Prescribing Optimal Prices for Airbnb Listings in New York City

Maya Nesen, Nidhish Nerur

MIT Machine Learning Under a Modern Optimization Lens

Contents

1 Introduction

As a host of an Airbnb listing in New York City—one of the most competitive apartment markets in the United States—how can you know which features of your listing will boost your ratings and, ultimately, your revenue? Through the use of multimodal data, our project provides a model that addresses this challenge by providing hosts with actionable guidance. Specifically, our model focuses on determining what factors of their listing they should optimize and, particularly, how to set the price of their listing to maximize customer satisfaction, increase bookings, and drive higher revenue.

2 The Data

For this project, we selected the AirBnB U.S. Properties Images dataset, which contains 53,063 rows of listings in the U.S., with 24,688 of them being for listings in New York City. Each row contains an Airbnb listing described by 29 features: *id*, *log_price*, *property_type*, *room_type*, *amenities*, *accommodates*, *bathrooms*, *bed_type*, *cancellation_policy*, *cleaning_fee*, *city*, *description*, *first_review*, *host_has_profile_pic*, *host_identity_verified*, *host_response_rate*, *host_since*, *instant_bookable*, *last_review*, *latitude*, *longitude*, *name*, *neighbourhood*, *number_of_reviews*, *review_scores_rating*, *thumbnail_url*, *zipcode*, *bedrooms*, and *beds*.

	log_price	property_type	accommodates	review_scores_rating	bedrooms	amenities
0	5.010635	Apartment	3	100.0	1.0	{"Wireless Internet","Air conditioning",Kitche...
1	5.129899	Apartment	7	93.0	3.0	{"Wireless Internet","Air conditioning",Kitche...
8	4.605170	Apartment	2	93.0	1.0	{Internet,"Wireless Internet","Air conditionin...
13	3.688879	House	2	89.0	1.0	{Internet,"Air conditioning",Kitchen,"Smoking ...
16	5.003946	Apartment	6	100.0	3.0	{TV,"Cable TV",Internet,"Wireless Internet","A...
...	...	...	...	...	...	...
53051	3.688879	Apartment	2	86.0	1.0	{TV,Internet,"Wireless Internet","Air conditio...
53052	3.912023	Apartment	2	60.0	1.0	{TV,"Cable TV",Internet,"Wireless Internet","A...
53053	4.700480	Apartment	2	92.0	1.0	{Internet,"Wireless Internet","Air conditionin...
53060	4.605170	Apartment	1	NaN	1.0	{}
53061	5.220356	Apartment	5	94.0	2.0	{TV,Internet,"Wireless Internet","Air conditio...

Figure 1: Subset of columns from our original dataset for NYC listings

Most of the columns are numeric, comprising our tabular data, and some text and image data. We focused on the tabular and text data for our models, and, if given more time, we would have included the image data as well.

2.1 Feature Engineering

After filtering the data only for the listings in New York City, we started by performing some feature engineering to build variables that could be helpful for our predictive modeling. The features we created are:

- `days_between_reviews`: the number of days between `first_review` and `last_review` dates of a listing;
- `price_per_capacity`: captures the price relative to the number of people the listing accommodates;
- `host_days_active`: the number of days that a host has been active on Airbnb;
- `distance_to_times_square`: measures the proximity to a major tourist attraction, built using the latitude and longitude coordinates of listings and calculating the Euclidean distance from Times Square;
- `room_capacity`: the number of people the bedrooms of a listing can accommodate;
- `price_bucket`: divides the listings into three roughly equal buckets for the *price* column, labeled as:
  - Low (0): $1 to $80
  - Medium (1): $80 to $150
  - High (2): $150 to $1999
- `rating_bucket`: divides the listings into three roughly equal buckets for the `review_score ratings` column, labeled as:
  - Low (0): listings with ratings of less than 90%
  - High (1): listings with ratings between 90% and 99%
  - Perfect(2): listings with ratings of exactly 100%

With these features, we could better capture various aspects of the Airbnb listings. Furthermore, we dropped features in the dataset that were either used to build these new features, did not make managerial sense, or could cause multicollinearity issues, such as $first\_review$ and $last\_review$, $zipcode$, and log_price.

The price and rating buckets, however, are our focal point. The price bucket column contains our treatment groups for when we later compute the rewards matrix, while the rating bucket is our outcome variable for our predictive model. To make sure the ratings and price ranges of listings were fairly evenly distributed, we splitted each of them into three roughly equal buckets. We attempted different numbers of buckets and different ranges of values for the price and ratings, but three buckets for each gave us the best distributions. For the price bucket, 8,472 listings are priced from $1 to $80, 8,829 listings are priced from $80 to $150, and 7,387 listings are priced from $150 to $1999. For the rating bucket, 3,672 listings had a rating of less than 90%, 9,716 had a rating between 90% and 99%, and 5,443 had a rating of exactly 100%. The range values for our buckets are not ideal because, for example, we equate or bucketize a listing with rating of 0% with a listing of rating 89%, which may have very different features

and, clearly, different levels of customer satisfaction. However, for the sake of our modeling, we implemented these distributions.

2.2 Data Cleaning

Our dataset was fairly clean to begin with, but we pre-processed our data to prepare it for analysis. Besides feature engineering and removing irrelevant or highly correlated variables, as mentioned previously, we:

- **Handled missing values**: Most of our missing values were in the ratings column, so we dropped all listings that did not have ratings so as to maintain outcome integrity and to avoid introducing noise or variability in the ratings. This reduced the number of listings in NYC in our dataset from 24,688 to 18,831. For fields with remaining missing values, we imputed values. For numeric features, we used Iterative Imputer which estimates the values of the desired feature based on the other features. For categorical features, we imputed the mode.
- **Standardized our features**: We standardized the numeric features using Standard Scaler to ensure that their values are on the same scale, and transformed our boolean features, like *host_has_profile_pic* or *instant_bookable*, into binary.
- **Binned categorical features into even groupings**: Some of our categorical features have many categories, but little numbers of listings in each category. Therefore, we grouped categories together to reduce the number of groups, which would avoid having too many columns once the features were one-hot encoded. For example, for the *neighbourhood* column, there are 190 unique neighborhoods, but we grouped them by "popularity", ranked by listing counts, into a *neighbourhood_group* column: the top 10% of neighborhoods are marked as "High Frequency", top 10% to 40% are marked as "Medium Frequency", and the remaining neighborhoods are marked as "Low Frequency". Similar groupings were performed on the *property_type* and *cancellation_policy* columns.
- **Retained top amenities and created binary columns for each**: The column containing all the amenities for each listing has over 17,000 unique amenities. Therefore, again to avoid having far too many one-hot encoded columns for this feature, we retained only the top ten amenities based on their frequency across listings and created indicator binary columns for whether each listing contained those amenities.
- **Applied one-hot encoding for categorical features**: We performed a simple one-hot encoding on the *property_type*, *room_type*, *cancellation_policy*, and *neighbourhood_group* features to use in our predictive model.
- **Combined text data columns**: Our dataset only contains two text columns: the name of the listing and the description written by the host. We concatenated these two columns into one *text* column to more easily perform text analysis later in our predictive model.

3 Building a Predictive Model

We began by building an initial model to predict the *review_score_rating* using the features of the listing, including the bucketized price. We used XGBoost which is a tree ensemble method, built on sequentially learning the errors of the previous tree. XGBoost has strong predictive power and enables us to easily incorporate our text data into the model. For each of the below iterations of XGBoost, we performed a train-test split with 80% of the listings for training and 20% for testing the model on its performance. This helps with the generalizability of our model, which is necessary for our counterfactual estimations in the following step. Our XGBoost models use the softmax activation function since we have a multi-class classification problem, predicting the probability of a listing falling in each of the three rating bucket groups.

3.1 XGBoost on Tabular Data Alone

We first built an XGBoost model on our tabular Airbnb data alone. Our model output an out-of-sample accuracy of 67.32%, meaning our model's rating predictions correctly determined the listings' true ratings 67.32% of the time. While the accuracy is not incredibly high, the model is still making reasonable predictions. The model also output a strong AUC of 0.8, which indicates that the XGBoost is correctly distinguishing between classes of ratings. This means that the model overall performs well in determining the probability for each listing of falling into each rating bucket, even though the raw classification accuracy is not as strong.

3.2 XGBoost on Text Data Alone

We next built an XGBoost model on our text data alone, i.e. the name and host description of our listings. In order to do so, we need to first convert the text data into numeric values in order for the model to process the information and make predictions. Therefore, we first utilized a pre-trained deep learning natural language processing (NLP) model called BERT to create embeddings from our text data. BERT first tokenizes the text data to split the text into smaller units or tokens, then creates embeddings to convert them into numerical values. BERT further uses transformers to capture contextual information for the token relative to the text as a whole. Using BERT in our code, our text data column was transformed into 768 numerical embeddings. We ran our XGBoost on these 768 embeddings to predict the ratings, which output an accuracy of 52.6%. The performance is fairly low on the text alone, but is expected since we are not including the tabular features, which add more granular information about the listings, and the descriptions themselves might not necessarily capture the true value and quality of the listing, especially because they are written by the host, further emphasizing the need for our project's overall goal of providing actionable guidance to hosts.

3.3 XGBoost on Combined Multimodal Data

Finally, we ran the XGBoost model on both the tabular and text embeddings together, which gave an accuracy of 67.1%. While slightly lower than the performance of the XGBoost on the tabular data alone, this combined model still provides solid predictions to use for counterfactual estimations.

4 Counterfactual Estimation

4.1 Rewards Matrix

The multi-modal classification models predict a particular rating bucket score for each listing, which enables us to perform counterfactual estimation. We generate a reward matrix that specifies the particular rating outcome resulting from each potential treatment policy, namely, the price P_0, P_1, and P_2 buckets.

The price field in our dataset was numeric initially, but as discussed in the feature engineering portion of our paper, we discretize this variable into buckets ranging from 1 to 80 dollars (P_0), 80 to 150 dollars (P_1), and 150+ dollars (P_2). We acknowledge that the first two ranges appear quite narrow while the third is expansive; however, with our dataset size of 18,831 NYC listings, the selected price treatment bucket sizes are roughly even. With more data and computational capability, we could experiment with more varied price ranges. Additionally, three prescription policy categories made the reward estimation process more efficient and reduced the risk of overfitting, as we introduce less complexity.

We manually create the rewards matrix using three multi-modal classification models to estimate the rating outcome across each price treatment. Specifically, we grouped our datapoints within the P_0, P_1, and P_2 buckets then built separate models to predict the rating outcome score. This enabled us to compute counterfactuals, which represent the likely rating outcome score if the Airbnb NYC listing had been priced in a different range. For instance, if we observe a listing priced in the P_0 range, then we need to estimate its discretized rating score if it had been priced in both the P_1 and P_2 ranges. Then, for our 18,831 NYC listings, we have the actual rating bucket for the observed price along with the estimated rating bin for the other price treatments. A small subset of our complete reward matrix is presented in Figure 2 below:

	treatment0	treatment1	treatment2
3	0	1	1
6	2	2	2
7	1	1	1
13	0	2	2
14	1	1	1

Figure 2: Subset of the estimated rating buckets.

5 Optimal Policy Trees

5.1 OPT with Direct Method

Given our rewards matrix and training data with tabular features and textual embeddings, we construct an Optimal Policy Tree. We perform an 80-20 split of the data with 80% of our observations in the training set and 20% in the testing set. Then, we fine-tuned the *max_depth* hyperparameter to account for the tree's complexity, testing depths from 1 to 5. Higher max depth allows for greater complexity with interactions between features, while lower max depth can make the tree simpler with fewer splits. Our Optimal Policy Tree is illustrated in Figure 3:

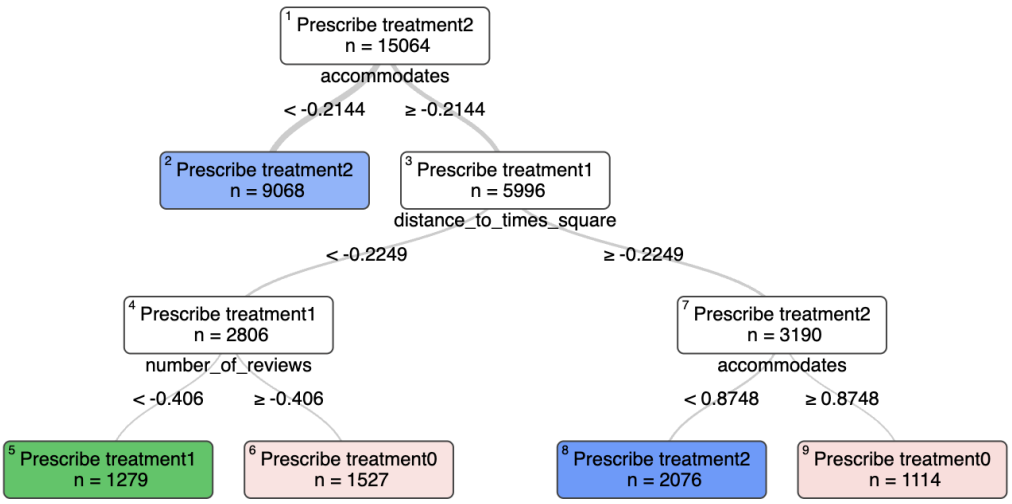

Figure 3: Optimal Policy Tree with Direct Method

The tree highlights that the number of people the Airbnb listing accommodates, distance to Times Square, and number of customer reviews drive which price to prescribe in order to optimize the rating range. Logically, these features make business sense in determining how to price an NYC listing. In particular, if the listing is geographically close to the popular Times Square attraction, accommodates more people, or has more reviews, then we would expect the listing to be pricier on average.

The features were standardized prior to creating the tree, and we convert them to their original scale to discuss the results. Even so, interpreting the specific decision rules is a bit challenging. For instance, the first split tells us that an Airbnb that accommodates fewer than two people should be prescribed the highest price range to maximize potential ratings. Furthermore, an Airbnb somewhat far away from Times Square that can accommodate fewer than four people should also receive the highest price. These rules are counterintuitive and require further investigation. The policy tree informs us that listings close to Times Square that can accommodate more than two people and have few existing reviews should be priced moderately. This rule

makes sense, as proximity and accommodations could raise customers' expectations of price while fewer reviews may lower their pricing estimates.

The average estimated rating outcome score under our prescribed treatments is about 1.326, whereas the average rating score under the treatment prices observed in the data is roughly 1.096. Thus, our tree provides a 20.94% improvement in the rating bucket score, raising both customer satisfaction and host revenue.

The primary issue with the direct method is the risk of treatment assignment bias. Hosts independently assign prices based on their expectations of demand, property quality, and more. There could be confounding bias as higher prices may be associated with greater ratings while lower prices are correlated with lower rating scores. To potentially account for these biases, we employ the doubly robust method.

5.2 OPT with Doubly Robust Method

The inverse propensity weights in doubly robust will re-weigh the treatment group rewards, reducing risk of treatment assignment bias. The direct method models predict counterfactual outcomes, which uses the relationship between covariates and the rating outcome bucket. Combining these two approaches creates a more robust model to predict rating scores across the price buckets. For this tree, we made the outcome binary rather than multi-class for ease of computation, so we have perfect rating as one bucket and all other scores in the remaining class. The Optimal Policy Tree with the doubly robust method is shown in Figure 4 below:

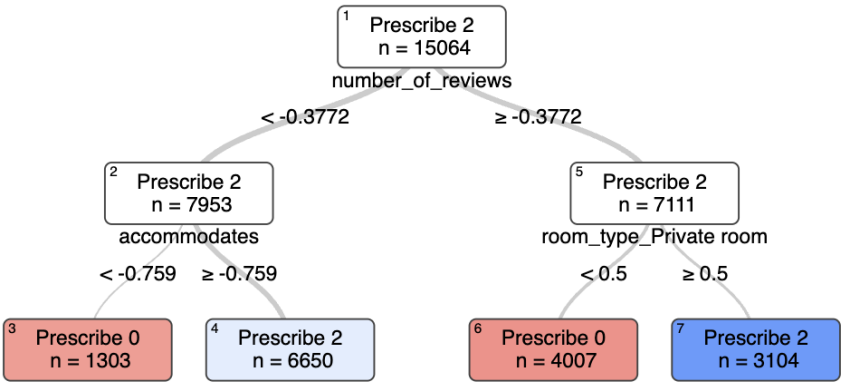

Figure 4: Optimal Policy Tree with Doubly Robust

Similar to our first tree, this one also suggests that number of reviews and accommodates are key drivers of the optimal pricing strategy to yield improved ratings. We also see that the room type being listed as a private room rather than a shared room is an important determinant of the prescribed price. Furthermore, we see this tree does not prescribe the middle price range of 80 to 150 dollars, and tells hosts to keep the price cheap or on the more expensive side.

The average estimated rating outcome score under our prescribed treatments is about 0.360, whereas the average rating score under the treatment prices observed in the data is roughly 0.293. Thus, our tree provides a 22.81% improvement in the rating bucket score.

6 Conclusion

6.1 Future Direction

To improve results, we would utilize image data, further fine-tune the price and rating buckets, incorporate customer reviews, and improve interpretability of the decision rules.

- **Image Data**: The Airbnb dataset provided images of one room in each listing, but we were unable to include them due to computational limitations. Specifically, the two optimal policy trees used 805 features (37 tabular and 768 textual embeddings), which already required significant computational power and the runtime was extremely slow. Incorporating embeddings for the images would have taken much more runtime and compute. We would definitely try to add on the images in the next iteration of the project.
- **Discretization of Price and Ratings**: We categorized our treatment price into three ranges, and the first two were narrow from 1 to 80 dollars and 80 to 150 dollars, respectively. With more data, we could create more buckets and apply our counterfactual methods. Additionally, we could utilize continuous price treatments with continuous counterfactual estimations to assess performance differences. Similarly, we would fine-tune the rating bucket outcome, as our lowest bucket consists of all ratings from 0 to 89 which may not make business sense. Naturally, a listing rated a 0 on average should not be treated the same as one with an 89 average score. With our limited dataset, we experimented with various rating buckets and would continue to update them.
- **Customer Reviews**: The Airbnb data had access to the host textual description of his or her listing, but not the actual reviews provided by the customer. We would try to extract the raw customer reviews from the Airbnb website filtered for NYC listings to improve our multi-modal models.
- **Improve Interpretability**: As discussed in the Optimal Policy Tree section, the rules arriving at a particular prescription can be somewhat difficult to interpret. Some of the results do not make much intuitive sense, while others are logically sound. We would try to experiment with other counterfactual estimation methods and compare both the decision rules and performance across multiple Optimal Policy Trees.

6.2 Contributions

For the report and presentation, equal work was contributed by Nidhish Nerur and Maya Nesen. Maya wrote the report Introduction, Data, and Predictive Model sections, and designed the report structure. Nidhish wrote the Counterfactual Estimation, Optimal Policy Trees, and Conclusion sections. Both of us read the entire report and collaborated on the presentation.

With technical tasks, both of us worked together on the overall approach and project organization. Nidhish wrote the data preprocessing, counterfactual estimation, and Optimal Policy Tree code. Maya led the treatment and outcome discretization structure and multi-modal model development to analyze the tabular and textual data. Maya adjusted these models to generate the rewards matrices. Nidhish trained, fine-tuned, and evaluated the policy trees. Both of us reviewed all code and worked closely throughout all aspects of the project.

7 References

OpenAI. (2024). ChatGPT (Dec 6 version) [Large language model]. https://chat.openai.com/chat.

Ren. Airbnb Properties and Image Dataset. Kaggle, 2024, https://www.kaggle.com/datasets/eren2222/airbnb-properties-image.