# Fundamentals of Information Theory

## Group Project Report

**Project Name:** Stock Market Analysis based on Information Theory

**Student Name：** Liangsheng Xu, Sikai Yin, Wan Wang (Team 4)

**Department:** Data Science and Information & Information Technology

**Teacher：** Prof. Lin Zhang

**FALL 2018**

**Shenzhen, China**

# Task 1 Stock Closing Price Forecasting

For the first task in this project, the stock price forecasting model needs to be built so as to forecast the closing price for the testing dataset. More importantly, it may provide reference for the second task of constructing automatic investment portfolio.

In this task, the training dataset containing the minute-based stock prices of each stock in the market is given, with timestamp from 2008-01-02 to 2009-11-30. Altogether, there are 55,500,855 records in the training data file, which corresponds to 512 stocks in the market. It is obvious that due to some stock market suspension factors, the records for each stock can vary greatly, leading to the NULL value problem in this training set. In the meanwhile, the test dataset includes the minute-based stock prices of each stock in the market with timestamp from 2009-12-01 to 2009-12-31. Based on price value for each minute, there are 5543 values that need to be forecasted by the model for each stock.

After this description of training and testing dataset, it is obvious that this task is a time-series forecasting problem. In order to give precious forecasting, reasonable forecasting model and algorithm need to be designed. After referring to papers in this field, our group finally selected the LSTM Recurrent Neural Network(RNN) model to conduct this forecasting task. This kind of network has loops in it, allowing information to persist. As an improvement of original RNN model, LSTM model resolves the problem of long-term dependencies[1]-[2]. LSTM was introduced by Hochreiter & Schmidhuber (1997)[3], and was refined and popularized by many people in following work. For each cell of LSTM, there are gates to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation and LSTM cell has three of these gates to protect and control the cell state. The three gates are as follows[4]:

1) Forget gate, which decides how much information to throw away from the cell.
2) Input gate, which decides which values should be updated and a tanh layer creates a vector of candidate values.
3) Output gate, which decides which part of Cell state should be included in the output.

Based on this LSTM model, many improvements has been conducted. For this project, our group chose a common paradigm called encoder-decoder (or sequence to sequence), whose goal is to produce an entire sequence of forecasting stock price. The Seq2Seq framework relies on the encoder-decoder paradigm[]. The encoder encodes the input time-series sequence, while the decoder produces the target time-series sequence. Such model has been refined over the past few years and greatly benefited from what is known as attention. Attention is a mechanism that forces the model to learn to focus on specific parts of the input sequence when decoding, instead of relying only on the hidden vector of the decoder's LSTM.[5]

Our group realized this LSTM-based Seq2Seq framework with attention mechanism by Python programming on Google TensorFlow platform. We then utilized this forecasting model to train and test our dataset. The first step involves data standardization, which means the raw stock price data in the dataset needs to be

regularized to zero-mean and unit variance for the forecasting model. Such regularization is realized through the following functions.

$$X_i' = \frac{X_i - mean(X)}{std(X)} \tag{1}$$

In this standardization function, $\mathrm{X_i}$ is the raw data in the training set, $\mathrm{mean(X)}$ denotes the mean value of all data in training set and $\mathrm{std(X)}$ is the sample covariance. After this processing, the standardized data $X_i'$ will be utilized as the input to train the forecasting model.

According to the project instruction, the prediction results should be evaluated through the Root-mean-square error (RMSE) function, which is defined as follows:

$$\mathrm{RMSE} = \sqrt{\mathrm{MSE}(\hat{X})} = \sqrt{E((\hat{X} - X)^2)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{X}_i - X_i)^2} \tag{2}$$

Therefore, the RMSE can be calculated after the forecasting result. In order to test the performance of our forecasting model, we will segment the training set into two parts. The first price data for the first 22 months will be utilized to train the forecasting model, while the data of the last month(November 2009) will be used as validation to compute the RMSE and evaluate the performance of our forecasting model.

