[Become a member](https://medium.com/membership?source=upgrade_membership---nav_full------------------------)

[Sign in](https://medium.com/m/signin?operation=login&redirect=https%3A%2F%2Fmedium.com%2F%40wesleyklock%2Fexpedia-hotel-recommendations-ea6a9d5fbaa7&source=post_page-----ea6a9d5fbaa7---------------------nav_reg-)





**Expedia Hotel Recommendations**

[](https://medium.com/@wesleyklock?source=post_page-----ea6a9d5fbaa7----------------------)

[Wesley Klock](https://medium.com/@wesleyklock?source=post_page-----ea6a9d5fbaa7----------------------)

[Dec 16, 2018](https://medium.com/@wesleyklock/expedia-hotel-recommendations-ea6a9d5fbaa7?source=post_page-----ea6a9d5fbaa7----------------------) · 14 min read

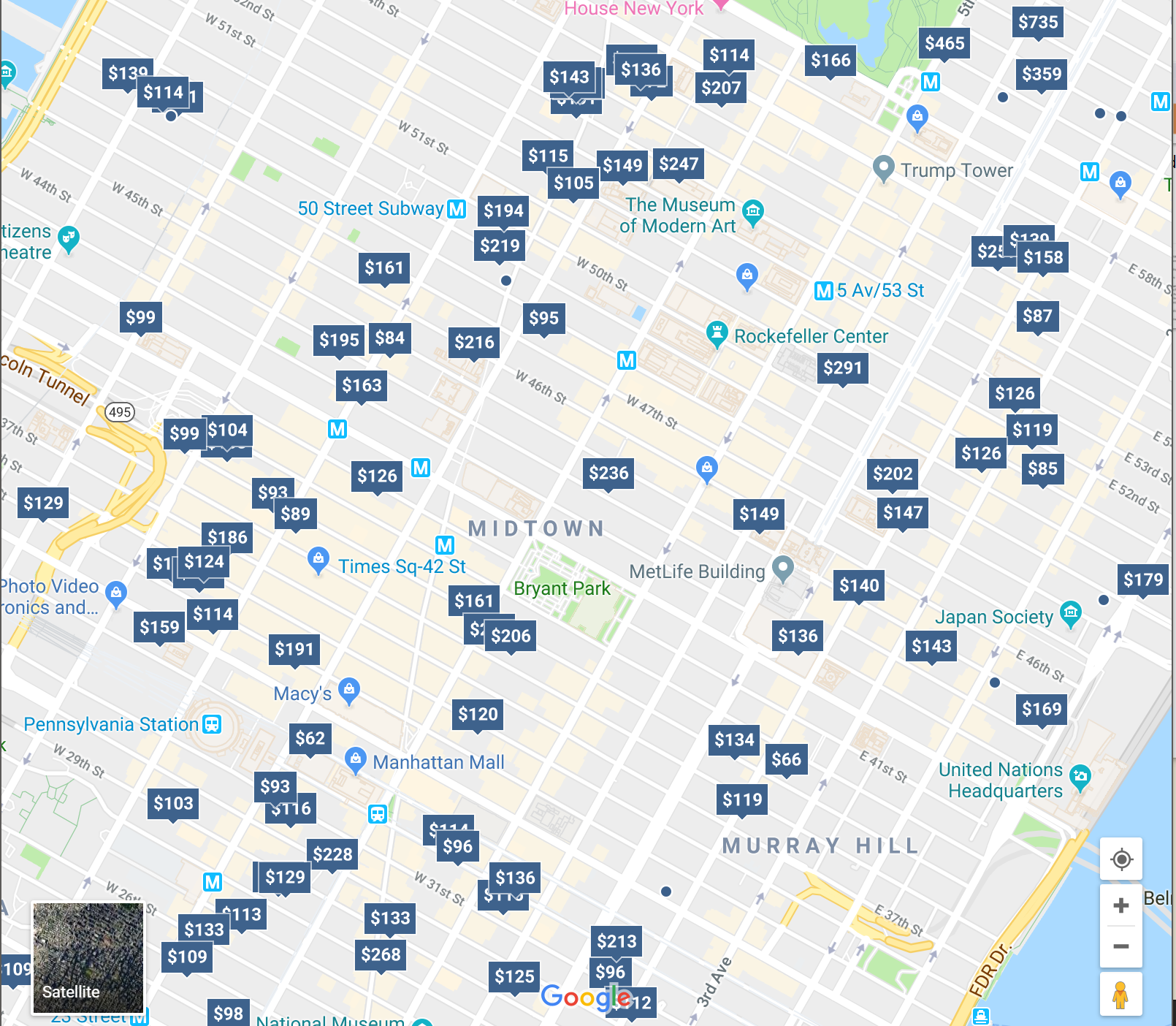
*Project by Genki Oji, Wesley Klock, Andriy Dashko, Emil Häglund*

