Cryptocurrency Market

machine learning for predicting price and trading strategy

OPIM 5671 Final Project

Jae Seong Cho

Ting Shen

Shan Xu

Junyuan Huang

1. Introduction

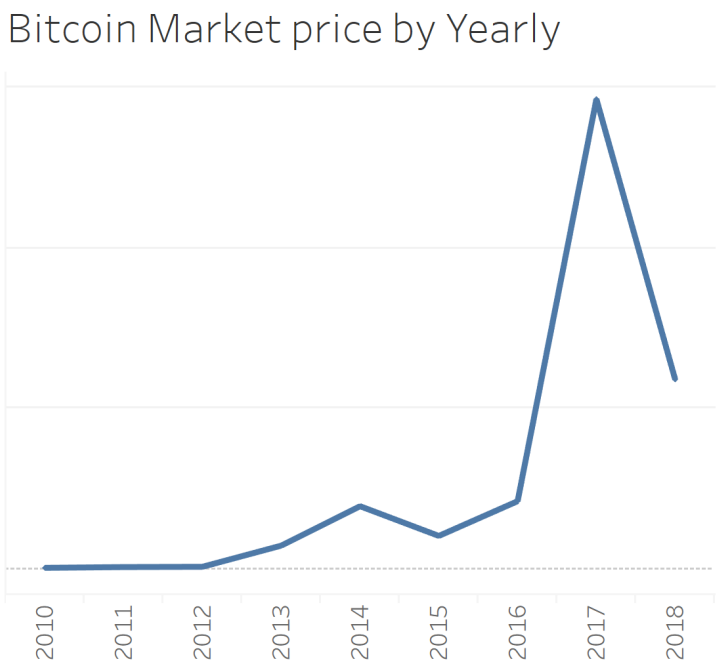
Cryptocurrency has been hottest topic around the world recently. The idea of blockchain that

enables people to transfer virtual money without the regulation of third party such as bank or government, mesmerized people. Also the reason of cryptocurrency being hot topic is because there are people who became millionaire by buying and selling cryptocurrency.

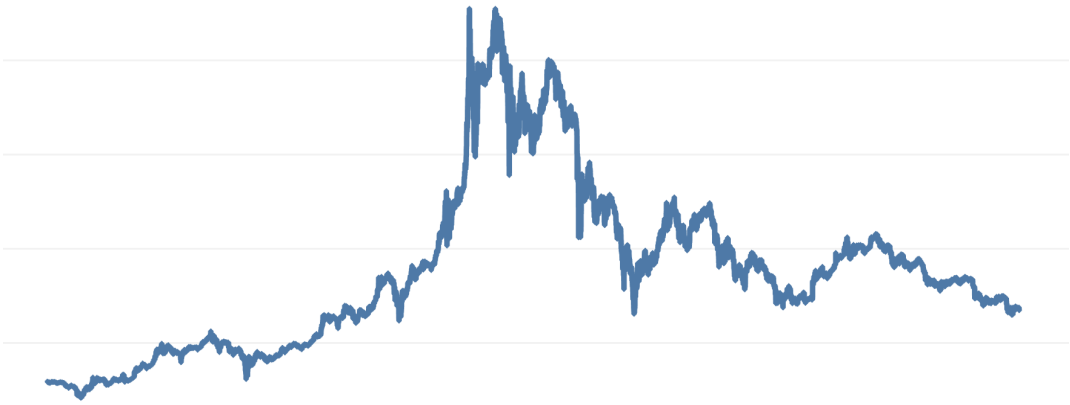
Because of cryptocurrency being hot topic and great potential it has in the future, we analyzed mainly bitcoin and also used ethereuem dataset for pair trading, which are one of the biggest cryptocurrencies in the market. We will use ARIMA model and Neural networks to forecast bitcoin prices and analyze the dataset to present strategy to make profit in cryptocurrency.