In the first trail, we took the stock with stock code 600000 as an example and input all the minute-based history price data to train the model. After training, we output 5543 forecasting data points and calculate the RMSE between forecasting value and real data. The model parameters and forecasting price curve result is shown as below:

Table 1. Model parameter settings for the minute-based forecasting

| Parameters | Value |
|---|---|
| num_layers | 1 |
| size_layer | 1280 |
| dropout_rate | 0.7 |
| timestamp | 1205 |
| attention_size | 100 |
| future_point | 5543 |

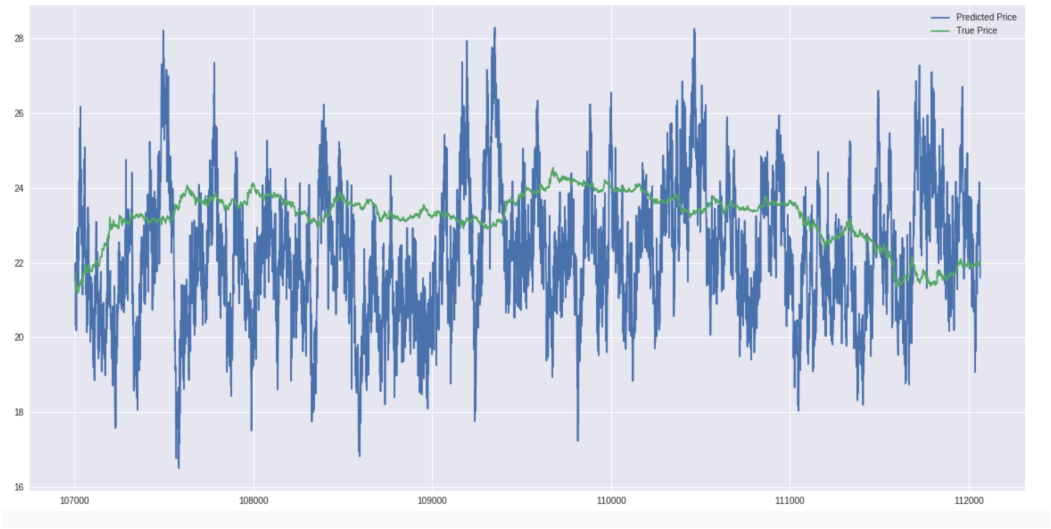


Figure 1. Forecasting result comparison for Stock 600000 by minute-based forecasting

As is shown, the prediction result of our LSTM Seq2Seq with Attention model(marked in blue) failed to simulate the trend or value of the real price data(marked in green). The RMSE for this prediction is 2.22748, which will lead to significant loss of money if utilized as the guidance of portfolio. More importantly, if we output the prediction line through all these 23 months, the curve will look like below:

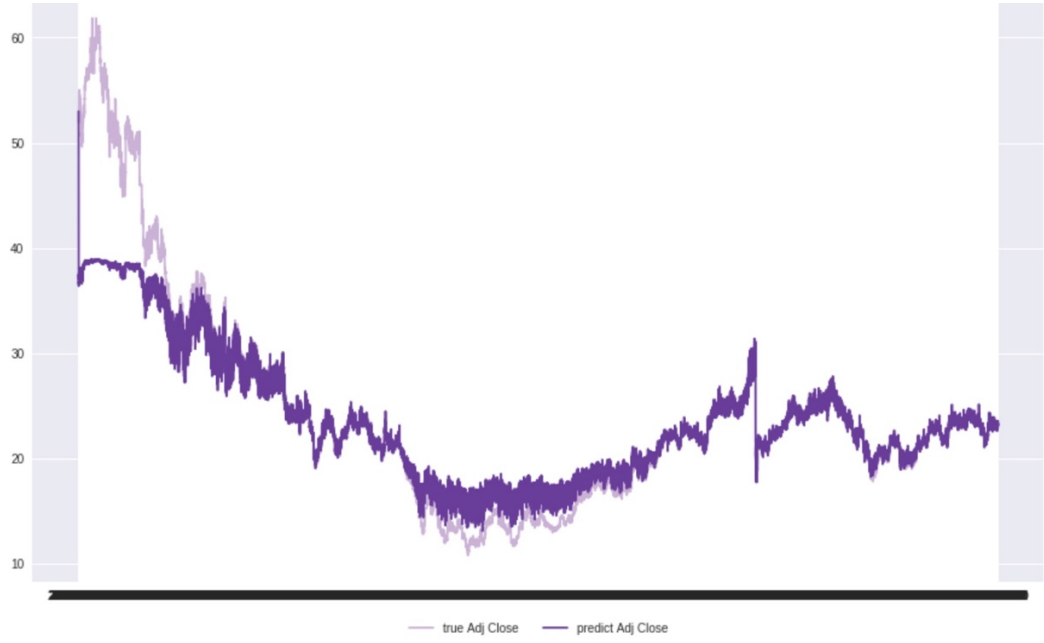


Figure 2. Fitting curve of forecasting model over all history data of Stock 600000

The fitting performance of the LSTM model during the first 22 months clearly shows that it learns the sharp fluctuation of stock price during the timestamp(1205 in this case), and then output similar sharp fluctuation in forecasting price data even between the adjacent two minutes, which is opposite to the real case. After this analysis, our group abandoned this minute-based idea of forecasting. Instead, we are looking into the forecasting performance of our model in the daily-based data.

To transform our raw minute-based data into the daily-based, we picked the

mean value of stock price during 241 minutes in one day as the representative price for this transaction day:

$$X_{daily} = \frac{\sum_{i=1}^{241} X_i}{241} \tag{3}$$

After this daily-based data processing, the parameter of our forecasting is now shown as:

Table 2. Model parameter settings for the daily-based forecasting

| Parameters | Value |
|---|---|
| num_layers | 1 |
| size_layer | 128 |
| dropout_rate | 0.7 |
| timestamp | 5 |
| attention_size | 25 |
| future_point | 25 |

We conducted the forecasting experiment on the Stock 600000 again, the result is shown as follows for the forecasting month:

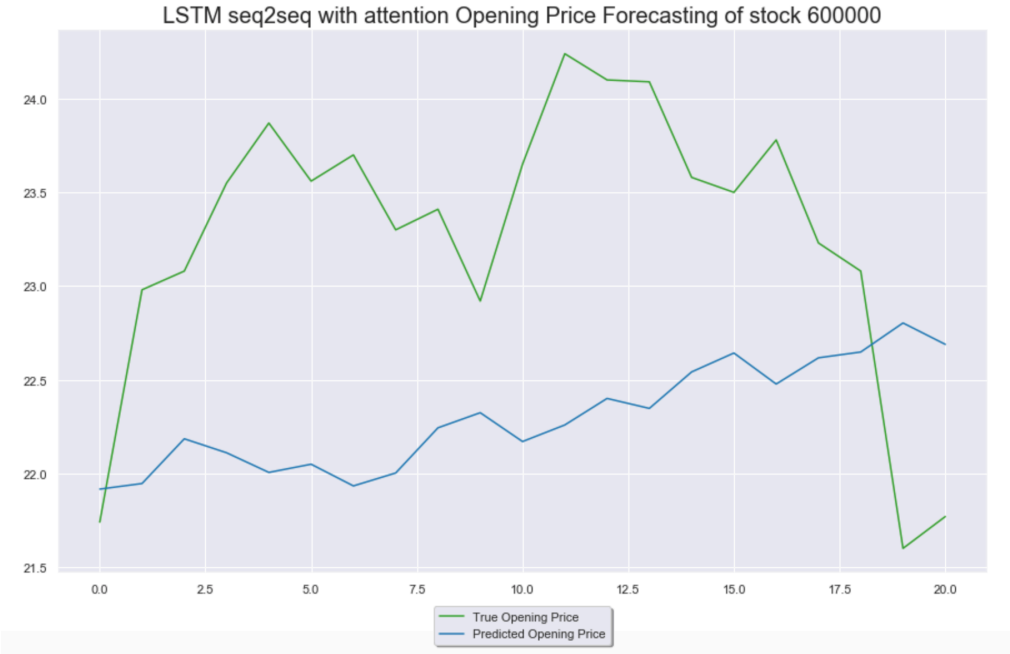


Figure 3. Forecasting result comparison for Stock 600000 by daily-based forecasting

