CS249-Personalize Expedia Hotel Searches

Team: Madata

Cheyun Xia 504422348
Mingrui Shi 104517541
Shengzhi Jiang 704514808
Haosheng Zou 804518325

1. Introduction

The project requires learning to rank hotel queries by Expedia.com. The dataset which is provided by Expedia.com, contains more than 50 features describing the hotel characteristics, location attractiveness, users' aggregate purchase history and competitor OTA information. There are two features in the training set: *click_bool* and *booking_bool* indicating whether a user has clicked or booked the hotel in the query. Hotels for each user query are assigned relevance grades as following: 5 for user booked a room; 1 for user clicked the hotel page and 0 for the user neither book nor click. The task is to train the model from the training set and predict a rank of query results where the hotels most likely to be booked rank on the top and followed by the ones most likely to be clicked.
The evaluation metric for the project is called Normalized Discounted Cumulative Gain (NDCG), which is commonly used in ranking.
We used Python scikit-learn library for the modelling in our project.

2. Related Work

The project was a Kaggle competition called *Personalize Expedia Hotel Searches - ICDM 2013.* Some of the top candidates in the competition have published papers or presentation slides regarding their approaches to this challenge.
The winner[1] of the competition has the missing values in the dataset imputed by negative values, which indicates that missing value is the worst case. The winner down sampled the negative points as well as bounded numerical values (outliers removed). After preprocessing on the data, the winner did some feature engineering. The most important features are position, price, and location desirability (ver. 2). Then the winner used Gradient Boosting Machines model and achieved NDCG score of 0.54026.
The 2nd place winner[2] of the competition did similar data preprocessing. However, they composed over 300 features based on the original 54 features. They used both linear model (Regression and SVM-Rank) and non-linear model (LambdaMART) and achieved NDCG score of 0.53919.
The paper[3] tried several different models: Logistic Regression, SVM, Random Forest, Gradient Boosting Machines, Factorization Machine, and LambdaMART. The team identified the most import features as price in USD, property star rating and property location score 2 which are the same result as the winner got. LambdaMART gave them a best NDCG score of 0.53102.
We have studied these top ranked approaches and developed our solution to the problem as following.

3. Feature Engineering

3.1. Feature Analysis

Given the dataset provided by Expedia, we first investigate the existing features to get some sense of the dataset in order to further process the dataset and the features.
The feature analysis part of our project is conducted in ***Feature Analysis.ipynb***.
Here, we just put same typical samples in the report, please refer to the notebook for more details.

3.1.1. 'prop_review_score' feature

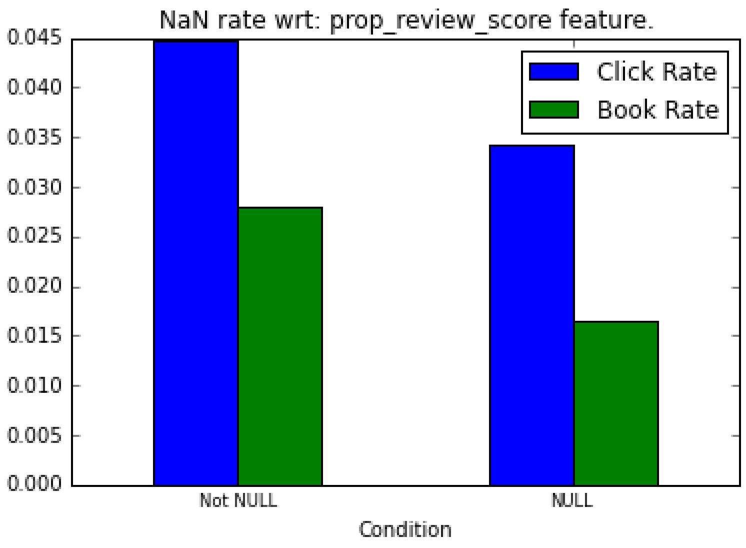

Figure 1 booking and click rate for NaN and non-NaN samples wrt. 'Prop_review_score' feature

