Reinforcement Learning for Portfolio Management

Youran Zhang
Student ID: 20460701
yzhanggm@connect.ust.hk

Chengming Wang
Student ID: 20450392
cwangbq@connect.ust.hk

Lingxiao Zhang
Student ID: 20475043
lzhangbr@connect.ust.hk

Yilan Yan
Student ID: 20475213
yyanao@connect.ust.hk

ABSTRACT

This project is about the stock investment with reinforcement learning on task 1 and task 2. It aims for the practice and improvements on the performance of using machine learning algorithms on the stock market.The project is separated into three tasks. Given several training datasets, we are going to implement each task using Python. Additional test data are also provided for the purpose of evaluation on our training model. In general, this project gives us an opportunity to gain some practical experience on building machine learning models and techniques on stock investment.

Keywords

Portfolio Management, Reinforcement Learning, Deep Learning, Quantitative Finance, Algorithm Trading.

1. INTRODUCTION

1.1 BACKGROUND

Considering the fact that the computer can be much faster for large calculations than humans. We would like to apply machine learning models on the stock market to capture data and perform calculations of technical analysis for all stocks itself, which is much more efficient than humans. Moreover, we need to improve the performance of using machine learning models in order to achieve the highest daily returns possible in real-time. In general, this project gives us an opportunity to gain some practical experience on building machine learning models and techniques on stock investment.

1.2 PROBLEM DEFINITION

Snowball is a social media that provides a platform on sharing the information of the investment philosophies, stock analysis and trading strategies. In this project, we are given several datasets from the Snowball and our goal is to implement machine learning algorithms, to design and improve our trading strategies based on these datasets. In general, our project mainly focuses on two parts, which are choosing stocks and portfolios. From the given five datasets, we need to come up with strategies on choosing stocks and portfolios.

1.3 RELATED WORK

Similarity search is a general form of searching in large space of objects based on similarity scores between any pair of objects. It is mainly used in database system to handle complex data like images and videos. In task 1, the data is stock quote data and the objective is to find similar trends in stocks. This kind of data can be viewed either as images which are snapshots of stocks at each time step, or as time series (vectors).

For image similarity search, in [1] it introduces siamese network to compare the similarity of two patches of images. This network takes two patches as inputs and they share the same set of architectures and weights. On the top two branches will merge and be fed into a series of fully connected layers. Basically, it works as a feature descriptor. But the time and memory it takes to go through the whole database may be huge. Thus it's not practical for this task.

For time series similarity search, in [2] it explains in full detail how similarity search in both static and streaming time series works. Not only the basic structure, but also advanced methods like R*-Tree. However, it does not include practical applications of similarity search methods to real life data. Whether it will perform well in real life remains to be seen.

In article [3], the paper presents how Q-learning works in daily trading, which mainly recommends financial decisions in advance. Experiments show that decision made in Korean market is efficient. Moreover, the result should be test in other methods in terms of risk and profit.

2. DESIGN AND IMPLEMENTATION

2.1 OVERVIEW OF SOLUTIONS

As described in the previous part, we have two choices to implement similarity search, either based on snapshots or time series. Consider the computation power of our laptops, we chose time series based method. To further reduce time and memory cost, we represent time series as binary arrays. We built a historical database to save up or down trends of each stock as binary arrays. Every day new data comes in and the database will be updated. With a sliding window, we search the database to compute and compare similarities. For the most similar windows, we compute their uprising probability and assign weights accordingly.

For strategies of choosing portfolios, we first decide to find patterns of trading strategies of investors from these datasets filtering out portfolios we would like to explore and learn. Then we build a set of reinforcement learning models with different targeted portfolios. Finally, our system will choose several models among all based on the feedback of the reward function.

