Predicting Bitcoin Price Series with Regression models and Neural Network models

Machine Learning, a Bayesian Perspective (EE4685)

Chuhan Wang 5289289
Delft University of Technology
c.wang-34@student.tudelft.nl

Jinchen Zeng 5268087
Delft University of Technology
j.zeng-4@student.tudelft.nl

April 6, 2021

Abstract

Bitcoin has become a hotspot in recent years while the price behavior stays unexplored for researchers. We researched the historical Bitcoin data set starting from the beginning of 2012 to the end of 2020. To predict the daily price series of Bitcoin, we employed different machine learning algorithms to achieve the prediction: the Linear Regression (LR) model as well as the Support Vector Regression (SVR) model. Meanwhile, we implemented two artificial neural network models: Multilayer Perceptron (MLP) and Long Short-Term Memory. We evaluated the results according to four different metrics related to regression: Root Mean Square Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE) and R^2 score (coefficient of determination). Besides, we discussed the results from two different prediction time points which showed a different performance in terms of different models. The LSTM model could fairly achieve a better performance in terms of two time periods' prediction results.

1 Introduction

Bitcoin is a worldwide cryptocurrency invented by Satoshi Nakamoto, a person or a group of people behind the mask. Bitcoin was first introduced to the world in 2009 and it has witnessed a fast growth both in transaction amount and fame since 2013. As a new entry in the market, its price behavior still remains unexplored, which leaves a huge space for further research [1]. Though bitcoin is officially admitted as a good rather than a currency and thus leads to a lot of controversies, its increasing value has had a great impact on the market, which makes the research on forecasting its future price quite interesting and exciting for both bitcoin traders and investors.

Our aim of the research was to predict the Bitcoin price, and which was the most appropriate way for prediction. We were interested in using machine learning methods to achieve the forecasting goal, and in our work, two types of models were implemented, which were linear regression models and neural network models, respectively. We implemented the Linear Regression (LR) model and Support Vector Regression (SVR). For neural network models, we implemented a simple Multilayer Perceptron (MLP) network at first, then we tried an improved algorithm: Long short-term memory (LSTM), which was more suitable for prediction based on time series. In this work, The data we used was a series of weighted prices from 2012 to 2021. The metrics we used for evaluation are RMSE, MAPE, MAE, and R2 score.

Apart from giving a comparison among these four models, we compared our work with some other recent studies. The first one we compare to was written by McNally et al. [3], who used RNN and LSTM models to predict bitcoin price. We also compare our results with the work of Fahmi et al., who tested many regression-based methods in [4], and the work of Karasu et al., who tested the performance of LR and SVR in [5]. In addition to that, Chen et al. [6] show how the data set would affect the performance, which is also discussed in the essay.

The structure of this essay is as follows: In section 2, an introduction of the methods is given, which consist of an introduction about the data set, an illustration about the data pre-processing, and basic ideas of the models we used. In section 3, we illustrated the metrics we use to evaluate the performance. In section 4, we showed the results and gave a discussion upon them. Section 5 is a conclusion of our work and suggestions about future work.

2 Methods

2.1 Collected Dataset

We tested all the algorithms on one data set, which is 'Bitcoin Historical Data' and available on Kaggle [8]. The CSV file contains a series of bitcoin trading from the end of 2011 (31/12/2011) to the end of 2020 (31/12/2020). The data columns consists of OHLC (Open, High, Low, Close), Volume in BTC and indicated

currency and weighted bitcoin price. Timestamps are in Unix time, while part of the data fields is filled with NaNs due to the period when there were no trading activities.

2.2 Data Pre-processing

Concerning both linear regression models and neural network models, we re-sampled the data set in the time unit of the day. Each sample presented one day. Since we only used the weighted price as the input feature of the model, the other columns were dropped to simplify the data set. Also, due to the non-activities in trading markets in some samples, the field of weighted price could be NaNs. To address the lack of sample points, we used the mean value of the weighted price of the day before and after the missing value to replace NaNs. Meanwhile, we used a MinMaxScaler() to re-scale all the samples from 0 to 1. The original Bitcoin price after pre-processing is shown in Fig.1 and the data set format is shown in Fig.2.

After ensuring the availability of all the data samples, to keep the temporal dependence between inputs (X) and outputs (Y), a sliding window was set to perform the forecasting. The window size was 100 days. To perform the prediction of Bitcoin price, we used a one-step-ahead forecast: the value in the next future time point was predicted by the input values from the past. We split the data set into training parts and testing parts. We chose 80% of the historical series price data set before 01/01/2020 as the training part with the remaining for testing.

