Expedia Hotel Recommendations using Machine Learning

**Team**: Shiv Lalapet, Alex Li, James Lu, Zander Tedjo

**Github**: https://github.com/alexlisa9/finalProject\_EE461P

Temporary Medium link: <https://medium.com/@zander.b.tedjo/expedia-hotel-recommendations-using-machine-learning-9a8eccd4ecba>

Abstract

We are trying to model hotel clusters as a function of user behavior, with reference to theExpedia dataset on Kaggle. We first do some data exploration and pre-processing, primarily in the form of heatmaps and visualizing the hotel clusters. We then tried multiple models and decided to go with K-nearest neighbors, XGBoost, and LightGBM. We used an evaluation metric called mean average precision with XGBoost coming in the highest at 0.4885. We finally evaluate our results to that of the winner of the Kaggle competition and discuss our next steps.

Introduction

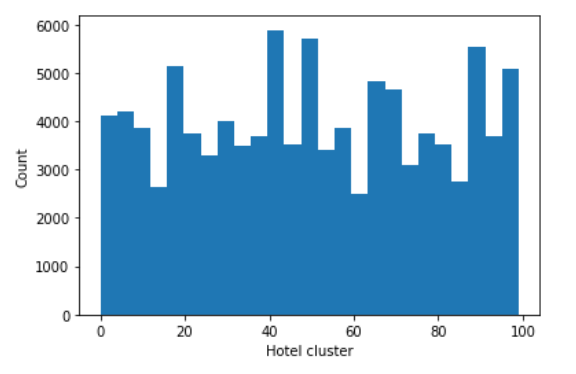
For our project, our group used data from a past kaggle competition hosted by Expedia. Expedia group is an online travel shopping company for consumer and small business travel. They are primarily known for their travel websites such as expedia.com and trivago. The goal of the project was to answer “Which hotel type will an Expedia customer book?”. Currently, Expedia uses search parameters to adjust their hotel recommendations, but there isn’t enough customer specific data to personalize them for each user.

Problem Description

The goal of this project was to contextualize customer data and predict, out of 100 hotel groups, which group the user would stay at. Each user would be given an ID, and based on the user’s data we would assign the user to a hotel cluster to recommend. Our provided data included data such as which website they used, the continent, whether or not they were on a mobile device and the user’s search destination. Overall, this was a multi-classification problem consisting of 100 classes.

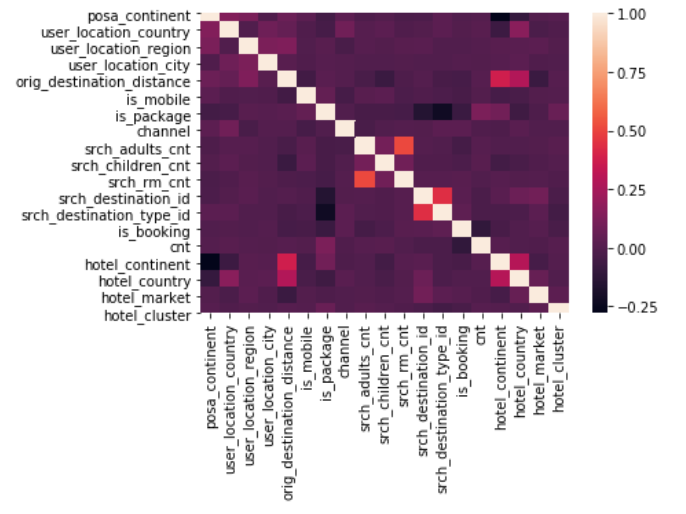
Exploratory Data Analysis

We started with 19 features, of which 5 are numerical and 14 are categorical. In Figure 1, we have the distribution of hotel clusters, which is the target class label. The hotel clusters range from 1-100, and from the graph we can see that they are not evenly distributed.



