CIS 520 Fall 2020, Final Project

Expedia Hotel Recommendations

Team Members: Team BYG

- Louise Yi; Email: `loyi@seas.upenn.edu`
- Valerio Galanti; Email: `vgalanti@sas.upenn.edu`
- Alon Benhaim; Email: `alonb@math.upenn.edu`

Abstract

This project aims to analyze and predict booking outcome based on user's searches on Expedia. We used the dataset consisting of 37.7 million samples of user interaction provided during the 2016 Expedia Hotel Recommendation Kaggle competition. We formulated this problem as a supervised multi-class classification problem. Because of the considerable size of the dataset, we used feature selection heuristics and down-sampling techniques to extract optimal data subsets for our machines to train on. We applied and evaluated each model's performance using the Mean Average Precision @ 5 method. We then explored the data leak exploited in the top-ranking submissions which significantly outperformed other, more traditions models seen in this course. Among our other methods, ensemble methods such as random forests and gradient tree boosting provided the most accurate results on held-out testing data.

1 Motivation

With the rise in hotel booking and vacation rental platforms, personalized recommendation systems have become a core strategy to attract and retain customers. Companies such as Expedia stand to gain tremendously from the use of recommendation systems, as targeted advertisements have the potential to incite new users onto the platform, and retain existing ones. Financially, better suggestions can lead more users to book hotels and Expedia to therefore receive their commission for helping a hotel find new clients. We found this problem particularly interesting as it could help us better understand the behaviour and patterns of users on such websites, and to explore the aspects guiding their booking decisions. In addition, the data set provided by Expedia is very interesting as it provides clusters where similar hotel for a search based on historical information and a variety of features including geographical and search criteria. We will hence consider applying a variety of machine learning methods to this problem, exploring similarity methods such as KNN, traditional multiclassification methods such as Logistic Regression or boosting, and finally ensemble methods such as Random Forests and boosting that have gained significant traction in online Machine Learning competitions.

2 Related Work

Considering the financial implications of personalized hotel recommendations, research on the matter is extensive [1–6]. Beyond academia, they have also become a popular topic in online competitions on Kaggle, for instance, where companies will set a challenge for users in hopes of improving on their own approaches.

Decision trees, KNN and Naive Bayes have been some of the most popular methods to predict the booking outcome. Aditi [1] suggested the hindrances in Naive Bayes to achieve good performance can be attributed to the fact that most features are one-hot vector encodings or suffer missing values. Cingel's team [7], on the other hand, noted that decision tree classifers have the advantage of being extremely efficient although being subject to over-fitting. As a result, their team selected an optimal subset of features counterbalancing the downside by reducing model complexity. Heng concluded that decision trees had a higher accuracy however it was not an ideal model because the precision result for various hotel cluster was significantly different ranging from 40 to 90 percents. KNN might be a great approach as it showed no variation among different clusters however KNN generally presented a relatively poor performance among multiple groups. [7] Multinomial logistic regression have been used as initial attempts by a few teams [8]. But there are concerns over logistic regressions due to its weakness for handling outlines and the assumption of linearity in the logit for variables and absence of multi-collinearity might not hold.

A few groups modelled this problem as a multi-class classification problem and build variables of classic support vector machines. Support Vector Machines are able to find pairwise interactions in data, which makes them well suited for recommending hotel clusters at the expense of significant computational resources. Xudong [2] worked on this problem as part of ICDM 2013 challenge with a different dataset containing additional features such as hotel prices, ratings, purchase history and OTA information. They were able to conclude that, in their particular dataset, price, rating and location features were among the most important features in determining user behavior. Their team applied random forests, gradient boosting, extreme randomized trees, factorization machines and deep learning for their predictions, concluding that combination of ensemble methods and individual ranking models showed the overall best ranking accuracy. On the other hand, Gao Huming [9] developed a hotel recommendation system based on collaborative filtering methods of clustering and rankboost algorithm which avoid cold-start and scalability problems existing in traditional collaborative filtering. In order to do so, they computed hotel rating matrices and Pearson correlations as a measure of user similarity to generate recommendations. [9].

