# The Automated Travel Agent: Hotel Recommendations Using Machine Learning

Michael Arruza, John Pericich, Michael Straka

June 6, 2016

**Abstract**

Expedia users who prefer the same types of hotels presumably share other commonalities (i.e., non-hotel commonalities) with each other. With this in mind, Kaggle challenged developers to recommend hotels to Expedia users. Armed with a training set containing data about 37 million Expedia users, we set out to do just that. Our machine-learning algorithms ranged from direct applications of material learned in class to multi-part algorithms with novel combinations of recommender system techniques. Kaggle's benchmark for randomly guessing a user's hotel cluster is 0.02260, and the mean average precision K = 5 value for naïve recommender systems is 0.05949. Our best combination of machine-learning algorithms achieved a figure just over 0.30. Our results provide insight into performing multi-class classification on data sets that lack linear structure.

## 1 Introduction

Recommending products to consumers is a popular application of machine learning, especially when there exists substantial data about the consumers' preferences. Kaggle has provided a dataset containing substantial information about 37 million Expedia users and has posed the challenge of recommending hotel clusters to these users.[1] The difficulty lies in taking user information that lacks linear structure and using it to recommend hotel clusters. Furthermore, the sheer volume of possible prediction classes also poses a challenge. Accordingly, this project focuses first on applying well-known machine learning algorithms to the dataset and then tweaking as well as combining these algorithms so that they produce the correct classifications on the dataset.

## 2 Related Work

A large amount has already been done and written about item recommendation systems. [3] A common technique for these systems is to use a user-item matrix combining features about the users and items along with user feedback for the items. However, these methods proved to be inapplicable for our project, as the anonymized nature of the target variables made it difficult to obtain relevant features for them. More applicable to our project, previous work has been done on hotel recommendation systems by GAO Huming and LI Weili, who showed good results using a combination of clustering and boosting algorithms. [5] While their results are not comparable to ours due to the large differences in the datasets used, it is notable that both their paper and ours show promise in using clustering and boosting for hotel recommendations.

## 3 Dataset and Features

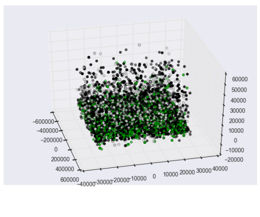


Figure 1: Three-dimensional PCA plot of the three most popular hotel clusters. Each is given a different color.

The data consisted of 37 million users and anonymized features including aspects such as number of children, number of rooms booked, destination visited, sate of check in and check out, location of user when booking, whether or not the booking was part of a package, etc. We were tasked with using this data to predict which hotel cluster the user was going to book into. Some feature engineering was used on the data set to extract more useful features. The date of check in, date of check out and date of booking were removed and replaced with length of stay (in days), month (valued from 1 to 12), year (valued 1, 2 or 3 as the data was from 2013 to 2014) and week (numbered from one to 52). The purpose was to discretize the dates into a format more amenable to our learning algorithms. It should be noted that the given data is known to have a data leak affecting roughly thirty percent of the test data and if used would boost our accuracy scores by roughly twenty to twenty five percent. However, for the purposes of this paper we opted not to utilize this data leak when testing our algorithms, as we felt that exploiting the leak would not yield interesting results generalizable to other problems. The size of the data set and limits on computational time also encouraged us to randomly sample subsets of the data to use when testing, as using the entire data set would have been prohibitively expensive.

For initial data visualization, we compressed the dataset into 3 dimensions using basic PCA dimensionality reduction. To reduce the visual clutter of the data, we only plotted the 3 most popular hotel clusters, as seen in Figure 1. This figure supported our hypothesis that user similarity would yield good results, as users who chose identical hotel clusters tended to be clustered together.

## 4 Methods

### 4.1 Baseline Methods

Our first algorithm used the Naive Bayes conditional independence assumption to rank hotel clusters.[2] This method outperformed the random benchmark put forward by Kaggle, and serves as our benchmark going forward for the relative success or failure of an algorithm.