In this article, we will show our approach to predict for Expedia users which hotel cluster they belong in. We first explain the motivation and parameters of the product. Then, we go over how we analyzed and combined the two datasets. Next, we go over models used. Lastly, we explain our results and conclusions.

**1. Motivations and Parameters 🌏**

Our main motivation for picking this project is our personal experience with finding and booking hotels. Between starred hotels, hostels, bed & baths, boutiques and more it is hard to know what is right place to stay. On my trip to NYC, I spent hours going through very different hotels.





The multitude of hotels in Midtown (Manhattan, NYC)

We wanted to build a model to help people find the type of hotel that is right for them. As a result we were pretty excited to find a [Kaggle Competition](https://www.kaggle.com/c/expedia-hotel-recommendations) hosted by Expedia that provided a great dataset to tackle this problem.

**Competition Scope**

Expedia makes a portion of its profit as a sales channel for many hotels (they take a portion of room payment when you find a hotel on Expedia) so they have an interest in creating good customize recommendations for users since they would be more likely to book through the platform. In that vein they created this competition to build a better engine to do this.

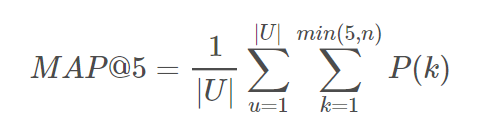
The goal was to contextualize customer data and predict the likelihood a user will stay at a 100 different hotel groups. So this is a large multi class classification problem.

The training data is from 2013 and 2014 while the test data is from 2015. This makes it such that any submission is actually evaluating future recommendations based on past recommendations.

**Submission Evaluation**

Submission are evaluated using the Mean Average Precision @ 5 metric.





Mean Average Precision @ 5

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

This creates a interesting twist on the competition in that it is much more important to get the correct answer in the top 5 predictions which allows for some margin of error than to always get the correct answer as the top choice.

**2. Analyzing the Data set and Pre-processing 🌎**



**Error! Filename not specified.**

Initial Search Information on Expedia Site

**A. Data Leak 🌊**

Before we went into how we approached the data we wanted to address one problem we encountered: in this Kaggle competition, some users discovered a data leak, which was later acknowledged by Expedia. This leak allowed people to match specific rows containing user and hotel location in the training data with those in the testing set, resulting in 33% of the test data being able to be predicted 90% of the time. In our solution, we decided not to exploit this data leak because we felt that exploiting this data leak would not add any value to learning and applying data science principles. As a result, our overall score would be lower than those on Kaggle, but we would be able to see how well we could predict the hotel clusters even without the data leak.

**B. Pre-processing**

The first step we took to manage our data was to down sample our data by a considerable amount. The original data consists of **37 million** rows which represent 1.2 million users, so we decided to down sample to 10 thousand users, which resulted in about **320 thousand rows**.

train.shape(319954, 27)

In our preprocessing, we wanted to address the fact that travel agents do not accurately represent a single user’s preferences. This is a preprocessing step that we did not find many solutions on Kaggle mention. To identify this noise from our data, we decided that any user that had over 20 bookings (in the two year timeframe from 2013–2014) would be considered a travel agent and will be removed from our dataset. This was around 7.8% of our downsampled dataset.

