Problem Chosen	2024 MCM/ICM Summary Sheet	Team Control Number
C		2211264

Quantitative Trading based on Neural Networks: Forecasting and Decision making

Summary

Today, mathematical modeling is widely used in financial markets to analyze past data of investments such as Bitcoin and gold to give advice on trading decisions. Through the analysis and processing of data, we build a quantitative trading strategy model based on neural network to make predictions and decisions to maximize profits and provide suggestions to a trader.

In task1, we use neural networks to make predictions and decisions with time series of the price of gold and Bitcoin. Firstly, we train **CEEMDAN-LSTM** forecasting model to analyze the time series of price. However, there is an obvious delay phenomenon of the prediction results of **CEEMDAN-SE-LSTM** forecasting model. To reduce the lag, we train CEEMDAN-SE-LSTM forecasting model , which introduces **sample entropy** to change the way of model reconstructing, and this model greatly reduces the lag of prediction of price. As for trading decision making, we use **fuzzy control system** and **BP neural network** to established **fuzzy trading rules**, and make trading decisions everyday with the forecasting results of CEEMDAN-SE-LSTM forecasting model.

In task2, we compare our strategy with other three common trading strategies. They are **No-trade investment strategy**, **Double moving average strategy**, and **The Bollinger Band Mean Reversion strategy**. Comparison results show that our strategy is effective.

In task3, When discussing the sensitivity of our strategy to the change of transaction costs, the **response matrix** set in our quantitative trading model can make the frequency of buying or selling respond to the change of transaction costsand thus we can find relative rules. We find that when the transaction costs are larger, the short-term operation is reduced, and vice versa. When discussing the impact of transaction costs on strategies, we make two conclusions based on literature review and historical market rules.

In task4, we introduced our strategy, model and results in detail to the trader in a memorandum.

Keywords: BP neural network Fuzzy trading rules CEEMDAN-SE-LSTM Forcasting and Decision making Quantitative trading strategies Moving average

Contents

1 Introduction

1.1 Problem Background

Nowadays, volatile assets are popular in market, because market traders want to maximize their total return. Gold and bitcoin are two kind of assets used as a commission for each sale and purchase. Bitcoin can be traded everyday, while gold only can be traded on market open days.

1.2 Problem restatement

For the given tasks, only the past stream of daily prices which are provided are allowed to be used to develop models to determine the purchasing strategy everyday. 1,000 dollars are given in the beginning, and a portfolio consisting of cash, gold and bitcoin should be considered in each transaction. We will accomplish the following tasks according to the given data:

1. Calculate the final price of the initial. 1,000 dollars on 9/10/2021 based only on the data before that day using our prediction and decision models.

2. Certify that our model gives the best strategy compared with other several strategies.

3. Determine the sensibility of the strategy to the transaction costs and the influence of the change of transaction costs to our strategy and results.

4. Using a maximum of two pages to convey our strategy, model, and result to the trader.

2 Preparation of the Models

2.1 Assumptions

We assume the following constraints to help us complete our model building:

1. The daily transaction occurs in the moment before the close of market, the transaction does not take time to happen.

2. In the moment before the closing of market, we can immediately buy bitcoin after converting some of the gold into cash, which almost takes no time, and similar transactions are the same.

3. In the moment before the closing of market, we can immediately use the model to make predictions and decisions about the price trend of the following day according to the closing price of the day and the price of the past. The time when the model runs for both prediction and decision making can be ignored.

2.2 Notations

The primary notations used in this paper are listed in Table 1.

Table 1: Notations

Symbol	Definition
imf	The first-order modal component obtained according to EMD method
res	Residual items
$SampEn(k,r,n)$	Sample entropy
$\overline{P}_{t,n}$	The average price during n days
$x_{1,t}^{m,n}$	The relative change of moving average price to moving average price
ed	Excess demand

3 The Models for forecasting of price and making trading decisions

3.1 Model 1 Price forecasting model

3.1.1 Basic model description

In order to work out the final price of the initial 1,000 dollars on 9/10/2021, firstly,we use **CEEMDAN-LSTM** forecasting model[1]to predict the price of bitcoin and gold in the next day. However, due to the time lag of the model, which can lead to inaccurate predictions, we introduce sample entropy as the basis for component reconstruction, and built the **CEEMDAN-SE-LSTM** forecasting model to achieve more accurate price prediction.

3.1.2 The principle of CEEMDAN-LSTM forecasting model

CEEMDAN is an improvement model based on empirical modal decomposition (EMD), representing data characteristics on different time scales by splitting sequences into a series of intrinsic mode functions (imf) . However, the unimproved EMD has a modal stacking defect, so CEEMDAN based on the white noise method of EEMD is proposed to improve the original data. The adaptive white noise solves the problem of modal stacking and excess noise residue simultaneously, and improves the decomposition efficiency. The CEEMDAN algorithm procedure is as follows:

1. Normal distribution white gaussian noise is added to the original time series:

$$y_i(n) = y(n) + aw^i(n), i = 1,2,3,...t \tag{1}$$

Where $y(n)$ is the original sequence,$w^i(n)$ is white gaussian noise,a is standard deviation for the noise, t is the number of noise.

2. The first-order modal component $imf_i^1(n)$ was obtained according to the EMD method, and the mean is taken as the first imf component, and the residual value of the first phase is calculated:

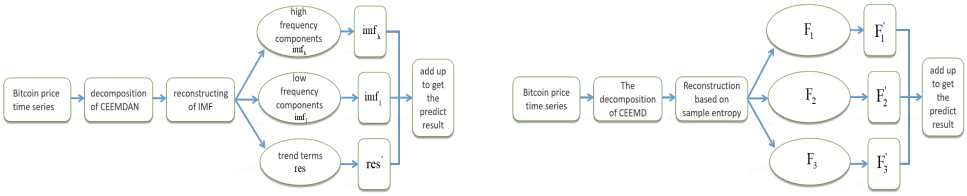
$$imf_1(n) = \frac{1}{t}\sum_{i=1}^{t} imf_1^i(n) \tag{2}$$

$$res_1(n) = y(n) - imf_1(n) \tag{3}$$

3. Similarly, take the residual value as the original time series, repeat steps1, step2. After that, EMD decomposition is perform after adding adaptive white gaussian noise to obtain $imf_2(n)$ and the corresponding residual value. Repeat the above steps until the residual amount can not be decomposed, which means the residual value is already a monotone function or constant. Finally we get k orthogonal imf function with the final trend term res:

$$y(n) = \sum_{i=1}^{k} imf_i(n) + res_k \tag{4}$$

Based on the sequence decomposition of CEEMDAN, CEEMDAN-LSTM is built to predict the price of bitcoin and gold. First, the bitcoin price's time sequence is taken as the input data, and the imf components and trend value(res) are obtained after CEEM-DAN decomposition. By reconstructing the imf components, three prediction components are obtained (high frequency components, low frequency components and trend terms). The three prediction components are input into the LSTM model for prediction respectively, and the final prediction results are obtained. The whole process of price prediction is in Figure 1.