Another simple method used was training an SVM. One issue we faced in applying SVMs to our problem was adapting the typically discriminative SVM algorithm to generate probabilities, which can then be used to create a ranking of hotel clusters. We ended up using a method described by Wu et al. relying on pairwise coupling of single class predictions, learning the class probabilities using cross-validation. [4]

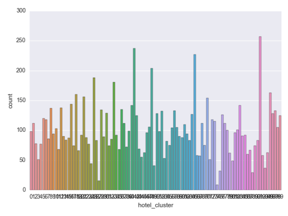


Figure 2: The distribution of the target hotel clusters.

In selecting the kernel to use for our SVM, we avoided extensive experiments with linear kernels due to the obvious lack of linear separability in the data. Indeed, initial explorations were found to perform poorly, with a 7% Map5 accuracy. Using a polynomial kernel was considered, but this was found to be prohibitively expensive in terms of computational time required. The rbf kernel $e^{-\frac{||x-x'||^2}{2\sigma^2}}$ was favored instead, both because of the speed of pre-existing implementations, and because of the Euclidean norm's potential to be interpreted as a similarity measure, which we thought might perform well due to the hypothesized importance of user similarity. We first normalized the training matrix to ensure that this interpretation would be valid. Parameters $C$ and $\gamma$ were tuned using 5-fold cross validation, selecting the parameters that gave the highest Map5 accuracy.

The best parameters were found to be $C = 10^{-6}$ and $\gamma = 10^{-5}$. The model still underperformed, and it was found in practice that lower values of $C$ and $\gamma$ reduced the sensitivity of the model to the data, and made both the predictions and map5 accuracy converge towards predicting the top 5 most numerous targets.

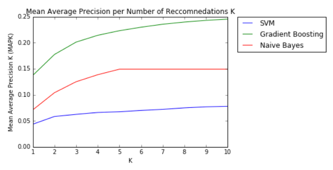


Figure 3: Mean Average Precision (using different values of k) for our baseline methods.

### 4.2 Weighted User Similarity

### 4.3 Gradient Boosting

Our first success came from using a form of ensemble tree boosting algorithm. This was chosen due to boosting's tendency to intelligently learn the non-linear structure of data. Our approach involved minimizing a softmax objective function to output a list of probabilities, one per class for a test user, which would then be used to get the top $k$ hotel cluster recommendations. We did this by using an off-the-shelf XGBoost package, which trains several classification and regression trees, at each step fixing the previous trees and adding a new one.

We experimented with several regularization terms and stepsizes, but found that they had no real impact on our results. For the number of rounds to train, we chose 6 as a reasonable trade-off between complexity of the model and time needed to train it.

Our most successful class of algorithms involve a novel combination of clustering, kernelized similarity and weighting distance, inspired by user similarity algorithms commonly used in recommendation systems. [3] First, we cluster all of training data based on the src destination id feature. The idea was that people who travel to similar destinations are more likely to book into the same hotel clusters. Originally we attempted to cluster based on hotel market (due to the relatively high correlation with the target hotel clusters, see figure 3) but this resulted in poor results, as the number of hotel markets represented in the data was too small,

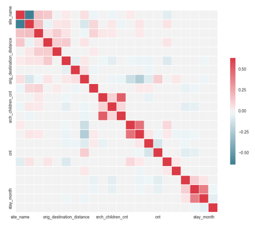


Figure 4: Pearson correlation matrix for the processed features.

producing clusters too big to be useful (the dataset contained 26543 unique values of destination ids, but only 2094 unique values for hotel markets). Once clustered together, we created a dictionary with src destination id as a key and the users in the training data that contain that id as a feature as the value. Once the dictionary is set up, we make predictions as follows. For each member of the test set, we hash into the group that share the same src destination id feature using the created dictionary. Once we have the group of training users, we then create a kernel matrix using some kernel function (many were attempted, and we discuss the ones we used later) and find the top 150 users in the group. Once we have these top 150 users, we give each user a score based on the similarity, and for each hotel cluster sum up the scores of the users in the top 150 who booked that cluster. The clusters with the ten highest scores are recommended for the user.