2. Data exploration

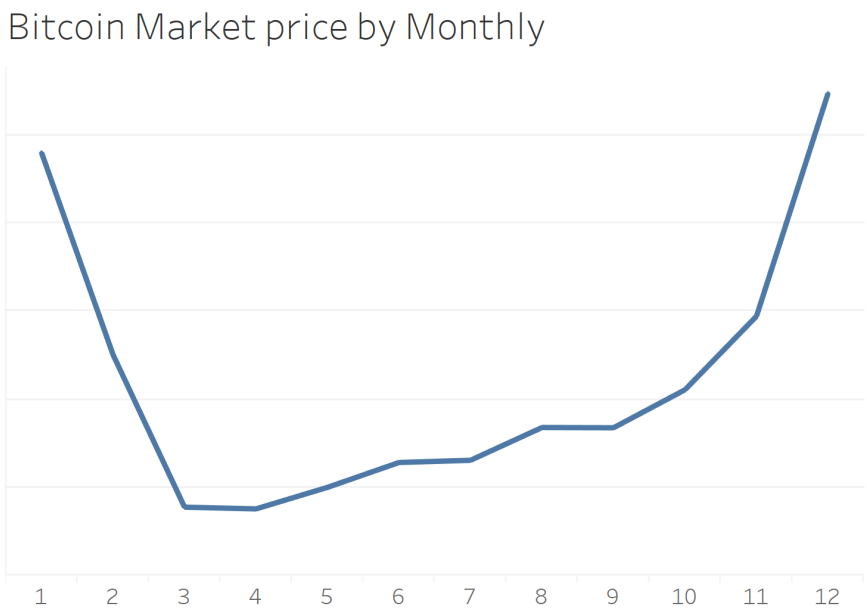
We have compared two datasets which are daily prices for bitcoin and 1 minute prices for bitcoin. Below is the graph for daily bitcoin prices.



We can see that there are sudden surge and decline for the bitcoin prices. We found that sudden surge happened mostly because of bandwagon effect. However, after governments put regulations or ban bitcoin, the prices of bitcoin dropped. After the regulation, bitcoin prices seems stationary compared to the sudden surge in 2017. Therefore, we chose another data set which has recent bitcoin prices with higher frequency. Below is the graph of the data set.



The bitcoin dataset does not seem to have trend. To verify if there are any seasonality, we created graph of monthly bitcoin prices.



There is actually seasonality in Bitcoin. As you can see from the graph above, at the start of the year, the bitcoin prices tend to drop. This is called January effect. Therefore, there is seasonality in the function.

The data set variables are presented in below table. The variables include:

○Open- Bitcoin price in Currency units at time period open

○High- Highest Bitcoin price in Currency units during time period

○Low- Lowest Bitcoin price in Currency units during time period

○Close- Bitcoin price in Currency units at time period close

■*"Closing price" generally refers to the last price at which a stock trades during a regular trading session.*

○Volume (BTC) - Volume of BTC transacted in time period

○Volume (Currency)- Volume of Currency transacted in time period

○Volume-weighted average price (VWAP)

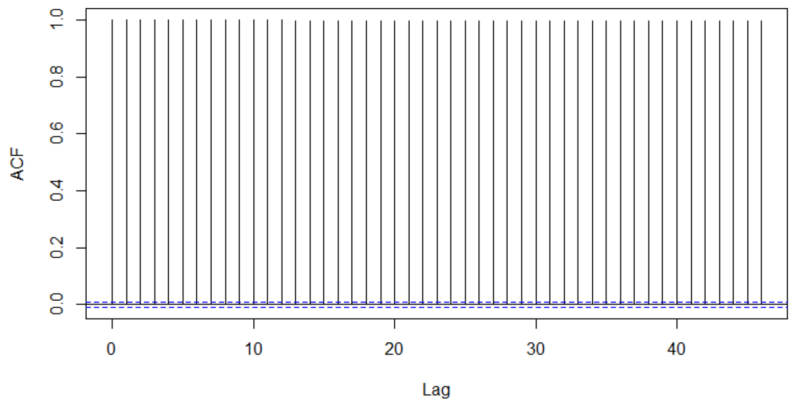
Close	High	Low	Open	Volume_(BTC)	Volume_(Currency)	Weighted_Price
296,540	296,558	296,016	296,127	1.159	343,244.14	296,257.67
296,260	296,260	296,011	296,215	2.735	810,100.81	296,221.52
296,621	296,859	296,621	296,636	8.186	2,429,080.19	296,733.40
296,709	296,880	296,666	296,666	2.082	617,838.34	296,785.11
295,911	296,154	295,842	295,842	2.613	773,264.43	295,929.75
295,996	296,080	295,777	296,080	3.488	1,032,515.89	295,993.71
294,387	294,525	294,250	294,250	11.575	3,406,793.63	294,334.03
294,400	294,800	294,237	294,778	18.201	5,359,863.26	294,481.80
295,159	295,200	295,159	295,200	0.830	245,014.77	295,198.52
295,335	295,345	294,911	295,112	0.144	42,459.29	295,022.12
295,800	295,800	295,494	295,600	8.548	2,527,633.91	295,703.03
295,395	295,550	295,108	295,200	1.090	321,926.25	295,252.53
294,179	294,261	293,978	294,160	6.236	1,833,737.63	294,035.21
293,639	293,811	293,362	293,811	3.531	1,036,307.27	293,455.07
292,362	292,554	292,362	292,400	4.951	1,447,849.51	292,427.96
290,600	290,600	290,040	290,442	27.820	8,078,513.66	290,381.86
292,186	292,431	291,748	292,183	11.641	3,399,319.80	292,018.18