(a) CEEMDAN-LSTM forecasting procedure

(b) CEEMDAN-SE-LSTM forecasting procedure

Figure 1: CEEMDAN-LSTM and CEEMDAN-SE-LSTM forecasting procedures

Because the high frequency component reflects short-term fluctuations and the original sequence influence is small and the mean is close to zero, we can screen the high frequency component by independent-samples T test, and then merge into a new sequence imf_h, which reflects random fluctuations. Except for the residual componentother residual components merge into low frequency component imf, which reflect various factors, such as fundamentals, technical adjustment on the sequence the trend term res reflects the trend of the original sequence. Specific steps are performed as follows:

1. Calculate the means of $imf_i(n)$, i=1,2,3, . . . , k.

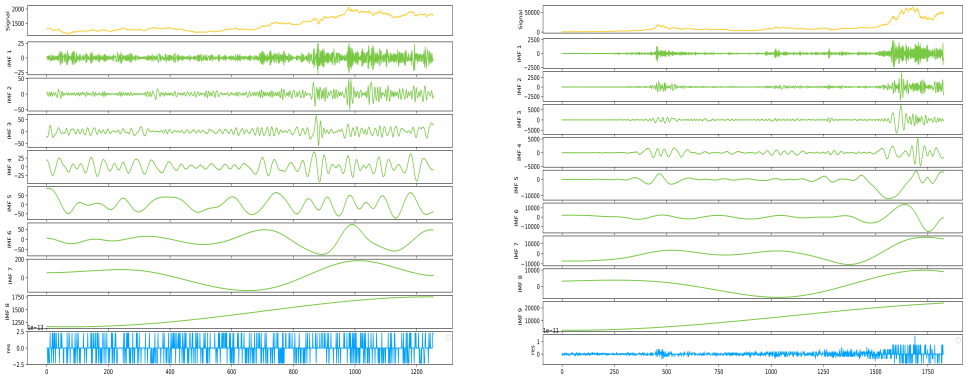
2. Independent-samples t test for $imf_i(n)$(significance level=0.05, mean is not zero).

3. After successive inspection, the first mean value of significant non-zero fractions is obtained ($imf_q(n)$),then we add up $imf_1(n)$ to $imf_{q-1}(n)$ to get the high frequency subsequence imf_h, after which $imf_q(n)$ to $imf_k(n)$are added up to get the low frequency subsequence $imf_k(n)$. res_k is still the trend term.

Refer to Torres [2] parameter Settings principles, parameters are set.When CEEMDAN decomposed the original sequence, white gaussian noise with standard deviation of 0.2 is added, and the number of addition is 500.

3.1.3 CEEMDAN-LSTM forecasting model training and forecasting results

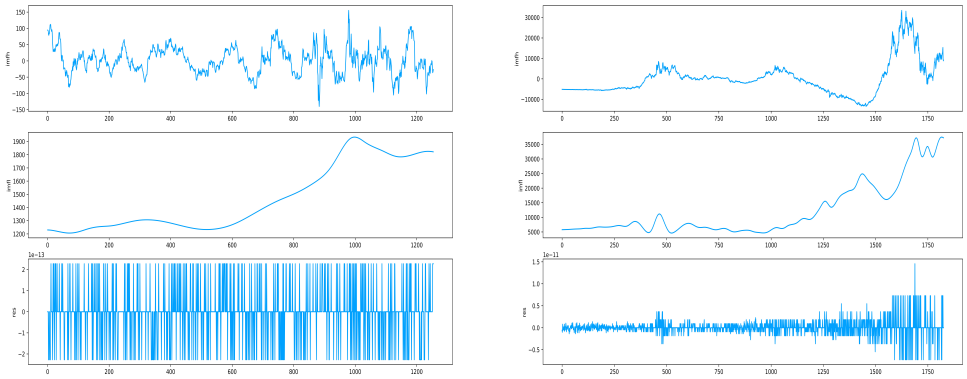
The data set of Bitcoin and gold are divided into train sets and test sets in 8:2 for debugging the model. The imf components obtained by CEEMDAN decomposition are shown in Figure 2.



(a) gold (b) Bitcoin

Figure 2: CEEMDAN decomposed components

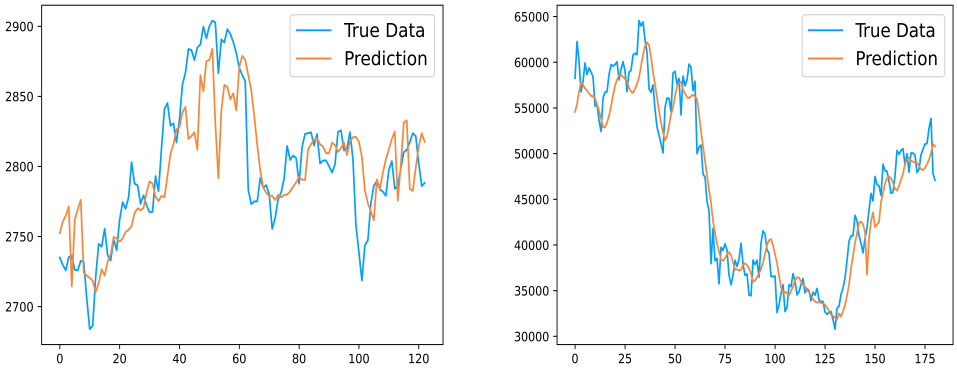
Before the reconstruction, we perform a t-test on each imf component, and reconstruct the 10 components into high-frequency components, low-frequency components and trend components according to the results, and the reconstruction sequence is shown in Figure 3.



(a) gold (b) Bitcoin

Figure 3: Components reconstruction results

The three-component prediction results are summed up, and the final prediction results are shown in Figure 4.



(a) gold (b) Bitcoin

Figure 4: Prediction results of CEEMDAN

3.1.4 The principle of CEEMDAN-SE-LSTM forecasting model

From the result we find that there is a obvious lag between the prediction of model and the actual situation. Due to this problem, sample entropy is introduced as basis for imf component reconstruction based on the CEEMDAN-LSTM model. Starting from the complexity of the time series, the sample entropy quantitatively describes the complexity and regularity of the system, so as to determine the magnitude of the probability of generating a new pattern. The larger the entropy value obtained, the more complex the time series, the greater the probability of generating a new pattern, and the simpler the sequence, the smaller the probability of generating a new pattern. The calculation process of sample entropy is as follows:

1. For a given time series $y(n)$, a set of k-dimensional vectors $y_k(1),...,y_k(n-k+1)$are arranged by ordinal number, where $y_k(i) = y(i)),y(i+1),...,y(i+k-1)(1 \le i \le n-k+1)$.

2. Define the distance of $y_k(i)$ and $y_k(j)$ as the absolute value of the maximum difference between the two corresponding elements, which is denoted $d[y_k(i),y_k(j)]$.

