Predicting and Comparing Bitcoin Prices using LSTM and ARIMA

Michael Tsao
Computer Science Dept.
Ryerson University
Toronto, ON CANADA
michael.tsao@ryerson.ca

Sai Nidumukkala
Computer Science Dept.
Ryerson University
Toronto, ON CANADA
sai.nidumukkala@ryerson.ca

Abstract

In this study, we use ARIMA and LSTM to predict the price of Bitcoin, a cryptocurrency based on time series. We also examine the effects of Bitcoin's volatility on traditional regressive and advanced neural network based models. There are many kinds of regression models including Autoregressive (AR), Moving Average (MV) and Autoregressive Integrated Moving Average (ARIMA). There are also deep-learning based algorithms such as 'Long Short Term Memory (LSTM)'. In this particular study, we compare ARIMA vs LSTM and through empirical results show that ARIMA is a superior forecasting model when it comes to a time-series based datasets. We also examine the effective use of data preprocessing techniques to improve model performance. Finally, we combine ARIMA and LSTM through an ensemble bagging technique to produce a model that improves upon existing models.

1 Introduction

Bitcoin is a cryptocurrency that utilizes a blockchain as a ledger for its transactions. Bitcoin is a time series data and predicting it is mainly a challenging task mainly due to unprecedented changes in economic trends and conditions. In the last few years, market volatility has caused a serious fluctuation in bitcoin prices. It has seen various price bubbles which peaked at 26,455.61 US Dollars and as of the current date is at 9,282.21 US Dollars. **[1]** Therefore, there is an increasing need to forecast market movement using several regression techniques in order to reduce price volatility and increase consumer participation.

The main objective of this paper is to forecast using a stochastic time-series based model. The most popular method is a univariate "Auto-Regressive Moving Average(ARMA)" for a single time series data where Auto-Regressive (AR) and Moving Average (MA) values are combined. Whereas, "Auto-Regressive Integrated Moving Average (ARIMA)" is a type of regressive model where differencing between AR and MA are taken into consideration.

In recent years, deep learning algorithms have improved over regressive techniques where relationships between variables and attributes are modeled in a deep and layered hierarchy. Some machine learning based techniques such as Support Vector Machines (SVM), Random Forests (RF) as well as deep learning algorithms such as Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) have gained popularity especially in a time series based forecasting problems. Deep learning methods are able to identify structure and pattern of data such as non-linearity and complexity in time series forecasting.

This paper will compare ARIMA and LSTM with respect to their forecasting accuracy and minimizing error. ARIMA was chosen mainly because of its non-stationary property when it comes to data modeling and analysis. Whereas, LSTM was chosen because of its ability to preserve and train the features of a given data over a long period of time.

The paper will provide an in-depth look into data processing including tweaking hyper-parameters and also improving the forecasting results using ensemble techniques for LSTM and ARIMA models. It will also conduct an empirical study and analysis of the findings as well as conclude the results and performance of LSTM and ARIMA as error reduction tools.

2 Time Series Forecasting

The main purpose of studying time series analysis is to make observations on the time-data path and also to build a model to describe the structure of data and predict the future values of the time series. Because, time series is used in many scientific fields in addition to stocks, there is a need to build an effective model with the goal of improving the forecasting accuracy.

Although ARIMA has been an industry standard for financial forecasting there are some serious limitations to its functionality. For example in ARIMA, it is hard to model non-linear relationships between variables. Also, there is a constant standard deviation in errors. To combat this, the common approach has been to integrate it with Generalized Auto-Regressive Conditional Heteroskedasticity (GARCH) model. **[2]** But in many time-series data including Bitcoin, it is challenging and hard to optimize the GARCH model.

New techniques have been developed to address the non-linear forecasting challenges in ARIMA including the rise of LSTM. LSTM (Long Short Term Memory) is a special architecture of Recurrent Neural Network (RNN) that was introduced by Hochreiter and Schmidhuber. **[3]** Even though, this deep learning technique is fairly new, it has experienced massive adoption in the financial and research world. The main idea was to build various portfolios of stocks including Bitcoin by adjusting the internal RNN layers and other hyperparameters to deliver a threshold level of return.

3 Background

This section of the paper will go into mathematical background of the time series techniques (ARIMA and LSTM).