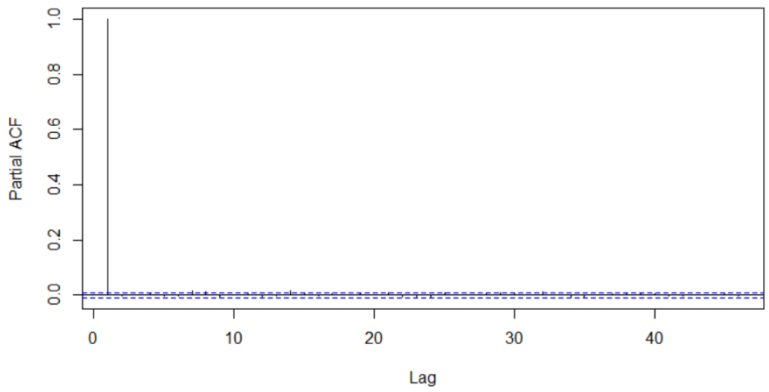
We only use Closing price for the bitcoin in this analyzation.

3. ARIMA model

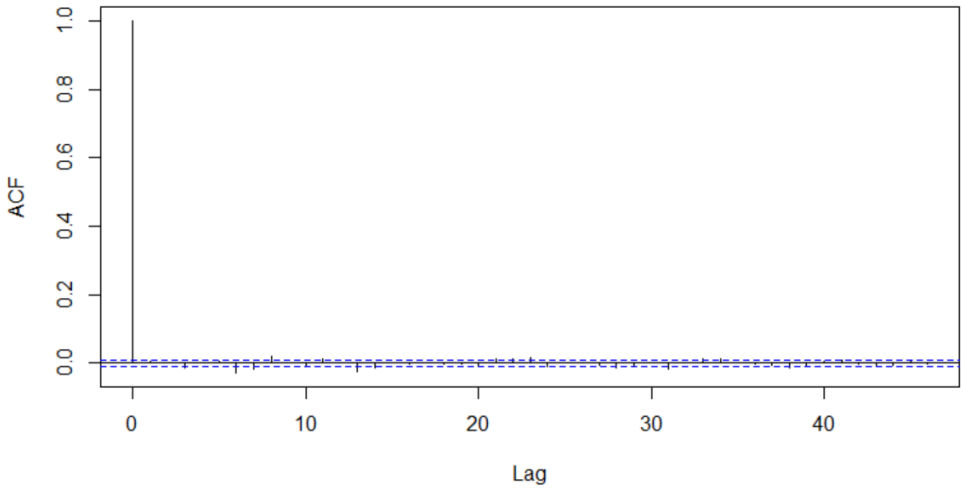
We used R programming in analyzing ARIMA model. First, we imported excel file of resampled data set which is 10 minutes interval for bitcoin prices. Among the variables, closing price is used for evaluating ARIMA model. The first step we took was to log the closing prices. Bitcoin prices are basically based on returns and returns are based on percentage. Therefore, we log on the values.

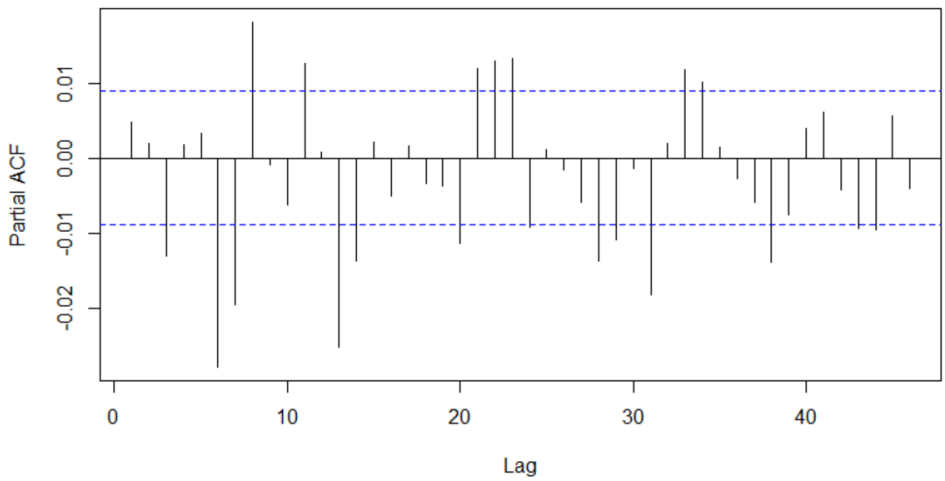
To evaluate if the data is stationary or non-stationary, we opened autocorrelation function (ACF) graph and partial autocorrelation function (PACF) graph.