In measuring user similarity, our first attempt with cosine similarity performed relatively poorly, as certain features serve as numeric representations of qualitative information. For example, the continent ids for two users may be close together numerically, but that does not imply the two continents should be considered similar. Thus, we instead opted to represent similarity through jaccard similarity. Thus, to this end, we opted to utilize the kernel function

$$K_1(x, y) = \frac{1}{m}\sum_{i=1}^{m} 1\{x_i = y_i\}$$

Which counts the number of features shared. Using this kernel, we define the kernel matrix to be the vector K where the ith element of K is the result of the kernel function defined above. We then sort K and extract the top 150 users.

Now that we have the sorted top 150 users and their similarity scores for each according to the kernel function, we need to assign each hotel cluster a score based on the number of times it appears in these top 150 users and how similar these users are. Lets define V to be a vector with 150 elements containing the hotel clusters (in order) of the top 150 users. The first function that was attempted is as follows:

$$Score_1(cluster) = \sum_{i=1}^{1} 501\{V_i = cluster\}exp(-\frac{i^2}{2\tau^2})$$

Where tau is some constant (60 worked best for us) and $V_i$ is the hotel cluster of the top ith user. The basis of the equation above is to weigh users who are really close to the test user more than the users who are far away, but still heavily weigh clusters that appear more often in those top users. It is based on the equation for locally weighted linear regression and was meant to test the hypothesis that similarity could be used to effectively make recommendations. As shown in figures 5 and 6, this method outperformed our benchmark by a noticeable amount, and made us move towards exploring other similar algorithms and scoring functions.

One important flaw with the previous scoring algorithm is that it weighs based on placement in the list of top 150 clusters. It does not account for what the difference between those elements are or how close they actually are to the test user; the ith place in the list is always weighed the same. To remedy is, our second attempt with kernel methods used the same kernel function, but the scoring function for each hotel cluster was redefined as follows:

$$Score_2(cluster) = \sum_{i=1}^{1} 501\{V_i = cluster\}K_i^{'e}$$

where $K'$ is defined to be the sorted K matrix (which, since we do this for each individual test example, can be considered a vector), and e is some constant. The larger e is, the more we penalize farther values (as the defined kernel function gives values strictly between 0 and 1). We tested several values, and found 5 to be the most effective. Using this scoring method based on the raw similarity score instead of merely placement, we saw a very large improvement in the accuracy of our predictions across all metrics, and this became the basis of our most successful algorithm.

We briefly digressed from our algorithm to attempt a different kernel method. For this kernel method, we attempted to split the feature vector into two vectors, one comprised of discrete features corresponding to qualitative data (country, location, user ids, etc.) and another vector comprised of more continuous and quantifiable data (length of stay, user distance from location, number of children, etc.). Lets call these two vectors split from an arbitrary vector x x' and x". We also normalize x" to have a norm of 1. Using this notation, we defined the new kernel function to be:

$$K_2(x, y) = c_1K_1(x', y') + c_2x''^Ty''$$

Where c1 and c2 are some positive constants such that c1 + c2 = 1. The idea behind this kernel was to account for similarity in continuous features when

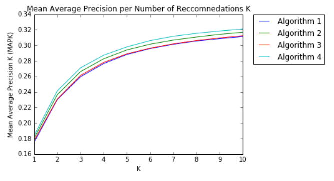


Figure 5: Mean Average Precision (using different values of k) for our kernel methods. The algorithms are numbered in order of discussion in this section.

weighing user similarity. The rest of the algorithm remained unchanged, to the
second iteration, but overall performed worse regardless of the parameters used, suggesting that similarity based on jaccard distance is more representative. Armed with this knowledge, we returned to the previous algorithm.