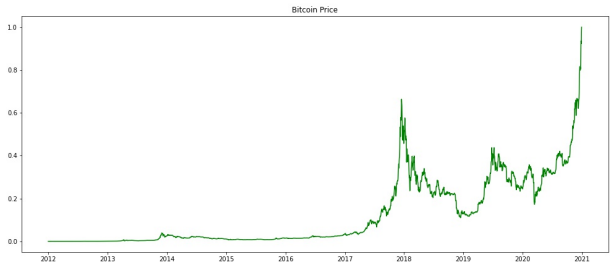

Figure 1: Bitcoin price after re-sampling

Timestamp	Weighted_Price
2011-12-31	4.471603
2012-01-01	4.806667
2012-01-02	5.000000
2012-01-03	5.252500
2012-01-04	5.208159

Figure 2: Dataset format

2.3 Prediction and Forecasting

We decided to implement two different methods: regression and common neural networks. Regarding regression models, we compared two statistical methods: LR and SVR. For neural networks, compared the performance of MLP Neural Network and LSTM Neural Network.

2.3.1 Linear Regression

Linear regression is an linear methodology for modelling the relationship between dependent variables and independent variables. In this section, we focus on the situation with only one independent variable (weighted price), and the dependent variable is the price to be predicted. Their relationship can be expressed as:

$$y = b_0 + b_1 * x \tag{1}$$

where y represents the predicted price, x represents the historical weighted prices, b_0 represents the intercept and b_1 represents the vector of slope coefficients. The aim of linear regression is to find a curve best fitting the data. To achieve the goal, a linear least square method named ordinary least squares (OLS) is involved. OLS chooses the most suitable parameter based on the principle that minimizing the sum of the squares of the differences between the dependent variable and the independent variable. The objective function for OLS is given as follows:

$$Min \sum_{i=1}^{n} (y - b_1 x_i)^2 \tag{2}$$

2.3.2 Support Vector Regression

In contrast to the objective that is mentioned above, the objective of Support Vector Regression (SVR) is to minimize the coefficients rather than the squared errors. In SVR, errors can be taken into account in the constraints, where the absolute error is required to less than or equal to a specified margin,ϵ, which is able to be tuned to gain the expected accuracy. The objective function of SVR and constraints are expressed as:

$$\begin{aligned} &Min \quad 1/2 * ||w||^2 \\ &Subject \quad to \quad |y_i - w_i x_i| \leq \epsilon \end{aligned} \tag{3}$$

SVR allows the flexibility to define how much error is acceptable in our model. Therefore, it can be regarded as an improvement of ordinary linear regression models.

2.3.3 MLP Neural Network

Multilayer perceptron (MLP) is a feed-forward network, where the information flows unidirectionally among three layers of neuron which are shown in 3. Connections among neurons have their weights and each neuron. Each layer has a nonlinear activation function except for the inputs.

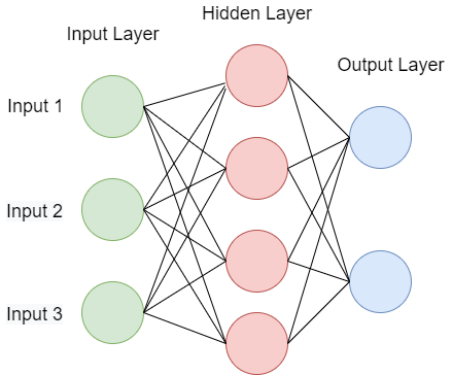

Figure 3: A MLP network

The performance of MLP depends on many things, such as the number of hidden layers, nodes, training data, etc. In our model, to avoid the low accuracy resulted from only one hidden layer, as well as an overfit because of applying too many layers, we finally decide on two hidden layers. The inputs are the historical weighted prices and the output is the price to be predicted.

2.3.4 LSTM Networks

LSTM is a significant variant of RNN. RNN works great on manifesting temporal dynamic behaviors, this is because that the special architecture of RNN allows it has a graceful performance on recognizing patterns in a series of data such as texts, handwriting, or real-time data generated from financial markets or other sources[2]. RNN has an architecture with loops, while the loops can be unrolled and the loop structure of the networks can be understood as multiple copies of the same network are organized in a row like a chain. Fig.4 (Colan, 2015) shows the architecture of RNN and how to see it from an unrolled perspective. In Fig.4 (Colan, 2015), x_t means the input of the network A and h_t represents the output.