As you can see from the graphs, the data set is obviously non-stationary. To make it stationary, we applied first difference. After applying first difference the graphs look like these.





To be certain about data being stationary, we used augmented dickey-fuller test to test unit root test for stationary. Below is the result of augmented dickey-fuller test for the log values of closing price

```
        Augmented Dickey-Fuller Test

data:  ln_bit_price
Dickey-Fuller = -1.0975, Lag order = 36, p-value = 0.9238
alternative hypothesis: stationary
```

and same values with first difference.

```
        Augmented Dickey-Fuller Test

data:  dif_bit_price
Dickey-Fuller = -37.331, Lag order = 36, p-value = 0.01
alternative hypothesis: stationary
```

The augmented dickey-fuller test for original values has p-value of 0.9238 which is greater than alpha(0.05). We fail to reject the null hypothesis which is non-stationary; therefore, the data set is non-stationary. The augmented dickey-fuller test for values with first difference, on the other hand, has p-value of 0.01 which is lower than alpha. Then we can reject the null hypothesis; therefore, the data is stationary.

Now we know that the first difference is necessary, we know that our ARIMA model will have d=1. Instead of plugging in random values for ARIMA model, R presents a function called auto.arima(). This function finds the best ARIMA model automatically measuring low AIC values.

In order to use auto.arima() code in R, we first transformed our log values of data set into time series data set. Since the frequency of the data set is 10 minutes interval, set frequency = 24(hours in a day)*60(minutes in an hour)/10. We used auto.arima code with time series data that we made with out the difference since auto.arima code will automatically insert difference if needed, and enforce seasonality in the code by inserting D=1. The result is showed below.

```
> auto.arima(ts_bitcoin, D=1)
Series: ts_bitcoin
ARIMA(0,1,1)(0,1,0)[144]

Coefficients:
          ma1
      0.0065
s.e.  0.0046

sigma^2 estimated as 5.258e-05:  log likelihood=168410.7
AIC=-336817.5   AICc=-336817.5   BIC=-336799.9
```

The ARIMA model is ARIMA(0,1,1)(0,1,0). Among all other ARIMA models, this model should have the lowest AIC value. The summary of the ARIMA model shows below.

```
> summary(Bitcoin.ARIMA)

Call:
arima(x = ts_bitcoin, order = c(0, 1, 1), seasonal = c(0, 1, 0))

Coefficients:
          ma1
      0.0065
s.e.  0.0046

sigma^2 estimated as 5.258e-05:  log likelihood = 168410.7,  aic = -336817.5

Training set error measures:
                       ME        RMSE         MAE           MPE      MAPE     MASE         ACF1
Training set -1.448847e-06 0.007240218 0.004513831 -1.097279e-05 0.0331479 1.500811 0.0004107149
```

The RMSE is 0.00724 which is low value for RMSE.

After having ARIMA model, we used forecast code in R to forecast closing bitcoin prices. After running the forecast code, we get forecasted values of however much the value we put in the code. Below is the graph of forecasted values along with the actual values.

We, however, noticed that the values are in log form:

```
         Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
335.4306       13.63984 13.63054 13.64913 13.62562 13.65405
335.4375       13.64078 13.62760 13.65397 13.62062 13.66095
335.4444       13.63792 13.62175 13.65408 13.61320 13.66264
335.4514       13.63769 13.61902 13.65637 13.60913 13.66625
335.4583       13.63903 13.61815 13.65992 13.60709 13.67098
335.4653       13.63742 13.61453 13.66030 13.60242 13.67241
335.4722       13.63515 13.61042 13.65987 13.59734 13.67296
335.4792       13.63829 13.61186 13.66472 13.59786 13.67871
335.4861       13.63912 13.61108 13.66715 13.59624 13.68200
335.4931       13.63965 13.61010 13.66921 13.59445 13.68486
335.5000       13.64161 13.61060 13.67261 13.59419 13.68902
335.5069       13.64146 13.60908 13.67384 13.59194 13.69098
```

Therefore, we used exponential function to change into actual values.

