CSE 516: Final Project

Due: May 8, 2017, by email to Sanmay

Note: You may work in teams of 2 or 3 for this project. Post a note on Piazza if you are looking for teammates. The final deliverable is a (maximum) 4-page PDF file prepared using the acmsmall format available at `http://www.acm.org/publications/authors/submissions`. Please e-mail one copy of this file per team to Sanmay by 5 PM CDT on Monday, May 8th. This is a HARD deadline. You may pick one of the suggested projects below or work on your own idea with prior approval from Sanmay. Note that your work will be judged on the quality of the writeup you submit!

On another note, this shouldn't be a huge amount of work, especially in teams; the idea is more to provide you with an opportunity to explore topics and demonstrate your creative range in thinking about problems in multi-agent systems. You can think of it more as a team take-home final than a major project in terms of effort.

1 A trading bot

Write a trading strategy for prediction markets mediated by the logarithmic market scoring rule (LMSR) market maker described in class (more background here as well: `http://blog.oddhead.com/2006/10/30/implementing-hansons-market-maker/`).

Your trading algorithm has access to noisy information about the true probability that an event will occur. The security payoff depends on the probability of the event in the manner described below. The information you receive is in the form of biased coin tosses from the true distribution; one thing to be aware of is that the true distribution can change over time, in the manner described below. Your algorithm has to compete against some other trading algorithms, also described below. We are providing a software framework in Python that will allow you to focus on just writing up your trading strategy and experimenting with it.

1.1 Underlying model

There are up to R rounds $1 \ldots R$, each representing a distinct time period. There is an underlying true probability that the event will occur, which is unknown to everyone, but can change over time. At round i, let p_i be the true probability of the event occurring. During this round i, your trader will be told the outcome of one Bernoulli trial (biased coin toss) with success probability p_i. The trader can then buy and sell the security (there are no inventory or cash restrictions – your trader can take arbitrary positive and negative positions).

1.1.1 Time evolution of p_i

The true probability is a jump process. p_0 is chosen uniformly at random between 0 and 1. At the beginning of a round, p_i is calculated as follows: with probability $1/R$, $p_i \sim N(p_{i-1}, \sigma_{\text{jump}})$, and $p_i = p_{i-1}$ otherwise. That is, at any time, with probability $1/R$ the true probability jumps, and

when it does jump, the new true value is drawn at random from a normal distribution centered on the present true value. If $p_i \leq 0$ or $p_i \geq 1$, the security liquidates prematurely at 0 or 100 respectively. Otherwise, the value of the security is $100p_R$ after round R. Note that this is somewhat different from the payoff being 0 or 1 depending on whether the event actually happens or not (think about why it's OK to use this model instead of actually having the event occur with probability p_R and pay off 0 or 1 in a simulation setting like the one here).

1.1.2 Parameters

For this assignment, $R = 100$ and $\sigma_{\text{jump}} = 0.2$.

1.1.3 Competing algorithms

We have provided code that implements various other types of trading bots. Specifically, there is code for a simple *fundamentals trader*, which receives the same information your bot does and trades in the direction of its belief, which is based on the information it has available to it. There is also code for two different *technical traders*. These are algorithms that do not use information about the fundamental value but instead try to profit from market movements using certain heuristics. In effect, they add noise to the market prices. Details on how to incorporate the different types of traders are in the appropriate readme file – you can control the proportion of fundamentals vs technical traders in the environment, and one of the goals of the project is to analyze performance as you vary this proportion.

The presence of these other traders allows you to potentially glean more information from the market price than would be available to just your own bot. On the other hand, these traders are themselves competing to make money off the market maker, just as you are. Finally, the technical traders add noise to the signal in the market price, but perhaps provide more opportunity for profit!

1.2 Software framework

We have provided a software framework for you in Python. Please look at it carefully and read the `README` file. The framework handles many tasks, including the generation of data for your bot to use in making trading decisions, handling the market making side of things, and collecting and printing aggregate data from single / multiple simulations.

1.3 Team composition

Given the code base we provide, it will be easiest if one of your team members is conversant in Python.

1.4 Questions to think about

Here are some questions you may want to explore in your writeup, but also feel free to analyze questions other than these.