*Figure 1 - Target Class Distribution*

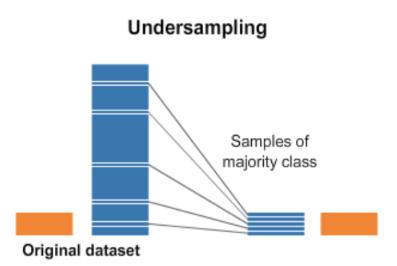
In Figure 2, we have the correlation heatmap. From this graph we can see that most features are not correlated, but there are a few that are. For example, srch\_rm\_cnt and srch\_adults\_cnt seem to be correlated as well as hotel\_continent and orig\_destination\_distance. This is important to know during feature selection to reduce the collinearity problem.



*Figure 2 - Correlation Heatmap*

Preprocessing

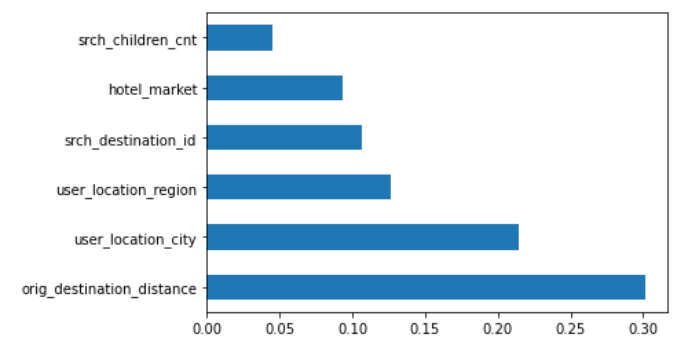
One part of our preprocessing was reducing the size of our data, which originally had 30 million rows. Since we had a very large dataset, we were able to remove samples that had missing information and still have plenty of data to work with. We also removed unnecessary features that were not relevant to solving our problem. We then resampled the data to get an equal distribution of target class labels and to create a smaller subset of the data so that we could process it more quickly due to limitations on computational resources.



*Figure 3 - Undersampling*

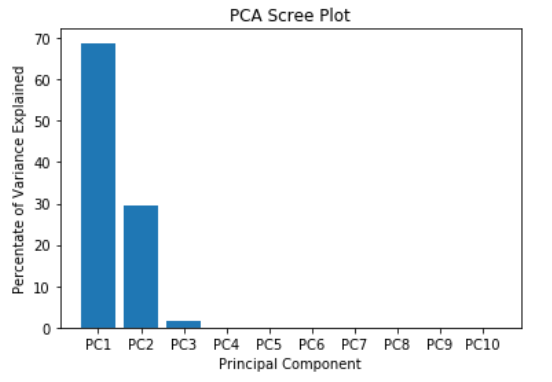
Feature Selection/Extraction

Next, we experimented with feature selection using chi sq and feature extraction using principal components. The figure below shows the feature importances. Orig\_destination\_distance is the most important feature. But we can see that almost all of the top features are related to location and geography. This makes sense because we are trying to predict the hotel cluster to recommend to the user.



*Figure 4 - Feature Importances*

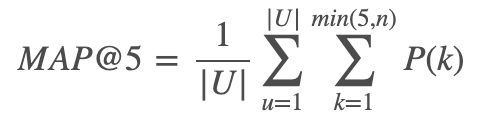
The graph below shows the PCA scree plot. Surprisingly, most of the information in our data can be represented by only three PCA components. This means that a lot of our features describe similar information.



*Figure 5 - PCA Scree Plot*

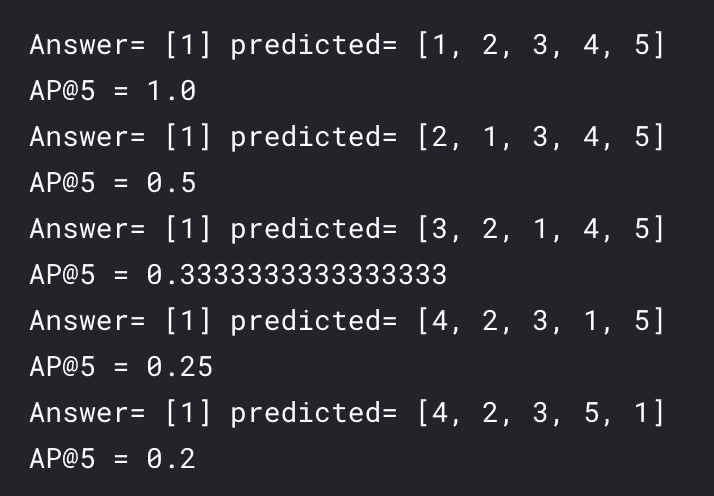
Accuracy Evaluation

The accuracy metric we used is Mean Average Precision @ Level 5. The formula for this metric is