# Remove travel agents  
# Get all unique user ids  
unique\_users = train.user\_id.unique()# Remove all non-bookings to make counting easier  
t1 = train[train.is\_booking != 0]for user in unique\_users:  
   
 # Count the number of rows under a single user  
 bookings = len(t1.loc[t1['user\_id'] == user])  
 if bookings >= 20:  
   
 # Remove the travel agent from dataset  
 train = train[train.user\_id != user]

**Shape with travel agents: (319954, 27)  
Shape without travel agents: (295058, 27)**

From here, to make processing dates easier, we converted the date/time cells into datetime format through pandas. Since the original challenge required that we use data from 2013 and 2014 to predict users’ preferred hotel clusters in 2015, we further split our new set of 10,000 users into training and testing sets by year; 2013 and part of 2014 for the new training set and the rest of 2014 for the testing set. Furthermore, we wanted to mirror the original training and testing set for our downsampled data, so we removed any rows that were non-bookings.

**C. User Reviews on Destinations Dataset 🌄**

In the challenge, we are also given a separate dataset that represents user reviews. None of this data is given any meaning that can be easily interpreted, since all the columns are unlabeled. This actually works to our advantage, because it allows us to use PCA to reduce the dimensionality without really losing any interpretability — because there is none to begin with.



**Error! Filename not specified.**

Output of the PCA

To work with and visualize our data better, we also additionally extracted features such as **duration of stay, check-in day, and check-out month**.

**Missing Data**

To figure out what values to replace missing data points with, we visualized the following features:

* Number of bookings in each month
* Number of bookings per each specific day of the month
* Number of bookings per duration of the stay

Then we replaced the missing values in these features in every row with the area of the highest frequency.

**Correlation**

Lastly, we wanted to figure out some limitations when it comes to selecting models, so we visualized the correlation of every feature with out target feature, “hotel\_cluster”.



**Error! Filename not specified.**

Notice how no features are strongly correlated with the target variable (hotel\_cluster) even after some normalization attempts. This suggests that a linear method would perform poorly. With this in mind let’s move onto the next step: Modeling.

**3. Models 🌍**

When decided which models to try and optimize parameters for we did some initial research on what models perform well for a classification problem with a large number of classes. Each model section is broken down into 3 parts: Motivation, Creation and Tuning, and Results.

**A. Base model 🚗**

The baseline model was meant to evaluate whether machine learning was even a good fit for this problem. The most simple method that we could try (beyond just randomly selecting clusters for each user) was to always select the most common clusters among the training set.

The above code gave us a list of the 5 most common clusters and the MAP@5 score for this method was **.07213**. This provides a baseline value for which to evaluate the other models.

most\_common\_clusters = list(ytrain['hotel\_cluster'].value\_counts().head().index)  
predictions = [most\_common\_clusters for i in range(ytest.shape[0])]print(metrics.mapk([[l] for l in ytest['hotel\_cluster']], predictions, k=5))

Now that we have formed a baseline, we decide to explore some simple models.

**B. Simple Models 🎒**

**K-nearest Neighbors**

***Motivation:***

KNN was a good simple model to try because it ‘trains’ very quickly by offsetting most of the computation to the actual testing portion. Additionally it is relatively intuitive how the model works.

***Creation and Tuning:***

The KNN model was first trained on the down-sampled version of the dataset. Using a gridsearch the value of k was tuned to 23 nearest neighbors.

With the best estimator:

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',metric\_params=None, n\_jobs=1, n\_neighbors=23, p=2,weights='uniform')

***Results:***

After this model was tuned on the smaller dataset, the MAP@5 score was very low. This was most likely a result of KNN’s being poor at generalizing given only a portion of the dataset. After expanding our downsample to a larger segment we were able to get a MAP@5 score of **.12458**. Which is fairly low performance compared to models that were better at generalizing

**Logistic Regression:**

***Motivations:***

We chose to test Logistic Regression because it was a simple model we learned in class that can handle non-linear decision boundaries, which we were clearly dealing with.

***Creation and Tuning:***

We optimized the hyper-parameters using a GridSearch method and tried several different solvers

log\_params={  
 'solver':['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],  
 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],  
}log\_grid = GridSearchCV(LogisticRegression(max\_iter=200), log\_params, scoring=map5scorer, cv=3, verbose=5)  
log\_grid.fit(X\_train, y\_train['hotel\_cluster'])

Ultimately the best estimator was:

LogisticRegression(max\_iter=200, solver='lbfgs', 0.01)

***Results:***

Ultimately logistic regression performed relatively poorly compared to other methods we used with a MAP@5 score of **.10200** .This is mainly due to the weaknesses of Logistic Regression for handling outliers and it’s assumptions not holding true for this dataset (there is a linear relationship between the logit of the outcome and each predictor variable).