2.2 INSIGHT OF SOLUTIONS

The objective of the first part is to choose stocks by assigning weights to them. We only took usage of the stock quote dataset. The feature we chose is the daily close price of each stock. It was further transformed into binary arrays. 1 represents uptrend and 0 represents downtrend. In this way we could use bit arrays to efficiently store these long binary arrays, thus saved large amount of memory. The model is basically a sliding window of M days. For each day and each stock, the window will slide through the

Source Code: https://github.com/cheersyouran/RL

database and compute similarity scores for each window. The similarity score is defined as the count of identical elements.

For portfolio management in the second part, we use reinforcement learning to allocate daily weights of portfolios. When it comes to choosing which reinforcement learning algorithm to implement, we have considered several algorithms such as Q-learning, Deep QNet, policy gradient and even some parallel method. For our project, we've tried three algorithms into practice, which are DDPG, Deep QNet and Q-learning. Finally, we decide to use Q-learning model, which we would like to explain the reasons below.

Since DDPG and DQN both use neural networks as approximate functions to simulate the Q-table, these two models work well when there is a great number of input features, such as images or the case of AlphaGo. However, considering the fact that we need to generate features by ourselves, which means the number of features is going to be limited. Moreover, using these two models costs too much time on improving the performance. For example, if we choose to use the DQN model and we add a new feature as part of our input, ending up with a poor performance. It's hard to tell either the neural network model or input features, which can be treated as two variables, causes this bad result. In order to seek for the influence of performance on a single variable, we would like to evaluate the features we added on our model. To evaluate a feature on a given model, we can use Q-learning to check the Q-table anytime. We can also write down and calculate the rewards distribution for evaluation if it's necessary.

2.3 IMPLEMENTATION DETAILS

2.3.1 Task1:Strock Trading

In the implementation phase, task 1 is split into three parts: building historical database, similarity search and weights assignment. In the first part, we represented up and down trend as 1 and 0, thus creating long binary arrays for each stock. To save memory, binary arrays were transformed into bit arrays. Stocks were grouped in industry so that we could easily search for stocks in the same industry during test time. So the historical database is just a large dictionary, keys are industries and elements are bit arrays of each stock which belong to this industry. In test phase, which is the second part, we built a sliding window of size M for each stock. This window will slide through bit arrays of stocks in the same industry as the current test stock. Inside each window, similarity score, defined as the count of identical elements will be computed. Windows of stocks which have the top scores will be chosen. In part three, we observe how these stock go afterwards and compute the uprising probability. Weights are assigned to these stocks according to the uprising probability. This will give us the weights of stocks in that test day. When a new test day came, we first update the historical database, then repeat the second and third part until the last day of test data.

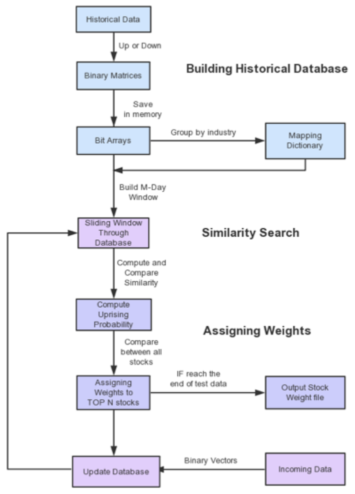


Figure 1. Flow Chart of Stock Trading

2.3.2 Task2: Portfolio Management

In this task, we assumed that each portfolio only focuses on several specific industries. We think our assumption is reasonable because most traders are not versatile among all industries and each trader has particular industries that are adept at. Based on this assumption, we generate a model for each industry, and divide a portfolio into limited industries based on their historical information.

In Reinforcement Learning, here are lots of Algorithms to generate a model, like Q-learning, DQN, Actor-Critic or some parallel methods. In our experiment, we choose to use Q-learning model for the following reasons:

- ➢ Like DDPG and DQN, they use a NN as an approximate function to simulate Q Table, it does work well when input features' number is very large, such as images, like what AlphaGo does. However, in our case, features must be generated by ourselves, which means features' number is limited. So If we use DQN, when we add a new feature to describe portfolio and get a bad result, we even don't know either the Neural network or features cause the bad result. To solve it we have to pay more attention to adjust a well performant Neural network, it's unworthy.
- ➢ When we use Q-learning, we can check the q-table at anytime, we can also write down and calculate the rewards distribution, which can help us to determine the influence of a new added feature.

