predicted with some level of accuracy.

The idea of a stock market is fairly simple. Publicly traded companies are listed on it, and people can buy or sell the shares of these companies. The trades consist of stocks and equity which represent an ownership in that company. In the modern day stock market, the majority of big traders use trading bots to facilitate the buying and selling of stocks. For these bots to handle crucial transitions of buying or selling of stocks, they need to be fed with historical stock market data, as well as the instructions on how to utilize the data for predictions. That is where machine learning comes in.

With an increased focus from researchers towards machine learning applications, it is quickly becoming the new base for stock prediction technology. The reason for this is simple. With a machine learning approach, algorithms are able to analyze extremely large data sets, and find patterns in them, much more efficiently, when compared to a human's hand calculations. This allows for a much faster decision making. Machine learning approaches also have the ability of finding patterns in the data that the human eye may not be able to see, thus allowing for new ways of portraying the data to predict new points. However, these systems primarily focus on looking at past trends to predict modern day stock prices.

In this paper we discuss our progress on working with the historical data of IBM, applying it to several machine learning algorithms and testing which algorithm is best applied to the problem.

II. Related Works

There has already been some work regarding the application of machine learning to the stock prediction domain. Vijh, et al.[4] provide an approach to the field where they use an Artificial Neural Network (ANN) to attempt to predict stock market returns accurately. Their work consisted of using a dataset which contains stock data over ten years, taking the first 8 years and using it as their training set and the last 2 years as their training set in an 80:20 ratio. Once they had their data set, they implemented the ANN and tested their implementation against multiple publicly traded companies. They found an root mean-square error (RMSE) of roughly 1.5%. With this error value, Vijh et al. conclude that a Machine Learning approach using an ANN can be effective in accurately predicting stock market prices.

Parmar et al.[1] implement a machine learning model that attempts to predict stock prices using a Regression based model as well as LSTM based learning. Both methods seem to have their own strengths and weaknesses when it comes to predicting stock prices. Liu & Zhou[3] provide and compare a KNN implementation and a KNN-NNA implementation. They found that the stock market forecast prediction accuracy is decreased using KNN, but given their test results still serves as a fairly accurate prediction model.

III. Problem

There are many machine learning algorithms that exist that can be applied to machine learning. The problem we are aiming to solve is to find which algorithm is best applied to this sort of problem. The algorithms we are aiming to test our data set against are:

1) Linear Regression
2) Neural Networks
3) K-Nearest Neighbors

Once these algorithms are applied on our data set, we plan on comparing the output of these data sets against each other. These comparisons consist of the correctness of the algorithms, or how well their predictions align with the actual trend of our test data set. The other metric we are aiming to compare these algorithms on is their performance, or how much computational resources an algorithm takes to run.

To test these algorithms we need a data. Therefore, we are sourcing our dataset from Kaggle, which includes the historical stock data from a large number of companies. The data set can be found here: https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs

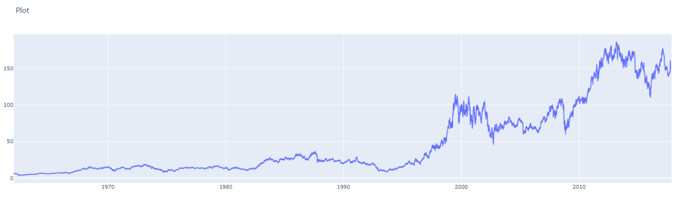


Fig. 1. Historical Data for IBM (1962-2017)

The above data set consists of many variables that makes it a good choice for a training set, as well as a testing set for our algorithms. These variables consist of:

- Date
- Open Price
- Close Price
- High
- Low
- Volume

The above variables consist of the date of the in which the data was taken, the open price, close price, high price, low price, and the volume of the stock on that day. This data is easily available in comma-delimited(csv) format to be read and applied to a machine learning algorithm.