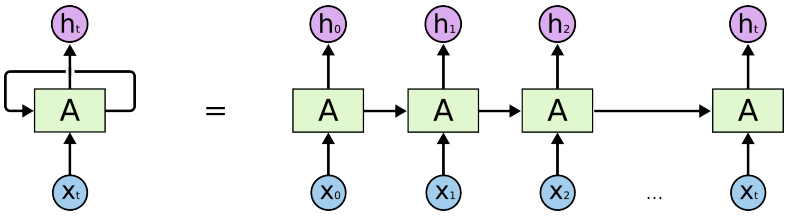

Figure 4: unrolling the RNN network [7]

The chain-like structure shows a strong relationship between the network and data in sequence or list, which leads to a question that whether RNN can learn useful knowledge from the previous information to boost the present work. The fact is that RNN can only handle this requirement when the gap between relevant past information and the present place is small enough. When the gap is bigger, it fails. The problem appealing here is called the "long-term dependency problem". Fig.5 (Colan, 2015) shows the case.

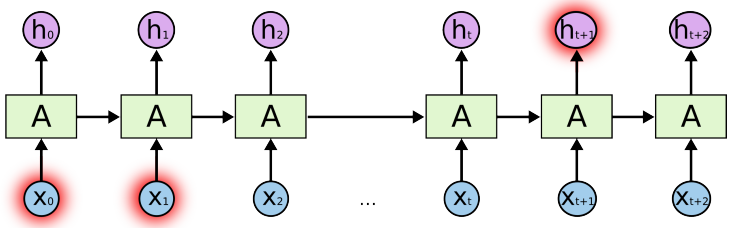

Figure 5: RNN fails to learn from past information when the gap is large [7]

LSTM is developed to solve this problem. LSTM was firstly introduced by Hochreiter and Schmidhuber in 1997 [2]. As a variant of RNN, LSTM can avoid long-term dependency problems successfully. Fig.6 (Colan, 2015) shows how LSTM works in one module(which will be repeated many times) and the notations are shown in Fig.7 (Colan, 2015).

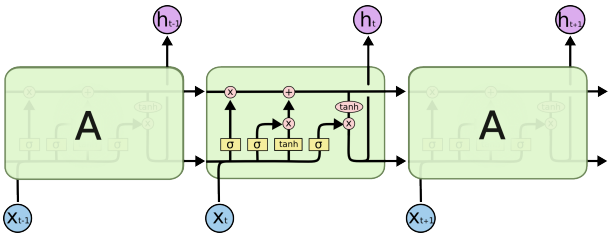

Figure 6: The repeating module in an LSTM [7]

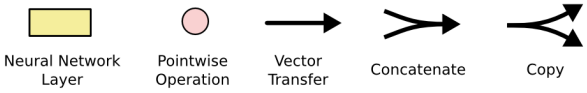

Figure 7: Notations [7]

The LSTM module consists of four main parts, which are a cell, an input gate, an output gate and a forget gate, respectively. The cell, which is the horizontal line running through the top of the above diagram, plays a significant role. From Fig.6 (Colan, 2015), we can know that it flows through the diagram, and its state is controlled by three gates. The process of how LSTM works is as follows: Firstly, decide on the information that is going to be "forget" from the cell state. The decision will be made by the forget gate layer, which is a sigmoid layer and its output will be either "1" or "0". An output "1" represents "Keep this message" and an output "0" represents "throw away this message". This layer makes the decision depending on the input and former states. The next step is deciding on what information is going to be stored in the cell state. To achieve this goal, firstly, the input gate layer decides which value should be updated. Then, a tanh layer creates a vector of new candidate values which will be added to the state. Finally, we combine the previous two steps and update the cell state. The reason why we apply the LSTM network to predicting bitcoin prices is that it is preferred classifying, processing, and predicting based on time series data, and what we have exactly is a series of real-time data.

3 Evaluation

To evaluate the performance and the effectiveness of different prediction methods, we chose the Root Mean Square, the Mean Absolute Percentage Error (MAPE), the Mean Absolute Error (MAE) and $R^2 score$ (coefficient of determination) expressed as:

$$RMSE = \sqrt{\sum_{i=1}^{N} \frac{(y_i - f_i)^2}{N}} \tag{4}$$

$$MAPE = \frac{1}{N} \sum_{i=1}^{N} |\frac{y_i - f_i}{y_i}| \tag{5}$$

$$MAE = \sum_{i=1}^{N} \frac{|y_i - f_i|}{N} \tag{6}$$

$$R^2 = 1 - \frac{SumSquaredRegressionError}{SumSquaredTotalError} = 1 - \frac{\sum(y_i - f_i)^2}{\sum(y_i - \overline{f_i})^2} \tag{7}$$

For both equations, y_i indicates the predicted value, while f_i indicates the actual value. N is the number of predicted time points also the length of the forecasting periods. These metrics are suitable for the evaluation of the regression models.