**Naive Bayes**

***Motivation:***

Naive Bayes is a relatively simple classifier that can natively be run on multiclass data, so we figured to at least try this type of classifier and see what kind of results we get initially.

***Creation and Tuning:***

We had fairly low expectations for this model because we were fairly sure the attribute independence assumption wouldn’t hold for this dataset. We used the Sci-kit Learn implementation of Naive Bayes called MultinomialNB and just fit a model with standard parameters.

***Results:***

Upon evaluating our first model with this classifier, we find it to be be even worse than the baseline model with a MAP@5 score of **.02690**. Thus, we opted to not do further tuning of the model.

**E. XGBoost 🚀**

***Motivation:***

XGBoost is a gradient boosting package popular in Kaggle Competitions. We decided to try using gradient boosting because it’s based on decision trees like random forest, but there a few key differences. Gradient boosting grows trees that are sequentially trying to reduce the error of previous iterations, so generally it should perform better and be more expressive than random forests with enough iterations. The downside to that is the algorithm is more expensive to run many iterations on with large datasets, so the performance increase over random forests can be hard to realize.

***Creation and Tuning:***

The XGBoost package was used to create our gradient boosting model, and it was trained on the downsampled version of the data because of runtime. It was tuned using a GridSearch cross validation on the parameters that control the growth of the trees, but the number of iterations had to be limited severely to get a reasonable runtime.

xgb\_model = XGBClassifier()parameters = {'tree\_method': ["gpu\_hist"],   
 'feval': ['map5eval']  
 'objective':['multi:softmax'],  
 'num\_class': [100],  
 'learning\_rate': [0.03, 0.05, 0.08, 0.1, 0.12],   
 'max\_depth': [4, 6, 8, 10],  
 'min\_child\_weight': [7, 9, 11],  
 'silent': [1],  
 'subsample': [0.6, 0.8, 1.0],  
 'colsample\_bytree': [0.6, 0.7, 0.8, 1.0],  
 'n\_estimators': [10],  
 'gamma': [0, 0.5, 1, 1.5, 2],  
 'seed': [42]  
 }clf = GridSearchCV(xgb\_model, parameters, n\_jobs=5,   
 cv=StratifiedKFold(ytrain.values.ravel(), n\_folds=5, shuffle=True),   
 verbose=2, refit=True)#Use X\_train, X\_train\_pca, X\_train\_trans(feature selection), X\_train\_combo(feature selection+pca)  
clf.fit(Xtrain, ytrain)

This gridsearch resulted in the parameters:

xgb\_params = {'tree\_method': 'gpu\_hist',   
 'objective':'multi:softmax',  
 'num\_class': 100,  
 'learning\_rate': 0.08,  
 'max\_depth': 8,  
 'min\_child\_weight': 9,  
 'silent': 1,  
 'subsample': 0.8,  
 'colsample\_bytree': 0.7,  
 'n\_estimators': 600,  
 'seed': 42  
}

***Results:***

Our XGBoost model performed second best behind the ensemble of random forest classifiers with a MAP@5 score of **.25289**. This is probably because the number of iterations during our tuning was severely hindered, so the optimal parameters selected may not hold for higher number of iterations. The final model uses more iterations than what was possible when searching through the parameter space, which improves results but not enough to beat our random forest classifier.

**D. Ensembling Methods 🌲**

***Motivation:***

While XGBoost offered a huge improvement over other models, this was only after some extensive tuning. With ensembling, we wanted to take an approach that required less hyper-parameter tuning. Ensembling methods like AdaBoost, Bagging, and Random Forest have several advantages for this type of problem. They are unlikely to over fit since the models are combining prediction from each model in a simple way which is a real issue when we are using such a big dataset (some properties trained might be very specific to 2013, 2014 and not so much for 2015). Additionally the models speed can be easily scaled by changing the **n\_estimators** hyper parameter.