3.1 ARIMA

ARIMA combines Autoregressive *(AR)* process and Moving Average *(MA)* process to build a composite time-series model. As seen in the jupyter notebook, *ARIMA(p,d,q)* requires various observation parameters in order to minimize errors during forecasting.

-*AR:* AutoRegression. A regression model that uses the dependencies between an observation and the number of lagged observations *(p).*
-*I:* Integrated: Makes the time series stationary by measuring the differences of observations at different times *(d).*
-*MA:* Moving Average: Approach that takes the dependency between observations and residual error when a moving average model is used to the lagged observation *(q)*.

AR model of order p, *AR(p)* can be written in a linear process as:

$$x_t = c + \sum_{i=1}^{p} \phi_i x_{t-i} + \epsilon_t \quad \textbf{[4]}$$

where x_t is the stationary variable, c is the constant and the terms in φ_i are autocorrelation coefficients at lags 1,2,p and ε_t . Also the residuals are the Gaussian white noise with mean 0 and standard variance

MA model of order q, *MA(q)* can be written in the form:

$$x_t = \mu + \sum_{i=0}^{q} \theta_i \epsilon_{t-i} \quad \textbf{[4]}$$

where μ is the expectation of x_t , the θ_i terms are the weights applied to the current and prior values of terms in the time series. Similar to AR model, we also assume that the residuals are the Gaussian white noise with mean 0 and standard variance

When we combine the two above models and add them together, we form the ARIMA model of order *(p,q)* .

$$x_t = c + \sum_{i=1}^{p} \phi_i x_{t-i} + \epsilon_t + \sum_{i=0}^{q} \theta_i \epsilon_{t-i} \quad \textbf{[4]}$$

Where φ_i and θ_i is not equal to 0 and standard variance is greater than 0. The parameters p and q are called the AR and MA orders. ARIMA is capable of dealing with non-stationary time series data due to its integrating step. Hence, the integrating component involves differencing the time series to convert a non-stationary time series into a stationary time series. The general form of ARIMA representation is *ARIMA(p,d.q).*

There may also be non-seasonal components that contribute to the model. Therefore, a general form of ARIMA usually incorporates both seasonal and non-seasonal components in the form *ARIMA(p,d,q) x (P,D,Q)S* , where p is the non-seasonal *AR* order, d is the non-seasonal differencing, q is the non-seasonal *MA* order, P is the

seasonal *AR* order, D is the seasonal differencing, Q is the seasonal *MA* order and S is the time span of repeating seasonal patterns.

The critical step in estimating seasonal ARIMA is to identify the values of *(p,d,q)* and *(P,D,Q)*. We should also use variance-stabilizing transformations and differencing. Then, use autocorrelation function (ACF) to measure the amount of linear dependence between observations in a time series that are separated by a lag *p*, and partial autocorrelation function (PACF) to determine how the number of autoregressive terms *q*, are necessary and then inverse the autocorrelation function (IACF) in order to detect overdifferencing. Similarly, we can determine the preliminary values of (*p,d.q)* along with their corresponding seasonal parameters *(P,D,Q)*. Also, the parameter *d* is the order of the frequency difference that occurs when we transition from non-stationary time series to a stationary time series.

3.2 LSTM

Long Short Term Memory (LSTM) is an architecture of the Recurrent Neural Network (RNN) with the capability to remember the values from earlier stages to be used in future use cases. The memorization of the earlier data trends is possible due to the presence of various gates as well as an incorporated memory line.

Each LSTM is a set of cells or system modules where the data streams are captured and stored. The cells typically resemble a transport line where one module is connected to another in order to convey data from the past and gather them for the present module. Due to the presence of gates, the data in each cell can be disposed, filtered and or added to the next cell. Therefore, the gates are usually based on sigmoidal neural network layer, which typically enables cells to optionally let the data pass through or be disposed.