where |U| is the number of user events, P(k) is the precision at cutoff k, n is the number of predicted hotel clusters.

Essentially, this metric evaluates the top k=5 class predictions for each input, and the earlier our target class is in the list, the higher the value of this metric is. This is different from other popular multiclass classification scoring formulas, such as the ROC AUC, because we can receive a positive score even if our target class isn’t the most likely prediction - as long as the target is somewhere within our top five predictions, we will receive some form of credit. In our models’ evaluations, we also included the accuracy rate and AUC scores for comparison.



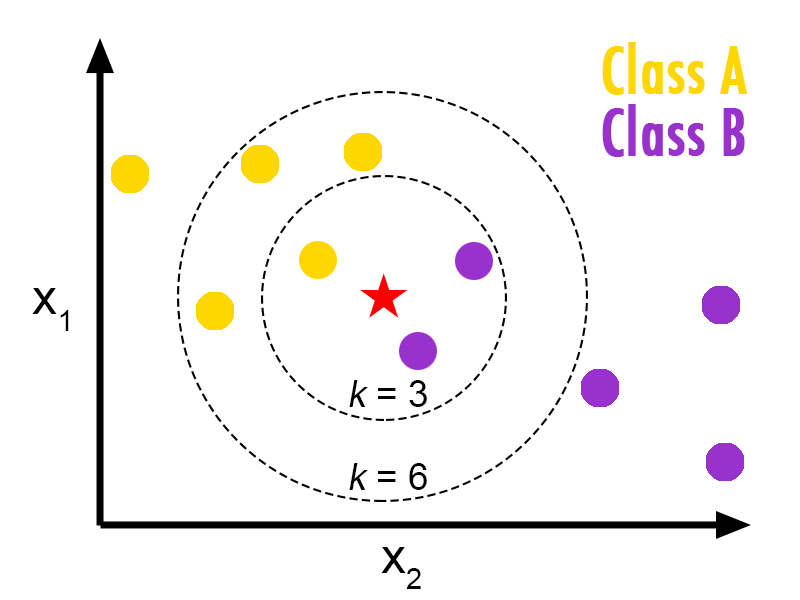
*Figure 6 - Mean Average Precision Breakdown*

This example shows the answer (target class) and list of predictions of a model. The target is 1, and the earlier 1 is in the predicted list, the higher our average precision is.

This method for evaluating accuracy gives a higher score if the target class is closer to the front of the predicted class list, and gives lower (but still positive) score if the target class is further from the front. MAP is a popular scoring system for recommendation engines - like Netflix or this project we did - because engines often output a list of what they think the user likes; if what they like is captured by our list, then we can say that our recommendation engine has been successful.

K-Nearest Neighbors

The first model we tried was K-Nearest Neighbors (KNN) because we wanted to see how a simpler multiclass model would perform in order to have a baseline for our next models. KNN is a supervised learning algorithm that assigns an input to the class that’s most common among its neighbors. The parameter K controls how many of the input’s closest neighbors the model takes into account, hence the name “K-nearest neighbors”