1. How does your trader estimate p_i at time i? What algorithm do you use to try and account for the probability of jumps?

2. At which periods does your trader choose to trade? How did you decide this?

3. How does your trader decide what quantities to trade?

4. How do you use the market price in order to improve either your estimate or your trading strategy based on the estimate?

5. Report the mean profit and standard deviation[1] of profits your trader achieves over at least 1000 simulation runs across a variety of different proportions of fundamentals vs. technical traders. Analyze these results in detail and explaining why you think they come out the way they do. (Strategies that make good average profit without having very high variance of profits are generally considered the best.)

6. Graphically depict the evolution of prices in one simulation run which you consider typical for your trader.

2 Matching papers to reviewers

Take a look at the Computer Science Conference Bidding Data available at `http://www.preflib.org/data/matching/csconf/`. In particular, you'll want to eventually work with the data for Conference 3 on this problem. Learn about the data and what the bids represent. Then, look at the paper (and code, if you wish) available here: `http://www.cis.upenn.edu/~cjtaylor/RESEARCH/projects/OptimalAssignment/OptimalAssignment.html`.

The key goal here is to take this idea for how to do paper assignment from the paper above, and apply it to the conference bidding data. There isn't necessarily an exact mapping between the exact forms of the two problems, but you can make reasonable assumptions to transform them. Following that, the main goal is to understand the qualities of allocations as a function of different parameters of the problem. Here are a few examples of things you can change about the input data in order to study questions about how preferences and constraints affect the qualities of allocations:

1. The number of papers each reviewer can be assigned at most, and the number of reviews that each paper needs at least.

2. The relative strengths of preferences in the different categories (yes/maybe/no-response) for each reviewer.

3. The range of preferences – for each reviewer, consider different ways of breaking up their preferences within each category and assigning different "points" to those.

One important question to think about is how you should measure allocation quality. Think about designing at least a couple of measures for the quality of the eventual allocation. While one of these (e.g. a total score) could be the input to the solver, you can also look at performance on the other metrics (e.g. what proportion of papers are assigned to those who did not express any interest in them, and what proportion of top choices do bidders get)?

[1] Controlling the standard deviation is essentially a way of controlling risk. You may be willing to sacrifice some expected profit in order to have less "downside risk" for your strategy. This is particularly important in the real world, where you don't get to run your strategy thousands of times if you run out of money!

3 Repeated congestion games

We saw congestion games, and, in particular, Braess' paradox, in class. The goal here is to try and understand whether agents playing the game *repeatedly* converge to equilibrium, and how different parameters may affect long-term outcomes. This can be a useful model of dynamics in road traffic networks, for example, where agents could be playing the same game repeatedly and have to decide on a path each time.

Develop an infrastructure where a fixed set of agents can play the versions of Braess' paradox with and without the "superhighway" on a repeated basis, and keep track of the choices and rewards of each agents at each point in time. The key implementation detail here will be an algorithm for learning about costs on paths and making decisions about which path to take based on estimates of these. There are only two or three possible paths, so you can restrict your focus to thinking of these as 2- or 3-action games. For the learning model, algorithms you can consider include *fictitious play*, multi-armed bandit algorithms like ϵ-greedy, UCB1, or Thompson sampling, or algorithms that maintain and update a prior over what the population of other agents will do and choose an action based on that. (You should look up these algorithms; they should all be simple to implement).

Again, feel free to tackle this in any way that you find interesting, but here are some suggestions for questions and issues to explore in this domain:

1. How are outcomes different in the two versions of the road network (with and without the "superhighway")?
2. How does the number of agents playing affect outcomes?
3. How are things different if you can see your own cost, but not the cost along every path at a given time, versus situations where you can see the cost along every path even ones you don't traverse?
4. What are the effects of agents using different learning strategies? (I suggest sticking to one strategy across the whole set of agents initially, but you're welcome to try mixing the learning algorithms and having different proportions of agents use different learning algorithms as an extension; one interesting possibility might be to fix the learning algorithms for all bue one agent and then find which algorithm performs best in)
5. What are the effects of different initial choices or prior beliefs for the agents?

4 Your own idea

Feel free to come up with an idea for your own project and run it by Sanmay if you're more compelled by that than any of the above options.