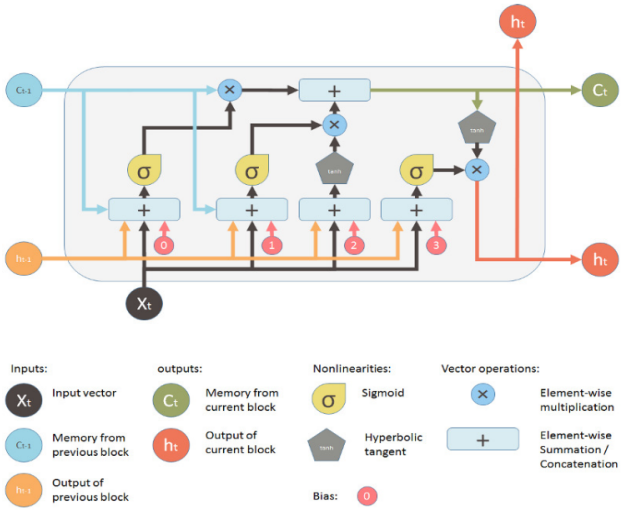
Each sigmoidal layer yields numbers in the range of zero and one, which determines the amount of data that needs to be let through each cell. Typically zero means “Nothing shall pass through” and one means that “Everything must pass through”.

In a typical LSTM, three types of gates are involved in controlling the state of each cell as shown in Figure 1.
-Forget Gate: outputs a number between 0 and 1. Where 1 means ‘keep this completely’ and 0 means ‘ignore this completely’.

-Memory Gate: it helps choose which new data needs to be stored in the cell. Usually, a sigmoidal layer called the ‘input door layer’ chooses which values need to me modified and then a tanh layer makes a vector of new candidate values that could be added to the state.

-Output Gate: it typically decides what will yield out of each cell. The yielded value is typically based on the cell state along with the filtered and newly added data.

Figure 1: LSTM Cell **[5]**



4 Problem Domain

The authors faced many problems with forecasting the Bitcoin Price. One of the many reasons could be due to its volatility in the open market. This introduced a lot of noise and spikes in the early years of its availability. For example, a huge public interest can influence the price unpredictably compared to normal patterns. The last spike occurred in late 2017 due to an increased popularity and a rising demand for using cryptocurrencies as a means of financial investments.

Due to this inherent nature of the time series data being polluted by noise, a robust machine learning algorithm must adapt to noise constraints during the forecasting process.

In this study, the Bitcoin data gathered would be in the form of a time series data. Although there are a wide range of classifiers and deep learning algorithms available, according to the paper ‘Bitcoin Price Prediction Using Machine Learning’ by Neha Mangla and Akshay Bhat in 2019 **[2]**, it was discovered in their research that LSTM and ARIMA classifiers performed the strongest with an accuracy

of over 50%. Therefore a decision was made to adopt the models in this paper's Bitcoin time-series study and to improve upon their findings through tuning of the hyperparameters as well as creating an ensemble method. This paper will also use the Mean Absolute Error (MAE) to perform forecasting error analysis.

5 Data Preparation and Cleaning

5.1 Data Gathering

For our dataset, we used a dataset put together from large bitcoin exchange services such as Coinbase and Bitstamp which provide minute to minute updates on the attributes. The timestamps are in Unix time and the data fields with missing activities are labeled NaN. The dataset has several columns with the following attribute descriptions:

- Timestamp - the time in unix time
- Open - price at market opening
- High - highest price for the day
- Low - lowest price for the day
- Close - price at market close
- Volume (Bitcoin) - amount of bitcoin at current date
- Volume (Currency) - amount of dollar value given the amount of bitcoin in circulation
- Weighted Price - the average price for the day

Timestamp	Open	High	Low	Close	Volume_(BTC)	Volume_(Currency)	Weighted_Price
1325317920	4.39	4.39	4.39	4.39	0.455580087	2.00000002	4.39
1325317980	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1325318040	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1325318100	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1325318160	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table 1: **Sample of Bitcoin Price Dataset (coinbaseUSD_1-min_data_2014-12-01_to_2019-01-09)**

6 Data Preprocessing

6.1 Data Cleaning

With the data that was obtained, a discovery was made that most of the data was not clean. As seen in Table 1, most of the rows contained a lot of NaN values and the models needed these values to be removed before proceeding to train the model. Therefore, in the LSTM model the decision was made to group the items by date. Whereas, in ARIMA, a forward fill mechanism was applied to get rid of the NaN values since the adjacent time-series data are very close in a stochastic model such as Bitcoin.

Additionally, a decision was made to limit the dataset in range 2014-2019 as there is more stable growth and less volatility in that range, when compared to the years 2012-2013. Also to reduce data noise, additional features like Google Trends and Sentimental Data were no longer considered.

Although many other data attributes like Volume(Currency) were related to the pricing models directly or indirectly, a decision was made to focus on the Average Weighted Price to help keep the models simple and efficient.