***Creation and Tuning:***

Firstly, we tried several different untuned decision tree ensembling methods on the downsampled data. We quickly found that Random Forest performed the best and focused our efforts there. Each of the following variations of the RF model was tuned using a grid-search.

from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import GridSearchCVrf\_params = {  
 "criterion": ['gini','entropy'],  
 "n\_estimators": [10,50,100,200,500,750,1000,2000],  
 "max\_features": ['auto','log2',None]  
}rf\_grid = GridSearchCV(RandomForestClassifier(), rf\_params, scoring=map5scorer, cv=3, verbose=10)  
rf\_grid.fit(X\_train, y\_train['hotel\_cluster'])

**Basic Random Forest**

We first started with a basic random forest model and tuned hyper-parameters. This resulted in a MAP@5 score of **.10405**.

**Binary Random Forest**

One of the largest weakness of Random Forest Classifiers is large class imbalances. In addition with many classes, the decision trees become very deep and complex which goes against the marginal differences that RF averaging is going for. To mitigate both of these weaknesses, we created a new model that trains a binary classifier RF for each class (of 100) and then compares the predicted probability of each RF Classifier and selects the top 5. This acts as an ensembling of 100 RF Classifier which are each ensembles of 200 decision trees. This nesting of ensembles does drastically increase runtime compared to the standard RF but improved performance drastically.

The code for this custom model is shown below.

unique\_clusters = y\_train["hotel\_cluster"].unique()  
count = len(unique\_clusters)  
for cluster in unique\_clusters:  
 print(count, "remaining")  
 ytrain["target"] = 1  
 ytrain["target"][ytrain["hotel\_cluster"] != cluster] = 0  
 predictors = Xtrain.columns  
 probs = []  
 clf = RandomForestClassifier(n\_estimators=200)  
 clf.fit(Xtrain, ytrain["target"])  
 preds = clf.predict\_proba(Xtest)  
 probs.append([p[1] for p in preds])  
 full\_probs = chain.from\_iterable(probs)  
 all\_probs.append(list(full\_probs))  
 count = count-1prediction\_frame = pd.DataFrame(all\_probs).T  
prediction\_frame.columns = unique\_clustersdef find\_top\_5(row):  
 return list(row.nlargest(5).index)rf\_preds = []  
for index, row in prediction\_frame.iterrows():  
 rf\_preds.append(find\_top\_5(row))

**CV Binary Random Forest**

A further modification on RF was Cross Validating each of the Binary Classifiers to tune their hyperparameters individually. This would allow the RF model to tune more closely to the training set on which of the binary classifiers needed a higher or lower number of n\_estimators.

**rf\_params = {  
 "criterion": ['gini','entropy'],  
 "n\_estimators": [10,50,100,200,500,750,1000,2000],  
 "max\_features": ['auto','log2',None]  
}rf\_grid=GridSearchCV(RandomForestClassifier(n\_estimators=200)rf\_params, scoring=map5scorer, cv=2, verbose=10)**for cluster in unique\_clusters:  
 y\_train["target"] = 1  
 y\_train["target"][y\_train["hotel\_cluster"] != cluster] = 0  
 predictors = X\_train.columns  
 probs = []  
 **rf\_grid.fit(X\_train, y\_train['hotel\_cluster'])  
 clf = rf\_grid.best\_estimator\_**  
 clf.fit(Xtrain, ytrain["target"])  
 preds = clf.predict\_proba(Xtest)  
 probs.append([p[1] for p in preds])  
 full\_probs = chain.from\_iterable(probs)  
 all\_probs.append(list(full\_probs))

***Results:***

Our Binary Random Forest Model performed the best with a MAP@5 score of **.28141**. While we thought the CV version of the Binary RF would perform better, it actually performed slightly worse with a MAP@5 score of **.27751** despite having a higher training set score. This is probably due to the low number of Kfolds the CV used (due to speed limitations). By increasing it beyond 2 we could probably have increased performance marginally past the .28 threshold but would face an exponential increase in training time. Additionally the increasing the n\_estimators parameter on each of the Binary Classifiers could create a similar effect.

**F. SVMs and Factorization Machines 🌃**

***Motivations:***

Support Vector Machines (SVMs) and Factorization Machines (FMs) are good at finding pairwise interactions in data, so they should be good at recommending a hotel cluster to a certain user. Though natively, they don’t support multiclass classification, there are techniques we felt we could explore to make them viably classify one out of 100 hotel clusters.