4 Result and Discussion

In our study, we selected the 100 data samples (100 days) equal to the size of sliding window size after 1/1/2020 for forecasting the price. Then we used the predicted prices compared with the actual prices to evaluate the performance and effectiveness.

4.1 MLP Neural Network

For MLP neural network model, we implemented the NN model with Python *Keras* library. Concerning "hyperparameter", we set the epoch size at 500 to avoid overfitting or underfitting issues. In the meantime, we used sets as the default value for batch size 32. Since MLP exploits the backpropagation computation for training the network, we kept our model as simple as possible which only contained two hidden layers. In

Keras, we used the Dense class to define the connected layer, while we specified the number of neurons and the activation function in each layer.

For the first two layers, we used ReLU as the activation function, and in the output layer, we used the Sigmoid function. The ReLu and Sigmoid activation functions are defined as:

$$ReLu = f(x) = max(0, x) \tag{8}$$

$$Sigmoid = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{9}$$

The detailed model was defined as:

- The model expected the input data with a series of 100 (sliding window) variables (the input_dim=X_train.shape[1, :])
- The first hidden and second layer had 16 and 8 nodes respectively with ReLU as the activation function.
- The output layer had 1 node with the Sigmoid as the activation function.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 100, 16)           32
_________________________________________________________________
dense_7 (Dense)              (None, 100, 8)            136
_________________________________________________________________
dense_8 (Dense)              (None, 100, 1)            9
=================================================================
Total params: 177
Trainable params: 177
Non-trainable params: 0
```

Figure 8: MLP model definition

For compiling the model, We specified the loss function to search for the weights for mapping the inputs to outputs during the training period, the optimizer was defined to search through network's various weights and any optional metric we expected to collect and report during training. In our model, we used mean squared error as the loss argument, defined in *Keras* as "mse". We defined the optimizer as the efficient stochastic gradient descent algorithm "adam" in *Keras*. The evaluation of loss function during the training session is shown in Fig.9.

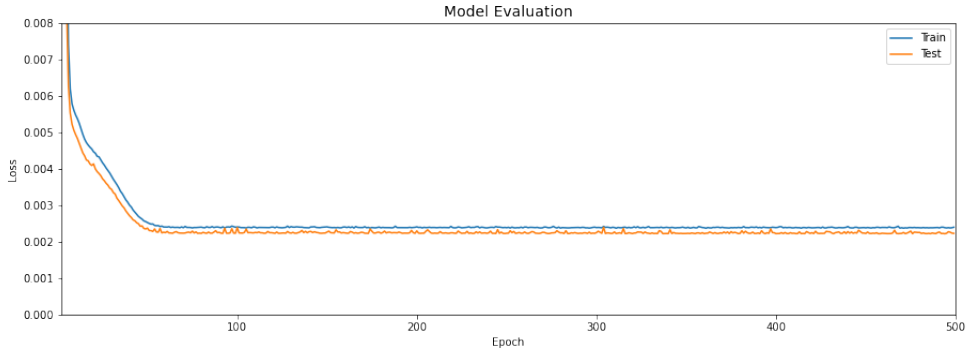

Figure 9: MLP model loss evaluation

The prediction result for 100 days from 01/01/2020 is shown in Fig.10. The evaluation metrics RMSE, MAPE, MAE, R2 score were 0.043, 0.143, 0.037 and 0.225 respectively.

4.2 LSTM Neural Network

For LSTM neural network model, we also used Python *Keras* library. Similarly, we chose the same "hyperparameter" values as we used in the MLP network model. Specifically, we set the epoch size to 500 and the batch size to the default value 32. We chose the model with 10 LSTM layers defined in *Keras*. Also, we complied the model with the loss function referred to MSE while the optimizer was "adam". The detailed model definition and the loss evaluation are shown in Fig.11 and Fig.12 respectively.

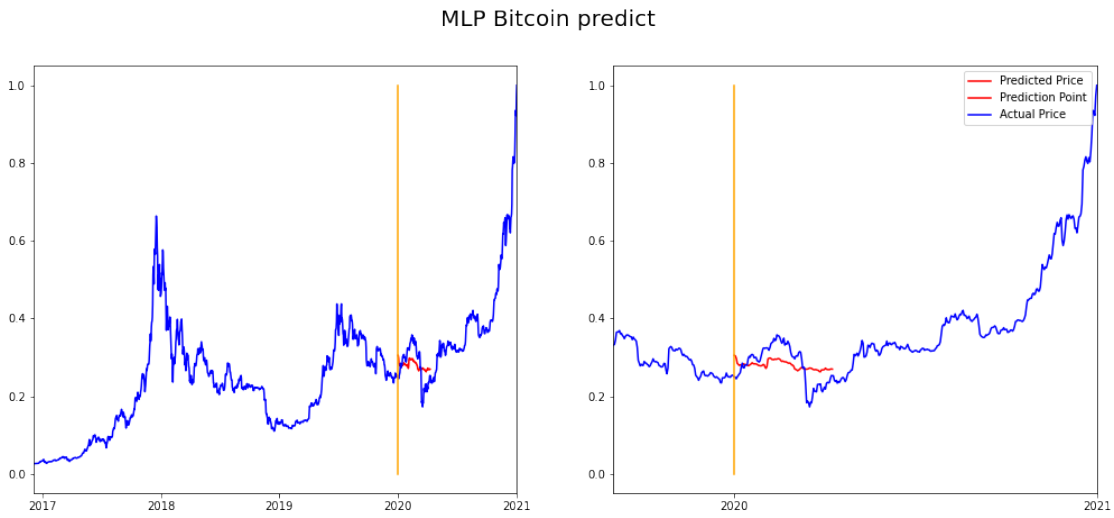

Figure 10: MLP model for Bitcoin forecasting

