Note: See ORIE 5355 Fall 2023 Project instructions for an overview of the project, as well as logistics. This document is intended purely as informational/suggestive for the strategy that you might undertake in each part – the project is open ended in terms of strategies you take.

Part 1: Pricing under competition

In this part, you must consider pricing in the presence of competition from your opposing team: if they offer low prices, then the customer will buy from them instead of buying from you. Thus, your task is to price high enough that you make revenue from a sale, but low enough such that the customer buys from you and not your competitor [Note that it does not matter how many *individual games* you win, just your revenue across games; thus, pricing near 0 for every item in every round is unlikely to be an effective strategy].

Note that after each customer, you will observe the price your opponent set, as well as the customer's decision.

There are many different aspects to consider and potential strategies to try when facing an opponent. I hope that this part teaches you the various complexities when thinking about applying game theory ideas to practice [we haven't delved too much into game theory in this class but have touched on ideas with competition]. In particular, I encourage you to think about the following:

1. Even before considering that your opponent is strategic and so can react to your choices, consider that they have some demand prediction/revenue maximization model that is outputting prices for the customer, just like you do. Their model might be similar or different to your model – can you learn/incorporate the tendencies of their model, such as the tendency to price items for certain customers higher than you do? In other words, can you predict the prices that your opponent will set for a given customer if your opponent faced no competition?
    a. How does this information help you set prices? Consider the following naïve heuristic. Suppose the customer valuation is v, and so optimal pricing without competition is $p = v$.. Then, with competition, instead of setting that price, instead set the price as $\alpha * p$, where α depends on how high your opponent prices tend to be. For example, if your opponent always prices things at infinity (equivalently, you have no opponent), then $\alpha = 1$. If instead your opponent sets very low prices, then set α close to 0.
2. Of course, your opponent is also strategic, and so you can't just learn a single α – you need to learn how they are taking competition into account. This task is like the "adaptive experimentation" we briefly touched on. One naïve strategy for adaptive experimentation is to adjust α after every customer. If the customer bought from you, then increase α. If they bought from your opponent or bought neither item, decrease α (your opponent set lower prices than you did). The amount that you increase or decrease α could depend on the exact prices they show (which we will provide you).
3. Of course, your opponent is also able to do such adaptive experimentation… You might be able to *take advantage of* such experimentation: set very high prices at first so that they increase their α, and then for later customers drop your prices to undercut them.

However, they might then take advantage of you trying to take advantage of them, and the cycle continues.

4. The goal is to maximize your revenue across games, *not* win an individual game against one competitor. Thus, as stated above, pricing near 0 for every item in every round is unlikely to be an effective strategy. However, maybe some of you will be tempted to set very low prices anyway. If you are worried about your competitor doing so (which will also affect your revenue for the round), you can “punish” your competitor by setting prices to 0 until they raise their prices (this will only work if their code raises prices if punished). For more information here, I encourage you to read about “tit-for-tat” strategies in Prisoner’s Dilemma game theory tournaments. [My guess is that this strategy is unlikely to be very useful, but it could be depending on how devious your classmates are…]

For further interest, I encourage you to read about “Level K” strategies in behavioral game theory. The idea is that “Level 1” means that you behave naively, as in Part 1 of the project (assuming no competition). “Level 2” means that you play optimally assuming your opponent is playing at “Level 1” (this is roughly what I suggest in the first item in the list). “Level 3” means you assume your level is playing at “Level 2” (what I suggest in 2^{nd} item in this list). This can continue to infinity, where each Level K assumes the opponent is playing at Level K-1. In practice, in games people often play a Level K strategy for a small K, instead of a formal game theoretic equilibrium (in which each player is playing optimally assuming the other is as well, roughly equivalent to setting K to infinity).

Testing strategy
One thing you might be wondering is how to test your agent? Your opponents might take a variety of strategies, and your agent should do well against them all. (Remember, only overall revenue across games matters). We recommend playing your agent against itself and against other dummy agents, for example:

- An agent that just outputs the optimal price without competition.
- The agents that we have added to the repository.
- An agent created by another team in the class (this is an exception to the rule that you should not possess code from other groups; you still cannot copy code, but you are allowed to transfer the agents into the same computer in order to simulate a game).

Part 2: Demand estimation and optimal pricing for two items, alongside competition.