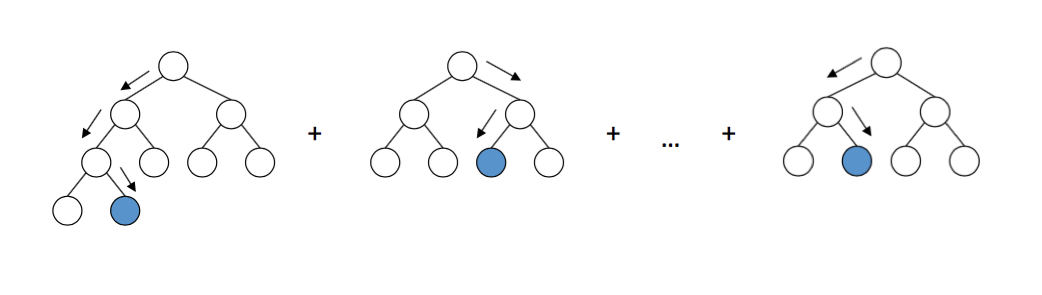


*Figure 7 - K-Nearest Neighbors*

We used cross validation to find our model’s best value for K, which we found to be equal to 5. With this parameter, we received an accuracy rate of .058, an AUC of .524, and a MAP of .225.

XGBoost

The next model we tried was XGBoost, which is a more complex model that uses gradient boosted decision trees. XGBoost combines weaker decision trees into a single strong learner iteratively using gradient descent.



*Figure 8 - Gradient Boosting Decision Trees*

We tuned a few of the parameters using cross validation, and these were the we values we found:

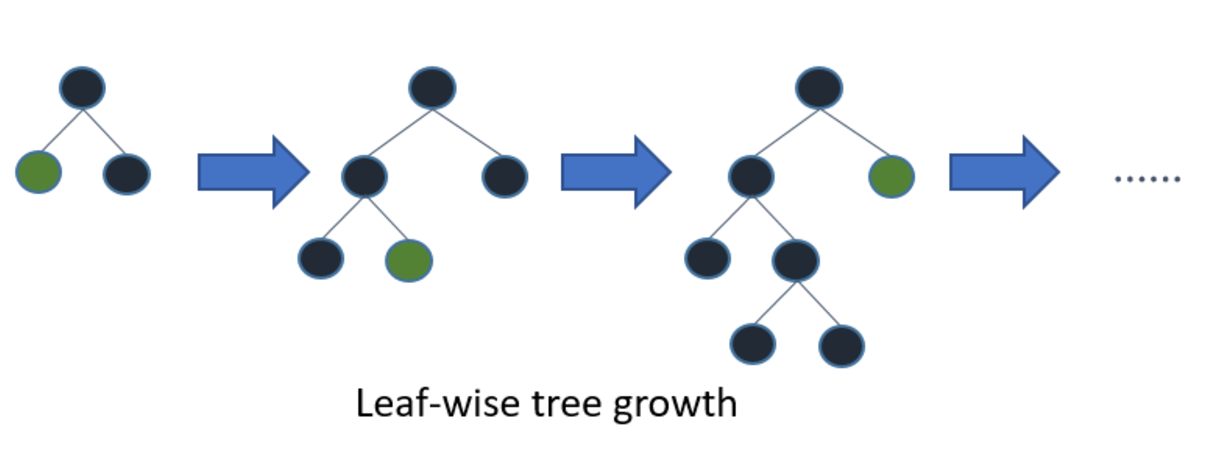
* n\_estimators = 400 (number of gradient boosted trees constructed)
* learning\_rate = 0.2 (boosting learning rate)
* max\_depth = 3 (maximum tree depth for weak learners, controls overfitting)
* min\_child\_weight = 1 (minimum sum of instance weight needed in a child)

There are several more parameters that can be tuned, but given the complexity of the model and our time constraints, these were the most optimal parameters we could provide. With these parameters, we received an accuracy rate of .151, an AUC of .571, and a MAP of .488.

While our accuracy rate is only 15%, keep in mind that the only metric we care about is the mean average precision. MAP takes our top 5 predictions in account, while the accuracy rate is calculated by only our top prediction.

Light GBM

Considering the dataset that we are using is substantially large, we decided to also create a model using Light Gradient Boosting. Light GBM uses a greedy decision tree algorithm that splits its decision tree in a leaf-wise manner to create a better fitted tree which is different from other boosting algorithms, like XGBoost, which splits its tree based on depth.