The RMSE of this forecasting is 1.54124, which is much lower than the original minute-based model.

Table 3. RMSE comparison between two forecasting approaches for Stock 600000

| Forecasting Method | RMSE |
|---|---|
| Minute-based forecasting | 2.22748 |
| Daily-based forecasting | 1.54124 |

Although the trend of the forecasting price still differs from the true one, our model now gives a relative precious result. We conducted this procedure for some other stocks, and it turns out that this could work well on some stocks in one-month price trends, like the following forecasting result for Stock 600004:

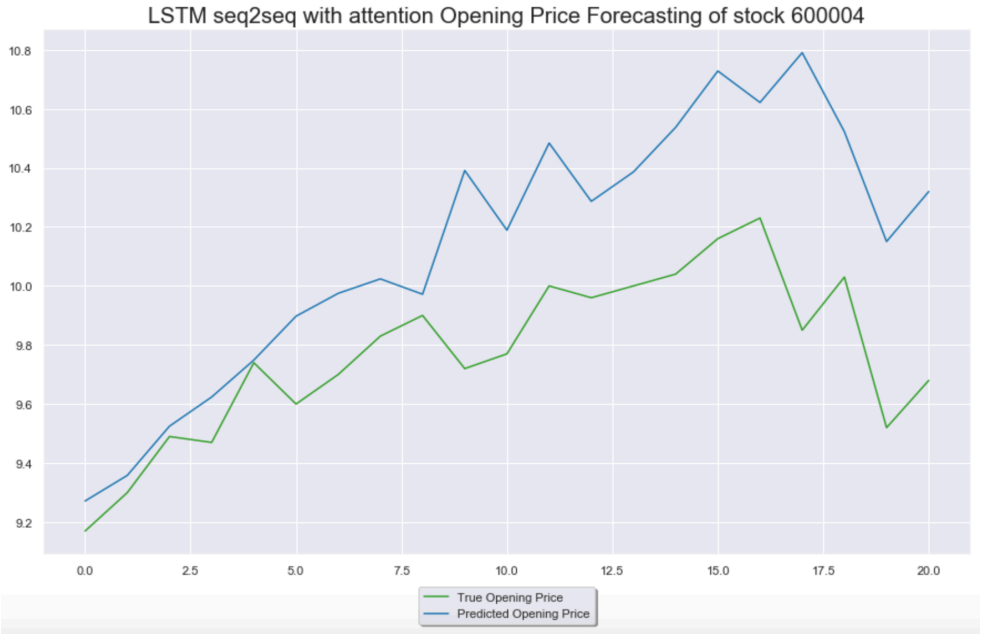


Figure 4. Forecasting result comparison for Stock 600004 by daily-based forecasting

Therefore, our task 1 finishes here with the finding that the minute-based forecasting is much harder than daily-based forecasting, especially in the forecasting precision in price data. Therefore, although the LSTM model with refinements is capable of time-series forecasting, its performance is not guaranteed for the time-series data with strong stochastic feature, like the stock price data in this case. The increase in data amount that is used for training doesn't guarantee the forecasting performance of the model.

As an inspiration and guidance for our task2 in designing investment portfolio, we ran our forecasting model over all stocks in the market and picked the one with highest increase ratio in the forecasted December 2009. This stock with highest forecasted increase here is 600359, which could serve as the baseline of our task2 investment.

