Introduction to Artificial Intelligence Writing 4

YiFan Li
ZeYuan Yang

November 19, 2019

1 Introduction

On today's financial market, increasing number of financial indexes are being created. Take the stock market as an example. Up till now, there have been over 200 generic rules that can suggest a buying or selling or selling decision based on the historical stock price time series [6]. Some of the most prevailing ones include moving average, relative strength indicator, stochastic oscillator, etc. However, these rules have at least three flaws. Firstly, all of these rules seems somewhat arbitrary, and are highly unstable. They could perform well on some stocks, but might fail on others. Secondly, it's impossible for a human trader to integrate all these 200 rules in his mind and make a trading decision. Thirdly, most of the indexes have some parameters as output, for example the time window for moving average, the choice of what number should be used for these parameters are still empirical.

Both stock market practitioner and academic researchers have devoted huge effort to determine good interpreters that can help make profitable decisions in the financial market. Additionally, with the rise of quantitative trading, which means trading stocks by computer program instead of a human trader, the need for algorithmically select good financial indexes have become more important.

To be more specific, the problem setting can be summarized as follows:

- Given a historical stock price time series in the past n periods, $P_t = [p_{t-n}, p_{t-n+1}, \cdots, p_{t-1}]$, where t represents the present,
- and a set of k trading rules suggesting "buy", "sell" or "hold" action based on the time series $R = \{r_1, r_2, \cdots, r_k\}, \forall r_i(P_t) \rightarrow s_i \in \{\text{sell, buy, hold}\}$,
- design an agent A who can integrate all the rules to make a final decision $A(R(P_t)) \rightarrow d_t \in \{\text{sell, buy, hold}\}$,
- that can maximize the profit in the following m period, which can be calculated as $\sum_{t=0}^{m} p_t^{\text{sell}} - p_t^{\text{buy}}$,

$$p_t^{\text{sell}} = \begin{cases} p_t & \text{if} A(R(P_t)) = \text{sell} \wedge A \text{ currently holds the stock} \\ 0 & \text{otherwise} \end{cases},$$

$$p_t^{\text{buy}} = \begin{cases} p_t & \text{if} A(R(P_t)) = \text{buy} \wedge A \text{ currently not holds the stock} \\ 0 & \text{otherwise} \end{cases}$$

We are not considering the service charge because it could make it difficult to design an evaluation benchmark that can be calculated without conducting another complete search.

2 Approach

We are aiming to solve this problem with genetic algorithm. On one hand, the problem setting can be considered as a search problem, which tries to find the best combination of trading rules from all possibilities. The evaluation function can also be very easily defined. On the other hand, there are already some previous studies using the same approach [1] [4]. We can based our project on their findings and dig deeper.

3 Implementation Plan

For the experiment, we will have to gather data and implement the algorithm.

3.1 Data

We can find the historical stock data for free on Alpha Vantage. It provides a lot of APIs and we can get the data via HTTP requests. The data pipeline, however, will have to be built by ourselves. We will use Python to grab data and save it as csv files for further use.

We plan to gather 3 years' data and use the first two years as the train set, and the last one year as test set. With this architecture, our agent does not only find the optimal strategy for the history, but also have some generalization ability and therefore have the potential to be applied on the market in practice.

3.2 Algorithm

We will use Python to implement the genetic algorithm and whatever strategies to be tested. We have found a Python package, gaft, that serve as a framework for running genetic algorithm.

What needs to be built from scratch includes all the empirical trading rules, and possibly some variant strategies of the genetic algorithm that are not included in gaft.

4 Analysis Plan

4.1 Performance Evaluation

The benchmark of our evaluation is an agent who can "predict the future". It knows the whole time series and make decision based on it, $A'(P_m, t) \rightarrow d'_t$. It can be implemented with the following trivial algorithm.

Algorithm 1: Benchmark Agent

```
Data: prices
Data: today
Result: decision
if prices[today + 1] > prices[today] then
    return buy;
else if prices[today + 1] < prices[today] then
    return sell;
else
    return hold;
end
```

The evaluation function used in the genetic algorithm will be $\frac{\text{Profit(genetic agent)}}{\text{Profit(benchmark agent)}}$ and the goal is to maximize it.

We may also choose some trading rules and build agents who make decisions based solely on each one of them to see how much improvement is brought by the genetic algorithm.

The runtime and memory usage will probably be measured, but generally they can be controlled by setting the maximum round of iteration and population. Therefore we will not emphasis too much on it.

4.2 Genetic Algorithms

During our research, we also found several variants for genetic algorithms which we will implement in our project. Three different variants are illustrated here, concerning three different parts of genetic algorithm.

The first variant adapts the format of the chromosome. Instead of traditional bit representations of integers in the chromosome, this variant may use other data types, for instance, arrays of real numbers. In some previous researches with the similar methodology as us, like [1], the bit representation is used, meaning that different trading rules can only have "selected" or "not selected", and those selected must have the same weight. With real number represented chromosome, we can do weighted average for the agent's final choice, which could probably improve the result.