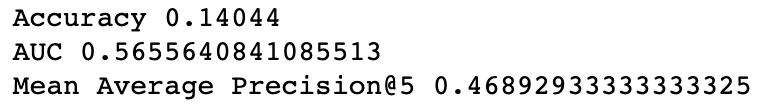


*Figure 9 - Leaf-wise Splitting Algorithm*

Light GBM is particularly useful for training large datasets at a much faster speed and is, on average, more accurate than other boosting models. However, in order to ensure peak accuracy while avoiding overfitting, we had to tune the hyperparameters as follows:

* Num\_leaves = 30: number of leaves to be formed in one tree
* Min\_data\_in\_leaf = 100: minimum number of data in one leaf

Setting num\_leaves to a value higher than 30 and min\_data\_in\_leaf to a value substantially larger than 100 would have led to an overfitted model. On the flip side, if num\_leaves were set to a value significantly lower than 30, the model may lose a substantial amount of accuracy. After we tuned more hyperparameters, we achieved good results from the evaluation of the model. The best results from the Light GBM model were:



*Figure 10 - Light GBM Results*

Evaluation

Best Mean Avg. Precision:   
 1. XGBoost: 0.4885  
 2. LightGBM: 0.4689  
 3. K-Nearest Neighbors: 0.2251

Of all the models we utilized, Xgboost performed the best, having the highest mean average precision of 0.4885. Cross-referencing our precision score with the other entries to the Kaggle competition, we would have placed slightly outside the top 50%. However, the difference in our precision is around .01 when our score is compared to those in the top 50 and top 10, and is significantly better than the precision score of the bottom quartile (circa 0.3082). By this metric, it is evident that we got close to our goal, but our model could still use further improvement.

Next Steps

The next steps would be to continue making improvements to the models. We could spend more time fine tuning the hyperparameters to increase the accuracy of the models, but this may not be the best way to improve the numbers significantly. Using more data could prove worthwhile, but the drawback will be an increase in time to run models, and given our computational resources we would need to upgrade these resources to realistically pursue this avenue. Additionally, we could take the route of creating an ensemble of models, containing some combination of Xgboost and a couple different models.

There is, however, another option that is guaranteed to improve results but is complex - we could follow the route that the winner of the Kaggle competition took. A few adjustments that we could look into incorporating would be, first, grouping features that are used uniquely for identifying user and hotel locations (for example, they grouped the features “userlocationcountry”, “userlocationregion”, and “userlocationcity” into a Tuple unit which was then used as a more precise input to their model). Secondly, they also used an implementation known as the Library for Field-aware Factorization Machines (LIBFFM), which has been used to win recent click-through rate prediction competitions. This implementation allowed them to create a separate factorization machine for each of the 100 clusters using the categorical features. Lastly, they also morphed this problem into a xgboost “rank:pairwise” problem in which the features input into their model was a combination of historical booking and click rates, scores from the LIBFFM model and a few more complex factors that would output their own unique method for scoring clusters.

In order to go this route, more research on the elements of their plan is required, in order to customize our model(s) heavily.

References

1. E. Rencberoglu, “Fundamental Techniques of Feature Engineering for Machine Learning,” *towardsdatascience.com*, April 1, 2019. [Online]. Available: <https://towardsdatascience.com/feature-engineering-for-machine-learning-3a5e293a5114>.
2. Kaggle, “Expedia Hotel Recommendations,” *kaggle.com*, 2016 [Online]. Available: <https://www.kaggle.com/c/expedia-hotel-recommendations/data>.
3. P. Khandelwal, “Which algorithm takes the crown: Light GBM vs XGBOOST?”, *analyticsvidhya.com*, June 12, 2017. [Online]. Available: <https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/?fbclid=IwAR2RKaO1nUU4wztJk9LUIT3HLQVwiX6On-2_dvuca1Gjq_ZCsEMC5LNaDEk>.
4. Scikit Learn, “Multiclass and multilabel algorithms,“ *scikit-learn.org* [Online]. Available: <https://scikit-learn.org/stable/modules/multiclass.html>.