## Task 2 Portfolio Strategy

To find a suitable portfolio strategy making the stock return as much as possible, we firstly analyzed our local stock data from 2009-11-01 09:30:00 to 2009-11-31 15:00:00 which contains 393 stocks and their prices in 5061 minutes, trying to figure out some rules that could simplify the problem. Due to the randomness of stock price, it is diffcult to find some long-term rules within a month, but it indeed follows a few short-term local properties. For example, generally speaking many stocks go up or down in a sequential time point. Therefore, we applied a simple local portfolio strategy which is to buy the rising stocks in the current minute and sell them all in the next minute if those stocks go up in the past few minutes.

1. **Algorithm 1: Invest all rising stocks in the market**

Based on the above strategy, we initially designed Algorithm 1 on our local stock data and the key ideas are shown below.

**Algorithm 1: Invest all rising stocks in the market**

**Do**

Define stock returns between 3 sequential time points $del_{21} = t_2 - t_1, del_{32} = t_3 - t_2$.

($t_1$, $t_2$, $t_3$ are opening prices of all stocks)

Find all the stocks whose $del_{21} > 0$ & $del_{32} > 0$ and view them as rising stocks.

Calculate percentage increase of rising stocks $p = (t_3' - t_1')/t_1'$.

($t_1'$, $t_3'$ are opening prices of all rising stocks)

Calculate money distribution to rising stocks $im = money * p/sum(p)$

Buy all the rising stocks in current minute and sell them in next minute.

**Until last min**

To get an initial idea about if Algorithm 1 works ,we made a test on local test stock data. The results are shown below with regard to the different selling time.

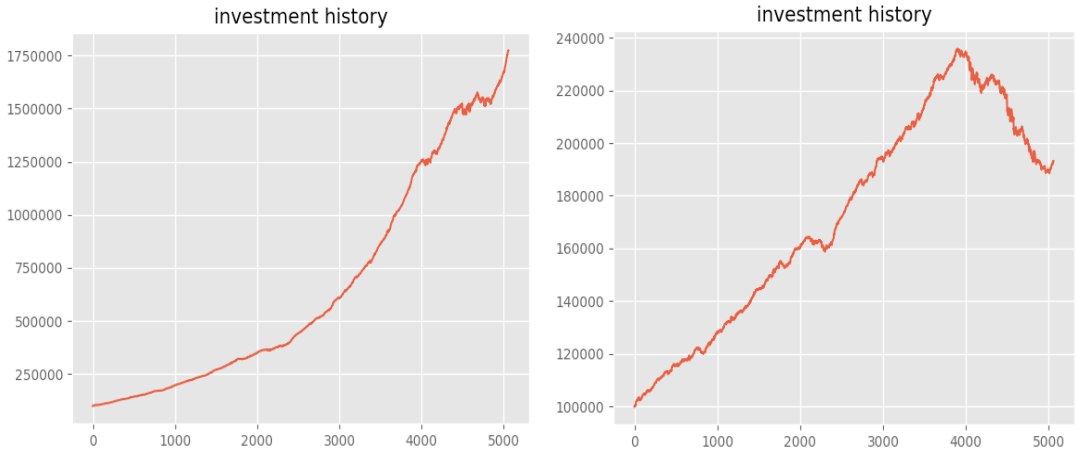


Figure 5. Final money based on current minute's transaction and next minute's transaction (stock returns between 3 sequential time points)

Considering the transaction based on current minute's money, we achieved the final money at 1773782.25 yuan. And if the transaction is based on next minute's money, the final money will be 193175.72 yuan. From the above result, we can draw a conclusion that Algorithm 1 works well on the test data. And also we can see that the selling time is very important because the final money varied a lot in terms of the different strategy.