```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 100, 1)]          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 10)                480
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 11
=================================================================
Total params: 491
Trainable params: 491
Non-trainable params: 0
```

Figure 11: LSTM model definition,[7]

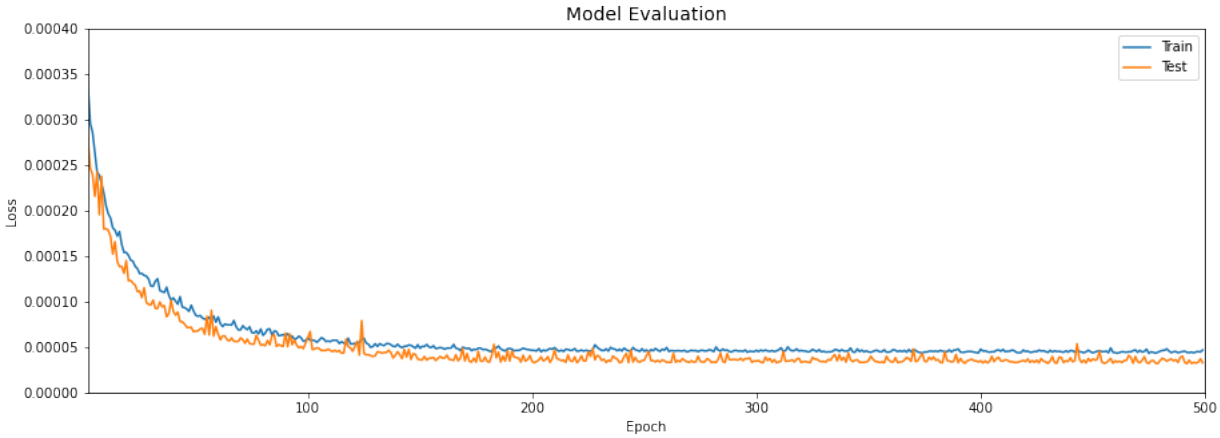

Figure 12: LSTM model loss evaluation

Similarly, we predicted 100 data points (100 days) after 01/01/2020, which is the same size as the sliding window. The predicted result is shown in Fig.13. The RMSE, MAPE, MAE, R2 score were 0.039, 0.116, 0.031 and 0.377 respectively.

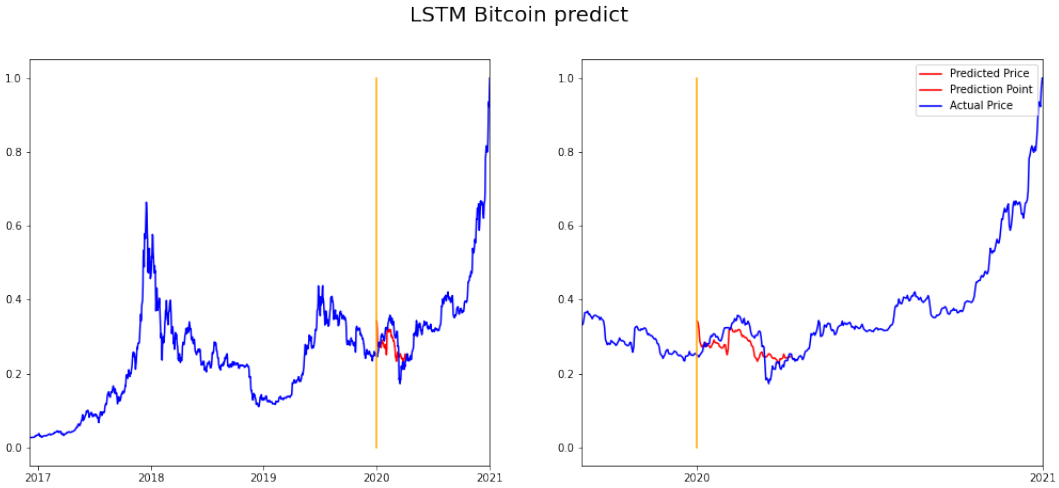

Figure 13: LSTM model for Bitcoin forecasting

4.3 LR and SVR

For LR and SVR models, we used Python *scikit-learn* library. We set the default value for the LR model and the SVR model. In SVR, we chose the default kernel value ('rbf') and the regularization parameter 'C' was set to be 10. The regularization parameter is used to reduce overfitting by applying a penalty to increasing the magnitude of parameter values. To find the most suitable 'C', we tried several values from 1 to 1000, finally, we narrowed it to around 10 which gives us the best performance in evaluation. Similar to MLP and LSTM, we predicted 100 data points (100 days) after 01/01/2020, which was the same size as the sliding window. The predicted results are shown in Fig.14 and Fig.15.

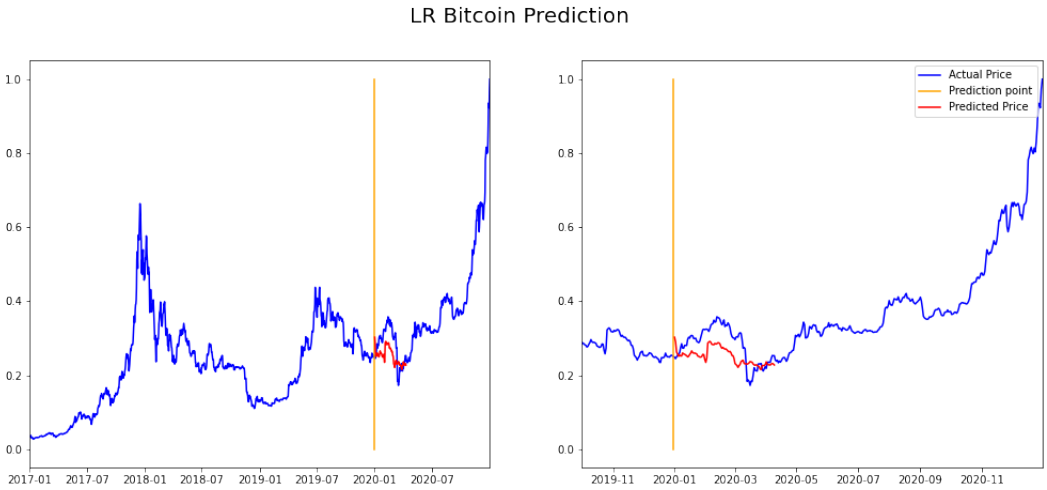

Figure 14: LR model for Bitcoin forecasting

The evaluation results are as follows. For the LR model, RMSE is 0.048, while for the SVR model RMSE is 0.036, which is considerably smaller. SVR has better performance concerning RMSE, MAPE, MAE, and R2 score. From the evaluation results, we can see that SVR works better than LR in predicting bitcoin prices.

The overall prediction evaluation result is shown in Table.1.

4.4 Discussion

As shown in Table.1, the SVR model outperformed all other three models with the lowest RMSE (0.036), MAPE (0.102) and MAE (0.028) as well as the highest R2 score (0.441). For linear regression approaches, the results obtained from SVR are much better than simple linear regression, since SVR minimized the coefficients

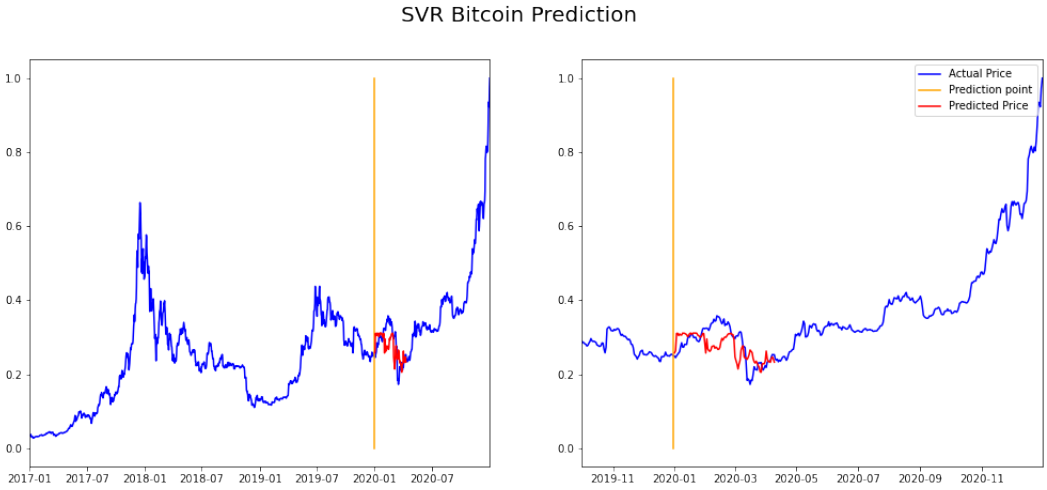

Figure 15: SVR model loss evaluation

Model	RMSE	MAPE	MAE	R2 score
LR	0.048	0.147	0.043	-0.007
SVR	0.036	0.102	0.028	0.441