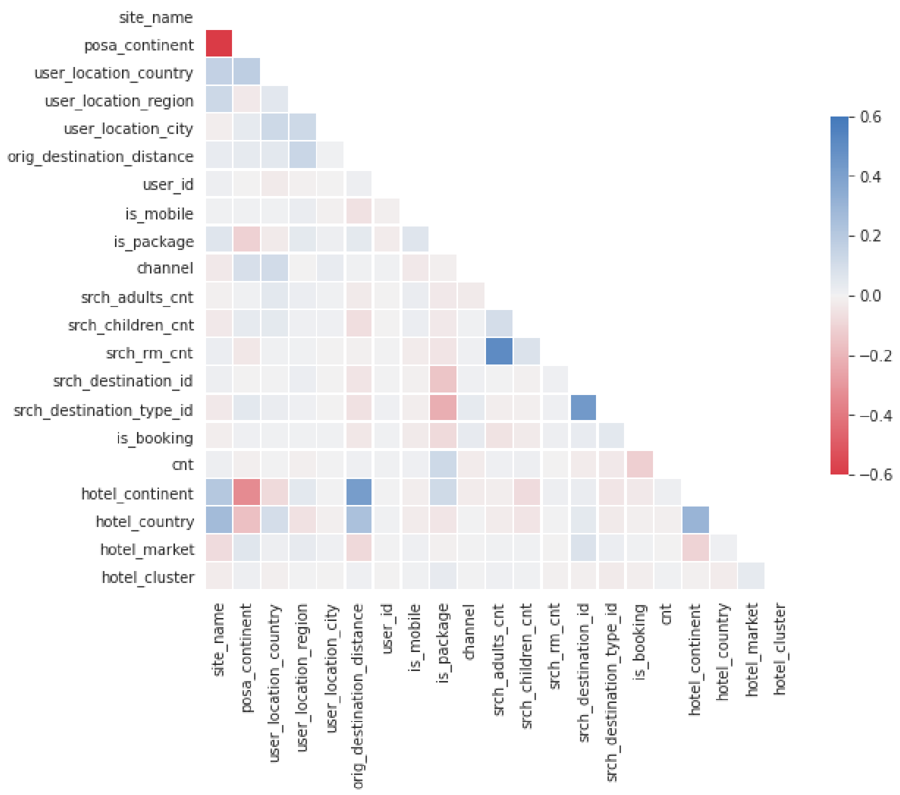
3 Dataset

3.1 Description

The dataset we are using was provided in the 2015 Expedia Hotel Reccommendations Kaggle competition and can be found at the following link: kaggle
The training set consists of around n = 37.7 million samples of user interaction with the website, each with an associated p = 24 features informing, among other things, about the user's current location, location of destination, medium used to access the website (i.e. mobile/desktop), search parameters, and whether that interaction led to a successful booking in one of the indexed hotel clusters. Similarly, the provided test set is similarly formatted, containing of n = 2.53 million user interactions. Besides the fact that the testing set does not contain labels that Kaggle wishes to keep for its internal model accuracy computations, the testing set consists of only searches that resulted in bookings, and thus foregoes the use of a boolean feature to indicate whether the search resulted in a booking.

3.2 Data Processing, Feature Selection, and Imputation

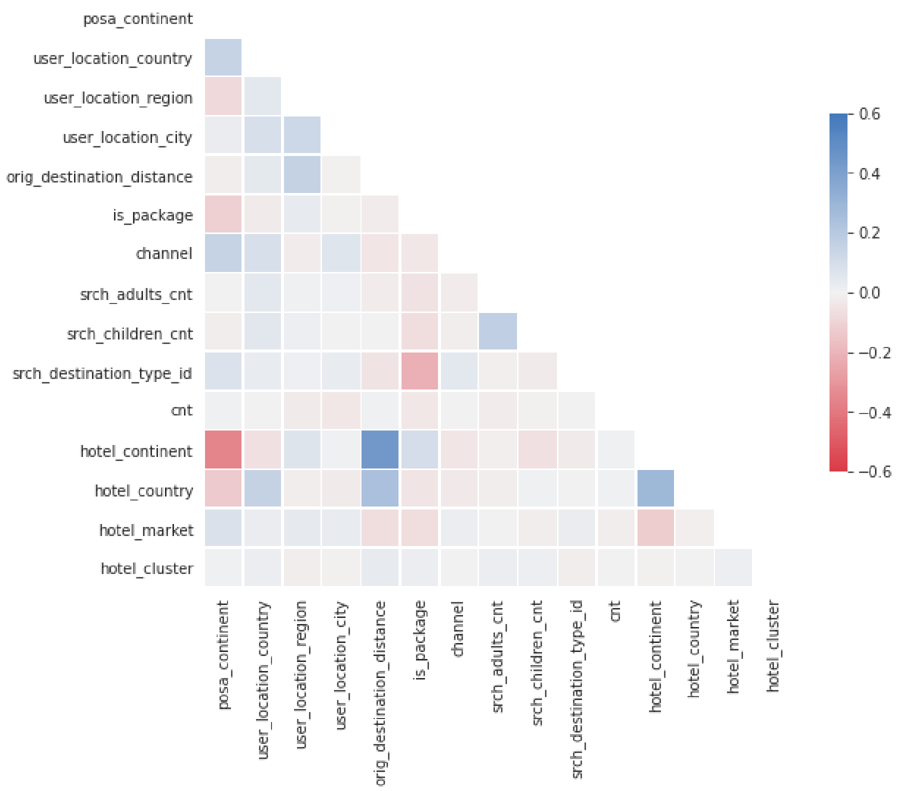
Since we are dealing with a large data-set that (free) colab memory does not allow us to load in its entirety, we had to consider various methods of feature reduction and selection in order to maximize the performance and feasability of our methods. To do so, we aided ourselves of both correlation maps as well as feature importance metrics as evaluated by random forest/gradient tree boosting methods. When considering the raw training set, we first obtain the following correlation heatmap as computed by the pandas and seaborn libraries:



As such, we observe that some features such as *site_name* and *posa_continent*, which is defined as the "ID of continent associated with *site_name*," are highly correlated with one another without being as correlated with our target variable, *hotel_cluster*, and could thus be omitted for training. In fact, we can remove both of these in favor of keeping simply *hotel_continent*. Similarly, it makes intuitive sense that *srch_adults_cnt* and *srch_rm_cnt* would be strongly correlated, as the number of adults involved in a booking is directly proportional to the number of desired rooms. We observe that $search_destination_id$, as well, is highly correlated with *search_destination_type_id* that is much more descriptive feature, having only 10 possible values instead of the 60k+ of *search_destination_id*'s.