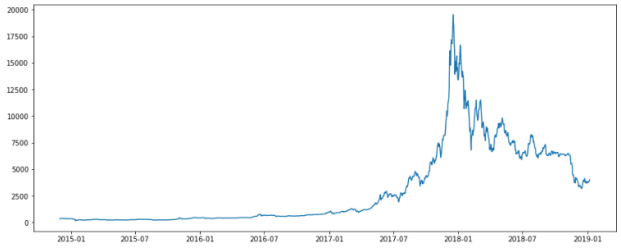
6.2 Data Normalization

Sampling through the data, an observation was made where using the given Average Price in the dataset resulted in a massive point difference. From the graph 1 (below), we see that for years (2015-2017) the price remains somewhat linear in the range $100 to $1800. But suddenly spikes to around $20,000 within years (2017-2018). Therefore to keep the machine learning models accurate, a decision was made to scale the data. In this study, we will be using the MinMax Scaling technique to keep the values in the range 0 to 1 to keep data across all the attributes relative and homogenous.

A MinMax scaling is typically performed with the equation:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Since the dates were listed in Unix time, a conversion was performed to make it more reader-friendly.



Graph 1: **Graph of Actual Price History from 2015-2019 using cleaned data**

7 LSTM and ARIMA Model Building

In this study, both LSTM and ARIMA models will be used to forecast and predict the Bitcoin prices for approximately the last 30 days (2018-12-09 to 2019-01-07) from the given dataset. Later a mean squared error (MAE) against the actual vs. predicted data points will be performed. This will help the study conclude between the strengths and weaknesses of LSTM and ARIMA as time-series forecasting and optimizing tools.

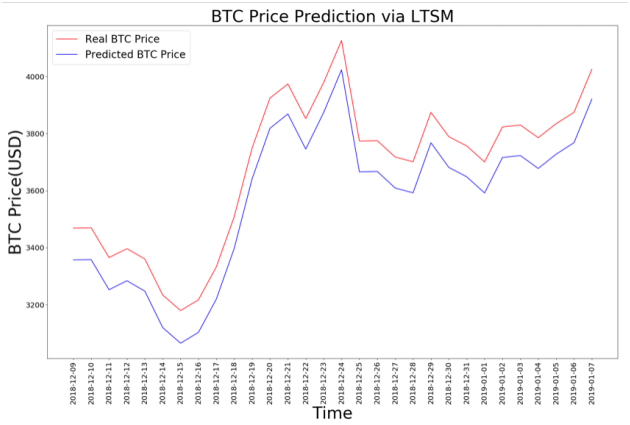
The initial hypothesis of this paper expects ARIMA to perform better than LSTM due to its proven time-series tracking record in the financial field, including the stock market. ARIMA is more suited for data with a moving average over time. We can see in Graph 1 (above) that this assumption holds true for Bitcoin prices. When we compare yearly averages or general trend of the data, we would find that the data is slowly growing over time. In other words, ARIMA can determine and analyze the change in growth of the price for some time period and use that prediction to determine the actual price for a given time period.

Compared to LSTM, there are similarities on how the data is imported and processed to train the classifier. For example, LSTM trains the models using batch sizes. This would be the equivalent of the period in ARIMA used for averaging the data over time. The only difference is that the neural network must implicitly learn of the increasing average price trend while ARIMA which is designed to explicitly predict based on that trend. This would explain why LSTM performs worse than ARIMA.

Overall, both models performed well. LSTM performed very well during training but ARIMA performed better in terms of predictions.

7.1 Long Short-Term Memory (LSTM)

Our initial results, from this performed really well. We got a mean squared error at around 100 to 10 dollars. The one issue with this model is that it produced some variance. We hope to address this when we combine the classifiers.



Graph 2: **Graph of Results from our LSTM classifier**

7.1.1 LSTM Model Hyperparameters

This section will talk about the effects of changing the hyper parameters for the LSTM model.

Using the Keras library documentation, we identified many parameters that could allow us to better fine tune our model to reduce our errors before applying the ensemble. We now list several notable parameters.

To see a full list of our empirical results for tuning the LSTM classifier. See attached file.