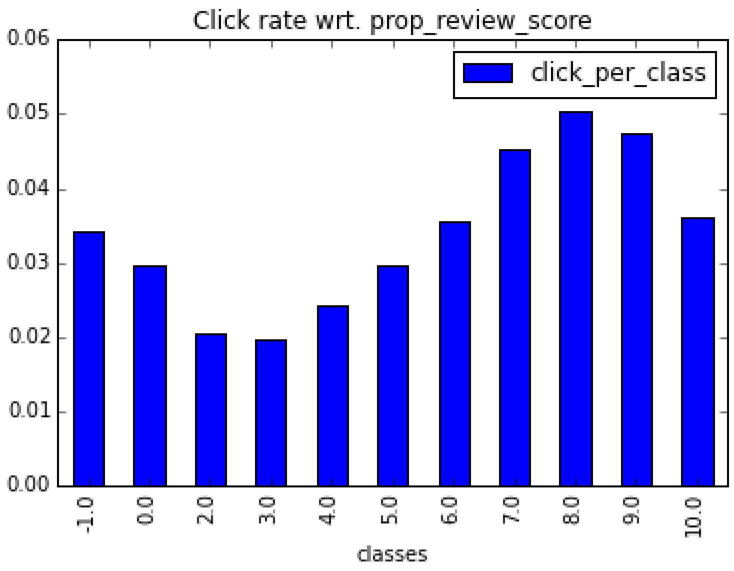

Figure 2 click rate for samples in different bins wrt. 'Prop_review_score' feature

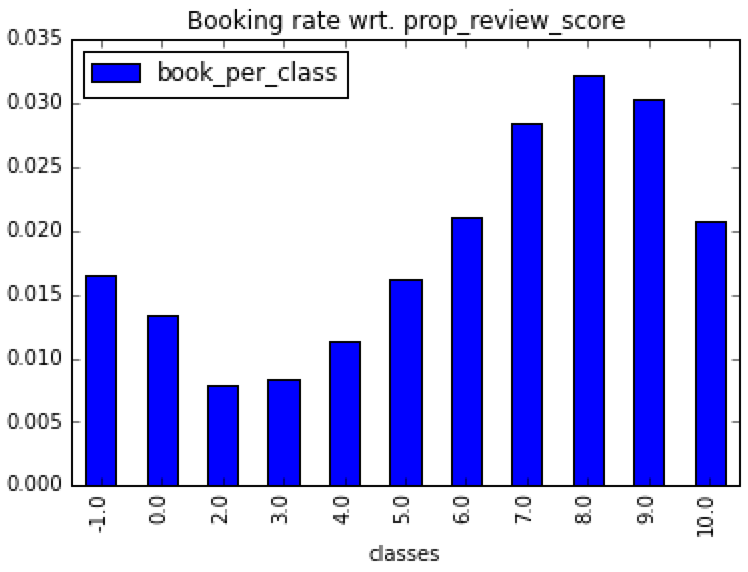

Figure 3 booking rate for samples in different bins wrt. 'Prop_review_score' feature

From the above figures we can easily find that samples with NaN values have lower booking and click rate against other sample. Also, from the distribution figures, NaN values is similar to the samples with 2.5-3.0 review score and samples with 0 values is similar to the samples with about 2.5 review score.So, here we assign NaN values with 3.0 and 0 values with 2.0.

3.1.2. 'prop_location_score2' feature

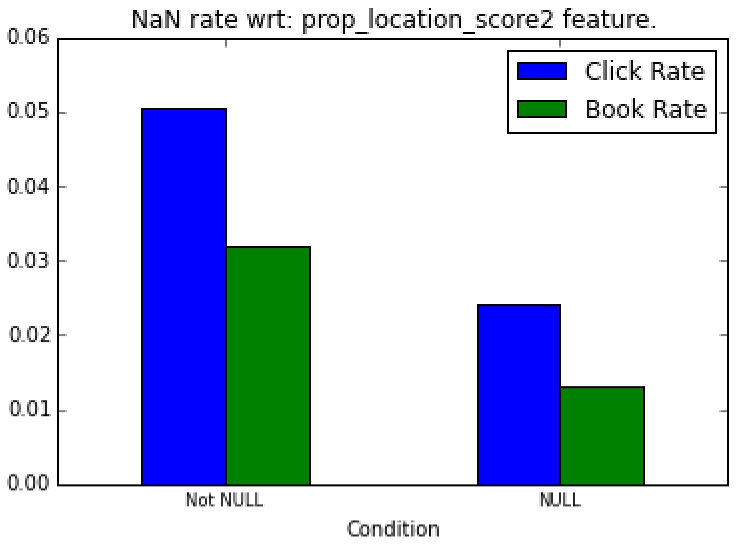

Figure 4 booking and click rate for NaN and non-NaN samples wrt. 'prop_location_score2' feature