Additionally, we dropped sessions that did not result in a booking, as we are only interested in predictions for those that do. The Expedia-provided test.csv file explicitly requires it, and does not even contain an *is_booking* feature. This approach is reasonable, considering that user behavior is likely to differ between those seriously searching for a place to stay or, say, browsing for resorts in the Maldives to distract themselves from their cold February in Philadelphia. Similarly, many of the unsuccessful sessions might result from the user not finding what they are looking for and moving on to a competing website, and should thus perhaps not be mimicked. Another benefit of removing all non-booking entries is that it significantly reduces the number of samples to train our models on, given that only about 8% of the given user-interactions (or roughly 3 million sessions) resulted in bookings. Other features we might consider dropping are *user_id*, for instance. New *user_id*'s are constantly created and, by themselves, are equivalent to noise for our classifiers.

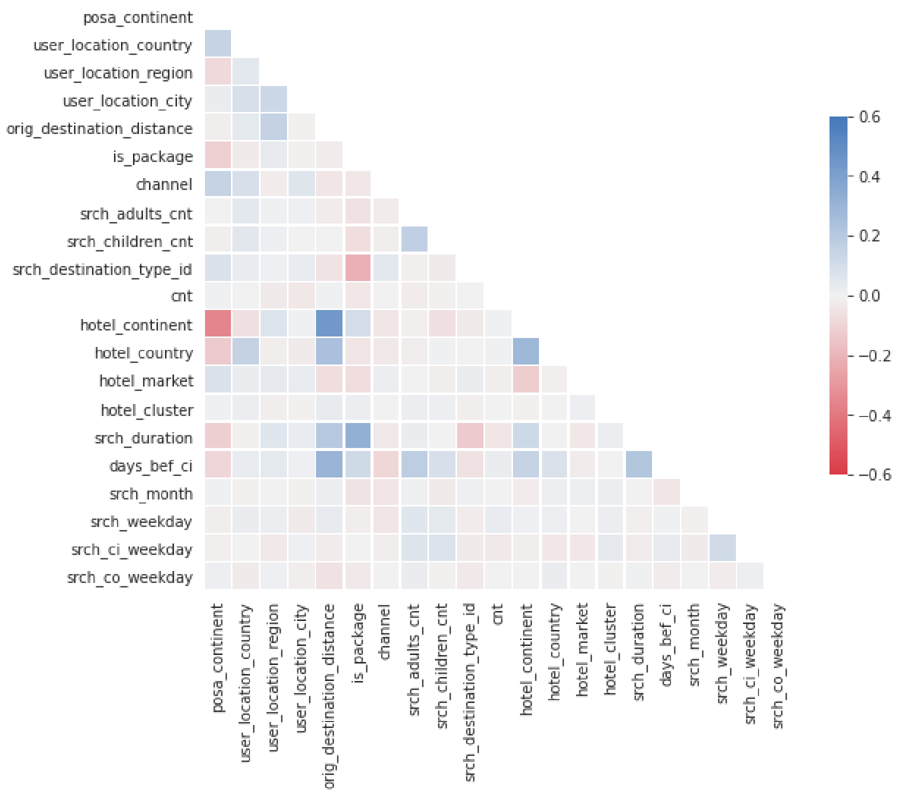
Once we have dropped those columns and kept only the entries resulting in a booking, we find a new correlation heatmap with less strong correlations that are not as straight-forward to eliminate.



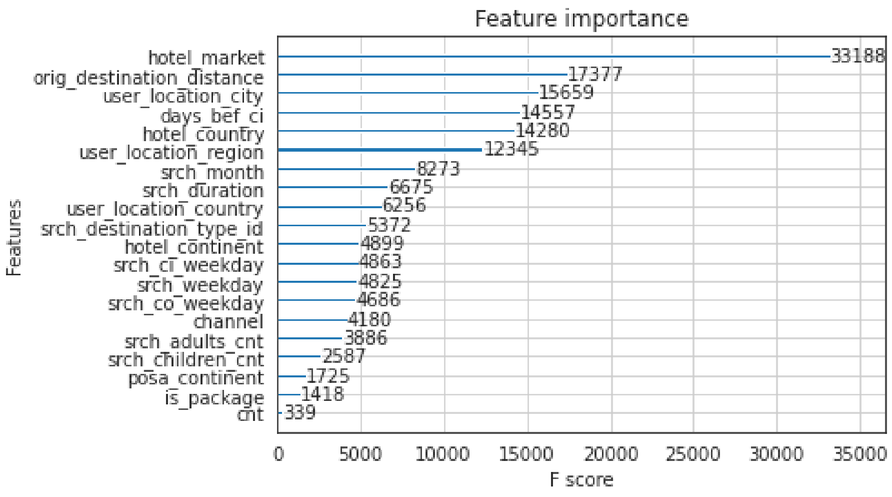
Additionally, we observe that in their current representation, dates, i.e. *date_time*, *srch_ci* (check-in), *srch_co* (check-out), are not of much use for our classifiers, and we would need to consider other embeddings.

For instance, the *date_time* field represents the current time of the search on Expedia. The exact date itself may not be the most relevant as people tend to book vacations, for instance, at different minutes and hours of the year. However, we can capture the general idea by instead keeping track of the month of the search. This embedding would better help capture the season and thus the type of hotel and destination for a portion of the users. Winter bookings might be more biased towards ski-resorts than they would in July, for example. In a similar fashion, the *search_ci* and *search_co* values, checking and checkout dates, are linked and can be merged into a single value indicating the duration of the stay, *srch_dur*, which can be better compared between users. In order to not lose information about the date the user is aiming to book the hotel, we ought to also add a field indicating the time between the current search and the checkin date, which we encode in $days\_bef\_ci$.