There are mainly four components for implementing a reinforcement algorithm, which are environment, action space, status space and rewards:

1. Environment. given a date, environment can provide us daily stock info and portfolio info. Also, we can check the history info of stock and portfolio.
2. Action Space. We have to design a set of discrete actions since we are using Q-learning model. The dimension is N+1, where N means N-different-portfolios and the left 1 means cash. It's worth mentioning that N portfolio is chosen by

IR(Info Ratio) function, To generate a IR, the least time period of a portfolio must lager than 7. So when a new portfolio comes in, we have to observe with 7 days and then the portfolio has the chance to be chosen to N dimension.

3. Status Space. We have created a set of features, such as MACD, two days' average, KDJ, BOLL etc. For each model, we select a single feature as inputs everytime and then test the performance of this particular feature. If the feature chosen results in a very good performance, we will add it into our status space. Follow these steps, we've tried and tested all the features we've created.
4. Reward. We use NAV growth rate of the next day as reward. For the cash dimension, the reward will always be 0.

Moreover, for the formula of Q value iteration Q(s,a) = R +r maxQ(s',A), we set r as 0 since we ignore the transaction fees. In other words, everyday, our model will try to find the action that have the best income for tomorrow.

Finally, we need to combine all the models we've created. To achieve this goal, we can also use reinforcement learning model to assign weights for each model. We can also give a average weight to each model. In our experiment, we use the later one.

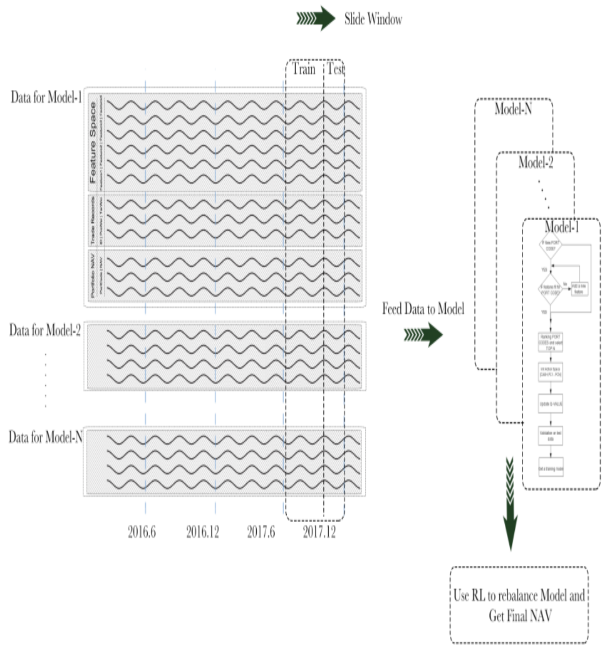


Figure 2. Flow Chart of Portfolio Management

3. EVALUATION

3.1 EXPERIMENTAL SETUP

For task 1, test data is from August 14th, 2017 to September 8th, 2017. We use a sliding window of size 10, which means we will look back for the last 10 days. For each day we chose 6 stocks from 200 stocks in total. The baseline is that suppose there's a man who is new to the stock market. The way he chooses stocks is by observing for the last three days which stocks were going up all the time. He chose ten of them randomly and gave them equal weights. He will do nothing for a month.

For task 2, we tested the performance using data from January 4^{t}o February 26th. Actually, we design a slide window to determine the data time period we do care about. If Test_Window = 10 and Train_Window = 30, that means we use past 30 days data as training data and use the past 10 days data as test data.

