Introducing MumeParrot

Let me introduce my friend MumeParrot (Mume + Parrot). It has been my proud stock trading bot for three years. So, I'm going to talk about how it works and why I trust it.

Shortly speaking, MumeParrot trades stock every day based on a quant algorithm (regardless of my sentiments), and by doing so, I believe it earns the money. So, which quant algorithm does it use?

The idea of MumeParrot started from a famous quant algorithm, Infinite Buying [1], from South Korea. The core idea of Infinite Buying is very simple: 1) given a seed of amount X$, divide it by 40 (i.e., 1/40 X$) and buy a certain 3x leverage stock (e.g., TQQQ) every day, 2) once the stock price goes over 10% of average unit price, sell all stocks, 3) repeat step 1 and 2 forever. Infinite Buying [1] states that we can steadily earn money by this strategy.

Thus, the assumption of Infinite Buying is that "the price fluctuation of 3x leverage stock always hits 10% of average unit price within 40 days". However, is this assumption really true? To answer this question, we evaluated the strategy. We started the step 1 on every single day in the history (of TQQQ), and counted the

cases where the stocks are sold in 40 days (i.e., price touching 10%). Surprisingly, the ratio of success was not that high as you can see in Figure 1. Only 68% of trials succeeded while 2066 (out of 6500) trials fail.

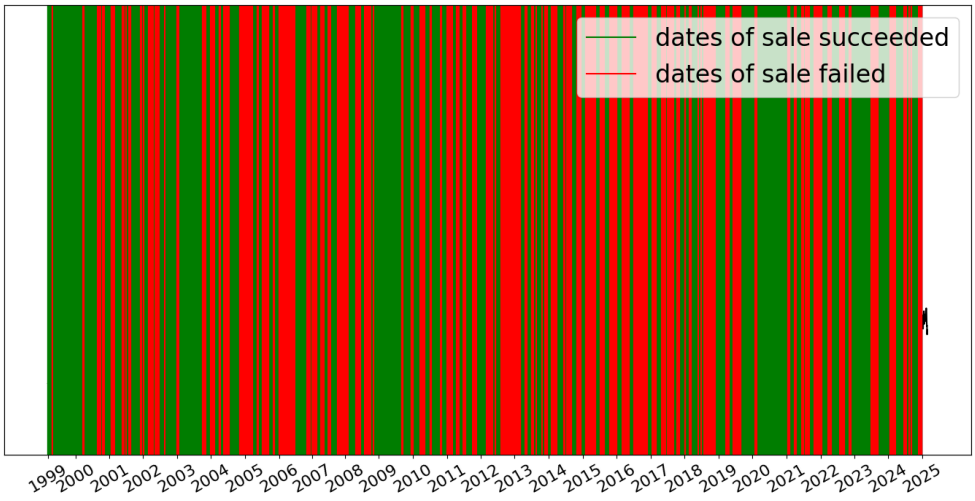

Fig 1. Sold and failed dates when the sell threshold is 10%. Only 68% of total trials succeed. For the period from 1999 to 2010, we reconstructed TQQQ history based on QQQ history.

Then, how can we reduce these failures? The easiest way would be to lower the sell threshold (i.e., 10%) to, for example, 1%. As you can see in Figure 2, the success rate increases to almost 99% by doing so. However, it actually loses the overall yield (as it sells the stocks at a lower price). The average return of rate (RoR) of the trials decreases about 70%—-i.e., from 9.5% per year to 2.6% per year.

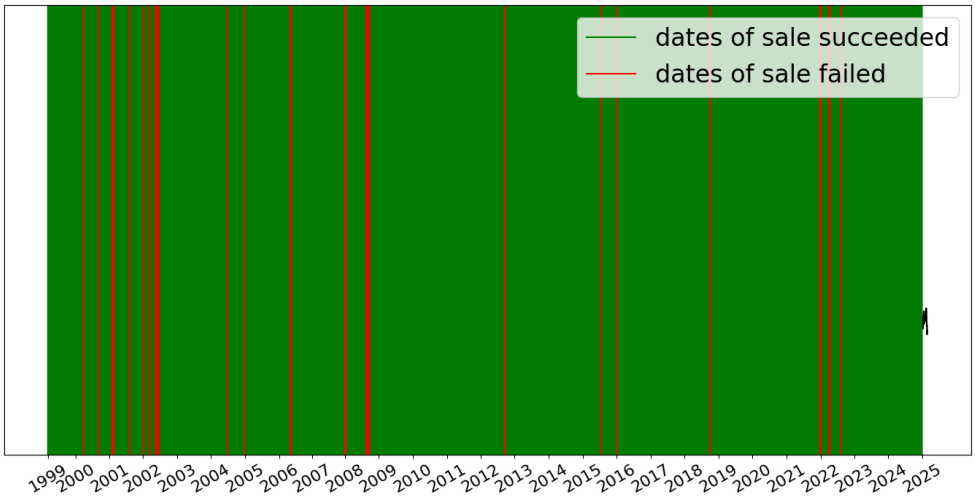

Fig 2. Sold and failed dates when the sell threshold is 1%. 99% of trials succeed.