IV. Methodologies

Linear regression

Linear Regression is one of the most basic, and commonly used machine learning algorithms used to predict the output of a continuous variable. It consists of two variables, one independent(x), and one dependent(y). Using this relationship, the algorithm calculates a best fit line. Here's what a sample linear regression graph looks like:

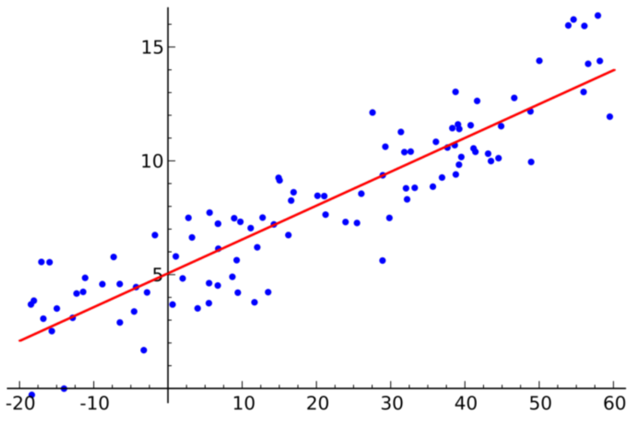


Fig. 2. Simple Linear Regression

The red line is based on the following equation:

$$Y = \theta_1 X_1 + \theta_2 X_2 + ... + \theta_n X_n \tag{1}$$

The purpose of the linear regression algorithm is to find the best fitting value for variables X_1 and X_2.

Cost Function: The cost function helps us calculate the best values for X_1 and X_2 which would further allow us to calculate the data points for a best fit line. To calculate X_1 and X_2, we need to minimize the error between the actual value, and the predicted value. This is done using the following equation:

$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

$$J = \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

Fig. 3. Minimize function

The difference between the actual values, and the predicted values would yield us the error difference. The square of that error difference, and the summation of all the data points, divided by the total number of data points gives us the Mean Squared Error(MSE). This can also be called the Mean Squared Error function. By applying the MSE to our variables, we can calculate its best values.

Gradient Descent: The gradient descent is a method for minimizing the cost function. By changing X1 and X2 at each iteration, we ensure that we are reducing the cost-function. It is expressed by the following formula:

$$b = a - \gamma \nabla f(a) \tag{2}$$

Here, Y(Stock Price) is a dependent variable. X_1, X_2, X_n (Features) represent the independent variables while the coefficients $\theta_1, \theta_2, ...\theta_n$ represent the weights.

Neural Networks

A Neural Network is a computational learning system that uses a network of function to understand and translate a data input of one form onto a desired output. The neural network learning algorithm learns input type from processing many labeled examples that are supplied during training and using this answer to learn what characteristics of the input are needed to construct the correct output. Once a sufficient number of examples have been processed the neural network can begin to process new unseen inputs and successfully return accurate results.

In general, Neural Network consists of three layers:

1) Input layers
2) Hidden layers
3) Output layers

In a Neural Network that only contains one hidden layer, the number of nodes in the input layer always depends on the dimension of the data. The input layer nodes connect to the hidden layer via links called synapses. The relation between every two nodes from input layer to the hidden layer has a coefficient called weight, which is the decision maker for signals.

The hidden layer nodes apply a sigmoid or tangent hyperbolic function on the sum of weights coming from the input layer, which is called the activation function, this transformation will generate values, with a minimized error rate between the train and test data using the SoftMax function. The values obtained after this transformation constitute the output layer of our Neural Network. These values may not be the best output, thus in this scenario, a back propagation process will be applied to target the optimal value of error. Then, the back propagation process will connect the output layer to the hidden layer, sending a signal conforming the best weight with the optimal error for the number of epochs decided. This process will be repeated trying to improve our predictions and minimize the prediction error. After completing this process, the model will be trained.

Stock prices are time series data so we chose to apply neural network algorithms like Recurrent Neural Network(RNN), Gated Recurrent Unit (GRU) and Long-Short Term Memory (LSTM).

