Dynamic Pricing in a Competitive Environment

Elliot Pickens (eep58) Andrew Yan (zy434) Candice He (sh2382)

Nikhil Garg
Data, People, and Systems (Cornell Tech)

December 15, 2021

1 Introduction

In this project we set out to build a system that could take in a set of covariates and output a set of personalized prices, where each set of covariates is said to have come from a hypothetical user. Additionally, we assume that our pricing system exists within a competitive environment. That is to say each user is expected to receive multiple price price offers. We want our system to be able to account for this. To do this we divide the problem into two parts: personalized pricing based on covariates and adaptive pricing according to competition.

2 Demand Estimation

To estimate demand in part 1 we tried a number of different techniques before ultimately settling on a combination of a neural network for prediction and kNN to impute missing embeddings. This was, however, only the last stage of our modeling process. Before reaching this final model we went through an extensive experimentation process first.

After splitting the data into a 70-30 train test split we started our modeling process by figuring out how to impute the missing user embedding values. To do this we tested three different methods: random forest imputation, kNN imputation, and MICE forest imputation using lightgbm. We found that the MICE forest imputation performed best, so we moved forward with it into the next part of our modeling process. After imputing the values and creating a complete list of user vectors we then computed the dot product between each user vector and each product vector to get two new similarity matching metrics to be used in our models.

The next question we turned to answer was how we would model demand given the data. We began this process by evaluating a number of different un-tuned prototype models. These prototypes were logistic regression, random forest, lightgbm, svc, xgboost, and Bayesian logistic (softmax) regression. Of these methods we found that xgboost and Bayesian logistic regression performed best so we pushed them forward to the next stage in the modeling process.

We used cross validated gridsearch to tune the models iteratively until we found what we believed to be the best sets of parameters. At this point both of the models were outputting very similar performance. Our initial thought was to select the logistic regression model as our final model due to its simplicity, but drawing predictions required sampling from the posterior predictive distribution and was too slow a process to be used in the context of the competition.

2.1 Covariate Bias

Considering the broader context within which our project is situated we decided to analyze exactly what our model was doing with the inputs we gave it. We wanted to know whether or not it was valuing certain customer metrics above others. For this project, where covariates are just abstract random variables it is not so important which ones the model uses, but if they were important demographic features (like race or gender identity) this sort of dynamic pricing based on the output of a machine learning model can quickly become very problematic.

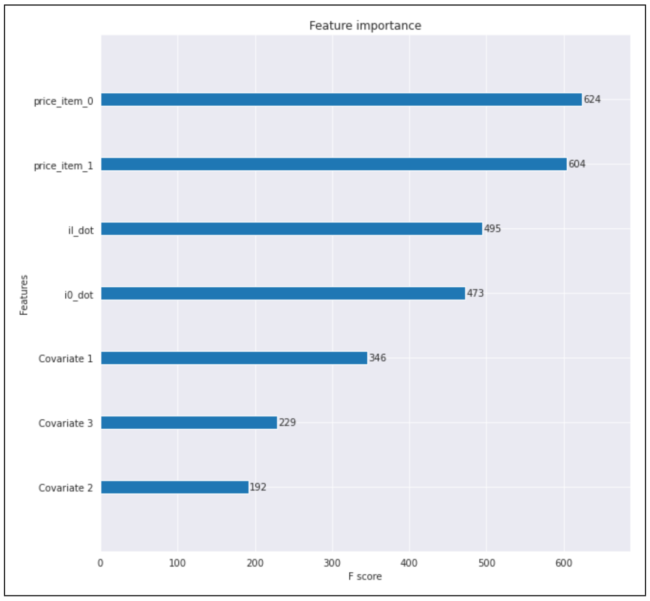


Figure 1: xgboost Feature Importance

As can be seen in figure 1 we found that our model did favor one covariate above the others (covariate 1). It also used user embeddings (through `i0_dot` and `i1_dot`), which could also include important potentially sensitive information. Of course, this is a very minor audit of model behavior, but it shows just how critical it is to not blindly roll out a black box when discrimination is plausible.

2.2 Model Alterations and Performance

Although we initially decided to use a xgboost model, we eventually used a SGD neural network implemented in `Pytorch` to make our predictions. We made this decision because we saw a modest boost in prediction f1 scores across all classes.