So, as you may notice, it is the trade-off between the fail rate and the average RoR. The lower the fail rate, the lower the average RoR, as all the real-world problem does. So, the goal comes to our mind: let's find the best strategy that minimizes the fail rate while maximizing the average RoR.

Interestingly, what we have discussed is actually modeling the market—i.e., building a function $f_{TQQQ}(s) = (r_f, r_r)$, where s is the sell threshold, r_f is the fail rate, and r_r is the average RoR. The function f_{TQQQ} that models the market will be determined by the price history of TQQQ. Given this function, we can define a metric (i.e., $(1 - \lambda * r_f) * r_r$ that we want to maximize—λ determines the weight between fail rate and RoR. The best s that maximizes the metric would be the sell threshold that we want to choose (as it minimizes the fail rate while maximizing the average RoR). And also, we can add more parameters, such as, an RSI threshold—-i.e., buying the stock only when the current RSI is lower than the threshold.

Hola! We converted a complex financial problem into an optimization problem. Now, what we have to do is to only find the best set of parameters that maximizes

our metric (i.e., minimizing the fail rate while maximizing the average RoR). Taking our previous example, using sell threshold and RSI threshold as parameters, the hyper-plane of our metric is determined as shown in Fig 3. Thus, we can confirm the sell threshold 6% with the RSI threshold 100 is the best. In real settings, we employ a few more parameters (e.g., responding to market volatility).

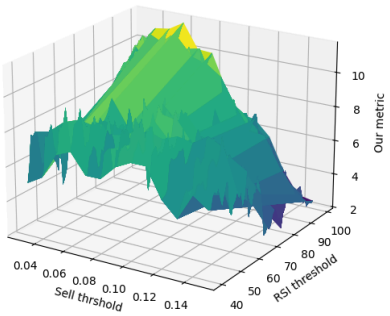

Fig 3. Hyperplane of our metric depending on sell threshold and RSI threshold. The higher value means lower fail rate and higher average RoR.

MumeParrot applies a genetic algorithm [2] to this problem, as it is a complex problem, and after a few moment, we can get the best set of parameters! For the results of optimization and our backtest implementation please refer to the mumeparrot-backtest link below. So, the last thing is to build a bot that faithfully performs this strategy on behalf of us, and you can check our bot, MumeParrot, here:

- Android: https://play.google.com/store/apps/details?id=com.mumemume.mumeparrot&hl=en
- iOS: https://testflight.apple.com/join/wBtRGB72

That's all done! Once configured, MumeParrot trades stocks on behalf of you on your own phone without any cloud interaction (except sending trading requests to stock brokers).

However, MumeParrot still has a lot of **open** problems, and I believe we can find better solutions together. For those who may be curious, here is our back-testing facility. Feel free to try it!

- mumeparrot-backtest: https://github.com/MumeParrot/mumeparrot-backtest

I also enumerate some open problems I found below:

- Handling the failed trials

Unfortunately, there is always a possibility of MumeParrot failing to sell the stocks. What do we have to do if MumeParrot fails to sell the stocks in a given cycle? Just holding the stocks is a dumb approach as 3x leverage stocks have volatility drag [3]. However, liquidating all stocks at once would also lose a fortune even if the stock price may go up after that.

- Liquidating stocks earlier

If MumeParrot can predict whether a market is going to be bearish due to any external factors, it would be good to liquidate the stocks earlier to avoid market crush.

- Responding to long term fluctuations

The characteristics of price fluctuation may vary throughout the time, so MumeParrot should also be able to respond to such variation and find better strategy time to time. However, current MumeParrot uses a fixed strategy that is optimized over the entire period.

This is all. Thank you for reading! Please fill free to contact me (jwhur19@gmail.com) if you have any questions!

[1] https://blog.naver.com/mortley/222577958223

[2] https://en.wikipedia.org/wiki/Genetic_algorithm

[3] https://en.wikipedia.org/wiki/Volatility_tax