MLP	0.043	0.143	0.037	0.225
LSTM	0.039	0.116	0.031	0.377

Table 1: Evaluation results of four models

rather than the squared errors while the errors needed to be constrained. For the neural network approach, the LSTM network outperformed the MLP network, this is because of the feature of LSTM which allows it to capture the long-term dependencies in a sequence. Because the MLP has the limitations constrained by the vanishing gradient problem and the ineffective recognition towards temporal elements of a time series task.

Since we only discussed the prediction result of 100 samples starting from 01/01/2020, however, there was an enormous increase occurred around 09/2020, to see the prediction performance for the bullish of the Bitcoin market, we also measured the results that contained 100 samples from 01/09/2020 with the same trained model as measured before. The measurements are shown in Table.2.

Regarding the prediction for the dramatic increase of the Bitcoin price, generally, the linear regression approach performed worse than the neural network approach which is the opposite of the previous results. The SVR model which had the best performance didn't fit with the dramatically increasing trend of Bitcoin, however, the LTSM model could learn the trend from the history data. Overall, all the models we chose in the previous section had problems with predicting anomalous increasing of the Bitcoin market, however, LSTM could be considered as an effective learner in both scenarios.

Regards with computation time or training time, two linear regression models hardly cost extra time to train the model. As for the neural network models, the training time also differs. The Multilayer Perceptron costs much less training time compared with LSTM due to the simple structure of MLP.

Model	RMSE	MAPE	MAE	R2 score
LR	0.251	0.430	0.218	-4.272
SVR	0.246	0.251	0.200	-4.061
MLP	0.186	0.317	0.161	-1.933
LSTM	0.143	0.237	0.121	-0.699

Table 2: Evaluation results for the bullish of the Bitcoin Market

We also compare our work with other papers. In [3], they used both the RNN model and LSTM model to predict the bitcoin price and their results show that LSTM works better than initial RNN(since LSTM is a variant of RNN), which proves that the mechanism of LSTM makes it a better performance dealing with time-series data. when the number of epochs is set to be 500, as same as what we set, the RMSE of their result is higher than ours. Although the difference between data sets might be responsible for that, it partly shows that our LSTM model works. We also have a look at the work of Karasu et al.