Above, we already gave advice on how to think about competition. Here, give advice on how to think about demand estimation and optimal pricing for 2 items without worrying about competition. Then, you should think about how to extend the advice for Part 1 into a setting with 2 items and demand estimation.

Suppose you want to do “single-shot” revenue maximization, with two items. Then, the revenue function looks like:

$$p_1 d_1(p_1, p_2, x) + p_2 d_2(p_1, p_2, x)$$

Where (p_1, p_2) are the prices, $d_k(\cdot)$ is the demand for item k, and x is the user covariates.

We suggest breaking this down into two tasks:

Demand Prediction

For a given set of two prices (1 for each item) and customer information, predict the probability $d_k(\cdot)$ that the customer will buy each item k (or buy nothing). We suggest doing so using a multi-class machine learning model that takes in as covariates:

- Prices (p_1, p_2)
- The demographic covariates available for every user

Then, you can use the purchase decisions provided in the training data as the true training labels. Here is an overview of the sklearn multi-class classification algorithms: 1.12. Multiclass and multioutput algorithms — scikit-learn 1.0.1 documentation

And then to extract the class probabilities, use: predict_proba, as in HW3.

- Warning: some of the multiclass "classification" algorithms only output probabilities that are either 0 or 1 for each class, even if you use predict_proba. I would recommend not using those algorithms.

Revenue-maximizing prices

Once you have learned $d_k(\cdot)$, the next step is to find a pair of prices that maximizes expected revenue:

$$argmax_{\{p_1, p_2\}} [p_1 d_1(p_1, p_2, x) + p_2 d_2(p_1, p_2, x)]$$

In HW3, you solved a 1-dimensional version of this problem, likely by "brute-force" – you created a list of possible prices, for each price p calculated the expected revenue using that price, and then selected the price p^* that maximized this revenue.

That same general approach will work here but your code will take a lot of time to run. If you consider 100 unique prices for each item, then you are considering 100*100 = 10,000 pairs of prices, and so would need to evaluate your machine learning model 10,000 times for each customer. Recall that we impose a constraint: your code will need to spend less than ½ seconds per customer on average.

Thus, you will likely need to find a more efficient way to find the optimal pair of prices $\{p_1, p_2\}$ in this task. We suggest one of the following approaches, or a combination thereof.

1. Instead of creating 10000 evenly spaced pairs of prices, sample 100 pairs of prices uniformly at random and simply choose the best pair of prices among these. It will often be the case that this pair will perform close to optimal. Instead of sampling uniformly at

random, you can also create a coarser grid, such that you only consider 10 price options per item, and so 10*10 = 100 pairs total.

2. Iterative refinement: As a first step, do the above. But now, select the best pair from the first step, and create a grid around it. And then repeat, sampling 100 points from this smaller grid or 100 pairs of prices evenly spaced in this grid. (For example, if the best prices from the first step is $\{p_1 = 3,\ p_2 = 4\}$, then create a new grid around $\{p_1 \in [2.5,\ 3.5],\ p_2 \in [3.5,\ 4.5]\}$). Repeat until you don't see an expected revenue gain.
3. Zero-order optimization. Formally, this task is a "zero-order optimization" challenge, where you don't (trivially) have access to the derivative of your objective function (revenue) with respect to the prices, since the value of your objective function is obtained from a machine learning model. However, you can still under certain assumptions (that roughly hold here, depending on your demand estimation model) still perform something that looks like gradient descent. The idea is to start at some initial pair of prices $\{p_1,\ p_2\}$ and estimate the gradient around that pair, by evaluating revenue at $\{p_1 \pm \epsilon,\ p_2 \pm \epsilon\}$. Then, take a step in the direction of the gradient, like you would in gradient descent.

Note that this problem is very similar to something you might have done or will do in other courses: hyperparameter optimization. In that problem, you want to find the optimal set of hyperparameters for your machine learning model (e.g., learning rate in neural network training), but evaluating how good your model does under each set of hyperparameters takes a long time, as it requires training a new machine learning model. The approaches suggested above are analogous to those used to perform hyperparameter optimization in practice.

Testing strategy

Suppose you want to evaluate/test your price optimization method, without worrying about competitor prices (remember that you have to submit a CSV for optimal prices for the test set assuming no competitor).

One suggestion is to create an Agent that just prices according to these optimal prices, and then run it against a dummy agent that always prices at [1 million dollars, 1 million dollars]. That will simulate pricing without a competitor because your prices will always be lower than this dummy agent's prices.