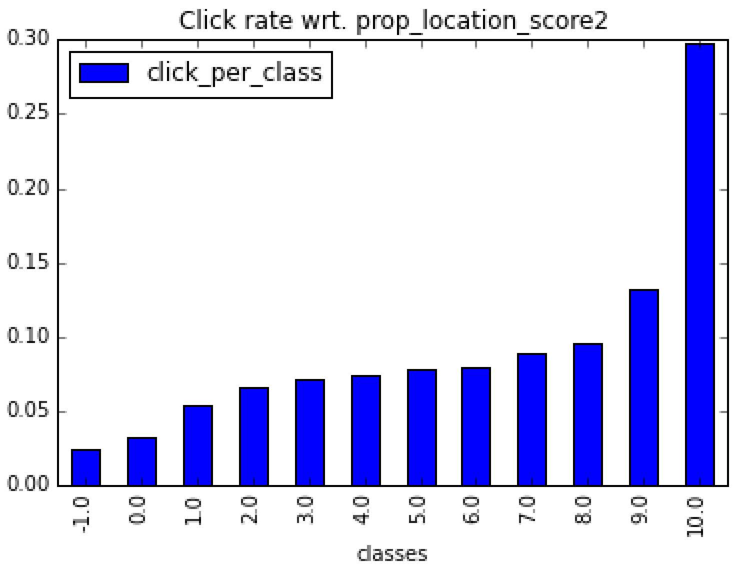

Figure 5 click rate for samples in different bins wrt. 'prop_location_score2' feature

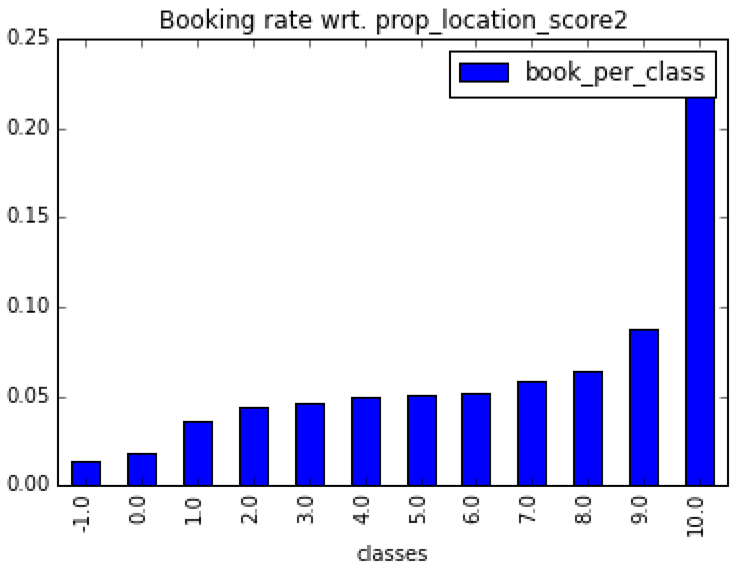

Figure 6 booking rate for samples in different bins wrt. 'prop_location_score2' feature

We can see from the NaN rate figure that samples with NaN value have lower booking and click rate than other samples. From the rate distribution figure, the samples with NaN value have the lowest rate among all the data.Therefore, we replace NaN value with 0.

3.2. Data Preprocessing

The dataset provided by Expedia.com is splitted into a training set and a testing set. The training data contains 399,344 unique search lists and 9,917,530 points. The test data contains 266,230 search lists and 6,622,629 points. We load the dataset into Python pandas DataFrame and observe that there are large number of negative instances (neither booked nor clicked). Moreover, some features have significant proportion of missing values (around 60% up to 98%) as shown in Figure 7.