**Error! Filename not specified.**

Second order FM equation

***Creation and Tuning:***

We used the Sci-kit Learn implementation of SVMs with a linear kernel and the fastFM package for factorization machines. Both required substantial time to run on even the downsampled data, and proper models would have to be an ensemble of 100 binary SVMs/FMs to classify the 100 different hotel clusters. Although they may have looked promising, SVM/FMs require too much computational resources to be viable for our project, even with AWS GPU. 😢 Regardless, it was a good opportunity to learn about FMs and hopefully we will get to use them again in the future.

**4. Results**

The TLDR version of the modeling we did is below:



**Error! Filename not specified.**

Results of the modeling section

**What do these numbers mean?**

However, the metric used, MAP@5 is fairly arbitrary when it come to evaluating the success of our model. Sure it could be said that our method is around 4 times better than the baseline method outline above, but if someone told you the MAP@5 score is .5, it is hard to know what that actually means.

It is more useful to look at the results of our best model in terms of placement of the correct cluster.



**Error! Filename not specified.**

Placement in Predictions over Percent of test dataset

From this graph we get much better sense of our performance using the Using our best model. Even the best model on the Kaggle Competition (that exploits the data leak) was unable to place 40% of the test data in the 1st recommendation.

Although we were able to perform comparatively well to the results of the Kaggle competition (with the data leak we would be able to place in the top 15% of the competition), the real-life application this model if far from perfect. Even when displaying the top 5 clusters (recommendations), only around **41.44%** of the time will the correct cluster (hotel type) would appear.

**5. Conclusions 🌏**

In conclusion, our resulting scores are much lower than those on Kaggle, which is expected because we tried to fit our models without exploiting the data leak. On Kaggle, people were able predict ⅓ of the data 90% of the time just from using the data leak.

We found that the ensembling of Binary Random Forests was effective in discriminating between a large number of classes. We believe that this model performed well because simplifying the problem into many binary classification problems allowed us to use random forest which is good for classification with non-linear feature relationships in a large multiclass setting.

Although in terms of the Kaggle competition our solution performed well, the real world applications of this are still lacking. In the future we would like to explore a different dataset or see if a time series implementation (using data on the browsing patterns of users as they navigate the site) might yield better results. Overall, this was a great experience where we learned how to optimize for a different evaluation metric and learned new things about models we went through in class.

Thanks for reading 😄!



**Error! Filename not specified.**

**References:**

**[Expedia Hotel Recommendations](https://www.kaggle.com/c/expedia-hotel-recommendations?source=post_page-----ea6a9d5fbaa7----------------------)**

**[Which hotel type will an Expedia customer book?](https://www.kaggle.com/c/expedia-hotel-recommendations?source=post_page-----ea6a9d5fbaa7----------------------)**

**[www.kaggle.com](https://www.kaggle.com/c/expedia-hotel-recommendations?source=post_page-----ea6a9d5fbaa7----------------------)**

**[Data preprocessing](http://www.cs.ccsu.edu/~markov/ccsu_courses/datamining-3.html?source=post_page-----ea6a9d5fbaa7----------------------)**

**[Incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data](http://www.cs.ccsu.edu/~markov/ccsu_courses/datamining-3.html?source=post_page-----ea6a9d5fbaa7----------------------)**

**[www.cs.ccsu.edu](http://www.cs.ccsu.edu/~markov/ccsu_courses/datamining-3.html?source=post_page-----ea6a9d5fbaa7----------------------)**

**[Introductory Guide - Factorization Machines & their application on huge datasets (with codes in…](https://www.analyticsvidhya.com/blog/2018/01/factorization-machines/?source=post_page-----ea6a9d5fbaa7----------------------)**

**[Introduction I still remember my first encounter with a Click prediction problem. Before this, I had been learning data…](https://www.analyticsvidhya.com/blog/2018/01/factorization-machines/?source=post_page-----ea6a9d5fbaa7----------------------)**

**[www.analyticsvidhya.com](https://www.analyticsvidhya.com/blog/2018/01/factorization-machines/?source=post_page-----ea6a9d5fbaa7----------------------)**

* [Machine Learning](https://medium.com/tag/machine-learning)
* [Data Science](https://medium.com/tag/data-science)
* [Hotel Booking](https://medium.com/tag/hotel-booking)
* [Expedia](https://medium.com/tag/expedia)
* [Kaggle](https://medium.com/tag/kaggle)