Another consideration is that the weekdays in which the search and bookings are made can potentially give us information about the purpose of the trip. Business trips will likely last less, be booked during weekdays, and consider different hotel clusters than say a summer family vacation. Perhaps some of these intuitions will be proven wrong, but we can verify this when considering feature importance estimations given by our models, in particular random forest methods. Adding these new features, we obtain a new correlation map, which indicates we are now only considering more subtly correlated features for our classifiers to pick up patterns on.

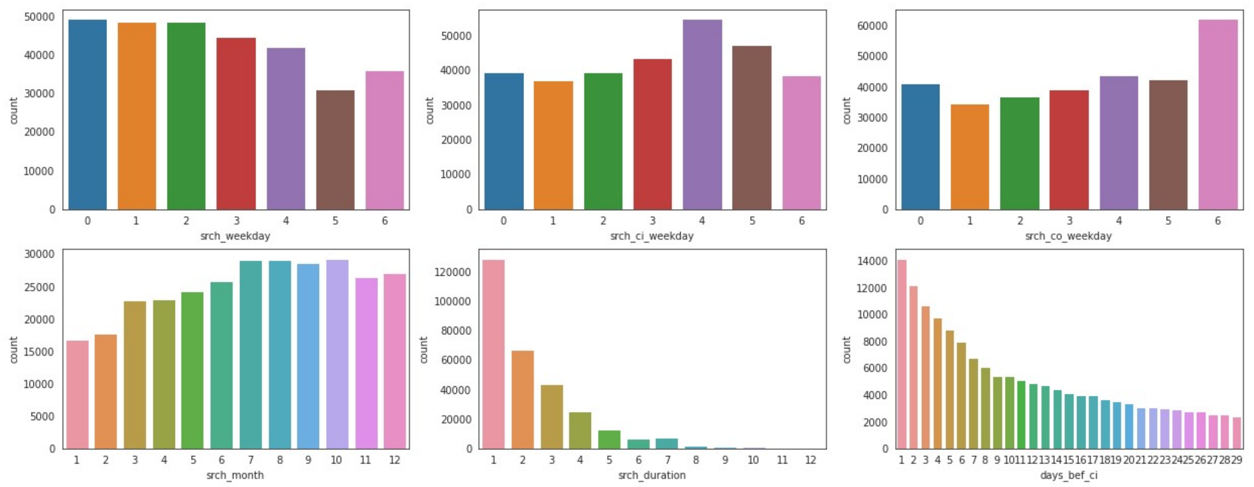


A final point of concern for this dataset was the significant amount of missing values for the *orig_destination_distance*, the geographic distance between user and destination. Of the 37.7 million entries, 13.5 million of them are missing a value for this feature and, as we will see later in evaluating our methods, information on location constitutes perhaps the single-most important feature in predicting *hotel_cluster*. We can therefore try to impute the *orig_destination_distance* which is missing the most entries of all the columns. We did so by considering the different imputations methods seen in class, i.e. zero, mean, most frequent, and especially regression-based imputation. We then combined all of these methods to clean the data we evaluated our models on. Initial testing showed that training XGBoost's Gradient tree boosting on 30% less, but feature-engineered and regression-imputed, data as above, improved the map@5 accuracy from 19% to 24%, giving the following feature-importance chart consolidating our choice of feature manipulations.

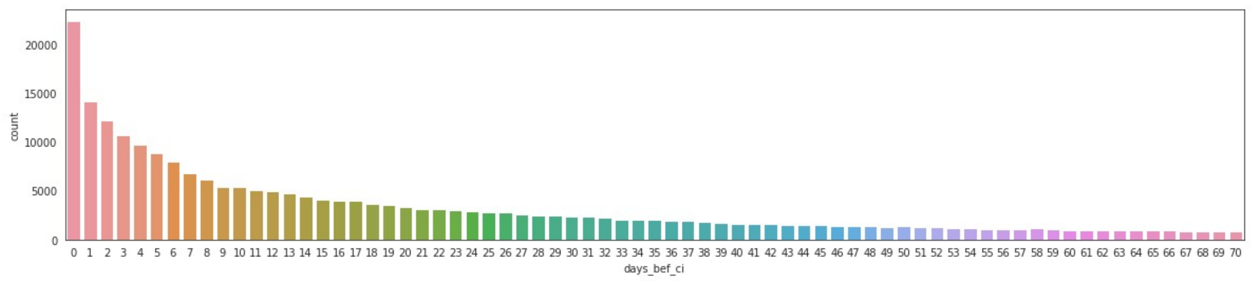


3.3 Summary statistics and visualizations

Perhaps the most interesting and understandable, in our opinion, part of this dataset were the temporal aspects of the different bookings given in the data. Below we provide a few plots on the interactions that resulted in bookings presenting the weekdays, months on which the searches were performed, the desired checkin and checkout days, the stay durations, as well as the number of days in advance that the bookings were made at.



This last plot in particular we find interesting because of how it illustrates the vastly different approaches users have to booking their stays and its variance's potential implication on classification performance.



4 Problem Formulation

The goal is to predict the likelihood a user will stay at 100 different hotel groups, given past user events on the website Expedia that include event attributes and the hotel group that the user searched for or booked in this event.