3. Given the threshold h, for each$i$, calculate the number of $d[yk(i),yk(j)] < r$, which is denoted $N_k(i)$ and we define:

$$B_k^r(i) = \frac{N_k(i)}{n-k+1} 1 \le i \le n-k, i \ne j \tag{5}$$

4. Calculate means of all the above defined values, and denote them to one variable B_k^r.

$$B_k^r = \frac{1}{n-k+1} \sum_{i=1}^{n-k+1} B_k^r(i) \tag{6}$$

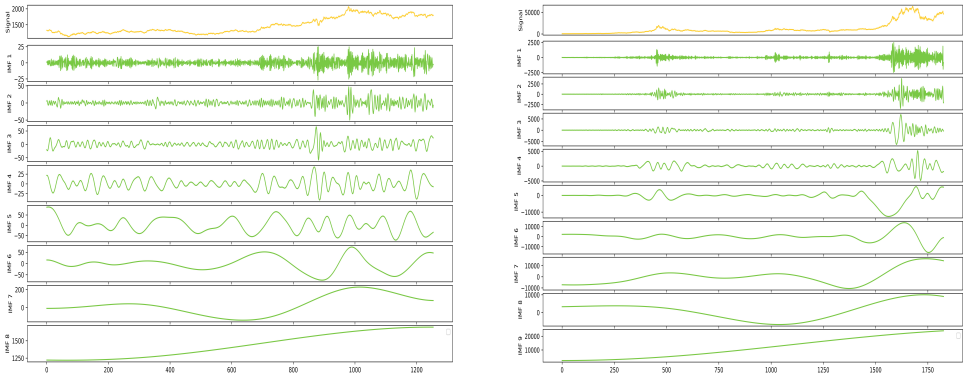
5. Repeat the above steps to get B_{k+1}^r, and when n is limited, the estimated value of *sample entropy* can be obtained, which is:

$$SampEn(k,r,n) = -ln(\frac{B_{k+1}^r}{B_k^r}) \tag{7}$$

Based on the characteristics of sample entropy in determining sequence complexity and new pattern probability, sample entropy is introduced as the basis of reconstruction for calculation . Unlike the previous CEEMDAN-LSTM model, which uses low frequencies and high frequencies as a basis for reconstruction, this model needs to calculate the sample entropy value. The closer the value of sample entropy, the more similar the representative components are, and the more consistent the fluctuations are. The forecasting process for this model is in Figure 1(b).

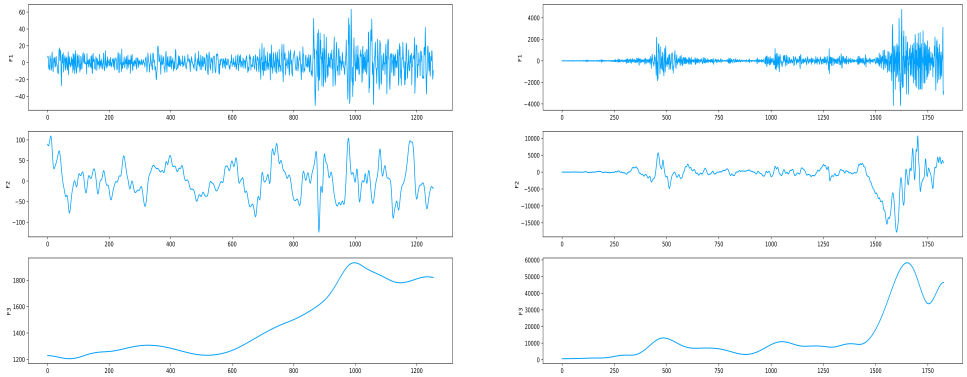
3.1.5 CEEMDAN-SE-LSTM forecasting model training and forcasting results

After calculating the sample entropy, we combined and overlapped the imf components with little difference in adjacent entropy values to obtain 3 consistency sequences, the results are shown in Figure 5.



(a) gold (b) Bitcoin

Figure 5: CEEMDAN-SE decomposed components



(a) gold (b) Bitcoin

Figure 6: 3 consistency sequences

This is followed by component prediction and summation. The three-component prediction results and the summation results are shown in Figures 6 and 7. It can be seen that the time delay effect has been greatly reduced.

Then we use the CEEMDAN-SE-LSTM model to predict the data, the prediction of the price of gold and bitcoin for each day refers only to all the data before that day, each prediction comes

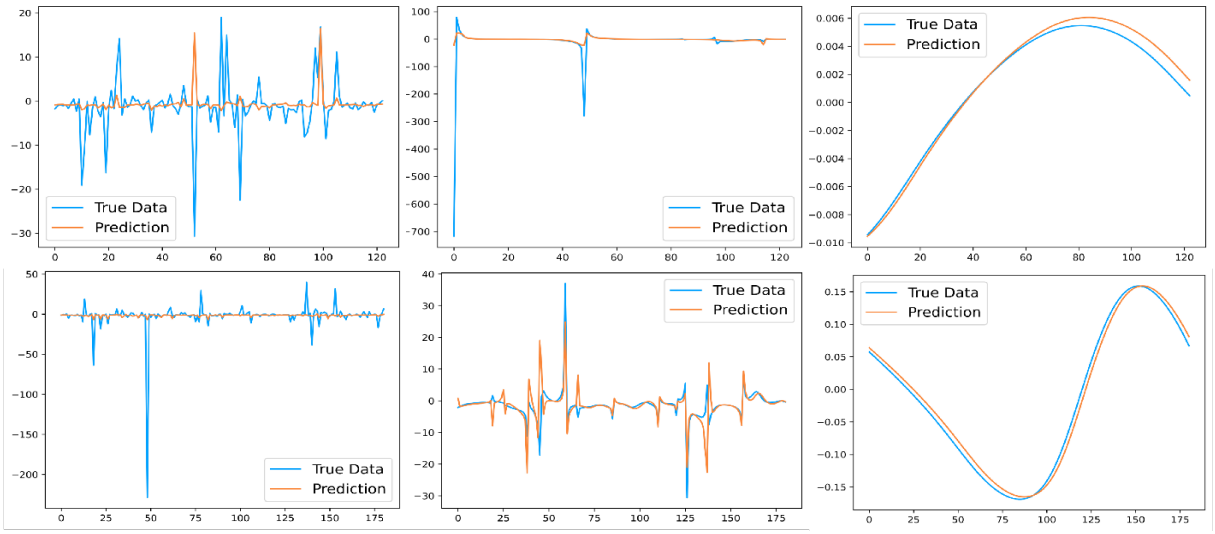
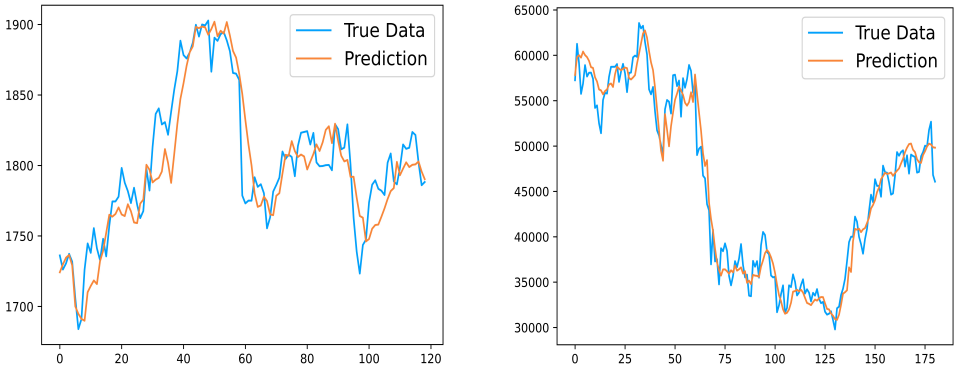


