Building an Improved, Less Risky Fundamentals Trader

LOUIS SCHLESSINGER, Washington University in St. Louis

A risk-controlling fundamentals-based trading bot is created to minimize risk and maximize the expected profit, especially given higher number of technical traders. It has lower standard deviation and slightly lower average profits in a variety of environments.

CCS Concepts: • **Applied computing → Law, social and behavioral sciences → Economics**

1 INTRODUCTION

Using the framework implementation of Hanson's Market Maker [1], I aimed to create a trading bot that would maximize expected profit while minimizing risk. The framework is a continuous double auction with a market maker (a price maker) who is almost always willing to accept both buy and sell orders at stated (but changing) prices. I made a variant of a fundamentals trader, which traded in the direction of its belief and used the information set it had available to it. At each time step, a new piece of information (0 or 1) was given to each trading bot. This information was given by a Bernoulli trial with success probability p_i. I therefore took the distribution to be approximately binomial. Between jumps, the distribution is binomial because it is made up of successive Bernoulli trials with a constant success probability. If there are no jumps, the entire distribution is perfectly binomial with a success probability p_0. My trading bot either buys or sells some quantity of the security, or takes no action at each round. I hypothesized that my bot could safely profit from the noise to the market prices that the technical traders create.

2 TRADING MODEL

2.1 Probability Estimation and Agent Belief

The estimation $\hat{p}_i$ of the true probability p_i was determined assuming that the information is a random variable X distributed binomially between each jump. That is, $X \sim B(n, p_i)$.

$$\hat{p}_i = \frac{x}{n}$$

My bot' trader's belief was given by the average of the market belief and the normalized $\hat{p}_i$. This gives equal weight to each beause the fundamentals traders make the market belief more accurate while the technical traders make it less so. This combination allows for the belief to be adjusted with more fundaments traders, but still update its belief according to $\hat{p}_i$.

$$belief = \frac{100 \cdot \hat{p}_i + market\_belief}{2}$$

To account for the jumps, the bot uses a sliding window method with length $L = 20$. That is, it checks at each time step if the normalized sliding window average $\bar{s}$ passes a jump threshold value. If a jump is detected, the bot resets its information as it is no longer valid to trade on.

$$\bar{s} = \frac{100}{L} \sum_{j=i-L}^{i} x_j$$

The jump is centered at the true probability p_i and is distributed normally with mean $\mu = p_i$ and standard deviation $\sigma_{jump} = 0.2$. Therefore, if $\bar{s} < belief - 100 \cdot \sigma_{jump}$ or $\bar{s} > belief + 100 \cdot \sigma_{jump}$, my bot assumes that, with high probability, a jump occured.

2.2 Trading Periods

The period at which the trading bot executes a trade depends on whether the bot calculates a profitable action. If there is a one, the bot executes that action otherwise it does not take one. To determine if there is a profitable action, the bot calls a function that optimizes the buy or sell expected payoff and returns the quantity at which it is optimized. After this, the bot compares the expected payoff from buying or selling and checks that the quantity associated with the bigger payoff is positive to determine the action. The bot then executes the best action with optimized quantity. Typically, the trading bot executes a trade when $n \cdot \frac{belief}{100} > 5$ and $n\left(1 - \frac{belief}{100}\right) > 5$ because, as explained below, the distribution's confidence interval is sufficiently narrow. It does not trade on scarce information of the true distribution. This also means that immediately after a jump was detected, it likely waits to collect more information on the new p_i.

2.3 Trading Quantities

The trading quantity was determined by a function which optimizes the expected profit and the quantity associated with the optimized expected profit. To model the quantity of shares bought or sold, it was necessary to capture some notion of confidence, risk aversion, and expected profit. The cost function of the LMSR is given by $C(q_1, q_2) = b \cdot \ln(e^{\frac{q_1}{b}} + e^{\frac{q_2}{b}})$ [2] assuming q_1 outstanding shares of outcome 1 and q_2 outstanding shares of outcome 2. To calculate the cost of buying Q_1 shares, the formula $C(q_1 + Q_1, q_2) - C(q_1, q_2)$ is used. The formula to calculate the payout of selling Q_2 shares is $C(q_1, q_2 - Q_2) - C(q_1, q_2)$. Both of these functions are monotonic; there is no global maximum for the cost buying shares or a global maximum for the payout of selling shares. Therefore, I decided to use the risk model that calculated the quantity of shares using an inverse relationship between quantity and the wideness of the confidence interval. If the confidence interval is narrow, the estimate has less uncertainty. If the confidence interval is wide, the estimate is not known precisely. Assuming a significance level of $\alpha = 0.05$, the bot calculates the confidence interval at each time step using an asymptotic normal approximation to the binomial distribution.

$$CI_{p_i} = \hat{p} \pm z\sqrt{\frac{1}{n}\hat{p}(1-\hat{p})}$$

The following function was used to estimate a maximum quantity where c_{upper} is the upper confidence level and c_{lower} is the lower confidence level.