Recurrent Neural Networks (RNN)

The classes of Neural Network that predict future value base on passed sequence of observations is called Recurrent Neural Network (RNN). RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Recurrent Neural Networks can be thought of as a series of networks linked together. They often have a chain-like architecture, making them applicable for tasks such as speech recognition, language translation, etc. An RNN can be designed to operate across sequences of vectors in the input, output, or both.

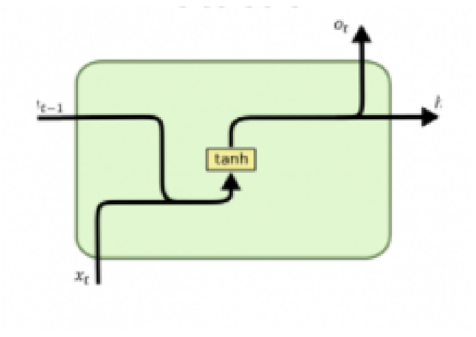


Fig. 4. RNN Architecture

The formulas that govern the computation happening in a RNN are as follows:

1) x_t is the input at time step t . For example, X_1 could be a one-hot vector corresponding to the second word of a sentence.
2) s_t is the hidden state at time step t. It's the memory of the network. s_t is calculated based on the previous hidden state and the input at the current step: s_t = f(UX_t +W$s_t - 1$) . The function f usually is a nonlinearity such as tanh or ReLU. s_t , which is required to calculate the first hidden state, is typically initialized to all zeroes.
3) o_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. o_t = Softmax(Vs_t)

There are a few things to note here:

- Unlike a traditional deep neural network, which uses different parameters at each layer, a RNN shares the same parameters across all steps. This reflects the fact that we are performing the same task at each step, just with different inputs. This greatly reduces the total number of parameters we need to learn.
- The above diagram has outputs at each time step, but depending on the task this may not be necessary. The main feature of an RNN is its hidden state, which captures some information about a sequence.

Gated Recurrent Unit (GRU)

Gated Recurrent Unit can also be considered as a variation on the LSTM because both are designed similarly and, in some cases, produce equally excellent results. GRUs are improved version of standard recurrent neural network. In GRU, these are two vectors which decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction.

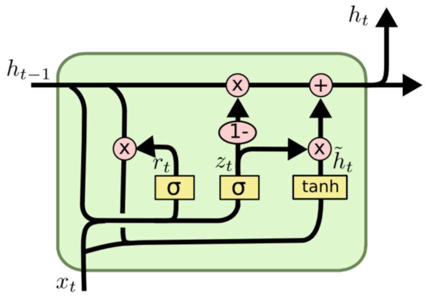


Fig. 5. GRU Architecture

We start with calculating the update gate z_t for time step t using the formula: $z_t = \sigma(W^z x_t + U^z h_(t-1))$ When x_t is plugged into the network unit, it is multiplied by its own weight W^z. The same goes for $h_t - 1$ which holds the information for the previous t-1 units and is multiplied by its own weight U^z. Both results are added together and a sigmoid activation function is applied to squash the result between 0 and 1.The update gate helps the model to determine how much of the past information (from previous time steps) needs to be passed along to the future. As the last step, the network needs to calculate h_t vector which holds information for the current unit and passes it down to the network. In order to do that the update gate is needed. It determines what to collect from the current memory content h'_t and what from the previous steps

$h_t - 1$. You can see how GRUs are able to store and filter the information using their update and reset gates.

Long Short Term Memory (LSTM)

Long short-term memory (LSTM) is a special kind of recurrent neural network, capable of learning long-term dependencies. It is used in the field of Deep Learning. It is commonly used for processing and predicting on the basis of time-series data. LSTM can store past important information, and forget the one that is not.
Following diagram describes the composition of Long Short Term Memory (LSTM) nodes.