Figure 7: Forecasting result of reconstructed components



(a) gold (b) Bitcoin

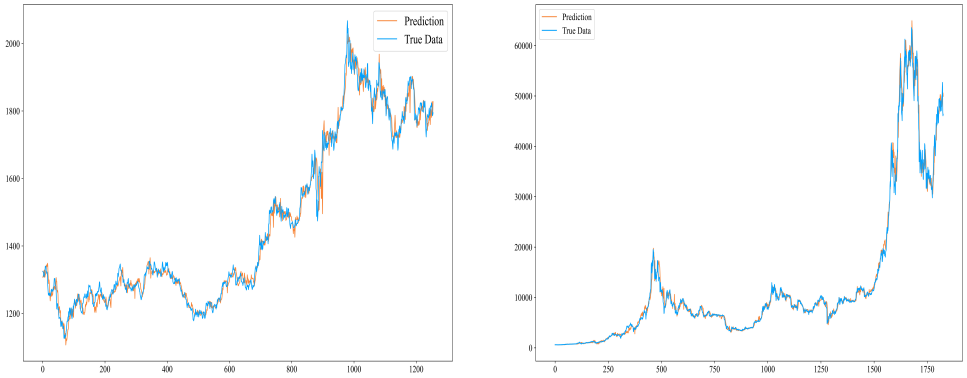
Figure 8: Summation results

from a newly trained neural network, and there is no connection between each network to avoid the recurrence of the training set and the test set. The results are shown in Figure 8. The accuracy of the prediction are: gold—0.9882 accurate, Bitcoin—0.9936 accurate.

3.2 Model 2 Trade decision model

3.2.1 Basic model description

To make trading decisions everyday(buy or sell bitcoin and gold or no trading at all) based on the prediction price of Bitcoin or gold produced by **model 1**, we develop **model 2** to make the technical analysis indicators fuzzed by a triangular fuzzifier and non-subjective transaction rules generated by **BP neural network**. As a result, we could find the best way to make trading decisions every day through BP neural network and maximize the benefits of initial 1,000 dollars on the last day.



(a) gold (b) Bitcoin

Figure 9: Forecasting results day by day

3.2.2 Fuzzy trading rules trained

The most common trading rule for technical traders is moving average(MA),which is based on different lengths. We define the value of moving average of bitcoin or gold as follows:

$$\overline{P}_{t,n} = \frac{1}{n}\sum_{i=0}^{n-1} P_{t-i} \tag{8}$$

Where P_t is the price of Bitcoin or gold today, $\overline{P}_{t,n}$ is the average price during n days. Then we set:

$$x_{1,t}^{m,n} = ln(\frac{\overline{P}_{t,m}}{\overline{P}_{t,n}}) \tag{9}$$

The formula (9) shows the relative change of moving average price of m to the moving average price which length is n. In this case, we choose (1,5) for (m,n). If $x_{1,t}^{(m,n)}$ is above zero, it indicates the price trend is upward. While the value of $x_{1,t}^{(m,n)}$ is negative, it implies a downward trend in price.

Next we turn the Moving Average indicator into the natural language of the fuzzy system. We define several fuzzy sets by the magnitude of the percentage amplitude change. They are "positive small (PS)", "positive medium (PM)", "positive large (PL)", "positive enormous(PE)", "negative small (NS)", "negative medium (NM)", "negative large (NL)", "negative enormous(NE)" and "zero (AZ)". The trigonometric membership function is shown in Figure 9.

w is a positive constant, which quantify the concept of"large", "medium", and "small" when investors describe price changes. For example, when $w = 0.01$, it means that investors may feel that the change of P_t is about 1%, and then consider the price change to be "small". When the change is about 2%, they consider the price change to be "medium"; when the change is about 3%, they consider the change to be "large"; and when the change exceed 4%, they consider the change to be "enormous". In fact, W is a sensory threshold parameter that plays an important role in the dynamic equation of prices which is to be discussed. In addition, W is also a parameter that reflects the trading frequency. The smaller W is, the easier it is for a trader to change from a trend taker to a contrarian taker, and the higher the trading frequency is, and vice versa.

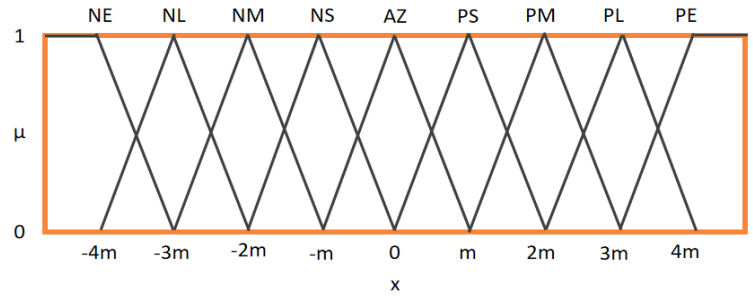


Figure 10: Fuzzy membership functions of PS,PM,PL,PE,NS,NM,NL,NE and AZ

Then, we define fuzzy sets that describes buy and sell signals. The value of *ed* is the strength of the buy signal or sell signal, where *ed* represents the excessive demand of the asset (demand supply). The symbol of *ed* can be positive or negative, and the 9 fuzzy sets defined *ed* are "buy enormous(BE)", "buy small (BS)", "buy medium (BM)", "buy big (BB)","sell small (SS)", "sell medium (SM)","sell big (SB)", "sell enormous(SE)", "and "hold (N)", and the images of their respective membership functions are shown in Figure 10.

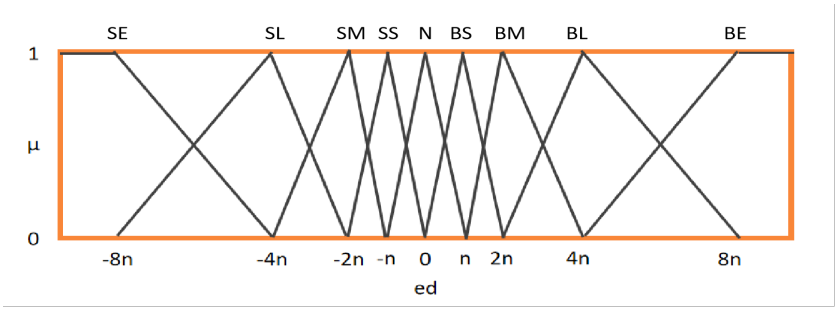


Figure 11: Membership functions of fuzzy set of excess demand BS,BM,BB,BE,SS,SM,SL,SE and N

The value of *ed* in Figure 10 represent the buy or sell power of traders who use the moving average heuristic as a trading strategy(the bigger the buy or sell power,the more the investor is inclined to buy (sell) the asset,and the greater the increase (fall) of the price).

3.2.3 BP neural network (Back Propagation algorithm) to train the fuzzy trading rules

We use BP neural network to train fuzzy trading rules.

Step1. Data obfuscation

At the t-moment, according to the formula (9) ,The setting (m = 1,n = 5) is brought into the corresponding 9 membership functions in fuzzy sets ("PS", "PM", "PB", "PE", "NZ", "NS", "NM", "NB", "NE"), and then the input signal matrix is obtained. Similarly, the predicted rise of the stock at $t+1$ moment is substituted into the corresponding 9 fuzzy sets("BS", "BM", "BB", "BE", "SS", "SM", "SB", "SE", "N") as *ed* and obtain the output signal matrix. The former matrix is put as the input signal of the training sample in BP neural network, and the latter matrix as the output signal of the training sample.