Formally, we are given a feature matrix $\mathbf{X}$ in which each row describes a user event and the columns are the features associated with this event and a set of labels $\mathbf{y}$ that corresponds to the hotel group that the user searched or booked. There are 100 hotel groups hence each label is in the set $y_i \in \{1, \dots, 100\}$. We formulated this hotel recommendation problem as a supervised multi-class classification problem, we want to determine an optimal classifier $h(x)$. We believe that in order to alleviate the problem, in their Kaggle competition, Expedia allowed to predict an ordered list of 5 hotel groups for each user event. Meaning given a user event $\mathbf{x_i}$ predict an ordered list of size 5: $[y_i^1, y_i^2, y_i^3, y_i^4, y_i^5]$. Hence, our problem is formulated as a supervised multi-label classification problem. A regular evaluation of multi-class classification can be done using simple accuracy metrics, that is, a correct prediction is an exact match between the prediction and the actual label but in a multi-label setting this would be a harsh evaluation metric. In this regard, Expedia suggested the Mean Average Precision @ 5 (MAP@5) loss function:

$$MAP@5 = \frac{1}{|U|}\sum_{u=1}^{|U|}\sum_{k=1}^{5} P(k)$$

where $|U|$ is the number of user events predicted and $P(k)$ is the precision at cutoff k. This means that we predict 5 ordered hotel groups for each user event. We get higher score the closer to the front of our guesses the correct cluster is, and zero score if the correct hotel group is not present. We decided to use this loss function to evaluate our methods since it was suggested by Expedia to alleviate the problem and because it would make it easy to compare our model to the leaderboard.

5 Methods

Our baseline method consisted always predicting the 5 most common clusters in that order and evaluates to an map@5 score of 0.020. We chose this to be our baseline because it is a simple heuristic that essentially guesses better than random but is still far from optimal for this problem. Because we are considering a supervised multiclass learning problem in which it is necessary to predict class probabilities in order to give our top 5 suggestions for the map@5 metric, we considered implementing the following methods:

- Baseline - most popular vote

  Our initial baseline is a relatively simple model that considers the most commonly booked hotels in the training set and repeats them for every new sample it sees. Because of its nature, it is likely to allocate one of its top 5 suggestions to the correct cluster for a new sample and therefore positively contribute towards the total map@5 score.

- SVM

  SVM was one of our first considerations for this problem as it is a comprehensible result with flexible parameters/kernels and was shown in class to generally perform better than Logistic regression for classification. We opted to use Scikit-learn's implementation, but found that its quadratic runtime in number of samples proved intractable for our computational resources, and other faster SVMs such as LinearSVC did not offer probability-predicting capabilities.

- Logistic Regression

  Similarly as for SVM, we chose this method because of its inherently probabilistic predictions and classification performance. We implemented it using Scikit-learn's LogisticRegression and LogisticRegressionCV (for cross-validation), as well as part of an ensemble in sklearn's parallelized BaggingClassifier functionality. Unfortunately the latter also appears to be computationally infeasible for our data and resources.

- KNN

  Similarly as for other recommender systems, we considered using a user-similarity based approach using KNN. We first implemented it using Scikit-learn, but quickly realised that the library's inefficiency would make it impossible to run it. With a small portion of the dataset, the performance is fairly poor compared to other models. We then looked at implementing Cython wrappers, and finally settled on using Deslib's facebook FAISS wrapper for similarity-searching, which provided between 60x and 600x speed-ups compared to sklearn's methods. Unfortunately we have yet to be able to parallelize this method to run on the entire dataset, as we are running into issues with python's GIL preventing query threading.

- Naive Bayes Model

  Naive Bayes is a simple classifier, which we implemented using Scikit-learn's Gaussian and Bernoulli Naive Bayes implementations, as they would provide simple and quick models to run and compare our models to.Gaussian Naive Bayes is used in cases when all features are continuous with the assumption of normal distribution. Bernoulli Naive Bayes assumes all features are binary with the assumption of multivariate bernoulli distribution. However we had a fairly low expectations for this model because key assumption of independence among features wouldn't hold for our dataset.

- Random Forests

  Because of the considerable size of the dataset, the inherent notion of information gain from splitting different features, and their ability to be parallelized over a significant number of cpu-cores, random forests appeared to be the ideal candidate for this classification problem. We used scikit-learn's implementation and are transitioning to using XGBoost's more efficient library.

- AdaBoost

  In a similar reasoning as for random forests, we used Scikit-learn's adaboost library to implement AdaBoost forests to evaluate and compare its suitability to this problem compared to random forests.

- Gradient Tree Boosting

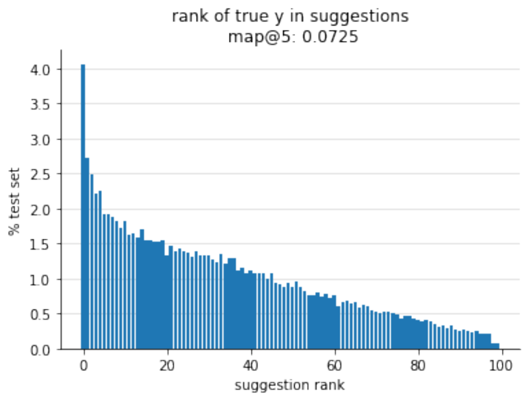
  Finally, we used XGBoost's gradient tree boosting implementations for this problem. These were particularly well suited for our application because of the library's tremendous support for efficient, flexible, scalable, and parallelizable implementations.