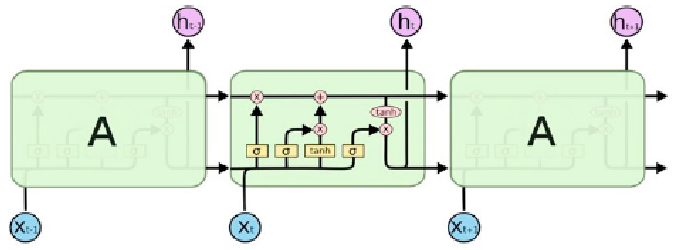


Fig. 6. The internal structure of LSTM

LSTM works in 3 layers process.

- 1^{st} Step is to decide which information is to be omitted from the cell in that particular time step. It is decided by the help of a sigmoid function. It looks at the previous state which is ht-1 and current input state which is Xt and computes the function.
- In 2^{nd} step, there are 2 layers. 1st is the sigmoid function and other is tanh function. In sigmoid function, it decides which values to let though which is 0 (let nothing pass through) or 1(let everything pass through). Hyperbolic tanh function gives weights to the values which helps decide the level of importance from -1 to +1.
- In 3^{rd} step. it decides what will be the final output. First it runs the sigmoid layer which decides which path of the cell state make it to the output, then put cell state to the tanh function to push the value between -1 and +1. And multiply with the output of the sigmoid key.

K-Nearest Neighbors (KNN)

K-Nearest neighbors is a simple yet effective machine learning algorithm when it comes to predicting new data points given trends in data. Typically KNN is applied to classification problems, for example classifying something as a certain type of object. While this is often the case, KNN can just as easily be applied to systems where the output is continuous. KNN primarily uses feature similarity to predict the value of new points. This is done by giving a value to each known data point and plotted mathematically, these being the training points. When we want to predict a new point, we plot that point on the graph and weight it based on how similar it is to it's neighbor points. The distance between the prediction point and the training points is then calculated. The most common way of finding the distance between two points is euclidean distance which can be found below:

$$d = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2} \tag{3}$$

Once the distance between the points is found, the closest k points are averaged together to find the value of the new predicted point.

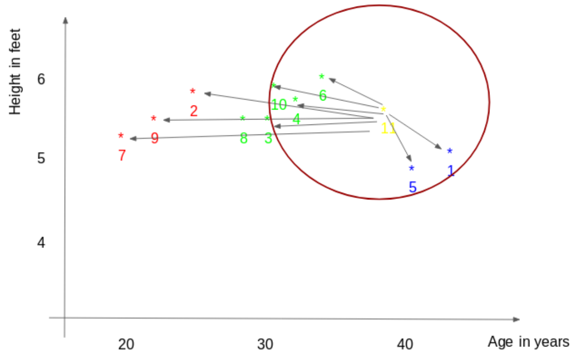


Fig. 7. Example of KNN

With the predicted value able to differ based on the chosen k-value, the optimal k-value must be found that is best attuned to the data set. This is found by calculating the root square mean error between the training and testing sets using different values of k. The goal is to reduce the value of k, and with doing this allows us to find the best value of k to most correctly predict new data points.

$$RMSE = \sqrt{\frac{\Sigma_{i=1}^{n} (\hat{y}_i - y_i)^2}{n}} \tag{4}$$

With stock predicting KNN is going to be implemented in the following way:

- 1^{st} step, assign a formula to translate each data point into a mathematical point on a a plot. This point will include the opening price, closing price, high, low, and volume of the stock.
- 2^{nd} step, calculate the optimal k-value for the data-set by testing the training set vs. the testing set.
- 3^{rd} step, predict a series of new points using the optimal k-value.

V. Model Setup and Results

These are the common steps that we follow to implement each of the models.

- 1^{st} step, collect data from the public data set available on Kaggle. The data set has been explained in detail previously in this paper.
- 2^{st} step, preprocess the data which includes, cleaning the data, and splitting it in training and testing set.
- 3^{rd} step, create a model based on the algorithm.

- 4^{rd} step, plot the closing price for the training data set, and the predicted closing price for the testing data set.

Linear Regression

We implemented our linear regression model with scikit-Learn, and used mean_squared_error and r2_score function for evaluating the regression model. First, we split the data in to training and testing data set. Then, we used the Linear-Regression function to run the Linear Regression algorithm on the data set. Below figure shows the result of linear regression model.