Step2. BP algorithm Training

The BP algorithm is used for network training so that the input signal corresponds to

the target output value. Set the learning function type, transfer function type, number of neurons in each layer, and number of hidden layers of the neural network. Ge Anderson and Sun Zhiqiang[3] mention that a network that has been repeatedly trained to achieve a preset precision can accurately respond to the input training set, and the rules of relationship between the two fuzzy sets will be stored in every parameter in the network. Interactions between networks form a tacit reasoning machine instead of subjective logical reasoning.

Step3. Unblur the data

When the trained network receives a new set of fuzzy signal inputs, it outputs a set of output signal data. Finally, the output signal is converted into a non-fuzzy variable according to the fuzzy converter of the transmitting signal,and it performed as the output of the entire fuzzy system. Figure 5 shows the entire process of mapping the fuzzy rules of a neural network.

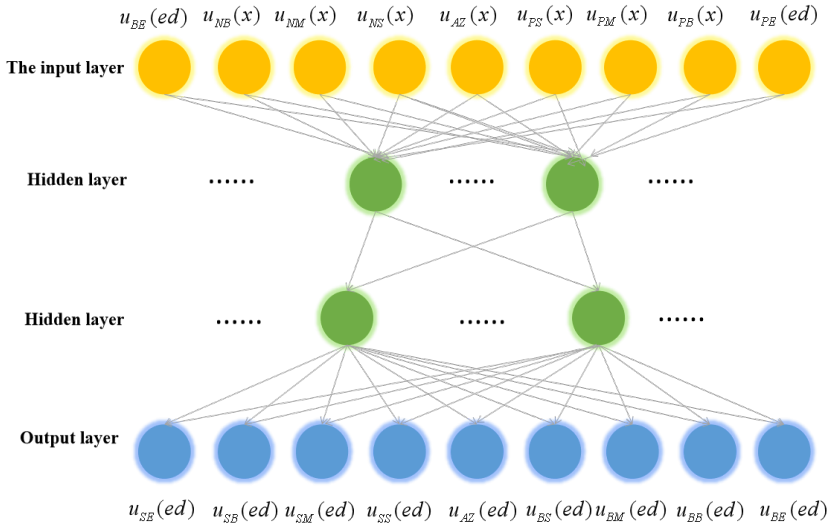


Figure 12: The structure of fuzzy trading rules generated by diagram BP network generates

We define that the center of *ed*'s membership function of fuzzy sets in output fuzzy sets is c_i.When the output matrix is obtained through BP neural network, we define the following formula (10) to solve the ambiguity and obtain *ed*, so as to obtain the guidance of the amount spent on Bitcoin and gold.

$$ed = \frac{\sum_{i=1}^{9} c_i u_{B_i}}{\sum_{i=1}^{9} u_{B_i}} \tag{10}$$

So the number of rules in the fuzzy transaction sets is 9, and c_i is the fuzzy center of BS, BM, BB, BE, SS, SM, SB, SE and N,the value of u_{B_i} is from the output matrix.

3.2.4 More details and improvements for model 2

Calculating $x_{1,t}^{m,n}$ according to formula (9), and substitute them separately into the subordinate functions of the corresponding 9 module sets to obtain a matrix of ambiguous input signals. Then the logarithm of the ratio of the price of the day after the day to the current price is calculated as an

excess demand *ed* corresponds to its division, calculate follow formula (11). The input and output signals of each set are used as the training samples for network learning.

$$ed = ln(\frac{P_T}{P_t}) \tag{11}$$

The neural network has two hidden layers, each with a number of neurons of 100. We train the neural network using data from before each day and predictions for one day after. After completing the training, the input signals from the test set are fed in and then we get the corresponding output fuzzy matrix, and according to the formula (10) the fuzzy matrix can be solved to obtain the value of *ed* and make a decision.

3.3 The final value of initial 1,000 dollars

The result of the stimulation of our models is a portfolio consisting of [C,G,B]=[29384.12, 1.55, 0] on the last day, equivalent to 32171.62 dollars.

4 Advantages of our trading strategy over common strategies

4.1 No-trade investment strategy

Compared with gold, bitcoin has bigger increase among the choosing period of time. So for a no-trade investment strategy, we invest all of 1,000 dollars in bitcoins on the first day and hold them until the last day, and convert the value of bitcoin into dollars on the last day. Through calculation, the final worth of the initial 1,000 dollars is 74589.7 dollars.

Although no-trade strategy in the case of given data earns more than our quantitative trading strategy based on neural network, very few people in daily life take no-trade investment strategy, and the strategy is not flexible which cares nothing about the market reaction.No trade strategy is likely to suffer great losses on the trading termination date, so it is not a desirable strategy compared with our strategy.

4.2 Double moving average strategy

Double moving average strategy is the most commonly used strategy in trading. The fundamental principle of double moving average strategy is that in the long run, the price of assets will revert to the mean. However, short-term averages can reflect more of recent changes than long-term averages. Based on the principle, when the short-term moving average crosses the long-term moving average from top to bottom, it indicates that the price of bitcoin or gold will drop and the strategy of going short could be adopted. When the short-term moving average crosses the long-term moving average from bottom to top, it indicates that the price of bitcoin or gold will arise and the strategy of going long could be adopted.

By doing calculations, we found that if half of the initial 1,000 dollars fund is used to buy gold and another half of the initial 1,000 dollars fund is used to buy bitcoin, the final value of 1,000 dollars is 28624.0 dollars which is lower than our strategy.

However, compared with our model, the double moving average model has a strong lag,it doesn't work for complex situations

4.3 The Bollinger Band Mean Reversion strategy

The Bollinger Band Mean Reversion strategy believes that the underlying price floats within the range enclosed by the upper and lower rail lines. Even if it breaks through the upper and lower rails in the short term, it will still return to the Bollinger Band in the long run. Therefore, once the upper and lower rails are broken, a buy and sell signal is formed.

When the stock price breaks above the upper bound, it is a sell signal, and when the stock price breaks below the lower bound, it is a buy signal. The calculation formula of the BOLL line:

$$Middle\ line = N\ day\ moving \tag{12}$$

$$standard\ deviation = \sqrt{\frac{\sum_{t=1}^{N}(P_i - MA)^2}{N}} \tag{13}$$

$$Average\ upper\ line = middle\ line + k\ standard\ deviation \tag{14}$$

$$Lower\ line = middle\ line - k\ standard\ deviation \tag{15}$$

where N= is 20 and k=2, P_i is daily price of bitcoin or gold.

Divide the initial 1,000 dollars into two parts of 500 dollars at the beginning, half for the trading of gold and half for the trading of bitcoin,.The final value of the initial 1000 dollars calculated by the Bollinger Band Mean Reversion strategy was 665.90 dollars

This shows that for volatile transactions such as Bitcoin, the common Bollinger Band Mean Reversion strategy is not appropriate, and this strategy is not suitable for long-term trading. Therefore, compared with our strategy, it is not universal.

5 The sensitivity of the strategy to transaction costs and the transaction costs' influence to strategy and results

5.1 The sensitivity of the strategy to transaction costs

For sensitivity analysis of our quantitative model, we conduct tests by changing transaction costs. Partial test results are shown in the Figure 12:

Where GTC is transaction cost of gold, BTC is transaction cost of bitcoin, GTT is transaction times of gold, BTT is transaction times of bitcoin, TT is transaction times of bitcoin, and TM is total money.