6 Experiments and Results

Beyond only using the map@5 metric for evaluating our methods that only keeps track of our top-5 predictions, we also implemented graphs that, for each method, count the number of times the true *hotel_cluster* booking was ranked as top choice number x along the x-axis. This allows us to gain a better understanding of how these different methods perform more generally and give an idea on possible convergence when used on a larger portion of the dataset. These statistics and visualisations are furthermore appropriate for this particular problem, as these rank statistics are particularly relevant for a service such as Expedia in which a user will likely see more than only the top 5 recommendations, but would be discourage to keep searching on the service past a certain threshold. All metrics and statistics given here are based on evaluation on 10% of the entire dataset, and can be reproduced using random seed 0, except for XGBoost because of its gpu-usage.

- Baseline – most popular vote

  This method performed surprisingly well in comparison to other methods we implemented, but its lack of specificity and proper feature-learning is clearly reflected in the linear progression of prediction rank for each true y. The corresponding images make sense considering that, if the testing set reflects the training set to some extent, the least occurring clusters will appear later in the suggestion rank of our baseline classifier.

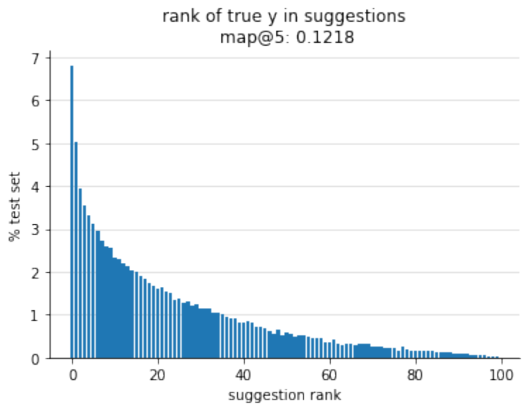


- SVM

  As mentioned previously, we were unable to run this method on our dataset given its runtime in number of rows.

- Logistic Regression

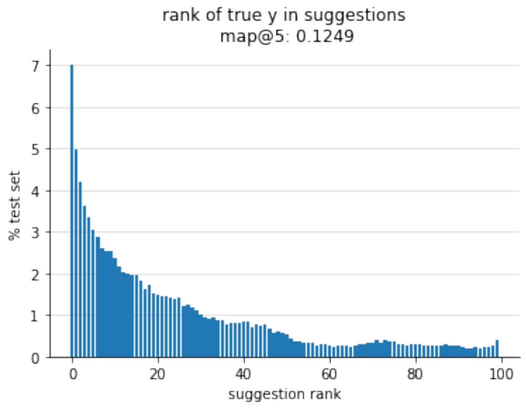
  In the case of Logistic regression, we use Scikit-learn's standard parameters and achieved the following results:



  In essence, our logistic regression classifier was able to correctly predict the true *hotel_cluster* as top choice nearly 7% of the time, 5% as second best-choice etc.

- KNN

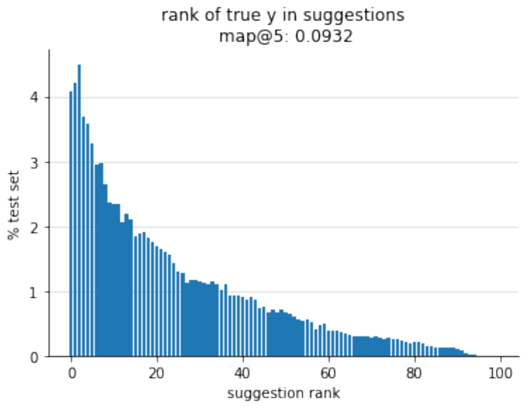
  In this case, we applied Cross-Validation to first determine the best value of neighbors k to choose from. We initially conducted this search with powers of two from 2^0 to $2^7 = 128$, and found that our best CV accuracy, 0.117, was given for $k = 128$. Then, retraining the model and evaluating it, we achieved the following results:



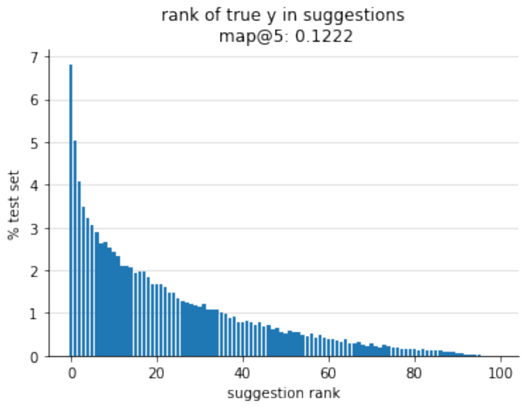
  As we can observe, while the KNN achieves similar map@5 performance as Logistic Regression, the curve indicates that KNN is much more likely to get wrong predictions very, very wrong, and justifies our use of these graphs for model performance evaluation/comparison.