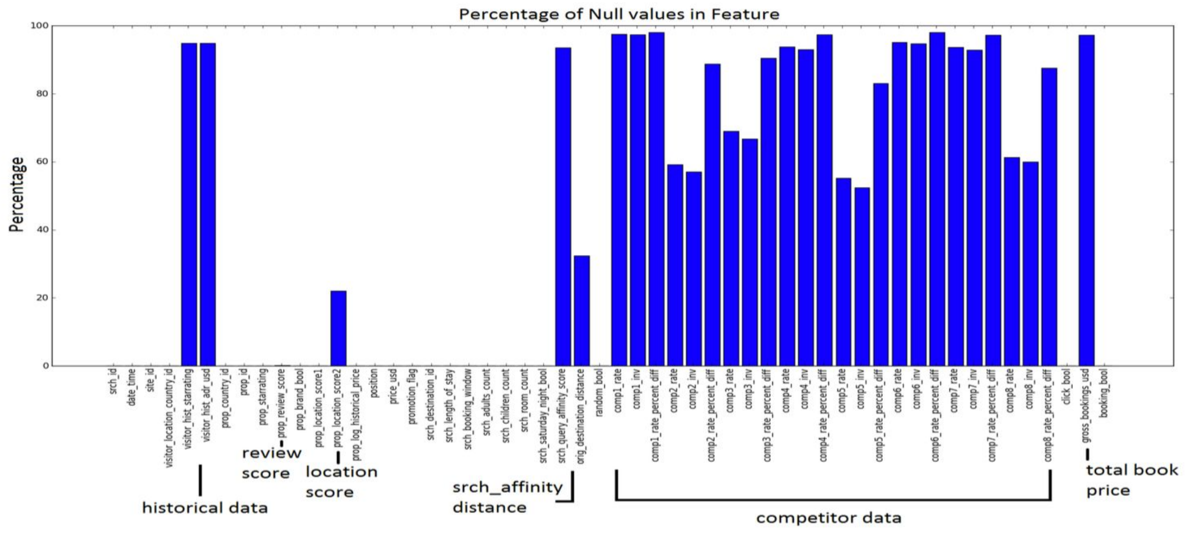

Figure 7. Percentage of Missing Values in Features

Thus before feeding the data into models, we have to work on the data to balance the positive and negative instances and fill the missing data.

3.2.1. Filling Missing Values

The features we use with missing values are treated in different manners as shown in Table 1.

Feature name	Type	Filled value
prop_log_historical_price	hotel	median
visitor_hist_adr_usd	user	median
visitor_hist_starrating	user	mean
comp_rate (1~8)	competitor	0
comp_inv (1~8)	competitor	0
prop_location_score2	location	0
srch_query_affinity_score	user	mean
prop_review_score	hotel	3
orig_destination_distance	location	median

Table 1. Filling Missing Values

The features are categorized into four different types: hotel characteristics, location attractions of hotels, user's aggregate purchase history, competitor OTA information. The reason for our specific way of dealing missing value comes from our understanding of the actual scenario. For example, if there is missing value in a hotel property location score, that hotel is less likely to be booked or clicked. So we fill the missing value with the lowest location score 0. Other ways of dealing with the missing value may generate better features, but it takes a lot of time to find the best way by trial and error.

3.2.2. Data Balancing

There are 3 categories that need to be predicted depending on the value of (click_bool, booking_bool): (0, 0), (1, 0), or (1, 1). Because of the fact that the size of the three categories varies significantly as shown in Table 2.

Clicked	**Booked**	**Neither**
0.04475	0.02791	0.95525

Table 2. Unbalanced data

The unbalanced nature of dataset could cause some problems. For linear models, the training set would train the model to always predict negative and yet still achieve accuracy of nearly 96%. By sampling and balancing the dataset, we could apply linear models to the problem and have shorter training time since the sampled training set is much smaller.

We first gathered all the positive(clicked or booked) data points. For each positive data point, randomly select a negative data point and put into the sampled training set. Thus we get a training with half positive and half negative data points.

3.2.3. Truncation

Upon exploring the features in detail, it is clear some features had significant outliers, '*price_usd*' for example. In our training set '*price_usd*' had a minimum of *0* and a maximum of *371493.6*.

For price related outliers, we bin them with an upper limit. These features include: ‘price_usd’, ‘prop_price_diff’, ‘user_price_diff’, ‘per_fee’, and ‘total_fee’.

3.2.4. Normalization

‘orig_destination_distance’,‘starrating_diff’, ‘prop_review_score’, ‘score2ma’, ‘score1d2’.Some features have very large or small values that needs normalization before fed into the models to reflect the relative distribution. The features we normalized are: ‘prop_location_score1’, prop_location_score2’

3.3. Feature Generation

In addition to existing feature given by the dataset, we try to generate more meaningful features.