$c_{range} = c_{upper} - c_{lower}$

$$confidence(c_{range}) = \left\lfloor \max_{c \in (0,1)} (\{0\} \cup \frac{1}{c_{range}^2} - 1) \right\rfloor$$

$riskiness = \beta$

$\max\_quantity = \lfloor \beta \cdot confidence(c_{range}) \rfloor$

This gave the plot shown in Fig. 1, from which the maximum quantity the bot could sell or buy was chosen. I found empirically that $\beta = 1$ and using an inverse square relationship to be optimal when $R = 100$.

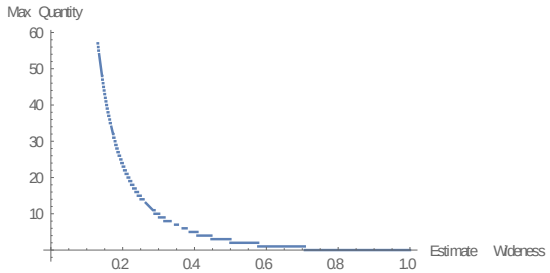

Fig. 1. Risk model for quantity of shares depicting an inverse square relationship with estimate wideness.

After finding the maximum quantity, the trading bot would decrement it until it found a profitable action. If the quantity reached zero, the bot would conclude that no action is profitable and wait for the next round. The following algorithm was used:

ALGORITHM 1: Optimal Quantity Algorithm

quantity ← *max_quantity*
while *quantity > 0*, ***do***
    if *action* ***is*** *'buy'* ***and*** *belief > buy_price(quantity)*
        return *quantity*
    else if *action* ***is*** *'sell'* ***and*** *belief < sell_price(quantity)*
        return *quantity*
    quantity ← *quantity* − 1
return *0*
end

2.4 Market Price Usage

The market price in general can lead to changes to any trader's demand curve. In this model, it was taken as half of an unweighted average. Assuming that my bot does not know how many technical traders or fundamentals traders, the market price had different levels of noise that was difficult, in general, to exploit. The market price has a higher correlation to p_i with more fundamentals traders.

3 RESULTS AND DISCUSSION

3.1 Results

I tested the simulation using $b = 250$ with 1000 samples by varying number of fundamentals traders N_f and the number of technical traders N_t, where $N_f > 0$ and $N_t > 1$. Those testing constraints were imposed due to the presence of the two types of technical traders. A summary of the results is found in Table 1. The results show that $\frac{N_f}{N_t}$ is approximately inversely proportional to profit. They also show that the standard deviation σ is constant at roughly $13{,}500$.

Table 1

N_f	N_t	N_f / N_t	Profit	σ	Maximum	Minimum
1	16	0.0625	9896.90	14103.88	106372.53	-103678.94
1	8	0.125	8894.78	13786.00	162738.92	-138074.20
1	4	0.25	7049.88	12499.80	137161.95	-103630.17

1	2	0.5	6214.21	12065.35	57848.98	-94612.34
2	2	1	4979.22	14912.79	362513.71	-82624.19
4	2	2	2516.91	13220.36	64288.54	-170102.87
8	2	4	2329.28	12092.13	116487.09	-152385.95
16	2	8	1611.81	12998.10	37505.28	-284011.39
32	2	16	1656.05	15091.29	56449.75	-286853.39

3.2 Performance Evaluation

This trader's strategy achieves a profit that is slightly less than the fundamentals trader and is on average positive, but has less variance as N_t increases. The naïve fundamentals traders are very susceptible to noise in the market because they do not detect jumps, which are expected to happen once per round because $E[jump] = \frac{1}{R} \cdot R = 1$. The technical traders only serve to add noise to the market, which this bot exploits. The strategy used allows for a confidence-based estimate of the true probability that is based on a common prior. The underlying model allows for a low standard deviation and controls for risk. It seems that with a higher N_f that market has prices closer to p_i, which means that there less opportunities to profit from an inaccurate market price. Fig. 2 shows a common run in which there is a single jump about halfway through. The market price follows the jump more quickly because N_f is sufficiently larger than N_t.

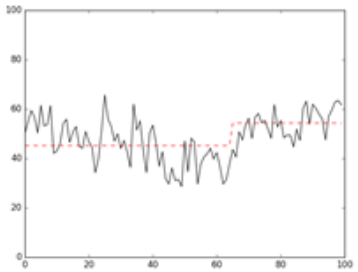

Fig. 2. Depiction of market prices in a typical simulation run (profit is 7331.90), $N_f = 5, N_t = 2$

If it were a real-world scenario and it was not possible to have 1000 runs, this trading bot would only be feasible if the bot is given a large sum of money relative to the security bid and ask price to account for the variance in expected payoff.

4 CONCLUSIONS

In summary, I created a robust trading bot that makes profit on average regardless of the number of fundamentals traders or technical traders. It contains a model for jump detection, a clearly calculated belief, and a model for risk. Although the average profit is not usually as much as the fundamentals trader, it has less variance. This sacrifice is one that is preferable because one does not usually know the proportion of N_f and N_t in a market.

REFERENCES

[1] Pennock, D. (2006, May 06). IMPLEMENTING HANSON'S MARKET MAKER. Retrieved May 07, 2017, from http://blog.oddhead.com/2006/10/30/implementing-hansons-market-maker/

[2] Step detection. (2017, February 13). Retrieved May 08, 2017, from https://en.wikipedia.org/wiki/Step_detection#Sliding_window