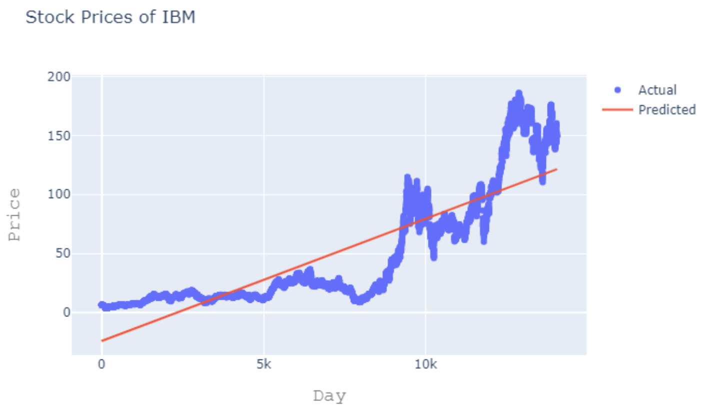


Fig. 8. Linear Regression on IBM stocks

This graph shows actual and predicted value for the training data set. The X-axis has the days and y-axis has closing prices of the stock. The red line is the predicted regression line. Based on that, we can calculate the scores for evaluating our model on training and testing data.

Metric	Train	Test
r2_score	0.7289	0.7321
MSE	660.2305	647.9331

As we can see, linear regression is not good at predicting stock prices. This is primarily because the algorithm works only based on the date, and the closing price. It doesn't consider any other factors in calculating the final prediction. As a result, the prediction stays constant based on how the stock has performed prior. In our case, since IBM's stock price has increased, the algorithm expects it to keep increasing. Due to this nature of linear regression, the fluctuations are not handled well. However, the data collected from computing the linear regression model can be quite useful in studying the past trends of a stock. It gives us a better idea of how the stock has performed over the years.

Neural Network

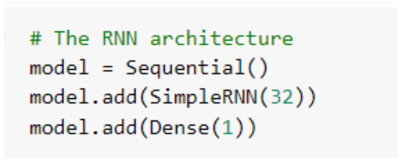
Each Neural Networks were implemented using *keras* library. Steps to implement NNs are as below,

- Data Processing
- Data Scaling
- Model Implementation
- Result Visualization

In order to implement Neural Networks, data processing is required which includes splitting data into training and test data. I used data from 1995-2016 for training and predicted price for year 2017. next step is to normalize data. Rather than feeding actual closing price to a model, we normalized them. Normalizing the value improves convergence. This formula was used to normalize them: $n_i = (p_i/p_0) - 1$. This gives the percentage changes from starting point. Once the model makes prediction we denormalize the data to get actual value using this formula: $p_i = p_0(n_i + 1)$. Then each neural networks were implemented and predicted closing prise were shown on graph.

Recurrent Neural Networks (RNN)

Recurrent Neural Networks can be implement using single-layer recurrent network with a small number of nodes and the model can be improved by adding more nodes or more layers. I implemented RNN using single layer using 32 input nodes.



```
# The RNN architecture
model = Sequential()
model.add(SimpleRNN(32))
model.add(Dense(1))
```

Fig. 9. RNN Model

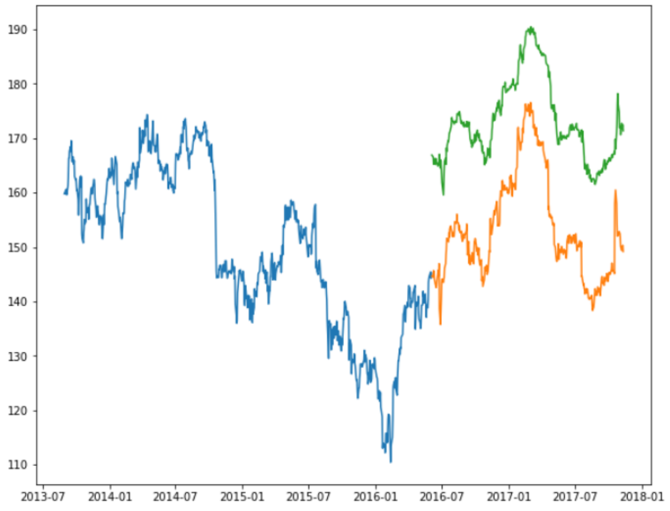
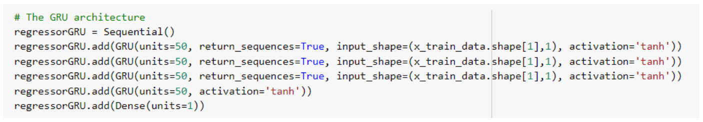