In Algorithm 1, how to determine whether a stock is increasing is a key point. Originally we defined stock returns between 3 sequential time points, but we also tested the situations that include 2 sequential time points or 4 sequential time points. And the final result is quite different:

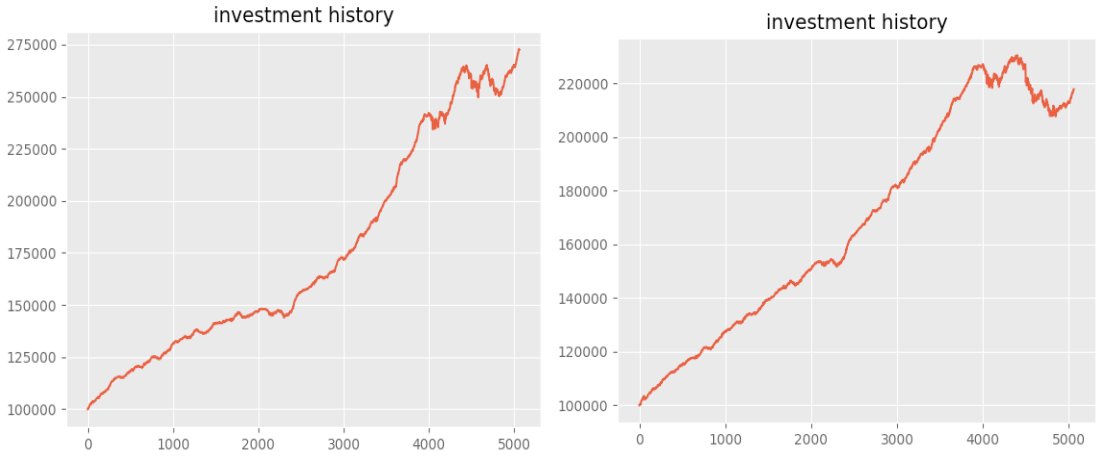


Figure 6. Final money based on current minute's transaction and next minute's transaction (stock returns between 2 sequential time points)

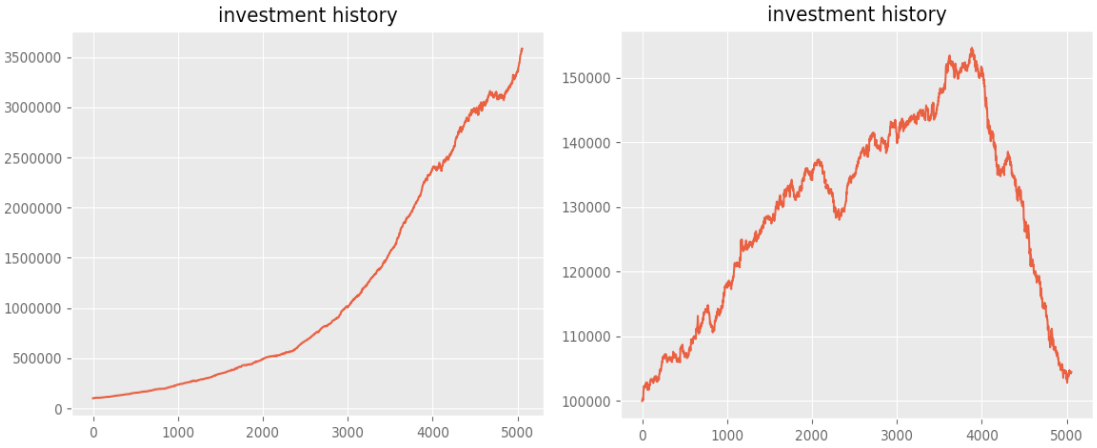


Figure 7. Final money based on current minute's transaction and next minute's transaction (stock returns between 4 sequential time points)

If we defined stock returns between 2 sequential time points, then the final money based on
current minute's and next minute's transaction will be around 275 thousand yuan and 220 thousand yuan. If it is 4 sequential time points, then the corresponding final money will be around 3.5 million yuan and 105 thousand yuan. Considering the robustness of this algorithm, we still choose to apply the original Algorithm 1 to API test.