- Naive Bayes Model

  In this case, we applied the different Naive Bayes implementations using their default features and found the following graphs for Gaussian and Bernoulli NB, respectively:



(a) Gaussian NB

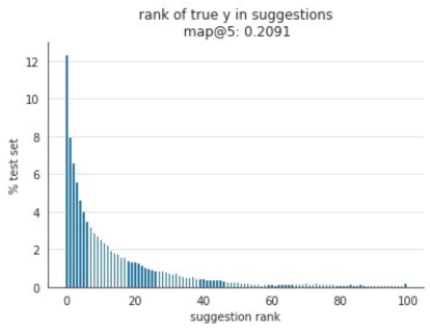


(b) Bernoulli NB

As we can see, while neither of these methods are particularly robust for our application, the Bernoulli Naive Bayes implementation significantly outperforms the Gaussian NB with lower suggestion percentages at every suggestion rank.

- Random Forests

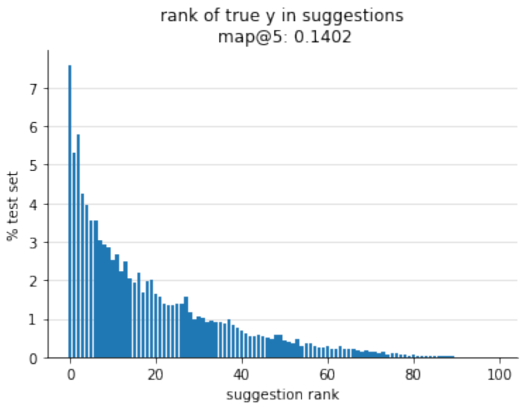
  Using Scikit-learn's implementation of Random Forests, we used Cross Validation to estimate the best parameter for the number of estimators out of the following options: $[1, 5, 10, 50, 100, 200]$. The best number of estimators was 200, with a cross-validation map@5 accuracy of 0.204, compared to the map@5 accuracy of 0.197 for 100 estimators. For this choice of hyperparemeter, we find:



  The much more pronounced $1/x$ aspect of the curve with a peak of 12% correct predictions for first suggestions alongside the nearly doubled map@5 accuracy indicates that random forest ensemble methods significantly outperform previously discussed methods for this task.

- AdaBoost

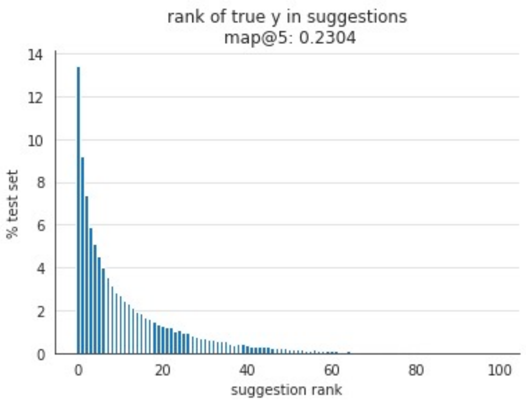
  Similarly, we performed cross-validation for the AdaBoost classifier with decision trees of max depth either 1 or 3, and number of estimators varying within the list $[1, 5, 10, 50, 150]$. CV showed that the best combination of hyperparameters for this task was 150 estimators with max depth 3 with a CV accuracy of 0.139. For this particular set of hyperparameters, the performance on the test set is shown as follows:



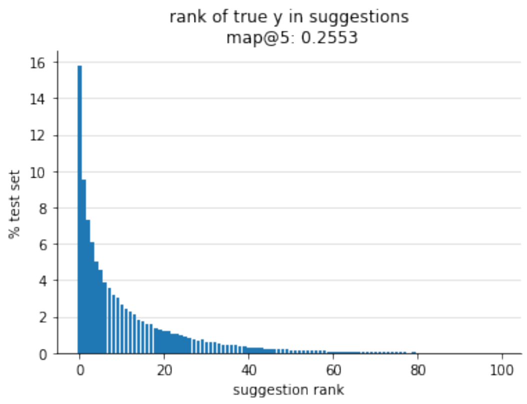
Given our data set and feature processing, it is conceivable that AdaBoost's performance might have suffered from trying to correct for some of the inherent noise of our dataset.

- Gradient Tree Boosting

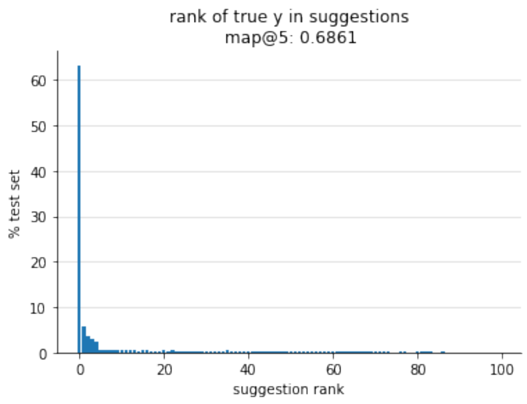
  Finally, we ran XGBoost's gradient tree boosting classifier with 300 estimator decision trees of maximum depth 5, minimum child weight 3, and prevented overfitting by setting an early stopping parameter of 20 rounds without map@5 increase on the validation set.