Through the test results, we can find that the model is not sensitive to the change of transaction costs, and the change of transaction costs can hardly cause the change of transaction times.

GTC	BTC	GTT	BTT	TT	TM
0.01	0.02	1084	1544	2628	1946
0.01	0.01	1084	1545	2629	2428
0.01	0.03	1090	1548	1548	1542
0.005	0.02	1084	1546	1546	2076
0.02	0.02	1082	1540	1540	1713

Figure 13: Partial test results

Therefore, we make the following modifications to the model:

Let X be the response matrix, X=$[1,4]^T$, and the modified matrix y= $X \cdot a$, where $a = [a_1, a_2]$, a_1 is the gold transaction costs and a_2 is the Bitcoin transaction costs. We use the modified matrix y to limit the trade, and test the modified model. Part of the test results are shown in Figure 13.

GTC	BTC	GTT	BTT	TT	TM
0.01	0.02	404	47	451	32171
0.01	0.01	334	130	464	1539
0.01	0.03	472	28	500	2537
0.005	0.02	590	58	648	33190
0.02	0.02	221	43	264	30826

Figure 14: Partial test results

After magnifying (or decreasing) the handling fee of Bitcoin or gold, the number of transaction of Bitcoin or gold will increase (or decrease) correspondingly. For the relationship between transaction costs and the number of transactions, we explored and draw Figure 14. So far we have established a transaction times - fee linear relationship.

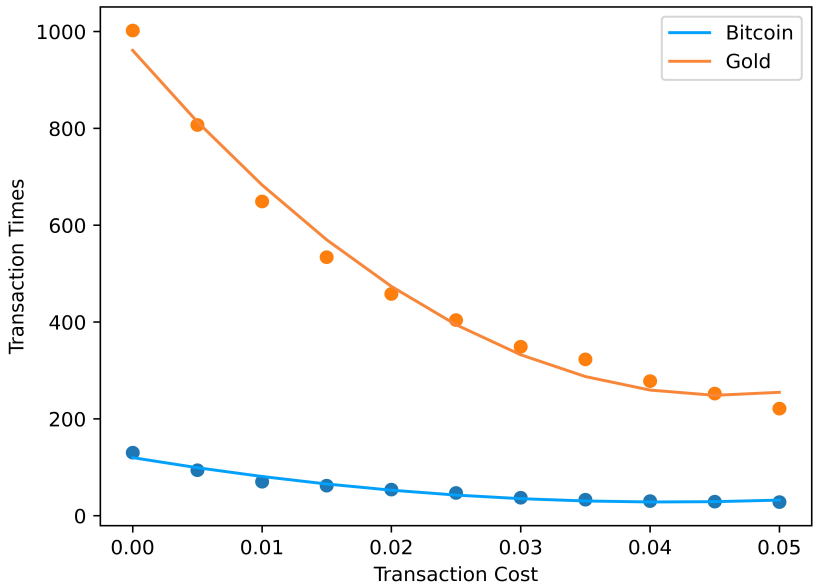


Figure 15: Partial test results

5.2 The change of transaction costs' influence to strategy and results

1. The increase of transaction costs has reduced the trading enthusiasm of investors, which in turn has a two-sided effect on the market. First of all, it will reduce the information efficiency of the market. Secondly, the liquidity level of the market will be reduced ([4]Loeta1., 204; Gombereta1.2016; Pomeranets Weaver2011). Noise traders not only bear the risk of price fluctuations, but also provide important counterparts to the market, which is an important source of market liquidity, The inhibition of investors, especially noise traders, will have a negative impact on the liquidity level of the market. The reduction of information efficiency and the reduction of liquidity make the market absorb new information and the ability to cope with shocks weakened, and a lower number of orders can greatly impact market prices, causing greater market fluctuations. In the case of increased transaction costs, due to the increase in the possibility of large market fluctuations,**our investment strategy should respond strongly to the large fluctuations that will occur, reduce the perception of smaller fluctuations, and go long or go short at the right time to avoid losses to the greatest extent.To be more specific, we can increase the dimension of membership functions of fuzzy sets (excess) and change the number of input and output neurons in BP neural network to make purchasing power judgment in a larger (or smaller) range, thus to maximize the benefits ofthe effect. What is more, the investment strategy is mainly based on long-term investment to avoid paying high fees for frequent transactions.**

2. By contrast, the reduction of transaction costs will stimulate the increase in transaction volume, thereby enhancing the liquidity of the market and the efficiency of information exchange. Compared with larger market fluctuations, smaller market fluctuations are more common.**The investment strategy model should mainly focus on short-term investments, which is more sensitive to changes in small fluctuations, and buys and sells smaller positions within the appropriate fluctuation range. To be more specific, we can predict futures trends more accurately and make effective judgments and decisions by increasing the number of neurons and the number of hidden layers in CEEMDAN-SE-LSTM neural network and BP neural network.**

Memorandum

To: Dear trader
From: Team 2211264
Date: February twenty-second, 2022
Subject: Quantitative strategy based on neural network

In the memo, we want to introduce you a quantitative strategy based on neural network.

Firstly,based on neural networks, CEEMDAN-SE-LSTM forecasting model is used for each day's trading predictions. It predicts tomorrow's data based on past and this day's data. Based on the results of daily prediction, the fuzzy trading rules trained by us can directly give the direct trading strategies of gold and bitcoin, so as to maximize the benefits of trading.MSE is used as the loss function, in combination with the Adam optimizer, to obtain an high accuary with low lag of price prediction. Compared to trading strategies that are common in the market, For example, The Double Moving Average strategy and The Bollinger Band Mean Reversion strategy, our strategy can better avoid the time delay in prediction and can better adapt to the changing market.

To be more specific about the steps of trading with our quantitative model:

Our codes are based on Python, which is object-oriented, easy to operate program and with high compatibility. To run our program, you just need to input the date you want to get the trading decision. Just make little changes of the parameters, you can smoothly run the program to get advice to make decisions ,thus maximizing benefits. As CEEDAN-SE-LSTM forecasting model needs certain data basis to predict the time series, there is no decision been made at the first ten days

Secondly,through our CEEMDAN-SE-LSTM forecasting model, the overall prediction accuracy of Bitcoin is 99.36%, and that of gold is 98.82%. From 9/11/2016 to 9/10/2021, every trade decisions using our quantitative model would have 32,171.62 dollars on the last day with an initial capital of 1,000 dollars, yielding 3117.162%.

Thirdly,in the case of high transaction costs, we can modify the coefficient of the response matrix to reduce the trading times of Bitcoins or gold, so as to decline the transaction costs. If the investments have large fluctuations, we can increase the dimension of membership functions of fuzzy sets (excess) and change the number of input and output neurons in BP neural network to make purchasing power judgment in a larger range, thus to maximize the benefits of the effect. For investments with small trend of ups and downs, we can predict futures trends more accurately and make effective judgments and decisions by increasing the number of neurons and the number of hidden layers in CEEMDAN-SE-LSTM neural network and BP neural network.

According to all those mentioned above, it is really worth to have a try!