3.3.1. Price Features

1. Price difference between property historical price and displayed price

$$ump = e^{prop\_log\_historical\_price} - price\_usd$$

2. Price difference between customer historical purchase price and displayed price:

$$price\_diff = visitor\_hist\_addr\_usd - price\_usd$$

3. Fee per person:

$$per\_fee = \frac{price\_usd \times srch\_room\_count}{srch\_adults\_count + srch\_children\_count}$$

4. Total fee:

$$total\_fee = price\_usd \times srch\_room\_count \times srch\_booking\_window$$

3.3.2. Score Features

The first two score is come from the reference, and we try to conduct experiment on the features in order to create new features, even the new feature are not that meaningful for human.

$$score1d2 = \frac{prop\_location\_score2 + 0.0001}{prop\_location\_score1 + 0.0001}$$

$$score2ma = prop\_location\_score2 \times srch\_query\_affinity\_score$$

Star rating difference between hotel the customer previously purchased and the searched hotel: $starrating\_diff = visitor\_hist\_starring - prop\_starrating$

3.3.3. Competitor Features

We sum up all competitor features to get a summed feature instead of using all competitor features.

$$comp\_inv = \sum_{i=1}^{8} comp\_inv(i)$$

$$comp\_rate = \sum_{i=1}^{8} comp\_rate(i)$$

3.3.4. Hotel Quality Features

Hotel quality features are created to indicated the hotel quality of each hotel, which means ,for each prop_id, we calculate click rate and booking rate as their hotel quality features.

$$hotel\_quality\_1 = \frac{click(prop\_id)}{counting(prop\_id)}$$
$$hotel\_quality\_2 = \frac{booking(prop\_id)}{counting(prop\_id)}$$

3.4. Target Generation

$$t = \begin{matrix} 0 \text{ if neither clicked nor booked} \\ 1 \text{ if clicked} \\ 5 \text{ if booked} \end{matrix}$$

$$t' = \{0,\ 1,\ 5\}$$

$$t'' = 1 \times click\_bool + 4 \times booking\_bool$$

$$t'\_w = 1 \times proba\_clicked \times t'\_clicked + 4 \times proba\_booking \times t'\_booking$$

The final weighted score is calculated by assigning 4 weight for booking probability while only 1 for click probability.

4. Models

We have used 4 models to learn the training data and predict the rank of hotels. The models are Logistic Regression, Random Forests, Gradient Boosting Machine and Extreme Randomized Trees. We used Python scikit-learn package for the models.

4.1. Logistic Regression

It is a well-known and well-studied simple linear model. We use it for classification. We firstly preprocess the data by merging the clicked and booked items within each query as positive instances, while the rest are negative. We apply our feature engineering work to the dataset and feed the training set into the Logistic Regression model.
We treat Logistic Regression model as a benchmark in our project. The simple model in scikit-learn package gives us a not bad NDCG score.
Some researchers[3] have customized the standard Logistic Regression model and tuned it to have much better performance on this problem. They adjust the weight in the cross-entropy error function with a parameter alpha to tackle the issue of data unbalance instead of our approach of randomly down sampling the negative instances. With this

customized version of Logistic Regression, they claim to have a NDCG score of 0.52, which is really amazing.

4.2. Random Forests

Random Forests is an ensemble learning method, in which each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features as shown in Figure 8.