Adaptive genetic algorithm is a common variant of genetic algorithm. It has already be implemented in different fields and makes a good performance.[2] [8] Instead of having fixed possibility of crossover or mutation, adaptive genetic algorithm will obtain distinct possibilities of crossover and mutation regarding to the population of each generation. For instance, Mahmoodabadi[7] designed the quasi sliding surface-mutation, defined as below:

$$\overrightarrow{X_i}(t+1) = \overrightarrow{X_i}(t) + (\overrightarrow{a} \times 10^{(\frac{-1}{\sqrt{|s|}})}), \text{where } s = 2 \times f(\overrightarrow{X_i}(t)) - f(\overrightarrow{X_i}(t-1))$$

$\overrightarrow{X_i}(t)$ represents a random chromosome in iteration t. $\overrightarrow{a} \in [0,1]^D$ is a random vector. $f(\overrightarrow{X_i}(t))$ represents the fitness value of $\overrightarrow{X_i}(t)$. The evolution is supposed to be optimized by adaptive genetic algorithm.

Elitism can be applied in the genetic algorithm as well. Not only organisms from the child generation will be kept, the best organism(s) from the parent generation will also be propagated into the next generation. Elitism can reduce genetic drift while also increase the selection pressure. [3]

Besides three mentioned variants, there are also many other variants of genetic algorithm, for example, parallel genetic algorithm. [5] We only plan to include two or three variants in our project.

4.3 Experiment Phases

The experiment will be conducted in two stages. In the first stage, the test dataset will not be used. We will only try to find the hyper parameters that leads to the best performance on the train dataset. In the second stage, we will run the genetic algorithm on the train dataset as before, but the trained agent will be applied on the test dataset to measure its final performance. The idea is borrowed from machine learning, that only optimizing on the train dataset could cause overfitting. The test dataset, which is invisible during the training state, can test the agent's generalization ability. We will also try to find the best hyper parameters for the second stage and compare it with those in the first stage, which hopefully could inspire us about how the choice of hyper parameters and strategies can serve as overfitting prevention.

5 Timeline

We will start working on our project right now. The whole project can be divided into three phases:

1. Data ETL and preprocessing
2. Algorithm implementation and experiment
3. Analysis and write-up

Each of these three parts requires much effort. We plan to finish the project in about three weeks.

We plan to finish the first part in this week, by Nov 24th. As mentioned in the Implementation Plan part, we will grab the data from Alpha Vantage. The result of API calls will be in JSON document, and our usable format is CSV (which can be easily loaded as a Python Pandas DataFrame). After that, the train dataset and test dataset will be splitted. The entire data preparation is scheduled to be done in five days.

After we have the clean data, we will implement the algorithms and run the program. As we mentioned before, we will choose the trading rules, build agents and then do the experiment. We will also try to apply the mentioned variants of genetic algorithm. Therefore, we are hoping to finish this part in the Thanksgiving break, which is due by Dec 1st.

Finally, we will analyze the result we get in the second part and write the final report. One week is planned for doing this task. To get insightful conclusions for the project, we plan to dig into the results first. Moreover, we will discuss what we find and revise our conclusions. How to present our conclusion is also a crucial portion of this part so visualizations may be required as well. If everything goes well, we will finish the draft of our project by Dec 8th.

In the last week of this semester, we will focus on refining our conclusion and revising our documentation. The last job of this project is to go through the whole essay and the algorithm to find where we can improve. The project is scheduled to be submitted by Dec 14th, so that we can have 4 days for emergency.

References

[1] F. A. Badawy, H. Y. Abdelazim, and M. G. Darwish. Genetic algorithms for predicting the egyptian stock market. In *2005 International Conference on Information and Communication Technology*, pages 109–122, Dec 2005.

[2] F. T. Chan, S. Chung, and P. Chan. An adaptive genetic algorithm with dominated genes for distributed scheduling problems. *Expert Systems with Applications*, 29(2):364 – 371, 2005.

[3] Chang Wook Ahn and R. S. Ramakrishna. Elitism-based compact genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 7(4):367–385, Aug 2003.

[4] D. de la Fuente, A. Garrido, J. Laviada, and A. Gómez. Genetic algorithms to optimise the time to make stock market investment. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1857–1858. ACM, 2006.

[5] D. E. Goldberg. Sizing populations for serial and parallel genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 70–79, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[6] J. Korczak and P. Roger. Stock timing using genetic algorithms. *Applied Stochastic Models in Business and Industry*, 18(2):121–134, 2002.

[7] M. Mahmoodabadi and A. Nemati. A novel adaptive genetic algorithm for global optimization of mathematical test functions and real-world problems. *Engineering Science and Technology, an International Journal*, 19(4):2002 – 2021, 2016.

[8] S. Q. Wu, M. Ji, C. Z. Wang, M. C. Nguyen, X. Zhao, K. Umemoto, R. M. Wentzcovitch, and K. M. Ho. An adaptive genetic algorithm for crystal structure prediction. *Journal of Physics: Condensed Matter*, 26(3):035402, dec 2013.