	Actual price	Forecasted Price
1	837768	838890.4
2	837500	839684.8
3	837076	837283.2
4	837702	837094.8
5	837500	838216.8
6	837999	836862.5
7	837128	834966.5
8	837122	837594.4
9	837800	838287.5
10	839265	838738.9
11	838932	840376.9
12	839250	840254.0
13	839500	841549.0
14	838501	840933.8
15	839200	840039.0
16	839892	840541.7
17	840500	838933.4
18	840522	838720.5
19	840850	838564.9
20	842552	836610.6
21	840267	836485.7

We made a data frame that consist actual price and forecasted price. Since we have both forecasted values and actual values, we used formula of getting percent error which is difference of actual value and forecasted value over actual value, to calculate percent error for our data sets. Then we got the mean of the percent error. The mean of percent error is 5.242076e-05. It is certainly low value and shows that the forecasted value is very similar to actual values.

4. RNN LSTM Model

There are a bunch of ways to model and predict Bitcoin price.We did some research and find out that RNN model works better with sequential data. But the simplest RNN model has a major drawback, called vanishing gradient problem, which means the network experiences difficulty in memorising words from far away in the sequence and makes predictions based on only the most recent ones and prevents it from being accurate. So we introduce the LSTM model to deal with this problem. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior. We implement the algorithm using Python with the help of several powerful machine learning libraries (e.g. sklearn, keres, pandas).

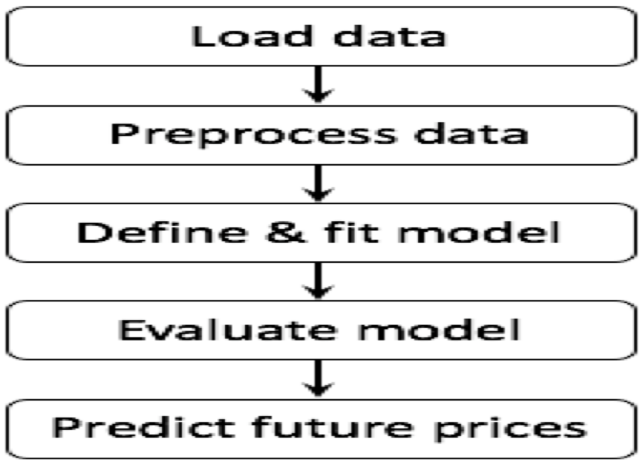


Figure 1. Workflow of LSTM approach

As shown in the figure 1, our implementation includes several steps. We first load the raw data into the memory and preprocess the data into expected format. Then, we define and train our model with the training data, and test with the remaining testing data. Later, we predict the price for future time. Next, I will introduce each step in details.

1) Load data

The dataset I used is from kaggle website. When searching for bitcoin historical data, there are a list of options and to cut down the execution time of the experiments, I selected the smallest one of 512k x 8 format: bitflyerJPY_1-min_data_2017-07-04_to_2018-06-27.csv. It includes eight columns as follows:

columns
timestamp
open
high
low
close
volume_(BTC)
volume_(currency)
weighted price

Table 1. Column information of Bitcoin raw data

The timestamp is in unix time and the interval is 1 minute. We focus on Close, which is the Bitcoin price in Currency units at time period close.

2) Preprocess data

Preprocessing step includes several transformation operations. To further cut down the execution time, we use an interval of 10 minutes rather than that of 1 minute. To do so, we sample the raw data by fetching every 10^{th} of the rows. In addition, as we mentioned, we are more interested in "Close" price, so that we extract this column only. Also, to convert the input to time-series data, we set "timestamp" column as the index of the 2D Pandas Dataframe in Python. We observe that, the range of close prices are quite large. To make input data more friendly to the model, we resort to a MinMaxScaler to normalize the prices between (-1, 1). To use the data as input for the model, we also reshape that to the required format. (e.g. 2D data to 3D data).

As usual, we split the transformed data into two parts: training data and testing data. In our experiment, we select 80% of the data as training data while leaving the remaining 20% of the data as testing data.

3) Define and fit the model

Now, we are ready to train our model using the above processed data. Instead of make wheels by ourselves, we use keras to build the model. Keras is a neural network library that provides a large sets of high-level APIs. It is written in Python and capable of running on top of many popular framework, such as Tensorflow and CNTK. With keras, this process is quite easy. In short, we only care about the high level logics and important parameters of the model, rather than details behind the operations.

We select LSTM networks in our experiment. Long short term memory networks (LSTM) are a special kind of recurrent neural network (RNN), capable of learning long-term dependencies. All RNNs have the form of a chain of repeating modules of neural network. The standard RNNs have a simple structure such as a single tahh layer. LSTMs also have the chain structure, but the repeating module has a different structure. Instead of a single neural network layer, they have four and interacting in a special way. To save space, we will not go into details.

In our model, we use 4 units as the dimension of the output state for LSTM cell. We use “sigmoid” activation function for LSTM cell. Also, we select “adam” as the optimizer that is used to minimize the loss function. To make a balance between accuracy and cost (execution time), we define 10 epochs, which means the model will iterate 10 times.