3.2 PERFORMANCE METRICS

The performance metrics for task1 and task2 are both NAV value.

3.2.1 Task 1: Stock Trading

For task 1, NAV value is computed as follow. Suppose we have n trading days and m stocks. For each day and each stock, we multiply the corresponding Change PCT of stock close price and weight. This will give us a vector for each day. By summing up values in the vector, we get the NAV change ratio. NAV value will start from 1 on the first test day and will change according to the change ratio computed for each incoming day

3.2.1 Task 2: Portfolio Management

For task 2, the computation is a little different. Suppose we have n trading days and m portfolios. For each day and each portfolio, we multiply the corresponding NAV change ratio from nav dataset and portfolio weight. This will give us a vector for each day. By summing up values in the vector, we get the weighted NAV change ratio. NAV value will start from 1 on the first test day and will change according to the weighted NAV change ratio computed for each incoming day.

Not only do we have NAV values as performance metrics. We also have winning probability, P&L ratio, annualized return, annualized volatility and sharpe ratio as additional performance metrics.

1. Winning probability is just the percentage of days or weeks or months where NAV value rises compared with last time period.
2. P&L ratio is the mean of NAV growth in winning time periods divided by the mean of NAV growth in losing time periods.
3. Annualized return is the annualized NAV value in the last time period.
4. Annualized volatility is the annualized standard deviation of NAV values.
5. Sharpe ratio is annualized return divided by annualized volatility.

Source Code: https://github.com/cheersyouran/RL

3.3 EXPERIMENTAL RESULTS

3.3.1 Task1: Stock Trading

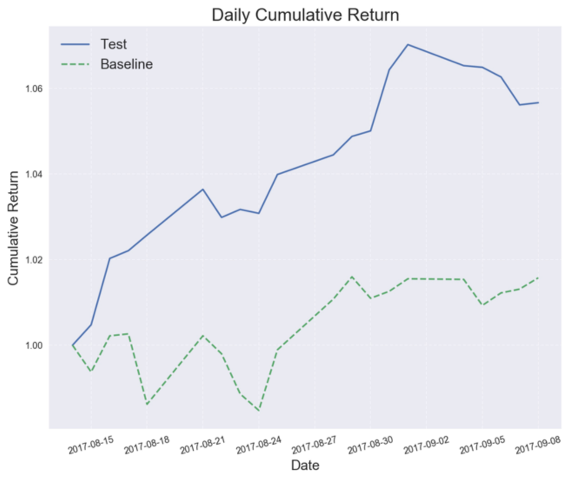
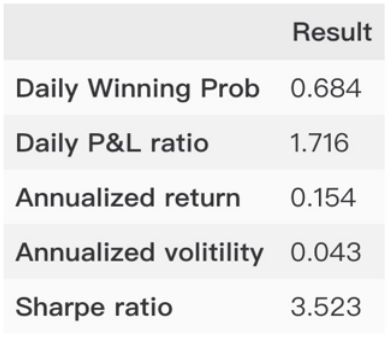


Figure 3. Daily Cumulative Return of Stock Trading



	Result
Daily Winning Prob	0.684
Daily P&L ratio	1.716
Annualized return	0.154
Annualized volitility	0.043
Sharpe ratio	3.523