Model Performance			
Model	DNP f1	Item 1 f1	Item 0 f1
Logistic Regression	0.838	0.884	0.852
Bayesian Logistic Regression	0.810	0.884	0.841
lightgbm	0.837	0.880	0.843
SVC	**0.848**	0.887	0.854
Random Forest	0.824	0.871	0.831
xgboost	0.829	0.8799	0.849
SGD NNet	0.845	**0.892**	**0.862**

Having completed the competition and evaluated the results this was probably not necessary. We made sure to tune the neural network to prevent overfitting and maximize performance, but given the potential errors of such a method and their lack of explain-ability (especially considering the ethical concerns previously touched upon) we believe this setup is sub-optimal in a real world setting.

We also had to drop the lightgbm imputation in favor of kNN imputation with (k=3). This was done purely for compatibility reasons, because the package we used was not a default package in Google Collab. We had to ensure our agent would work without errors within the competitive environment (which was hosted using Collab), so replacing the MICE imputation was necessary.

2.2.1 Occam's Razor Approach

Taking this all into consideration if we were to have redone this project we would have likely just stuck with a basic logistic regression. We saw that our Bayesian logistic regression was able to compete with the other models easily, but disregarded it too quick due to computational issues in favor of more complex alternatives. We should have just returned to a more basic logistic regression model and tuned it to get the best possible performance.

We also could have possibly improved our results by spending less time on imputation. It worked fairly well, but we could have further reduced complexity in our approach while improving robustness and transparency by not having the deal with it. If we had further time we would have liked to have tested running two separate logistic regression models (one for users with available user profiles and one for users without such data), as well as just sticking with a single logistics regression model based on the core covariate data.

3 Price Search

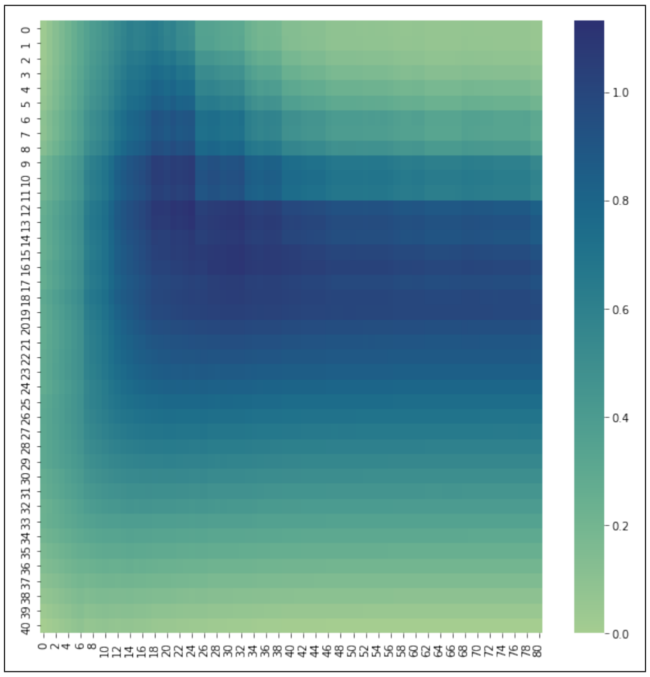


Figure 2: Example Price Search Heatmap

After selecting our model of choice we had to establish a process for finding the optimal set of prices given our demand predictions. The core of our approach was to check estimated revenue at different points by dotting our predicted purchase probabilities with prices, and then returning the set of prices we found would give the most expected revenue. Our process for selecting prices was broken down into the following steps:

- Run a semi randomized search to find a good possible starting point
  - To do this we start at a random point and check the six "mini grid" points surrounding it and then choose the one that gives the best increase in max revenue
  - We repeat this search process until a (likely local maximum) has been reached
  - If we haven't hit a iteration cap, restart the search at a new random point
  - Return the best result across all searches
- Draw a region around the most promising price point returned by the previous search

- Search that region using a gridsearch that iteratively zooms in and becomes more discerning
  - For example, if the random search returns (1,1) we choose the 2×2 region that has (1,1) at its center[1]
  - Then run a coarse gridsearch that divides the region into n^2 blocks to find a new best price
    - Where n is chosen in advance
  - We then zoom into a smaller region around the new best point and repeat the gridsearch, but with $(2m)n^2$ search points (where m is the number of gridsearches that have been run)
  - The search is complete when we no long see revenue increase or a max number of iterations has been achieved

4 Part II Competition

4.1 Overview

In this competitive marketplace environment, there are a series of head-to-head competitions. Each competition consisted of two pricing agents offering personalized prices to some number of simulated users arriving one by one in a queue. The customers/users will always buy from the team that provides them with the maximum utility, or neither if both teams prices are inconsistent with consumer preferences. Thus, each agent will receive either 0$ or the price of the item the user values most highly at each time step in the completion. To account for this and the fact that we are not considering the "cost" of the products, nor any no capacity constraints on those items, we decided to first implement a strategy of adaptively lowering and raising our prices based on how much we have outsold our competitors in the last 30 rounds. We want to avoid being outbid by other opponents who may offer a lower-than-expected price, and focus on beating each team we face individually.

4.2 Group-Sampling

Since each customer's buying choices are independent, one may think to develop an discount rate (alpha) strategy, which updates their pricing every round based on their profit in the previous round. That's what we saw in the "dummy_fixed_prices_adaptive" agent: if there's no profit being made, lower the price; if made profit, raise the price. However, there is an obvious flaw: its very susceptible to outliers. An outlier decision could be caused by either (1) the nature of randomness in the purchasing decision by some customers. (2) an "attack" by the opposing team that gives out extremely low or high prices. In both situations, the naïve adaptive strategy would adjust the alpha discount ratio very sensitively, and it would not be an accurate reflection of how our pricing is performed. Thus, to solve such a model, we decided to implement a group-sampling strategy: after every round, we would record the purchasing price and purchasing decision for the previous 30 customers, and run some simple analysis to determine how our model been performing for the past 30 rounds.

4.3 Discount rate (α) & Adaptation

Our guess for all participating strategies is that there will be a mix of various pricing schemes. Some of the teams might consistently offer a higher price while others may consistently offer a lower price. To deal with such variation in prices, we want our agent to learn how the opposing team has behaved in the last 30 rounds and provide a discount rate on our items that can be best to react to whichever team we are competing against while still outselling the opponent. The goal is to find the minimal discount rate so that we are on average consistently outselling the other team, but still getting the maximum price that we can for each item. Here we are treating each item's price independently.

[1]The size of the initial region can be chosen based on the problem at hand.

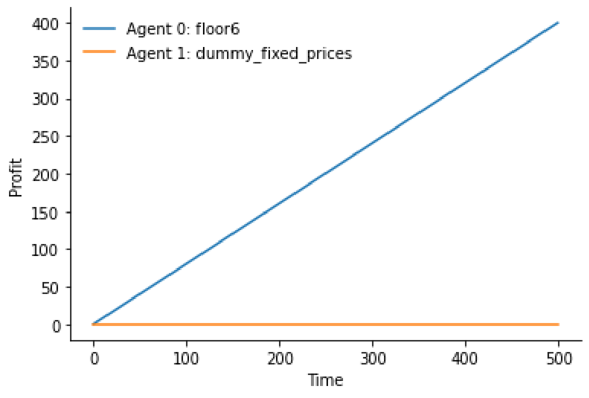
That is, if we are not outselling our items by more than 1 in the last 30 rounds, we lower our prices by $max(0.04,$ the average price differences between our price and the opposing team's).This allows a fast application of discount rates if our prices are significantly higher than the other teams. On the other hand, if we are in total domination and outselling more than 10 copies of an item in the last 30 rounds, we would raise our price by 0.07. This aggressive pricing strategy will guarantee our prices outsell our competitors.

Outlier removal To avoid some teams providing high or low prices that would affect the average difference in prices, we removed outliers among prices offered by the opposing team.

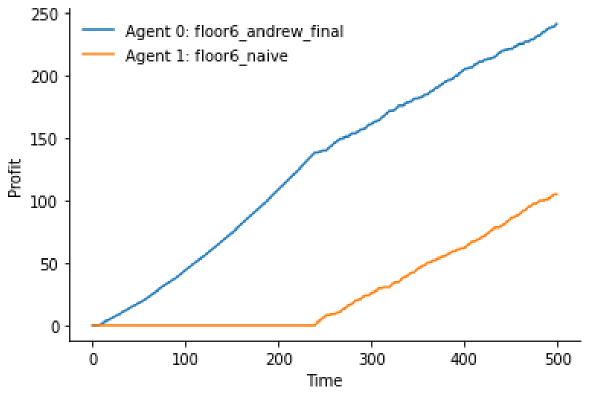
Lower bound Since some of the teams might be lowering their price close to 0, in that case, we are not getting any profit no matter how much we alter our pricing, thus there is no lower bound on how much the discount should be, as long as the final prices are above 0.

Upper bound In the case of raising our prices when our prices are significantly lower than the opposing team, we want to close such a gap by raising our price while still beating the opposing team. but also we want to be careful to never make our prices more expensive than what our original pricing model (from part 1) returned. Any price higher than that might lead customers to turn away. Thus, the maximum price for items would be the original maximum prices from part 1.

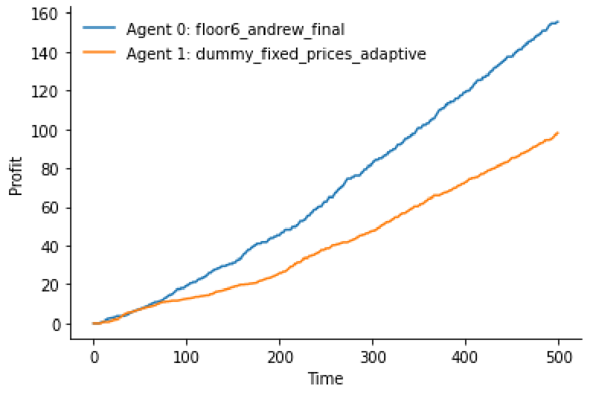
Discount rate parameter tuning we determine the discount rate by looking at graphs of how fast our prices are changing after every 30 rounds when benchmarking against other agents.



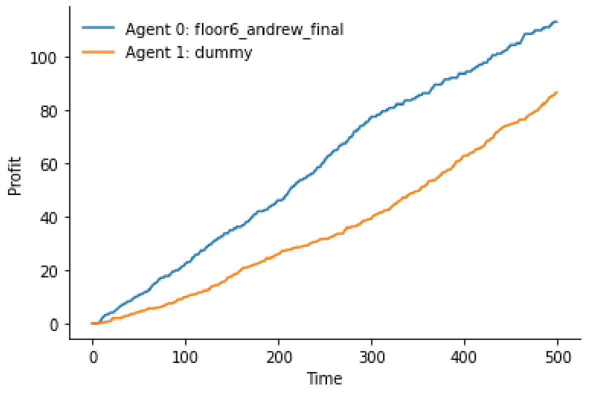
(a) Head to head against `dummy_fixed_prices`



(b) Head to head against naive prices from part 1



(a) Head to head against `dummy_fixed_prices_adaptive`



(b) Head to head against dummy

4.4 Benchmark Testing

We tested our agents against all three variations of the dummy agents, and the naïve static pricing result of our own from part 1 to evaluate how our agent is performing. Based on those results, we tuned our parameters to find the optimal discount rate. From the results, our model destroyed the dummy_fixed_prices and won against all the other agents by a great margin. When we compared our adaptive pricing to what we have in part 1, our agent wins while still earning a considerable amount of revenue without prices dropping too low. We can see these results in 4.3 and 4.3.

4.5 Assumptions about the Competition

Our assumption about the opposing team is that most of the teams will not develop a super sophisticated strategy in such a marketplace. And many of them may think about the idea of simply "lowering the price" to outsell others. Thus we want our agent to be adaptive to only lower prices against those who may do the same, and not continue lowering our price when we are outselling the opposing team.

This is, of course, a trade off between higher potential revenue and a higher chance of winning customers. For our agent we chose to go with the form, because we thought most teams might offer a low price. Therefore, we believed that as long as we could win most of the head to head matches, we would have a chance at winning the overall competition by a small margin. The result showed, however, showed this was not the case.

4.6 Exploration Vs Exploitation

Since our strategy is an adaptive one that lowers prices based on the previous 30 rounds, we are not too concerned with not being able to win in later the rounds. To give us a head start, we initialized our discount rate to be -0.2 for item 0 and -0.5 for item1. That way right from the beginning, we can secure more sales.

After the initial leader-board was posted before the 12/6 official competition, we saw that our results were not the best. This inspired us to think about ways of improving our model further. One approach we tried was by adding reinforcement learning.

4.7 Reinforcement Learning

Initially we tested out a basic reinforcement learning method, but chose not to go forward with it in favor of the adaptive sampling method. Why did we choose the adaptive sampling method first? Because it beat reinforcement learning head to head and was more consistently beating dummy variables during our testing. But we did not give up, and instead chose to combine the two methods!

We eventually wrapped the adaptive sampling method within a reinforcement learning framework, because the adaptive sampling strategy will only change prices if we are either outselling the opposing team by more than 10, or less than 1. Thus for the states in between, our prices remain fixed. This sometimes creates problems where the overall revenue curve becomes parallel during these stretches. We introduced reinforcement learning, so our agent could continue to adapt at each time step, and not just after each sample.

Specifically, we used a Q-Learning approach to do this. We defined a 9×9 Q table where the states were defined by how our previous price offerings for each product compared to the opponents. One state was (Our price was higher for item 0, Our price was lower for item 1), for example. Our actions mirrored these states, meaning that the agent could choose to raise, lower, or keep the same price for each item at each step.

We trained this model against the other agents we had developed by filling out our Q table using Bellman equations taking in the previous reward and future expected reward for out action state pairs. During this training phase we used a high exploration value (often not selecting the current expected optimal value), so the agent could learn its environment, but dropped that value down to 0.01 (1% chance of selected a non-optimal value according to the Q-table) during competition.