Optimizer:
Optimizer are algorithms that is used to reduce the error between epochs which would result in a better approximation of the curve. From the Keras documentation, our choices were between optimizer: Stochastic Gradient Descent (SGD), Adam, Adagrad and RMSProp.

Overall, we found that RMSProp gave the lowest error, the best fit and the least variance compared to other optimizers. SGD produced a very high error due to the optimizer getting stuck at a local minimum.

Loss Function:

Loss Function is the function that is used to calculate the error. It is used by the optimizer to reduce the error.

Overall, we found that mean squared error produced the most relevant error as it directly relates to the price. In addition, it also allowed reaching the local minimum faster compared to other faster compared to other loss functions.

Activation Function:
Activation function introduces nonlinearity to the neural network.

During our testing, the Sigmoid activation appears to be better at returning a lower loss compared to other activation functions which gets stuck at a local minima producing higher errors and more variance.

Dropout Rate:
Dropout Rate is the rate at which neurons are turned off in the network. With LSTM, this is accomplished by adding an extra node in our neural network.

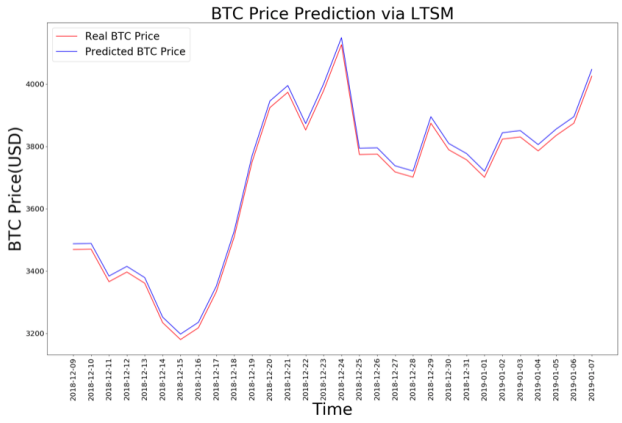
Our results showed that any amount of drop off did not affect our model in any meaningful way. Drop off was omitted from the final model.

Number of Neurons:
Defines the number of neurons in the hidden layer to use in the network.

Our results show that increasing the number of neurons did make a better fit but overall increased the variance too much to consider it for our final model. We would settle on 4 neurons as they provided the best trade off between fit and variance.

Batch sizes and Epochs:
Epoch defines the number of cycles through all the training points. Batch sizes define the size of the group of points that will be averaged to perform the gradient descent.

In our testing, we found that having a higher epoch size over fitted the data and a higher batch size proved to have a faster gradient descent but reached a local minimum. For our final model, we chose a batch size of 5 and an epoch of 100.

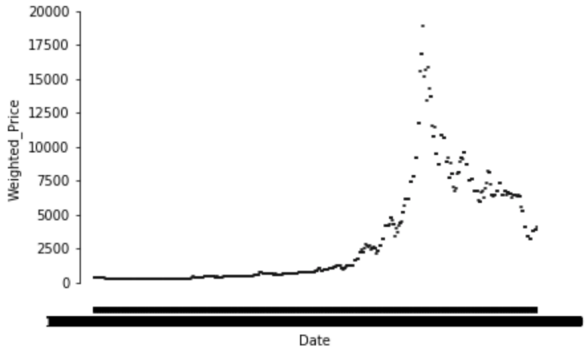


Graph 3: **Graph of Results from our LSTM classifier with tuned parameters**

7.2 ARIMA (Auto Regressive Integrated Moving Average)

As discussed in the Background section earlier, ARIMA is a class of models that captures temporal structure in time series data. ARIMA is a linear regression based forecasting approach.

Time Series analysis in ARIMA requires data to be stationary. Based on Graph 4 (below) this is clearly not the case.



Graph 4: **Weighted Bitcoin Price vs. Date**

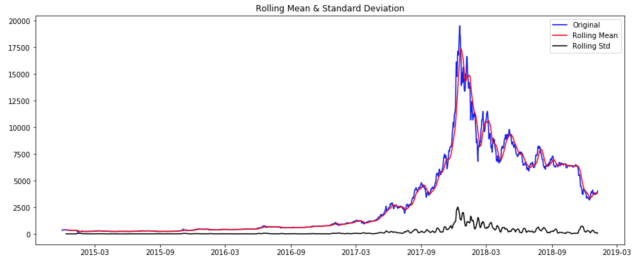
Dickey Fuller Test:
A statistical test that is used to test the null hypothesis that a unit root is present in an autoregressive model.