[5], the results in this paper show that SVR has higher performance than LR, which also match our conclusions from our results. In [6], Chen et al. tested a lot of machine learning models, including LR, SVM, LSTM, LDA, QDA, RF, XGB, on two different data sets. One of the data sets is the bitcoin daily price, while another one is the bitcoin

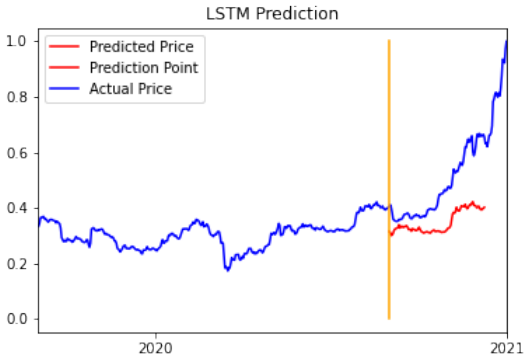

Figure 16: LSTM model prediction

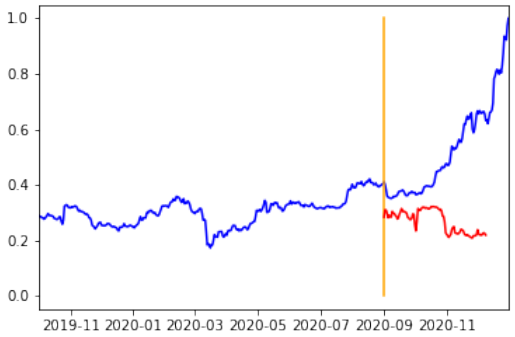

Figure 17: SVR model prediction

5-minute interval price. Their results show that, when using the daily price data, linear models have better performance than neural network models for low-frequency data with high-dimensional features, which is the bitcoin daily price data in their work, while neural network models outperform linear models for bitcoin 5-minute interval price data, which is a more high-frequency data. Since the data set, we use in our work is low-frequency, it turns out our conclusion that linear models (SVR) performs better than neural network models (LSTM) is reasonable. Besides, it also reminds us that sample's granularity and feature dimensions should be considered, which was ignored by us due to the limitation of the data set.

5 Conclusions

For predicting the Bitcoin price, we employed two approaches: linear regression and neural network. Each approach included two models. We compared two different scenarios regarding the prediction time point. With two different prediction time points, the best-performed model changed from SVR to LSTM. In general, the LSTM neural network could be considered as an effective method towards predicting the Bitcoin price, since the capability of LSTM to recognize the longer-term dependencies. However, there are still limitations. One limitation is that the highly-consumed training time due to the complexity of the training model. One another limitation is that the prediction of the dramatic increase could cause insignificant performance. Except for those, the balance between overfitting and underfitting could be a tricky issue which is common for most neural network models.

6 Limitations and Future Work