After highlight some important parameters, we define a Sequential model that stacks input, hidden and output layers. We then add the LSTM cells to our model, and compile the model with optimizer and loss function discussed above.

```
Train on 34787 samples, validate on 6139 samples
Epoch 1/10
 - 77s - loss: 0.0113 - val_loss: 1.8967e-05
Epoch 2/10
 - 76s - loss: 5.3860e-05 - val_loss: 1.4431e-05
Epoch 3/10
 - 76s - loss: 5.3834e-05 - val_loss: 1.4258e-05
Epoch 4/10
 - 76s - loss: 5.3736e-05 - val_loss: 1.3904e-05
Epoch 5/10
 - 76s - loss: 5.3368e-05 - val_loss: 1.3489e-05
Epoch 6/10
 - 76s - loss: 5.3542e-05 - val_loss: 1.6597e-05
Epoch 7/10
 - 76s - loss: 5.3521e-05 - val_loss: 1.4016e-05
Epoch 8/10
 - 76s - loss: 5.3507e-05 - val_loss: 2.1722e-05
Epoch 9/10
 - 75s - loss: 5.3415e-05 - val_loss: 1.4737e-05
Epoch 10/10
 - 83s - loss: 5.3048e-05 - val_loss: 1.7105e-05
train MSE =  4.3e-05
test MSE =  0.000117
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (1, 4)                    96
_________________________________________________________________
dense_1 (Dense)              (1, 1)                    5
=================================================================
Total params: 101
Trainable params: 101
Non-trainable params: 0
```

Figure 2. Execution log

4) Evaluate the model

We train the model with the training data, and then test the model with testing data. We calculate the MSE of the model against both training data and testing data. The above screen snapshot of the execution log shows the model summary.

5) Predict future prices

To predict future prices, we use two approaches and compare their behaviors. The first approach is moving window algorithm, which predict one step ahead, adjust the input to include this

step result, and predict the next step. In addition, the model provides a way to predict future steps in one batch by indicating “steps” parameter.