A simple AutoRegressive Model:

$$y_t = \rho y_{t-1} + u_t$$

where yt is the variable of interest, t is the time index, p is a coefficient and ut is the error term. A unit root is present if p=1.

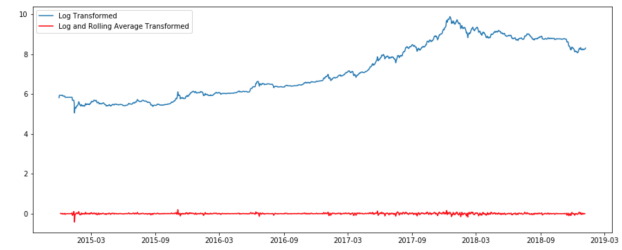
To be able to successfully make the data stationary, we needed to bring it down below the Critical Value (1%) of -3.434794. As we can see in the Graph 5 (below), at present the p value is at 0.379781 (very high). Therefore, the p value is too high to reject the null hypothesis.




```
Results of Dickey-Fuller Test:
Test Statistic                  -1.801372
p-value                          0.379781
#Lags Used                      24.000000
Number of Observations Used    1474.000000
Critical Value (5%)              -2.863503
Critical Value (1%)              -3.434794
Critical Value (10%)             -2.567815
dtype: float64
```

Graph 5: **Dickey-Fuller Test on Standard Data**

Hence, a log transform was applied to flatten the curve to a near linear line.




```
Results of Dickey-Fuller Test:
Test Statistic                -1.435010e+01
p-value                        1.025185e-26
#Lags Used                     6.000000e+00
Number of Observations Used    1.486000e+03
Critical Value (5%)           -2.863487e+00
Critical Value (1%)           -3.434758e+00
Critical Value (10%)          -2.567807e+00
dtype: float64
```

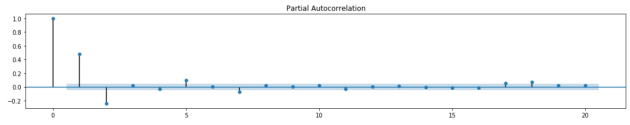
Graph 6: **Dickey Fuller Test on Log-Transformed Data**

Rolling Average:
A rolling average is a calculation to analyze data points by creating a series of averages of different subsets of the full data set.

By applying the above statistical methods of rolling average, log transform and date finetuning, we were able to bring down the p value to 1.025185e-26 which is far below the Critical Value (1%) of -3.434758e+00 as can be seen in Graph 6 (above). Therefore, the data is now stationary.

To correlate the data, ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) were used to justify AR, MA, ARIMA models and help calculate the p,q and d values.

A total of 20 lags and the PACF function was used to visualize the confidence level. Notice from Graph 7 (below) that the first entry to break the upper confidence is the second dot (graph starts from 0). Therefore, the values of p = 2, q = 2 and the standard value of d =1 were determined.



Graph 7: **PACF confidence chart**

Finally the resulting ARIMA data points were fitted and the cumulative sum was used to predict the last 30 days (2018-12-09 to 2019-01-07) from the given dataset. The result can be seen in Graph 8 (below).

Graph 8: **Graph of Results from our ARIMA classifier with tuned parameters**

The initial hypothesis of this study predicted that ARIMA would perform better than LSTM based on the research paper by Neha Mangla which supports this presumption. **[6]** Based on the findings of our own study above, the hypothesis appears to be true.

8 Improvement via Ensemble Bagging

Our proposal for using an ensemble methodology would be to perform bagging (weighted average of the results of the two classifiers based on the number of votes). The weights we choose will be proportional to the accuracy averaged over the course of several trials.

According to research by Bauer and Kohavi, it was found that given a series of weak learners we can improve the overall predictions using a bagging or random forest technique. **[7]** Our rationale for doing bagging is because when researching about the topic we saw that RNN and ARIMA classifiers had accuracy rates in the 50% range and we noted that there were no previous work for this problem that performed an ensemble learning between a RNN and an ARIMA classifier. Since the two models were considered weak learners we wondered on how we can combine the weak learners to create a more effective learner.