Our research also has some limitations. One limitation is that we broke down the data set into low-frequency daily samples which caused the lower number of training samples, which might be the reason why the linear regression model (SVR) outperformed neural network models in one scenario. Defining different frequencies of the data set could achieve better performance with our neural network models. Also, lots of recent studies took the Bitcoin price prediction problem from the perspective of classification, which also should be included in our future work. Aside from those, it was indicated in the recent findings that the performance could benefit a lot from implementing the parallelization of algorithms on a GPU, which showed that there could be 70.7% performance improvement regarding the LSTM model compared with the same algorithms on CPU. The multivariate forecasting was implemented in Uras N's study [1] by using multiple features of the data set including close and volume features. It was proved that multivariate forecasting could achieve a better result with the cost of considerable training time, which also should be included in our future work.

7 Related Materials

Code: https://github.com/Mmmelvil/EE4685-Machine-Learning-A-Bayesian-Perspective-Assignment-3

References

[1] Uras N, Marchesi L, Marchesi M, et al. Forecasting Bitcoin closing price series using linear regression and neural networks models[J]. PeerJ Computer Science, 2020, 6: e279.

[2] Hochreiter S, Schmidhuber J. LSTM can solve hard long time lag problems[J]. Advances in neural information processing systems, 1997: 473-479.

[3] McNally S, Roche J, Caton S. Predicting the price of bitcoin using machine learning[C]//2018 26th euromicro international conference on parallel, distributed and network-based processing (PDP). IEEE, 2018: 339-343.

[4] Fahmi A, Samsudin N, Mustapha A, et al. Regression-based Analysis for Bitcoin Price Prediction[J]. International Journal of Engineering Technology, 2018, 7(4.38): 1070-1073.

[5] Karasu S, Altan A, Saraç Z, et al. Prediction of Bitcoin prices with machine learning methods using time series data[C]//2018 26th signal processing and communications applications conference (SIU). IEEE, 2018: 1-4.

[6] Chen Z, Li C, Sun W. Bitcoin price prediction using machine learning: An approach to sample dimension engineering[J]. Journal of Computational and Applied Mathematics, 2020, 365: 112395.

[7] Colah. Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[8] Zielak. Bitcoin Historical Data. https://www.kaggle.com/mczielinski/bitcoin-historical-data