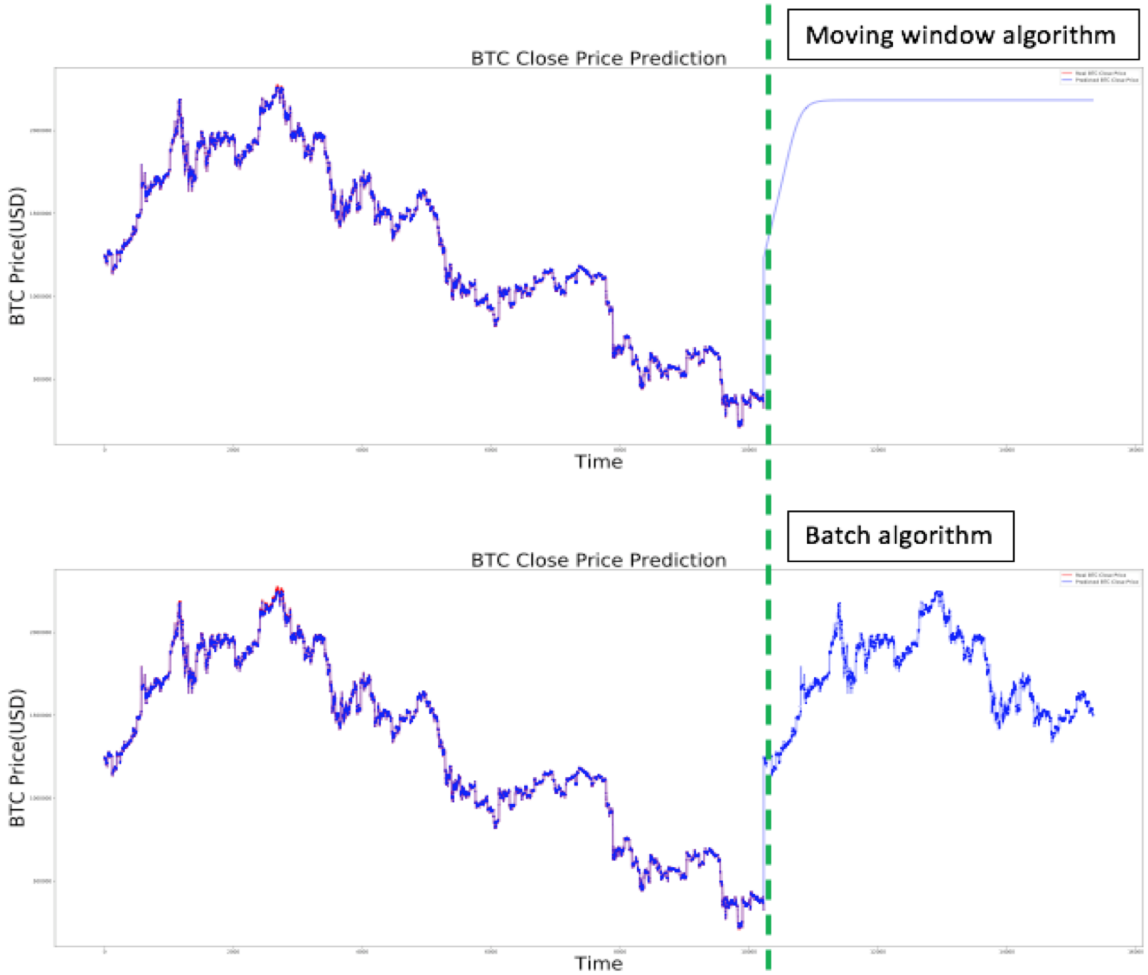


Figure 3. Evaluation of model prediction

Figure 3 shows the testing and prediction results. X-axis shows the time with an interval of 10 minutes. Y-axis shows the prices in real range. Red line and blue line shows the real prices and predicted prices, respectively. And we use a green dotted line to separate testing and predicting parts. Since these two approaches only affect the prediction part, their testing part (before the green dotted line) are the same. As shown in the figure, real Bitcoin close prices matches well with predicted prices, and our model achieves a RMSE of 0.000117 for testing data. For prediction parts, their behaviors are different. The batch approach seems to repeat the patterns in previous stages, while the moving window algorithm seems quite flat, which indicates the later prices are highly rely on previous prices.

5. Trading strategy:

A success model of trading strategy is based on a good predicting price model. We haven't build an trading strategy on the RNN, LSTM model or ARIMA model yet. However, we explore and develop several trading strategies that potentially can work out.

Logistic Regression and SVM strategy :

We explored the every past 7 days price of bitcoin as our features to train, and our prediction is whether tomorrow's price is up or down. It was a classification problem. The result is the training set performs too good to be true, and the testing set performs ugly bad.

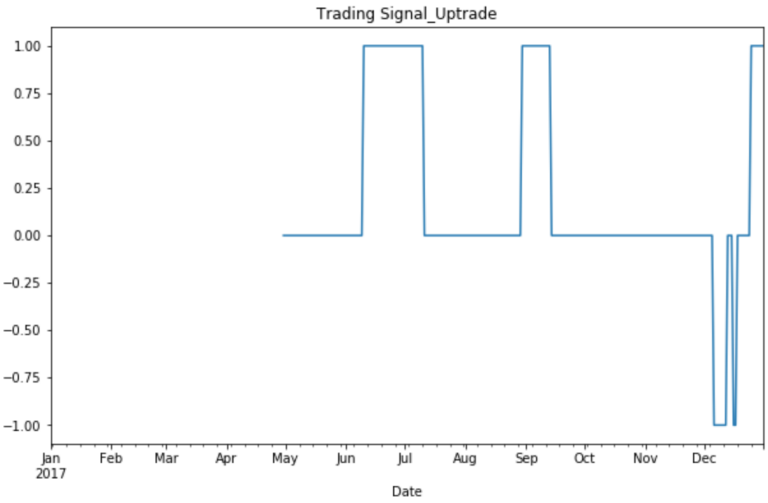
Pair Trading:

Pairs trading is a market-neutral trading strategy that matches a long position with a short position in a pair of highly correlated instruments such as two stocks, exchange-traded funds (ETFs), currencies, commodities or options. Pairs traders wait for weakness in the correlation and then go long the under-performer while simultaneously short selling the over-performer, closing the positions as the relationship returns to statistical norms.

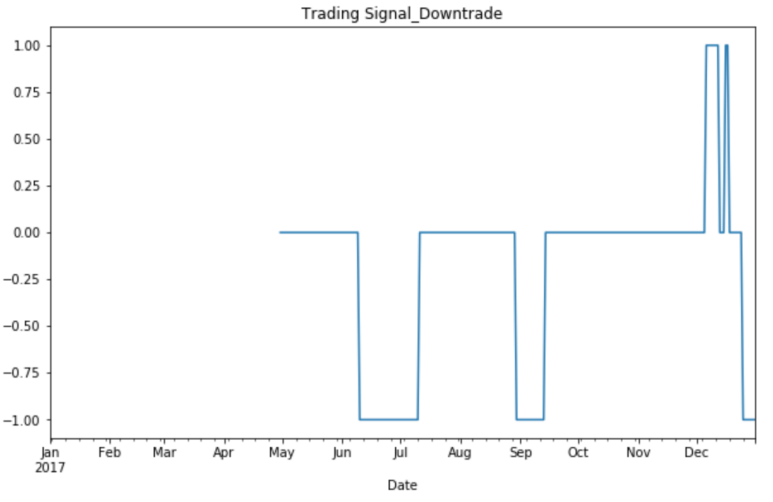
In our cases, we set the long or short position range is +1.5 sigma or - 1.5 sigma, +0.5 sigma or - 0.5 sigma as our closing position line.