The alternative ensemble method would be to choose a Random Forest which would also require sampling additional data. According to Bauer and Kohavi, bagging and random forest performs relatively the same task and since we will just be dealing with at most seven features we concluded that the difference is negligible between the two methods in our case. **[7]**

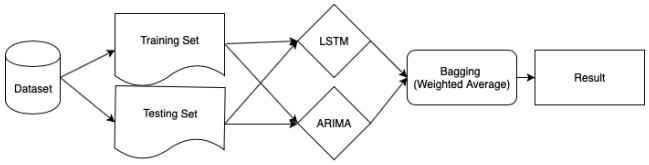
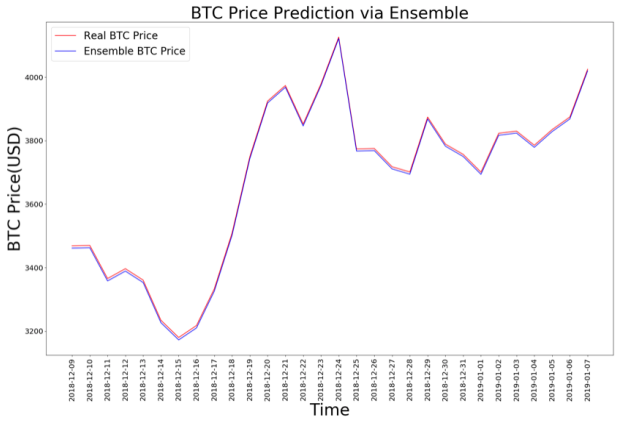


Figure 2: **Overview of the Ensemble Learning Pipeline**

8.1 Analysis of Ensemble

When performing the ensemble weighted average over our predicted values, we were able to obtain a smaller mean squared error of 5.75 which is lower than our other classifiers individually. Our variance, also reduced our error to the range of approximately 50 -> 5.

Overall, during our trials with bagging it was able to reduce our errors by half. This will be very helpful in correcting the variance obtained from our models over the course of retraining.



Graph 9: **Graph of Results from our Ensemble learning classifier**

9 Future Improvements

In this section, we propose several other ways to further improve our model or suggest a completely separate approach that may be of interest for some.

9.1 Blockchain Transactions

One of the initial approaches, we thought about improving our model was via the rate of transactions being performed on the blockchain. Our theory was that during a period where there are price bubbles, blockchaining data points would help deal with the volatility in the Bitcoin pricing.

9.2 Social Media / Sentiment Analysis

A more advanced method of improving our predictions was to take into account the quantity of bitcoin trading activity and use it to apply a bias to our model.

Additionally, we can also apply a sentiment analysis to social media activity and apply a bias over our predicted values. This assumes that more positive comments over bitcoin would have an effect over the increase in predictions.

10 Conclusion

In conclusion, we were able to produce accurate learning models using both LSTM and ARIMA and obtained very reasonable forecasting results. In addition, we were able to apply an ensemble bagging technique that improved upon

both the models and reduced our error and variance rates by half.

The results from this study shows that we can successfully predict with relative accuracy the average price of Bitcoin for approximately 30 days, given a robust dataset. Despite its relative accuracy, we do not recommend that this model be used as a guide to help Bitcoin investors. The main reasoning being, the dataset is small when compared to stocks on the NASDAQ and several data fields had NaN values which were grouped or forward-filled for this particular study. Also, as discussed before there is always sentimental data that usually causes huge spikes or downfalls in the market. Because our model ignores such data it would not be considered a fairly accurate Bitcoin forecaster.

ACKNOWLEDGMENTS

We would like to acknowledge the work done by Kaggle User: Zielak whose initial implementation of both RNN and ARIMA were adapted in our study and improved upon to obtain the results for this project.

REFERENCES

[1] https://www.coinbase.com/price/bitcoin
[2] https://faculty.washington.edu/ezivot/econ589/ch18-garch.pdf
[3] https://www.bioinf.jku.at/publications/older/2604.pdf
[4] Siami-Namini, Sima, et al. “A Comparison of ARIMA and LSTM in Forecasting Time Series.” 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 2018, doi:10.1109/icmla.2018.00227.
[5] http://ceur-ws.org/Vol-2280/paper-06.pdf
[6] Mangla, Neha. (2019). Bitcoin Price Prediction Using Machine Learning.
[7] E. Bauer and R. Kohavi, "An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants," Machine Learning, vol. 36, (1), pp. 105-139, 1999.