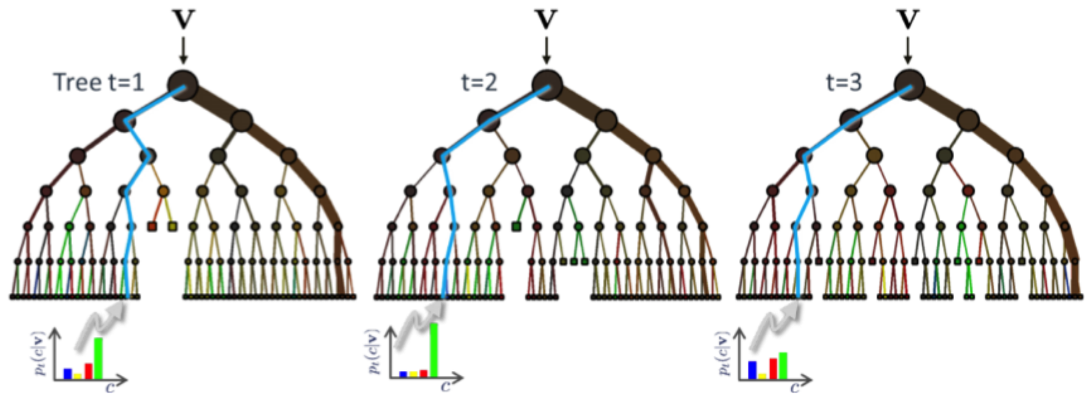

Figure 8

We have tried Random Forests with 200 trees up to 2000 trees. Due to the limit of our PC (limited memory and CPU), more than 800 trees will cause a kernel crash on IPython notebook or got the process killed by the OS in terminal. With 200 trees, we could achieve NDCG score of 0.487, which is slightly better than Logistic Regression. Some group[3] shows that they could achieve much better score (0.52) when running 3200 trees.

4.3. Gradient Boosting Machine

Gradient Boosting Machine (GBM) is a machine learning technique for regression problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. The gradient boosting method can also be used for classification problems by reducing them to regression with a suitable loss function.GBM has advantages of naturally handling of data of mixed types, its predictive power and the robustness to outliers in output space. The disadvantages on the other hand is its scalability.
We have run GBM with 1000 trees and achieved better result than Logistic Regression.

4.4. Extreme Randomized Trees

In extremely randomized trees, randomness goes one step further in the way splits are computed. As in random forests, a random subset of candidate features is used, but instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule. This usually allows to reduce the variance of the model a bit more, at the expense of a slightly greater increase in bias.
We use 200 trees in our model, which achieves similar results as Random Forest.

5. Evaluation

The evaluation metric in this ranking problem is NDCG[4] score as mentioned above. It measures the performance of a recommendation system based on the graded relevance of the recommended entities. It varies from 0.0 to 1.0, with 1.0 representing the ideal ranking of the entities. This metric is commonly used in information retrieval and to evaluate the performance of web search engines.
Here we list the best score we have achieved with three models in Table 3.

Logistic Regression	**Random Forests**	**Gradient Boosting Machine**	**Extreme Randomized Trees**
0.48334	0.49118	0.48855	0.48102

Table 3. Best scores in our project

We have also tested the performance under other conditions, for instance feed the model with all the data (unsampled) and treat clicking and booking equally important. However, these tests generate quite poor NDCG scores (around 0.38) due to the reasons discussed in section 3.
We believe with our models, we could have achieved NDCG score higher than 0.5 if we have more time to study and try the features and have larger servers to run our models with larger parameters (like Random Forests with 2000 trees).

6. Conclusions

In this project, we made our attempts to solve the problem of learning to rank on Expedia hotel search results and achieved some good results. The challenging part of this project is that the dataset is quite large with some characteristics that made it hard to feed into model and learn directly. So we have preprocessed the data and did feature engineering work to make the data fit our models.
The lesson we learned in this project is that linear model like Logistic Regression could be powerful but need more time and effort in feature engineering. Tree based rankers

like Random Forests and Gradient Boosting Machine are robust. Down sampling negative instances improves training time and predictive performance. Feature engineering makes a lot difference in the performance of prediction. Generating relevance features out of the original ones benefits the result.

7. References

[1] O. Zhang. Dropbox - Expedia Winner Presentations - 1st Place, 2013. URL https://www.dropbox.com/sh/5kedakjizgrog0y/AABhKazV866m3uDGa2QYoFsoa/ICDM_2013/3_owen.pdf?dl=0.
[2] J. Wang and A. Kalousis. Dropbox - Expedia Winner Presentations - 2nd Place, 2013. URL https://www.dropbox.com/sh/5kedakjizgrog0y/AAAahznTeGL_VKvq_jBpDMiCa/ICDM_2013/4_jun_wang.pdf?dl=0.
[3] Xudong Liu, Bing Xu, Yuyu Zhang, Qiang Yan, Liang Pang, Qiang Li, Hanxiao Sun, and Bin Wang. Combination of diverse ranking models for personalized expedia hotel searches. CoRR, abs/1311.7679, 2013.
[4] *NDCG*, URL https://www.kaggle.com/wiki/NormalizedDiscountedCumulativeGain