Following part of the code we submitted should show when trigger the trading signal clearly.

```
In [128]:
data['position_1'] = np.where(data['zscore'] > 1.5, 1, np.nan)
data['position_1'] = np.where(data['zscore'] < -1.5, -1, data['position_1'])
data['position_1'] = np.where(abs(data['zscore']) < 0.5, 0, data['position_1'])

In [129]:
data['position_1'] = data['position_1'].fillna(method = 'ffill')

In [130]:
data['position_1'].plot(ylim=[-1.1, 1.1], figsize=(10, 6),title = 'Trading Signal_Uptrade')
Out[130]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1eaeee80>
```




```
In [131]:
data['position_2'] = -np.sign(data['position_1'])

In [132]:
data['position_2'].plot(ylim=[-1.1, 1.1], figsize=(10, 6),title = 'Trading Signal_Downtrade')
Out[132]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1ee46da0>
```



6, Summary and future further Improvement:

For now, the pair trading is our safe bet. In future, we can try to develop a trading strategy model based on our predicting price model RNN- LSTM and ARIMA. The model can be further improved by the cross validation to find better hyper parameter. The two strategy model (SVM and logistic Regression) fails to be profitable means we have a lot to do and extract better features.

This new cryptocurrency market, unlike stock mature market, is hard to find an effective model to value the intrinsic value. Therefore, the technical market analysis is going to be the dominant method for a while. We can explore tons of different market technical indicators, such as CCI, Bollinger Band, and Rate of change and so on to predict price or develop trading strategy. We also can use google trend to quantify trading, trying to find if the search amount of volume of bitcoin related to the bitcoin price.

On the other hand, the un-defined intrinsic value of bitcoin might eventurealy correlated to how many people is going to adopt it as daily uses. We can also explore the amount of bitcoin active wallet users to see if it can be sign of the bitcoin value.

Furthermore, in such a volatile market, only talk about how return it is but ignore how big risk facing is meaningless. So introducing some core idea of risk management from tradition financial market field is necessary, such as using sharpe ratio or information ratio to monitor and gauge our trading strategy and system.

Data source: https://www.kaggle.com/mczielinski/bitcoin-historical-data?almost-there=true&token=lcXUBgt0pJdQptiQ6a-zgzviYOg81NxR2gQ7OO1MDjbOxV58ORB7OUtyzA38YV99_Dw4Zakf6Tu3U8mc-ql5CeFVCjVGnaEbsrzIEQLamqzWuL3GgRp0vO6-dXZCT5z--23nKkovv22OXuZc1fWhgFU3RINV07TeYzXGlzBSaCYzb5efA0-lmi9UcdfPZHhfkt9ctBJPDEouKEMQWldMul4HIUfvEJKGw2_GAUUL6VmfSet5fP-q5kQiTYSRKNK84TFxi_vzrl8nCMH9EDKlNUw15Smcv7glYhSx_OsEaXjUYahSTtEqO3GeICIYc0cliUQ26Um1XN-Md18eXE5H6-vK4vfJMSEjJj0ezISYB_o_5Ju4RZOgiiYZlZ5AD8Bl22XD_iMLzZ4k68ORFTN9Gpu-Sabx4A7KwbmSxQspMxpHoD7_svT-CanhG-bWs1aT-4uB1S72LO2539GDQjxHUq9NNEOOLl_lxhYO7Kskf6uOEJGHQyrRF9BidglmXdFfYJJ1y9NWEiFYKdHHt_UATcwa8jJ-9aa-cLmatuFLhusHhSjNlMBE9CL5hS6ZgutJDMtWNe2MRouh2K12dmRUtfKoFCA1#bitflyerJPY_1-min_data_2017-07-04_to_2018-06-27.csv

https://www.kaggle.com/sudalairajkumar/cryptocurrencypricehistory

References

https://towardsdatascience.com/bitcoin-price-prediction-using-lstm-9eb0938c22bd

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

http://intelligentonlinetools.com/blog/2018/04/03/machine-learning-stock-market-prediction-lstm-

keras/

https://www.nature.com/articles/srep01684

https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators