ISOM 676: Machine Learning

Predicting Factors Influencing clicking of a hotel during a search

Team OneHotEncoder: Neha Bansal, Nirja Mistry, Cassio Salge, Grace Zeng

Introduction

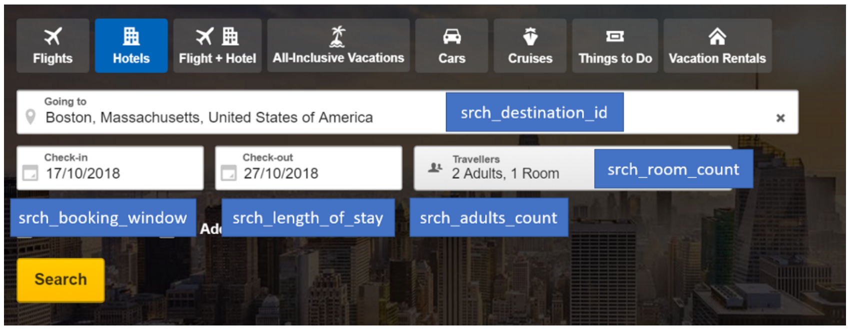
Background & Business Understanding

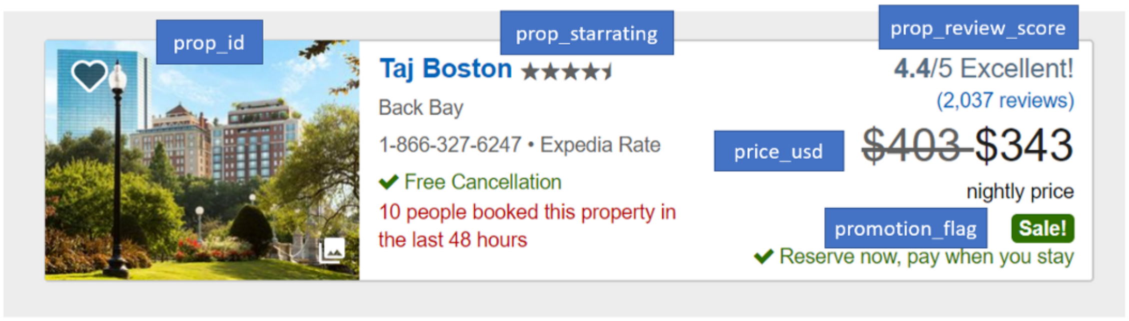
In the past decade the evolution of the internet has led to the development of stronger e-commerce channels. Online tourism has seen great growth in the past couple years and travel expenses are the third largest cost-block in companies after labour and IT. Several prominent Online Travel Agencies (OTAs) are already offering personalized services to help customers choose their hotels. Some of the most prominent OTAs are Kayak, TripAdvisor, Airbnb, Travelocity, and Expedia. Expedia is the world's largest OTA and powers search results for millions of travel shoppers every day. Expedia's award-winning Expert Searching and Pricing (ESP) technology delivers the most comprehensive flight options available online. ESP also allows customers to dynamically build complete trips that combine flights, Expedia Special Rate hotels and other lodging, ground transportation, and destination activities.

In this competitive market matching users to hotel inventory within their willingness to pay is very important since users easily jump from website to website. As such, having the best ranking of hotels for specific users with the best integration of price competitiveness gives an OTA the best chance of winning the sale. Our **goal** is to apply machine learning techniques to predict whether a user will click on a hotel or not based on a particular set of characteristics and preferences. Using these predictions we will be able to rank hotels based on likelihood of purchase, giving Expedia a better idea of which hotels are preferred. This can be beneficial for Expedia as it will allow them to see which hotels are preferred based on user preferences and provide a better selection of hotels and competitive rates on such hotels to improve their sales and market share in the OTA industry.

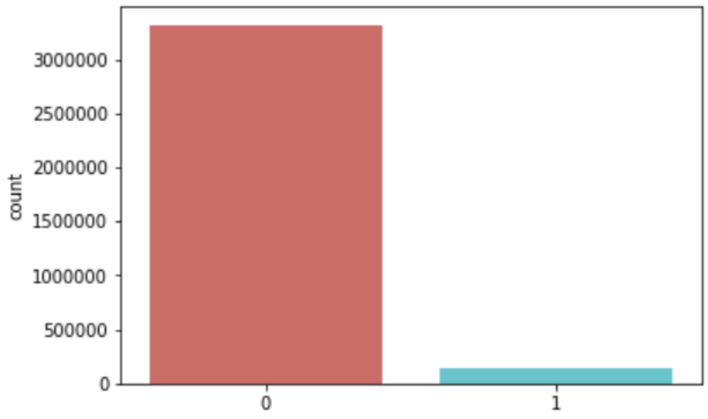
The Data

For this analysis we are using data provided by Expedia as part of a Kaggle competition. The data was already split into training and test sets. The training data has ~10 million rows where each row represents one hotel in a search result and contains features about the hotel characteristics such as location, rating, occupancy, attractiveness, promotions, and how the price compares against competitor OTAs. The test data has ~6 million rows and contains the same columns as the training data barring 4; *position, click_bool* (our target variable), *gross_bookings_usd*, and *booking_bool*. The dataset consists of one target variable column and 53 attributes, of which majority are of integer/float type with one datetime object. To gain a better understanding of what the columns mean past the data description provided by Kaggle, we went to Expedia website.



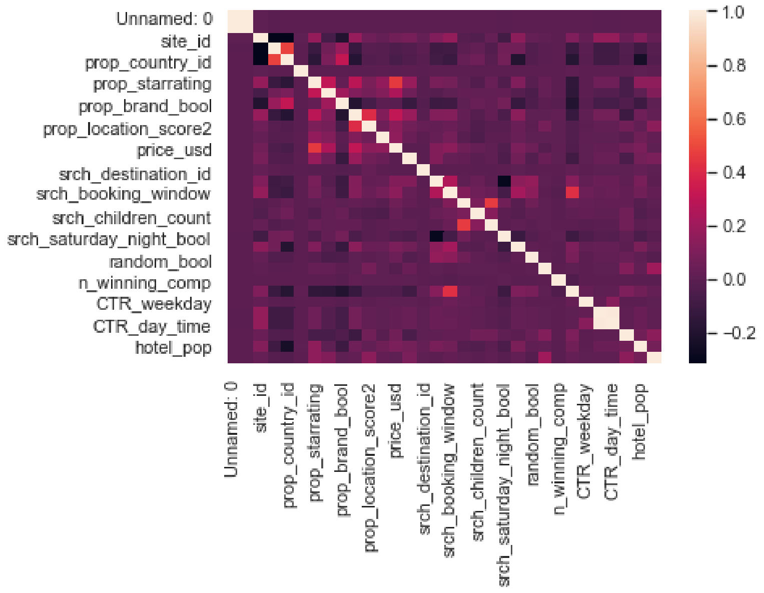


To start off the exploratory data analysis process, we looked at the basic structure of the data, summary statistics, and distribution charts for each of the 53 variables in the dataset and the target variable - *click_bool.* The data is completely anonymized, including things like the location of the search and the destination location. We were able to decipher that the largest country was the United States and it has 61% of all observations. Out of those, 58% of the searches were made by consumers also located in this country so we hypothesized that this country had to have a large territory. This combined with the price currency also suggested that the largest country was the United States. There were quite a few variables with more than 30% of missing data and we came up with a plan on how to deal with these discussed in the next section.



Our target variable is whether a customer clicks on a hotel or not. The click-through rate for this dataset is 4.3% resulting in a severely imbalanced class distribution. However upon some research we found that this was extremely common in OTA and decided to balance it as a result of it being representative of the population and implementing techniques during the modeling phase to account for this. On average, the price_usd that received a click is always lower than those that did not get a click.

We made sure to look for multicollinearity within the dataset and built a heatmap to better visualize this. We can immediately see that some variables are more correlated with each other but for the most part we don't see too much multicollinearity. Using this basic understanding of our columns along with the data description provided by Kaggle, our next steps were to prepare the data for modeling.



Data Preparation

We split up the data preparation into four steps; data cleaning and dealing with data discrepancies (missing data), balancing the dataset, feature engineering, and feature selection.

Data Cleaning

We started off by listing the columns that were present in both the training and the test dataset and removed any columns that were not found in the latter. For our specific project, we focused only on predicting the click rate for Expedia and thus, decided to ignore any column that pertained to the "booking" aspect of the dataset. Next, we identified all the columns that had a large number of null values and decided to remove columns that had over 90% of missing values from our analysis to avoid any discrepancies and missing data. Through these processes, we ended up with 48 columns in our training dataset.

Since our business problem deals with hotels in different locations and ranging across a wide set of prices, we wanted to pay attention to remove any extreme outliers that would affect our analysis and reduce accuracy. We noticed that hotel prices ranged from $75 to over $7 millions and were right-skewed. We extracted the interquartile range of the price variable and used it to derive the upper bound of price as $335. However, this captured less than 95% of the total data and appeared a little conservative so we decided to increase our upper bound to $599, which captured 99% of the data and was a realistic price for luxurious hotels.

Lastly, for our cleaning process we still had some variables - review and location score for the property and the distance from user's location to vacation destination, that had less than 30% null values and we chose to replace them with the median for the column using the imputation method. This method preserves the distribution of values in the dataset.

Balancing the Dataset

As we have a severely imbalanced dataset, we decided to use downsampling to fix the problem. Downsampling means training on a disproportionately low subset of the majority class examples in order to prevent it from dominating the learning algorithm. Due to our extremely large data size, the downsampling could also help us to reasonably decrease the data size inputting to the model and reduce the run time. To do this we first separated the data into two dataframes based on each class. We then took a sample of the majority class to match that of the minority class. Lastly, we combined the two dataframes back to having a more balanced dataset. This process resulted in a balanced training data set of approximately 880 thousands instances.

Feature Engineering

We created four new features to more accurately represent the data using a combination of mathematical approaches and business understanding.

- We created a click-through rate for each day of the week ('*CTR_day*') and then each hour of the day ('*CTR_hour*') and multiplied them to give us an interaction term to highlight how popular a certain day/time is.
- To highlight competitors, we multiplied the column of competitor availability and that of competitor price comparison with Expedia. We counted the number of competitors who won against Expedia's result, such that the competitors had room availability for the specific dates and a lower price.
- We also calculated the base click probability for each search result with the intuition that if fewer results appeared in a search, it would increase the probability that a result gets clicked. The base click probability is calculated by 1 divided by the total number of results appeared in a search.

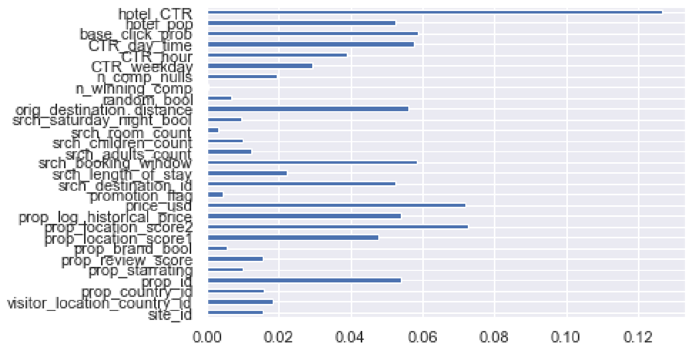
- Lastly we created a variable that indicated the popularity of a hotel. We did this by looking at how many times a hotel appeared in a search and merged that with how many hotels are in each country. We used the following formula to calculate popularity:

$$Popularity \ = \ Total\ \#\ of\ results\ of\ Hotel\ X\ within\ Country\ X\ /\ Total\ \#\ of\ results\ within\ Country\ X$$

Feature selection

After finalizing our base variables by cleaning and transforming them, we decided to use logistic regression and random forest to run feature selection and find the most important variables in the dataset that will accurately help us predict whether a user will click on a hotel listing or not.

For the Random Forest we set up a sequential feature selector that implements forward selection and backward elimination to form a feature subset. At each stage, this estimator chooses the best feature to add or remove based on the cross-validation score of an estimator. We used a cross-validation technique and evaluated our model using ROC-AUC to see which set of features yielded the best AUC. After running this feature selector we were able to extract the feature importance and combined that with the ROC-AUC for each feature which better guided our feature selection process.



Alongside, we ran a binomial logistic regression model and also used that to calculate the Variance Inflation Factor (VIF) to detect any multicollinearity within our variables. We observed that the variables - *CTR_weekday* and *CTR_hour* had a VIF higher than 5 and decided to remove them from our analysis since we already had another variable - *CTR_hour_day*, that described the datetime aspect for us. Lastly, we also built a lasso regression model which provides feature selection and recommends to drop certain variables whose coefficients shrink to low values, close to 0. Our L1 model dropped the following variables since they were not important - star rating for property, booking window from search to stay, and null values for Expedia competitors. Now that we had all our variables in place and had determined which ones we wanted to use for our analysis, we moved onto the modeling process using GridSearch to optimize the hyperparameters and provide us the best score in terms of ROC and AUC.

Modeling

We have a supervised learning binomial classification problem. We decided to build multiple different models using various algorithms, chose the optimal features for each algorithm, conducted hyperparameter tuning for each model and based on the results, picked the best model for our business problem. For all models we used forward selection which allows us to add features one by one till we get an ideal F-1 score and AUC. Using this we were able to use the optimal features for each algorithm and also see which features had the highest predictive power. For hyperparameter tuning we incorporated cross-validation using a stratified k-fold cross-validation in a GridSearch. We chose to use this method to ensure that the model is not biased towards a certain class in the training dataset.

Logistic Regression

We started our analysis by running a Logistic Regression. We chose to start with a logistic regression because it is a simple yet effective way to analyze results. It gives quick predictions which can be used in real-time to handle clicks. Because we have a relatively large dataset, it is cost effective and suitable for this problem as it shows improved sensitivity to class imbalance problems. The response variable is a log function of odds of an event happening taking on a value between $-\infty$ and $+\infty$. We optimized the hyper-parameters using a GridSearch method and tried several different solvers. For the logistic regression the optimal estimators are C=1, penalty='l1', solver='liblinear'. Using these parameters we ran the model on the training and test set and got an F-1 score of 0.62 and an AUC of 0.65.

Neural Network

We wanted to run more advanced techniques based off of what we've learned in class and so we decided to proceed with a Neural Net. We know that the outcome is modeled by the hidden layers where each hidden layer has some weight parameter associated with it. In order to be able to model non-linear data we usually use a nonlinear function such as logistic to be able to introduce this aspect in the data. We started by creating a model that had 3 hidden layers - 2 layers using an activation function of 'relu' and the last layer had an activation function of 'sigmoid'. All of these layers were compiled together and optimized using the 'adam optimizer' which is very similar to stochastic gradient descent and iteratively updates the weights of the nodes. We also used binary cross-entropy to measure the loss between predicted and actual values since it is the ideal method for a binary classification problem.

Finally, we also used gridsearch to find the optimal values of the parameters such as the batch size, number of epochs, and active units. Since the model required a substantial amount of time to run for any required size of sample, we were not able to get the final evaluation score for this model. However, from running the baseline model, we received an AUC score 0.50.

Support Vector Machine

Support vector machines (SVM) try to find the line that best linearly separates the data such that it is farthest from the closest training point in each of the two classes. While SVM can be slow on large datasets, we still chose to implement a SVM because it works well for extreme case binary classification. Again, we utilized GridSearch to get the optimal parameters; kernel=”rbf”, class_weight=”balanced”, gamma=”auto”, C=1, tol=0.001, decision_function_shape=”ovr”. Using these parameters we got an F-1 score of 0.68 and an AUC of 0.61. We had to run SVM on a sample of the data because the model required a substantial amount of time to run even on smaller samples of the dataset.

Random Forest

Random forest is an ensemble method where a large number of relatively uncorrelated individual decision trees are created, each individual tree giving a class prediction. The class with the most “votes'' will become the model prediction. The key to random forests is the low correlation between the models. Uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. Random forests are a strong modeling technique and much more robust than a single decision tree. They aggregate many decision trees to limit overfitting as well as error due to bias and therefore yield useful results. For the random forest, the optimal parameters were criterion=“entropy”, max_depth=10, max_features=“log2”, n_estimators=100 giving us an F1-score of 0.71 and AUC of 0.69.

XGBoost

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. We wanted to use it because it has roots in decision trees which are simple to explain to non-technical stakeholders. The difference between this and regular decision trees is that XGBoost grows trees that are sequentially trying to reduce error from previous iterations, i.e. the next iteration of a model is trying to solve the previous iterations errors. We tuned this using a GridSearch cross-validation but had to limit the number of iterations and sample size initially as the runtime was too long. Using this, the optimal parameters were objective=”binary:logistic”, booster=”gbtree”, eta=0.1, max_depth=5, subsample=0.8, colsample_bytree=0.8, eval_metric=”logloss”, giving us an F-1 score of 0.71 and an AUC of 0.79.

Model	Motivation	F-1 Score	AUC
Logistic Regression	Simple model that can handle nonlinear decision boundaries	0.62	0.65
Neural Network	Capture non-linearity within dataset	N/A	0.50
Support Vector Machine	Works well with large datasets but requires a lot of computational power	0.68	0.61
Random Forest	Aggregating decision trees to limit overfitting	0.71	0.69
XGBoost	Performs really well but can be difficult to tune	0.74	0.79

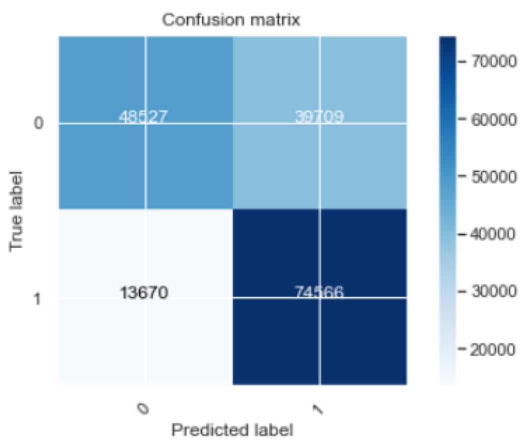
Model Evaluation

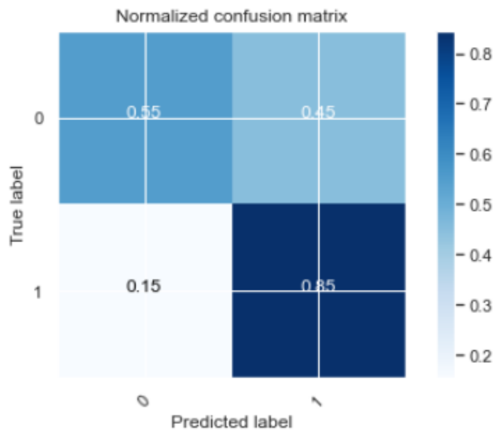
We chose to evaluate our models on F-1 score because F1 acts as the harmonic mean of Precision and Recall. We also chose to look at the area under the curve (AUC) because it very simply measures the ability of a classifier to distinguish between classes. Based on a five-fold stratified cross validation, we have one model that stands out on both these metrics.

	Parameters	Features	F1-Score	AUC

XGBoost	objective = binary:logistic
	booster = gbtree
	eta=0.1
	max_depth=5
		hotel_CTR
		base_click_prob
		prop_location_score2
		promotion_flag
		price_usd
			0.74	0.79

	subsample=0.8
	colsample_bytree=0.8
		prop_starrating		





We found that using ensemble methods was crucial in creating a model that effectively discriminated between the two classes. XGBoost performed the best and we believe it did so because gradient tree boosting models are robust to outliers and hence efficiently deal with class imbalance problems. Our model accurately predicted someone clicking on a hotel with 74% F1-score. This can help improve Expedia's ability to better suggest hotels to customers and in turn boost revenue.

Expedia can use this model to extract the most informative features that tell you what exactly customers are looking for in hotels and what makes them click on one hotel versus another. From our model we saw that;

- **Popularity:** Hotel click-through rate (*hotel_CTR)* was the most informative attribute. This variable tells us which hotels are most clicked on and so we can generate a list of top hotels clicked on and see if there are similarities amongst them. For example, if we see a lot of hotels from International Hotel Group (IHG) or Hilton Group, then Expedia can partner with such hotels to have a better selection on their page - increasing revenue for both Expedia and the hotel group.
- **Price:** Next we see that the price (*price_usd*) is an important factor when people are booking their vacations. Nobody wants to pay more than they absolutely have to for a hotel, but getting the best price has historically meant checking dozens of websites to compare pricing, and even then you couldn't be certain that you didn't miss a better deal. By understanding the customer's

willingness to pay and combining that with promotional information, Expedia can maximize revenue from each customer by providing the best price and product pair.

- **Quality:** Lastly, the number of stars (*prop_starrating)* a hotel has is also a predictor for whether they will click on it or not. This can be used by Expedia as a quality assurance metric for allowing new hotel groups or new independent hotels onto their page. We know that the most commonly searched rating is 3 stars so adding more hotels in that portfolio could also help Expedia boos revenue.

Deployment

Based on the CRISP-DM, we suggest that Expedia continue running these models in an iterative manner. This could be running the models to keep learning the most important features people are looking for in hotels, but also matching hotels to people's preferences using Collaborative Filtering. We recommend Expedia establish an experimentation framework and A/B test the performance of the model against objective metrics. As well as ensure that enough tracking is available to accurately debug and evaluate model performance a posteriori.

While our model performed reasonably well with this limited dataset, we do think it could be hard to generalize the results in the real world at this stage. Expedia could use this data to further study whether the people that clicked on a hotel booked that hotel or not to further understand what factors go into selecting a hotel. Expedia could do a cost-benefit analysis from using a cost-benefit matrix. In this case we hypothesize that predicting a booking but not actually booking is more costly for Expedia than predicting they don't book and they do. The cost comes from two sources; loss of a customer and potential profit loss. As we mentioned, the OTA business is a very competitive market so if a customer doesn't find what they're looking for they will take their business elsewhere. Expedia can lose potential revenue generated from this customer. One thing to note is that as Expedia's goals and business plans change, they will have to reevaluate the cost function to make it more accurately represent the business needs. Additionally the cost function can differ by time of year due to the seasonality of the travel industry. Seasonality is a measurable feature since it has significant economic importance and can be seen through the number of visitors, how much they are willing to spend, and the number of hotel rooms available. For example, someone may come to the Expedia page to book a hotel during the holiday season but there may be a limited inventory of hotels within their price range.

<u>Appendix 1</u> - New Features Created

Comp1: comp1_rate * comp1_inv; it represents the competition result between Expedia and competitor 1. The value of -1 indicates the competitor 1 is winning, where competitor 1 has a lower price than Expeida's result and has availability of equivalent room type. Comp 2 to comp 8 are the corresponding columns for competitor 2 to 8.

n_winning_comp: the number of competitors winning in the competition. In another word, the number of value -1 among the columns of "comp1" to "comp8".

n_comp_nulls: the number of nulls among the columns of "comp1" to "comp8".

CTR_weekday: the average click-through rate by weekday. It is an interim variable for the calculation of CTR_day_time and is not included in the modeling.

CTR_hour: the average click-through rate by the hour of day. It is an interim variable for the calculation of CTR_day_time and is not included in the modeling.

CTR_day_time: the product of CTR_weekday and CRT_hour. It refers to the clicking trend of a specific hour of a specific weekday.

srch_count: the number of results appeared in each search. It is an interim variable for the calculation of base_click_prob and is not included in the modeling.

base_click_prob: The base click probability of each result in a search. 1/srch_count. When there are only a few results in a search, the probability that a result gets clicked is higher.

hotel_pop: Total number of results of Hotel X within Country X / Total number of results within Country X