Fig. 10. RNN on IBM stocks

As you can see in the graph, training data is shown in blue, test data is in orange and green line shows the predicted price for the test dataset.

Gated Recurrent Unit (GRU)

In practice the GRU and LSTM layers frequently achieve similar results, although LSTM layers did slightly outperform. However, GRU layers are generally preferred due to being less computationally expensive. I implemented GRU model using multi layers.



```
# The GRU architecture
regressorGRU = Sequential()
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(x_train_data.shape[1],1), activation='tanh'))
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(x_train_data.shape[1],1), activation='tanh'))
regressorGRU.add(GRU(units=50, return_sequences=True, input_shape=(x_train_data.shape[1],1), activation='tanh'))
regressorGRU.add(GRU(units=50, activation='tanh'))
regressorGRU.add(Dense(units=1))
```

Fig. 11. GRU Model

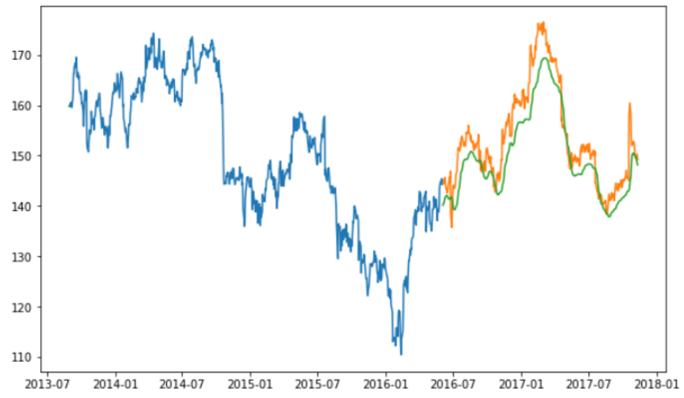
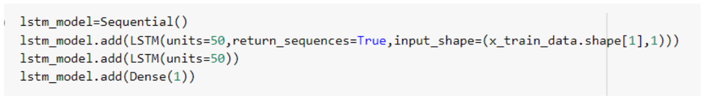


Fig. 12. GRU on IBM stocks

By looking at the predicted stock value in green we can see that GRU gives better result than RNN.

Long Short Term Memory (LSTM)

Long-Sort Term Memory models are able to use long-term dependencies. As you can see in below figure that I used multi-layer LSTM model. These advance layers do better job of retaining older information to use in future predictions. The downside of using multi-layer is that, they have tendency to overfit. The solution to that is to add dropout. However for LSTM it is not recommended because in fitting to time-series data, each node may carry information that you don't want to be dropped.



```
lstm_model=Sequential()
lstm_model.add(LSTM(units=50,return_sequences=True,input_shape=(x_train_data.shape[1],1)))
lstm_model.add(LSTM(units=50))
lstm_model.add(Dense(1))
```

Fig. 13. LSTM on IBM stocks

Below figure shows stock prediction using LSTM model

X axis on the graph represents days and y axis shows stock closing price of the stock. Orange line indicates actual stock price while the green line indicates the predicted stock price. From looking at the all Neural Networks graph, we can conclude that LSTM gives better performance than RNN and GRU.