References

[1] Xun Z , Lai K K , Wang S Y . A new approach for crude oil price analysis based on Empirical Mode Decomposition[J]. Energy Economics, 2008, 30(3):905-918.

[2] TORRES M E,COLOMINAS M A,SCHLOTTHAUER G, et al.A complete ensemble empirical.

[3] Ge Z X, Sun Z Q. Neural Network Theory and Matlab R2007 Application. Beijing: PublishingHouse of Electronics Industry, 2007.

[4] Habermeier, KandKirilenkoA2001Securitiestrans- action tax esand finan cial marketsIMF WorkingPapers

Appendix : Program Codes

Here are the program codes we used in our research.

prediction.py

```
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
import numpy as np
import pandas as pd
import lstm, time
def train_lstm(path,normalized,sql_len,num):
    X_train, y_train, X_test, y_test, data_test,y_ordinary =
    lstm.load_data(path + '.csv', sql_len, normalized,num)
    model = Sequential()
    model.add(LSTM(input_dim=1, units=50, return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(100, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))
    model.add(Activation('linear'))
    start = time.time()
    model.compile(loss='mse', optimizer='adam')
    print('compilation time : ', time.time() - start)
    model.fit(X_train, y_train, batch_size=128, epochs=20,
    validation_split=0.05,verbose=0)
    predictions = lstm.predict_point_by_point(model, X_test)
    lstm.plot_results(predictions, y_test)
    return predictions,y_test,data_test,y_ordinary

def predict_data(name,sql_len,num):
    imfhpredictions, imfhtest, data_test, y_ordinary1 =
    train_lstm(name + "imfh", True,sql_len,num)
    denor_imfh = lstm.denormalise_windows(imfhpredictions,
    data_test, sql_len)
    imflpredictions, imfltest, data_test, y_ordinary2 =
    train_lstm(name + "imfl", True,sql_len,num)
    denor_imfl = lstm.denormalise_windows(imflpredictions,
    data_test, sql_len)
    respredictions, restest, data_test, y_ordinary3 =
    train_lstm(name + "res", True,sql_len,num)
    denor_res = lstm.denormalise_windows(respredictions,
    data_test, sql_len)
    print(list(np.array(denor_imfh)+np.array(denor_imfl)
    +np.array(denor_res)))
    y_ordinary1 = [float(s) for s in list(y_ordinary1)]
    y_ordinary2 = [float(s) for s in list(y_ordinary2)]
    y_ordinary3 = [float(s) for s in list(y_ordinary3)]
    finalimf = list(np.array(denor_imfh) + np.array(denor_imfl)
```

```
    + np.array(denor_res))
    y_ordinary = list(np.array(y_ordinary1) + np.array(y_ordinary2)
    + np.array(y_ordinary3))
    lstm.plot_results(finalimf, y_ordinary)
    return 0

def get_len(t):
    if t<10:return 2
    if t<100:return 5
    else:return 20

if __name__ == "__main__":
    Bitprediction = []
    for i in range(12, 1826):
        Bitprediction.append(predict_data("Bit", get_len(i), i + 1))
    Bitprediction = pd.DataFrame(Bitprediction)
    Bitprediction.to_csv("Bitprediction", index=False, header=False)
```

lstm.py

```
import time
import warnings
import numpy as np
from numpy import newaxis
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")

def plot_results_multiple(predicted_data, true_data, prediction_len):
    fig = plt.figure(facecolor='white')
    ax = fig.add_subplot(111)
    ax.plot(true_data, label='True Data')
    print('yo')
    for i, data in enumerate(predicted_data):
        padding = [None for p in range(i * prediction_len)]
        # print('type_data',type(data))
        plt.plot(padding + data, label='Prediction')
        plt.legend()
    plt.show()

def plot_results(predicted_data, true_data):
    fig = plt.figure(facecolor='white')
    ax = fig.add_subplot(111)
    ax.plot(true_data, label='True Data',c = '#00a1ff')
    plt.plot(predicted_data, label='Prediction',c = '#f98639')
    plt.legend(fontsize=15)
    plt.savefig(r'predBH.jpg',dpi=1000, bbox_inches = 'tight')
    plt.show()

def load_data(filename, seq_len, normalise_window,num):
    f = open(filename, 'r').read()
```

```
    data = f.split('\n')

    sequence_length = seq_len + 1
    result = []
    for index in range(num - sequence_length):
        if index + sequence_length > len(data):break
        result.append(data[index: index + sequence_length])
    if normalise_window:
        norm_result = normalise_windows(result)
    else:
        norm_result = result
    result = np.array(result)
    norm_result = np.array(norm_result)
    row = round(0.8 * result.shape[0])
    data_test = result[int(row):, :]
    y_ordinary = result[int(row):,-1]
    train = norm_result[:int(row), :]
    np.random.shuffle(train)
    x_train = train[:, :-1]
    y_train = train[:, -1]
    x_test = norm_result[int(row):, :-1]
    y_test = norm_result[int(row):, -1]
    x_train = np.reshape(x_train, (x_train.shape[0],
    x_train.shape[1], 1))
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
    return [x_train, y_train, x_test, y_test, data_test,y_ordinary]

def denormalise_windows(normdata, data, seq_len):
    denormalised_data = []
    wholelen = 0
    for i, rowdata in enumerate(normdata):
        denormalise = list()
        if isinstance(rowdata, float) | isinstance(rowdata, np.float32):
            denormalise = (rowdata + 1) * float(data[wholelen][0])
            denormalised_data.append(denormalise)
            wholelen = wholelen + 1
        else:
            for j in range(len(rowdata)):
                denormalise.append((float(rowdata[j]) + 1) *
                float(data[wholelen][0]))
            wholelen = wholelen + 1
            denormalised_data.append(denormalise)
    return denormalised_data

def normalise_windows(window_data):
    normalised_data = []
    for window in window_data:
        normalised_window = [((float(p) / float(window[0])) - 1)
        for p in window]
        normalised_data.append(normalised_window)
```

```
    return normalised_data

def predict_point_by_point(model, data):
    predicted = model.predict(data)
    predicted = np.reshape(predicted, (predicted.size,))
    return predicted

def predict_sequence_full(model, data, window_size):
    curr_frame = data[0]
    predicted = []
    for i in range(len(data)):
        predicted.append(model.predict(curr_frame
        [newaxis, :, :])[0, 0])
        curr_frame = curr_frame[1:]
        curr_frame = np.insert(curr_frame, [window_size - 1]
        , predicted[-1], axis=0)
    return predicted

def predict_sequences_multiple(model, data, window_size, prediction_len):
    prediction_seqs = []
    for i in range(int(len(data) / prediction_len)):
        curr_frame = data[i * prediction_len]
        predicted = []
        for j in range(prediction_len):
            predicted.append(model.predict
            (curr_frame[newaxis, :, :])[0, 0])
            curr_frame = curr_frame[1:]
            curr_frame = np.insert(curr_frame,
            [window_size - 1], predicted[-1], axis=0)
        prediction_seqs.append(predicted)
    return prediction_seqs
```