Figure 4. Result Statistic of Stock Trading

3.3.2 Task2: Portfolio Management

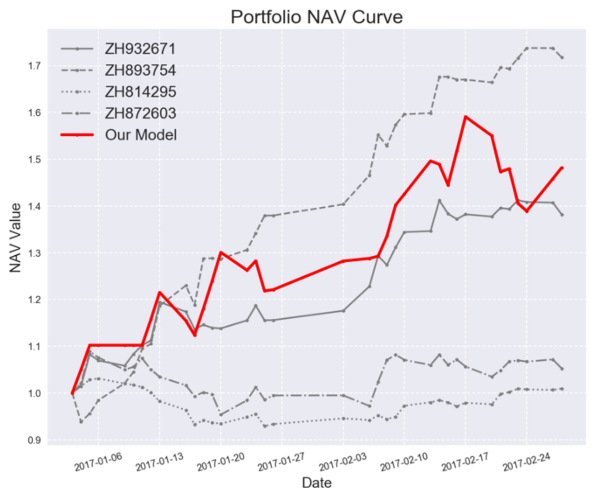
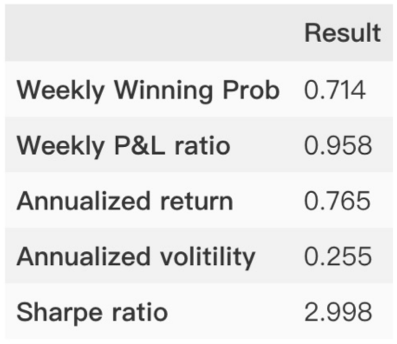


Figure 5. Daily Cumulative Return of Portfolio Management



	Result
Weekly Winning Prob	0.714
Weekly P&L ratio	0.958
Annualized return	0.765
Annualized volitility	0.255
Sharpe ratio	2.998

Figure 6. Result Statistics of Portfolio Management

3.4 ANALYSIS OF RESULTS

3.4.1 Task 1: Stock Trading

For task 1, at the beginning, the baseline is tumbling heavily, but it ends up well. Test performance which is based on similarity search outperforms the baseline. But here we see that the performance is not stable enough. In the first days of September, the NAV curve is going down. This unstableness is mainly due to the small size of historical data since similarity search really requires a huge size of historical data to have plenty source data to search from. Also, the chaos and complex nature of stock market really affects the performance. Representing stocks as binary arrays does save memory and running time, but it limits the amount of information we can have. Thus these binary arrays are far from representing the stocks.

3.4.2 Task2: Portfolio Management

For task2, our model works relatively well in this time period, yet we cannot handle the condition when the NAV sudden drop. Actually it's very dangerous since in real market, investor will be panic about such condition, if we cannot stop loss timely, it will cause a heavy result.

4. CONCLUSION AND FUTURE WORK

4.1 CONCLUSION

In conclusion， for task1, we used similarity search. In each trading day， we used similarity search to compute and compare similarities with a sliding window, then computed the uprising probablity of the most similar windows and assign weights. With the test data, the test result has much better performance than the baseline although a little unstableness because of the small size of historical data in the beginning.

For task 2, we used reinforcement learning to allocate weights automatically. we first decided to find patterns of trading strategies of traders, from these filtering out portfolios we would like to explore and learn. Then, Q-learning was used to train some models,which could be treated as a trader that target on a specific industry. Finally, we also used reinforcement learning to rebalance the model and get the final NAV. The test result in figure 3 performs better than the major single portfolios in the stock markets, yet we cannot handle the condition when the NAV sudden drop.

Source Code: https://github.com/cheersyouran/RL

4.2 FUTURE WORK

For task 1, because of the computation power of laptops, we use bit array. we need more dataset to make the similarity search work better. Moreover, because of large time of searching and memory costs of our database, we will use python generators to solve the memory issue.

Similarly, for task 2, Since we use Q-learning to train models, we need to compute Q-table and information ratio for each single day, which leads to large time costs. We do not consider the transaction fees in trading.

5. REFERENCES

[1] S. Zagoruyko and N. Komodakis, "Learning to compare image patches via convolutional neural networks," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015, pp. 4353-4361.

[2] Maria Kontaki, Apostolos N, Yannis, Similarity Search in Time Series Databases, Data Engineering Lab, Department of Aristotle University, 54124, Greece.

[3] Jae Won Lee, A Multiagent Approach to Q-Learning

for Daily Stock Trading, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, Vol 37, 2007

Source Code: https://github.com/cheersyouran/RL