K-Nearest Neighbors (KNN)

Before applying K-Nearest Neighbors to the IBM stock data, we had to solve for a value of K that best applied to the data set. We did this, by testing the values of k in the range of [1, 90] to see which one provided the lowest Root

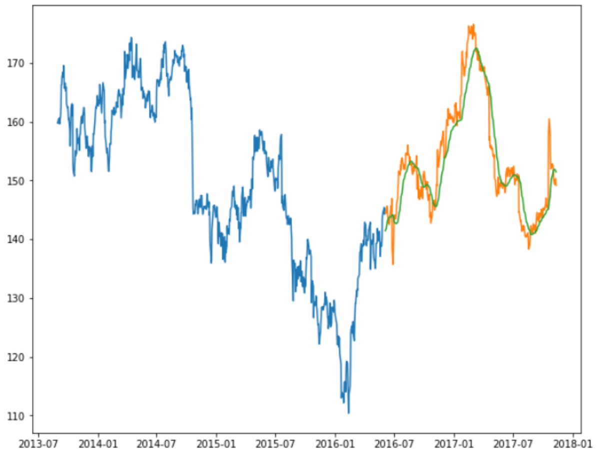


Fig. 14. LSTM on IBM stocks

Mean Squared Error (RMSE). To calculate the RMSE for the range of k-values, we segmented the data-set into an 80:20 ratio, the first eighty percent of the data being considered the training data-set and the remained twenty percent being the testing set. The below graph shows the k-value plotted against the RMSE.

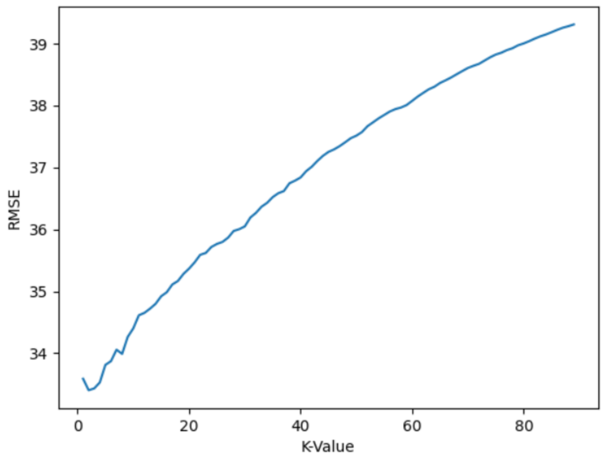


Fig. 15. RMSE for K-Values

As seen in figure 15, the optimal k-value that produces the lowest RMSE for the IBM data-set is 2. With the k-value found, it is now time to predict the stock prices using the calculated k-value. The k-value is then used to predict new stock data as seen below:

Figure 16 shows that K-Nearest Neighbours reliably predicts stock-values for ranges that is has already seen, since it is able to average the k-nearest points to solve for the new point. Where KNN struggles, as seen in the figure, is predicting new points that do not already exist in a range that has been seen before. This can be seen with the green line, or the predicted

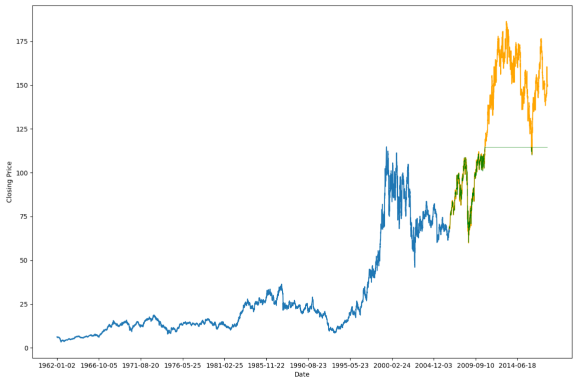


Fig. 16. Estimated Stock Prices

prices flat-lining once the data becomes out of range of the training set.