5 Conclusion

Performance Evaluation By looking at the head-to-head results of our model performance against others, we can see that we are winning the majority of the competitions, and even some by a great margin. This proves that our strategy works in not letting the opposing team have a great advantage over us. However, we are winning while enduring a high cost in the form of low average revenue. The revenue earned by our agent is lower than 400 for more than half of the matches (5). This drags down our overall average revenue.

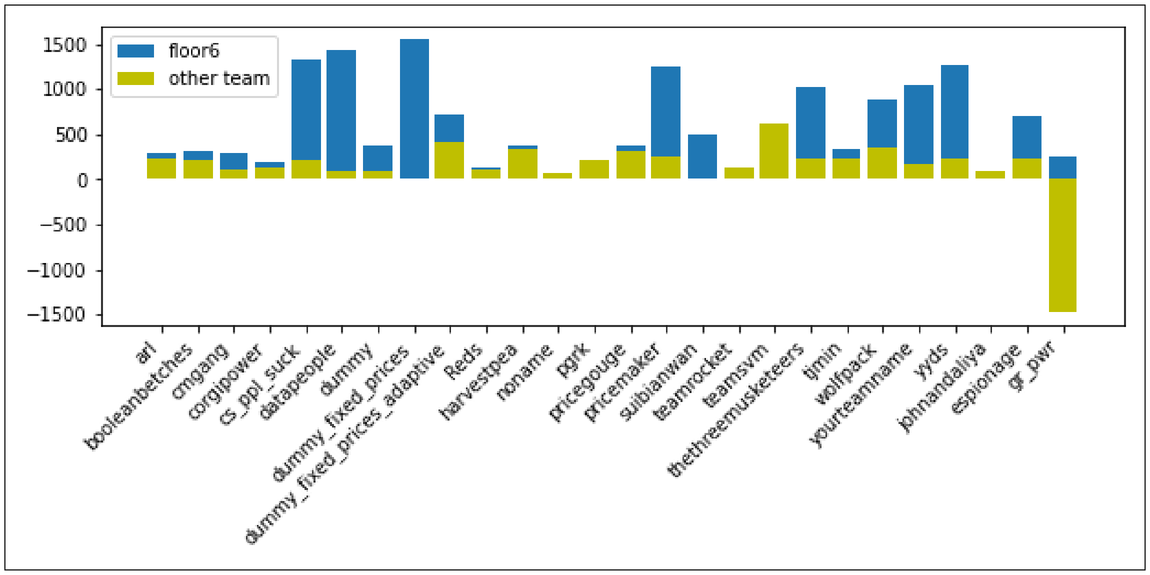


Figure 5: Head to head comparisons

Potential Strategy In the overall average revenue ranking, we were ranked 22nd among the 28 teams. To perform better, we might need to change our strategy-focus in a new direction. Instead of focusing on lowering our prices to outsell the opposing team, we could have a strategy (1) where if the opposing team is lowering their price on a certain item, we stop lowering ours and focus on the other item. (2) tuning our current model to be smarter in raising our prices. Instead of only raising prices when outselling in 10 out of the 30 previous rounds, we can try a strategy to increase our price dramatically back to our original evaluation when the pricing discount pushes the price down to 0. This might help to solve the deadlock situation where both teams are selling their items at an incredibly low price. If it is detected that the other team is lowering prices, instead of trying to beat them, we might stay at the same price with them. (3) If given the performance data of the team's history vs other teams, we might be able to categorize what type of strategy the opposing is using and try to identify a more customized counter-strategy.

Key Takeaways The competition shows that the world is chaotic: we not only need to focus on building an accurate and efficient model, but we also need to observe and react to how other people behave in the same environment. There are various strategies across the space, and choosing the right strategy is far more important than improving the of accuracy of a model in the real world. This project provided us with a basic understanding of how a current marketplaces use recommendation systems for pricing in the real world. Such an understanding will provide us (1) the framework to implement pricing/recommendation-related projects in the future (2) the mindset to always iterate and rework our models to best fit and adapt to the problems at hand.

Project Feedback While the concept of including a recommendation system in this competition is very intriguing, it would be nice to dig into more details on building such a system. For example, given the user covariates, how do we create effective user and item embeddings? Another idea would be to include price surging and capacity constraints in this project. It would be interesting to model certain situations (like low supply), to determine the correct premium to place on certain items.

Overall we enjoyed this project as it not only helped us utilize what we learned in class in a pseudo-real-world project but also gave us a detailed understanding of many methods that can be used in competitive marketplaces.