bp train.py

```
# encoding: utf-8
import datetime
import numpy as np
import math
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
bitcoin = pd.read_csv("../Pre/zBCHAIN-MKPRU.csv")
print(bitcoin)
Bitprediction = pd.read_excel("Bitprediction.xlsx")
def fuzzy1(x):
    m = 0.06
    NE = np.piecewise(x, [x <= -4 * m, -4 * m < x <= -3
    * m, x >= -3 * m],[lambda x: 1, lambda x: -1/m * x - 3,
    lambda x: 0])
```

```
    NL = np.piecewise(x, [x <= -4 * m, -4 * m < x <= -2 * m,
    x >= -2 * m],[lambda x: 0, lambda x: 1 - abs(x + 3 * m)
    / m, lambda x: 0])
    NM = np.piecewise(x, [x <= -3 * m, -3 * m < x <= -m,
    x >= -m],[lambda x: 0, lambda x: 1 - abs(x + 2 * m) /
    m, lambda x: 0])
    NS = np.piecewise(x, [x <= -2 * m, -2 * m < x <= 0,
    x >= 0],[lambda x: 0, lambda x: 1 - abs(x + m) / m,
    lambda x: 0])
    AZ = np.piecewise(x, [x <= -m, -m < x <= m, x >= m],
    [lambda x: 0, lambda x: 1 - abs(x) / m, lambda x: 0])
    PS = np.piecewise(x, [x <= 0, 0 < x <= 2 * m, x >= 2 *
    m],[lambda x: 0, lambda x: 1 - abs(x - m) / m, lambda x: 0])
    PM = np.piecewise(x, [x <= m, m < x <= 3 * m, x >= 3 *
    m],[lambda x: 0, lambda x: 1 - abs(x - 2 * m) / m, lambda x: 0])
    PL = np.piecewise(x, [x <= 2 * m, 2 * m < x <= 4 * m,
    x >= 4 * m],[lambda x: 0, lambda x: 1 - abs(x - 3 * m)
    / m, lambda x: 0])
    PE = np.piecewise(x, [x <= 3 * m, 3 * m < x <= 4 * m,
    x >= 4 * m],[lambda x: 0, lambda x: 1 / m * x - 3, lambda x: 1])
    return np.array([NE,NL,NM,NS,AZ,PS,PM,PL,PE])
def fuzzy2(x):
    n = 0.2
    SE = np.piecewise(x, [x <= -8 * n, -8 * n < x <= -4 * n,
    x >= -4 * n],[lambda x: 1, lambda x: -0.25 / n * x - 1,
    lambda x: 0])
    SB = np.piecewise(x, [x <= -8 * n, -8 * n < x <= -4 * n
    ,-4 * n< x <= -2 * n, x >= -2 * n],[lambda x: 0,
    lambda x: -0.25 / n * x + 2,lambda x: -0.5 / n * x - 1,
    lambda x: 0])
    SM = np.piecewise(x, [x <= -4 * n, -4 * n < x <= -2 * n
    , -2 * n< x <= -n, x>=n],[lambda x: 0, lambda x: 0.5 /
    n * x + 2,lambda x: -1 / n * x - 1,lambda x: 0])
    SS = np.piecewise(x, [x <= -2 * n, -2 * n < x <= 0, x >= 0]
    ,[lambda x: 0, lambda x: 1 - 1 / n * abs(x+n), lambda x: 0])
    N = np.piecewise(x, [x <= -n, -n < x <= n, x >= n],[lambda x:
    0, lambda x: 1 - 1 /n  * abs(x), lambda x: 0])
    BS = np.piecewise(x, [x <= 0, 0 < x <= 2 * n, x >= 2 * n],
    [lambda x: 0, lambda x: 1 - 1 / n * abs(x - n), lambda x: 0])
    BM = np.piecewise(x, [x <= n, n < x <= 2 * n, 2 * n < x <= 4
    * n, x >= 4 * n],[lambda x: 0, lambda x: 1/ n * x - 1,
    lambda x: -0.5 / n * x + 2, lambda x: 0])
    BB = np.piecewise(x, [x <= 2 * n, 2 * n < x <= 4 * n,
    4 * n < x <= 8 * n, x >= 8 * n],[lambda x: 0, lambda x:
    0.5 / n * x - 1, lambda x: -0.25 / n * x + 2, lambda x: 0])
    BE = np.piecewise(x, [x <= 4 * n, 4 * n < x <= 8 * n,
    x >= 8 * n], [lambda x: 0, lambda x: 0.25 / n * x - 1
    , lambda x: 1])
    return np.array([SE,SB,SM,SS,N,BS,BM,BB,BE])
```

```
def get_time(t, a):
    t = str(t)
    return (datetime.datetime(int(t[:4]),
    int(t[4:6]), int(t[6:])) +
    datetime.timedelta(days=a)).strftime("%Y%m%d")

def cal(point,tdelta):
    if point:
        p0 = bitcoin.iloc[tdelta].values[2]
        p1 = bitcoin.iloc[tdelta - 1].values[2]
        p2 = bitcoin.iloc[tdelta - 2].values[2]
        p3 = bitcoin.iloc[tdelta - 3].values[2]
        p4 = bitcoin.iloc[tdelta - 4].values[2]
        p5 = bitcoin.iloc[tdelta - 5].values[2]
        x=math.log(p0/(p1+p2+p3+p4+p5)*5)
        return fuzzy1(x)
    else:
        p_t = Bitprediction.iloc[tdelta + 1].values[0]
        p0 = bitcoin.iloc[tdelta].values[2]
        ed = math.log(p_t/p0)
        return fuzzy2(ed)

def get_len(t):
    if t<10:return 2
    if t<100:return 5
    else:return 20

def get_train(tdelta):
    x = []
    y = []
    for i in range(0,tdelta - 2):
        x.append(cal(True,tdelta - i))
        y.append(cal(False,tdelta - i))
    x_test = x[:1]
    y_test = y[:1]
    x_test = np.array(x_test).astype(np.float32)
    y_test = np.array(y_test).astype(np.float32)
    X = np.array(x[1:])
    X = X.astype(np.float32)
    Y = np.array(y[1:])
    Y.astype(np.float32)
    return X,Y,x_test,y_test

def trainbp(t):
    model = Sequential()
    layer1 = Dense(100, input_shape=(9,), activation='linear')
    model.add(layer1)
    layer2 = Dense(100, activation='linear')
    model.add(layer2)
    model.add(Dense(9))
```

```
    model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])
    X_train, Y_train, x_test, y_test = get_train(t)
    history = model.fit(X_train, Y_train, epochs=30, batch_size=512,
    validation_data=[x_test, y_test],verbose=0)
    y_predict = model.predict(x_test, batch_size=512)
    acc = history.history['accuracy']
    loss = history.history['loss']
    acc_val = history.history['val_accuracy']
    loss_val = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    plt.title('Accuracy and Loss')
    plt.plot(epochs, acc, 'red', label='Training acc')
    plt.plot(epochs, loss, 'blue', label='Validation loss')
    plt.plot(epochs, acc_val, 'green', label='Training acc_val')
    plt.plot(epochs, loss_val, 'black', label='Training loss_val')
    plt.legend()
    plt.show()
    score = model.evaluate(x_test, y_test, verbose=0)
    print(score)
    return y_predict.tolist()[0],y_test.tolist()

if __name__ == "__main__":
    edbit = []
    for i in range(12, 1825):
        a, b = trainbp(i)
        edbit.append(a)
        print(i, b, a)
    pd.DataFrame(edbit).to_csv("edbit.csv")
```