In API test, the details of test dataset are:

- Timestamp: 2009-12-01 09:30:00 to 2009-12-31 15:00:00
- Timestamp number: 5543
- Stock number: 506
- Stock without NAN: 331
- Stock with NAN: 175

Firstly, we obtained a baseline result of 137061.72 yuan by using a simplified Algorithm 1, and the ranking is shown below:

| Team id | Best result(Final money) | Last result(Final money) |
|---|---|---|
| 5 | 1074188.27 | 1074188.27 |
| 7 | 378324.17 | 378324.17 |
| 9 | 176173.75 | 176173.75 |
| 12 | 157030.02 | 144721.15999999997 |
| 4 | 137061.72 | 137061.72 |
| 8 | 123371.40999999989 | 123371.40999999989 |
| 2 | 120800.0 | 120800.0 |
| 13 | 115492.15 | 115490.29999999994 |
| 10 | 104424.72 | 104424.72 |
| 3 | 103295.44 | 59858.53000000005 |
| 0 | 100000.0 | 100000.0 |
| 1 | 100000.0 | 21854.890000000043 |
| 6 | 100000.0 | 100000.0 |
| 11 | 100000.0 | 100000.0 |

Figure 8. API test ranking

After that, we applied the formal Algorithm 1 to API test. Unfortunately, we didn't get the final result in this test because the bug and delay of online server. However, the final money of the third day was returned and there is the screenshot of it. We can see that the final money reach to about 140 thousand yuan only within two days. Therefore, it is estimated that the final money of this month could come to around 700 thousand yuan with regard to current increasing speed.

```
2009-12-03 10:24:00
140381.5699999988
529.3599999989038
2009-12-03 10:25:00
140631.39999999892
438.71999999890886
2009-12-03 10:26:00
140712.56999999893
353.14999999892643
```

Figure 9. The screenshot of API test feedback

**2. Algorithm 2: Invest top 10 rising stocks**

Furthermore, we also researched some other portfolio algorithms. In Algorithm 2, we invest money equally to top 10 rising stocks.

**Algorithm 2: Invest top 10 rising stocks**

**Do**

Define stock returns between 3 sequential time points $del_{21} = t_2 - t_1, del_{32} = t_3 - t_2$.

($t_1$, $t_2$, $t_3$ are opening prices of all stocks)

Find top 10 stocks whose $del_{21} > 0$ & $del_{32} > 0$ and view them as rising stocks.

Distribute money equally to top 10 rising stocks.

Buy all the rising stocks in current minute and sell them in next minute.

**Until last min**

**3. Algorithm 3: Invest top 10 rising stocks with risk**

In Algorithm 3, we invest money equally to top 10 rising stocks with risk which is defined as the variance of stock price in the past minutes. The motivation of introducing risk is to find those stocks which increase steadily rather than those

fluctuating in high volatility.

**Algorithm 3: Invest 10 rising stocks with risk**

**Do**

Define stock returns between 3 sequential time points $del_{21} = t_2 - t_1, del_{32} = t_3 - t_2$.

($t_1$, $t_2$, $t_3$ are opening prices of all stocks)

Take the variance of price as the investment risk.

Find **10 rising stocks with lowest risk** whose $del_{21} > 0$ & $del_{32} > 0$.

Distribute money equally to the above 10 rising stocks

Buy all the rising stocks in current minute and sell them in next minute.

**Until last min**

In conclusion, from the different application of the algorithms, we can see that for Algorithm 2, we just chose the stock rising within three time points and distributed our money according to the rising speed so that we could have a great result. After that we defined the risk, if the rising speed was too high from the average, we would have a riskier investment. Then it limited our investment so the profit was obviously less than Algorithm 2.

## Reference


[1] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, TU Munich, 1991.

[2] Y Bengio, P Simard, P Frasconi. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks and Learning Systems, 5(2), 157-166, 1994.

[3] Hochreiter S, Schmidhuber J. Long Short-term memory. Neural Computation, 1997,9(8):1735-1780.

[4] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio. On the difficulty of training recurrent neural networks. ICML, 2003.

[5] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. ArXiv, 2014.