With the KNN model implemented against the IBM data-set, it can be concluded that that a regression based implementation of KNN at it's base form does not do a good job at predicting stock values outside of it's trained range. A better application of KNN is one where it has a fully fleshed out data-set with some holes in the data that you are trying to fill. With these findings it can be concluded that regression based KNN is not viable for predicting stock prices with a volatile data set. It is much better tuned to datasets without much variation, with new data points being easily extrapolated using existing data. The rational for this is, KNN is known to struggle with large irregularities in data, as well as is very sensitive to it's initial data-set. Because of these two factors, KNN does not apply well to the IBM data-set. However, KNN on stock data without much variation might prove to be more effective.

VI. Future Work

With stock prediction being a very complex task, which is extremely chaotic in nature, there is a lot of future work that can be applied to this topic. Given other work currently being done in the field by researchers as well as our own, there is a chance for the Neural Network portion of this paper to be expanded. This can be done by combining different network types, as well as processing that data using different algorithms before applying it to a network to create a more versatile model.

VII. Conclusion

As we know predicting stock price is a very complex task and has uncertainties. Even though our testing shows LSTM performing well on the given dataset, there is no guarantee it will perform well in real time. The prediction model we implemented are only based on the past data. There are many other factors that affects stock price like: the company's history, current news, or political environment. These factors can be added to out current model as parameter to get more reliable stock prediction.

VIII. Appendix

Task Distribution:

- Ojas - Linear Regression
- Prachi - RNN, GRU, and LSTM
- Ryan - K Nearest Neighbors

Each member was responsible for taking one machine learning algorithm and implementing and testing it against the data set.

References

[1] Parmar et al., ”Stock Market Prediction Using Machine Learning,” 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC), Jalandhar, India, 2018, pp. 574-576, doi: 10.1109/ICSCCC.2018.8703332.

[2] S. Sarode, H. G. Tolani, P. Kak and C. S. Lifna, ”Stock Price Prediction Using Machine Learning Techniques,” 2019 International Conference on Intelligent Sustainable Systems (ICISS), Palladam, India, 2019, pp. 177-181, doi: 10.1109/ISS1.2019.8907958.

[3] San-hong Liu and Fang Zhou, ”On stock prediction based on KNN-ANN algorithm,” 2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), Changsha, China, 2010, pp. 310-312, doi: 10.1109/BICTA.2010.5645310.

[4] Mehar Vijh, Deeksha Chandola, Vinay Anand Tikkiwal, Arun Kumar, Stock Closing Price Prediction using Machine Learning Techniques, Procedia Computer Science, Volume 167, 2020, Pages 599-606, ISSN 1877-050, https://doi.org/10.1016/j.procs.2020.03.326.

[5] K. Abhishek, A. Khairwa, T. Pratap and S. Prakash, ”A stock market prediction model using Artificial Neural Network,” 2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12), 2012, pp. 1-5, doi: 10.1109/ICCCNT.2012.6396089.

[6] W. Chen, Y. Zhang, C. K. Yeo, C. T. Lau and B. S. Lee, ”Stock market prediction using neural network through news on online social networks,” 2017 International Smart Cities Conference (ISC2), 2017, pp. 1-6, doi: 10.1109/ISC2.2017.8090834.

[7] Huyen Giang Thi Thu, Thuy Nguyen Thanh, Tai Le Quy, ”A Neighborhood Deep Neural Network Model using Sliding Window for Stock Price Prediction”, Big Data and Smart Computing (BigComp) 2021 IEEE International Conference on, pp. 69-74, 2021.

[8] R. Iacomin, ”Stock market prediction,” 2015 19th International Conference on System Theory, Control and Computing (ICSTCC), 2015, pp. 200-205, doi: 10.1109/ICSTCC.2015.7321293.

[9] R. Verma, P. Choure and U. Singh, ”Neural networks through stock market data prediction,” 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA), 2017, pp. 514-519, doi: 10.1109/ICECA.2017.8212717.

[10] I. Jahan, S. Z. Sajal and K. E. Nygard, ”Prediction Model Using Recurrent Neural Networks,” 2019 IEEE International Conference on Electro Information Technology (EIT), 2019, pp. 1-6, doi: 10.1109/EIT.2019.8834336.