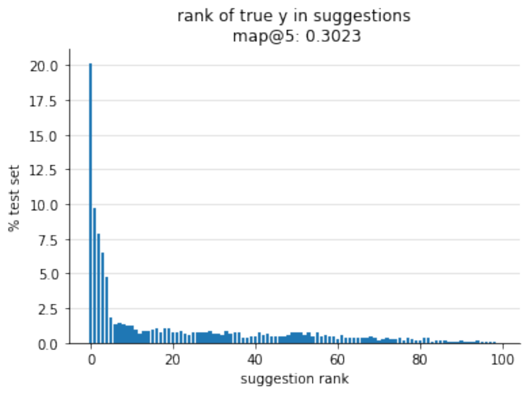
With an map@5 of 0.23, the true y appearing as a first suggestion nearly 14% of the time, and over 40% of the time within the top 5 suggestions, Gradient Tree boosting's performance on the testing set outperforms all of the other methods evaluated so far. Given that this was the most promising model that we were also computationally able to run on the entire data set, we further ran it with the data split 70-10-20 for training-validation-testing. This increased our accuracy on the held out dataset, as can be seen in the following graph:



Finally, we considered other approaches that top-scoring users on the platform adopted. We discovered that most of these submissions relied on an inherent issue in the data that made it possible to predict the correct hotel cluster for a subsection of the users with near certainty. Expedia acknowledged this issue but continued the competition nonetheless. The data contained very precise information on user location and distance to booked hotel that made it possible to precisely determine which particular user a search belonged to. We explored these solutions and considered adapting and combining them with our own methods. The main observation we made is that, while the leakage made for extremely precise predictions on training and testing datasets that shared users as in the full train/test files, performance significantly dropped when this assumption was lifted and we considered more random subsections. The first graph below shows, for example, the performance on a subset mimicking the full files:



On a more randomized section of the data, we see performance being halved:



We note however that the leakage solution above only provided suggestions for up to the first 5 ranks which contributed to the map@5 score, whereas the remaining noise can be attributed to a Naive Bayes model that it was supplementing. Nevertheless, we can conclude that the leakage relied primarily on pre-existing user data, and that it is less likely to perform well in areas where Expedia might, for instance, try to expand its reach and adapt to a more varied customer base.

7 Conclusion and Discussion

Conclusion on the performance of the models

We explore the Kaggle dataset and achieve an accuracy of 0.255 using the gradient tree boosting model and 0.258 using the random forest model. Ensemble methods as a whole tended to perform better on this task, as they were able to better leverage certain features to inform on likely predictions. However, the dataset in and of itself was particularly difficult for standard machine learning algorithms, as it presents data that is not necessarily as easily comparable. Dates or id's for thousands of locations do not present tangible correlations or predictive capabilities for the final hotel cluster label. This became evident when we considered more heuristic based approaches that addressed and exploited the data leak. These methods placed an emphasis on locality and relied heavily on the assumption that the users from the testing set had already booked trips in the training set. As such, while they were sometimes over twice as effective in predicting outcomes for recurring users compared to our ML methods, their accuracy halved when the data was not as skewed, whereas our other methods stayed consistent.

Model	**Test MAP@5**
Baseline	0.072
SVM	N/A
Logistic Regression	0.122
KNN	0.125
Gaussian Naive Bayes	0.093
Berloulli Naive Bayes	0.122
Random Forest	0.209
AdaBoost	0.140
Gradient Tree Boosting	0.255
Leakage solution	0.68

Table 1: Model Performance

Summary and lessons learned

The most challenging part of implementing the solution was to extract features and down-sample the data set given the 38 million data samples. As a result, we considered various methods of feature reduction and selection in order to maximize the performance and feasibility of our methods. During the data exploration process, we observed high correlations among some features and some sessions that are not relevant in a booking such as userid and datetime. The other challenge we met was the significant amount of missing value of *orig_destination_distance*. We thus imputed *orig_destination_distance* by considering multiple imputation methods seen in class i.e. zero, mean and regression-based imputation. This helped us improve the map@5 accuracy from 19% to 24%.

During the process, we also explored the data leak pertaining to the *orig_destination_distance* attribute. Expedia confirmed it is possible to find the correct *hotel_clusters* for the affected rows by matching a combination of multiple features including *user_location_region*, *hotel_market* and *user_location_country*. We found out after applying the data leak solution combined with ensemble learning model, the performance has been significantly improved to 0.64. However this leakage solution does not generalize well as it only exists in the data provided by Expedia. As a result, this solution is only acted as a reference to compare with the other submissions on the Kaggle leader board.

In the future, we would like to explore how to better combine heuristic-based approaches from clever observations made in the training and testing data with more general ML methods.

References

[1] Aditi A Mavalankar, Ajitesh Gupta, Chetan Gandotra, and Rishabh Misra. Hotel recommendation system. *arXiv preprint arXiv:1908.07498*, 2019.

[2] X. Liu, B. Xu, Y. Zhang, Qiang Yan, Liang Pang, Q. Li, Hanxiao Sun, and Bin Wang. Combination of diverse ranking models for personalized expedia hotel searches. *ArXiv*, abs/1311.7679, 2013.

[3] `https://www.kaggle.com/vanausloos/expedia-model-1?select=train.csv`.

[4] `https://www.kaggle.com/jiaofenx/expedia-hotel-recommendations`.

[5] `http://cs229.stanford.edu/proj2016spr/report/065.pdf`.

[6] `https://medium.com/@wesleyklock/expedia-hotel-recommendations-ea6a9d5fbaa7`.

[7] `https://towardsdatascience.com/predicting-hotel-bookings-on-expedia-d93b0c7e1411`.

[8] Gourav G Shenoy, Mangirish A Wagle, and Anwar Shaikh. Kaggle competition: Expedia hotel recommendations. *arXiv preprint arXiv:1703.02915*, 2017.

[9] G. Huming and L. Weili. A hotel recommendation system based on collaborative filtering and rankboost algorithm. In *2010 Second International Conference on Multimedia and Information Technology*, volume 1, pages 317–320, 2010.