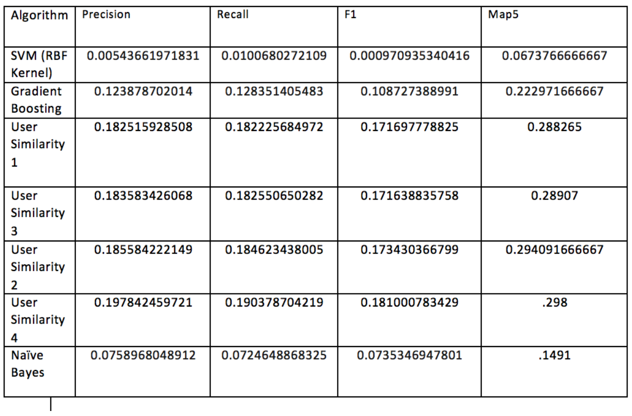
Our most effective algorithm, based off of the second kernel algorithm, was very similar, with one main difference. We took advantage of the fact that the hotel market and is package features are very highly correlated with hotel cluster (as can be seen in the correlation matrix in figure 3), so we performed the same algorithm but used a slightly modified kernel function:

$$\frac{1}{m+1}\sum_{i=1}^{m}(1\{x_i = y_i\} + 2 * 1\{x_i = y_i AND i = hotel\_market\_feature\} + 1.5 * 1\{x_i = y_i AND i = is\_package\})$$

Which weighed the hotel market feature twice as much as other features, and is package 1.5 times as much when accounting for the jaccard distance. Using this kernel method and the same algorithm as before, we attained the highest total accuracy on our predictions

## 5 Results and Discussion

Overall, the best methods were the methods that utilized user similarity and kernels to recommend hotel clusters that other similar users booked. Gradient Boosting was also effective, but mean average precision seemed to hit a hard cap at .25 regardless of the parameters used or size of the dataset sampled. As seen in Figure 3, these methods also took longer to converge given increasing values of $k$, suggesting the predictions are more nuanced. The SVM performed very poorly, as did other basic machine learning methods attempted on the data.



| Algorithm | Precision | Recall | F1 | Map5 |
|---|---|---|---|---|
| SVM (RBF Kernel) | 0.00543661971831 | 0.0100680272109 | 0.000970935340416 | 0.0673766666667 |
| Gradient Boosting | 0.123878702014 | 0.128351405483 | 0.108727388991 | 0.222971666667 |
| User Similarity 1 | 0.182515928508 | 0.182225684972 | 0.171697778825 | 0.288265 |
| User Similarity 3 | 0.183583426068 | 0.182550650282 | 0.171638835758 | 0.28907 |
| User Similarity 2 | 0.185584222149 | 0.184623438005 | 0.173430366799 | 0.294091666667 |
| User Similarity 4 | 0.197842459721 | 0.190378704219 | 0.181000783429 | .298 |
| Naïve Bayes | 0.0758968048912 | 0.0724648868325 | 0.0735346947801 | .1491 |

Figure 6: Precision, recall, f1 and "MAP5" scores for each algorithm across the chosen subset of the data. (MAP5 refers to mean average precision over 5 recommendations)

## 6 Conclusion and Future Work

This project provides an excellent case study for applying machine learning algorithms to large data sets lacking obvious structure. It also embodies the challenge of recommending items about which we have no features. In addressing these challenges, we demonstrated that a creative combination of user similarity matrices and Jaccard similarity outperforms gradient boosting—a technique currently well-known for winning Kaggle competitions. For future work, we recommend using ensemble stacking methods to combine predictions from various algorithms.

## References

[1] *Expedia Hotel Recommendations.* 15 April 2016. Web. https://www.kaggle.com/c/expedia-hotel-recommendations

[2] Pedgregosa et al. "Scikit-learn: Machine Learning in Python." *JMLR* 12, 2825-2830, 2011. 21 May 2016. Web.

[3] Jure Leskovec, Anand Rajaraman, Jeffrey D. Ullman. *Mining Massive Datasets.* Cambridge: Cambridge University Press, 2014. Print.

[4] Wu et al."Probability Estimates for Multi-class Classification by Pairwise Coupling." *Journal of Machine Learning Research* 5 (2004): 975-1005. Web. 28 May 2016. Cambridge: Cambridge University Press, 2014. Print.

[5] GAO Huming, LI Weili."A Hotel Recommendation System Based on Collaborative Filtering and Rankboost Algorithm." *Second International Conference on MultiMedia and Information Technology* (2010): 317-320. Web. 21 May 2016.

[6] "Introduction to Boosted Trees. *XGBoost.* Web. 21 May 2016.