Personalized Expedia Hotel Searches

Fengyuan Sun[2719197], Tim Stolp[2719117], and Max van den Heuvel[2719988]

VU University Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, the Netherlands
https://studiegids.vu.nl/en/2017-2018/courses/X_400108
{f.sun, t.stolp, m.r.a.vanden.heuvel}@student.vu.nl
Group 70

1 Introduction

These days time spent on web pages becomes shorter and crucial in determining a visitors intention [1]. In the area of recommender systems and especially in the competitive market of online travel agencies, it is of great value to match shoppers to store inventories. Having the best ranking of hotels, personalized for users, gives an online travel agency the best change of winning the sale.

The main task of this paper is predicting what hotel a user is most likely to book. Companies like Expedia[1], from which the dataset originates, can then apply this prediction to organize the search results in the most suitable way. The dataset contains information about a search query of a user for a hotel, the properties of the hotels that were presented, and whether the user clicked and booked it.

To predict what hotel properties a user is most likely to click on, research on previous predicting methods is done to find prominent predictors and approaches. This is explored in the related work section of this report. This section is followed by the introduction of the dataset and the identification of useful findings via exploratory data analysis. We then explain the feature engineering process and evaluate our models. Finally, we discuss some encountered difficulties and the lessons learned.

2 Related work

Since this task of predicting hotel rankings is a Kaggle competition, numerous previous approaches exist from other competitors. We present some approaches from the top-5 to gather insights from previous methods. The following information was acquired from presentations of winning competitors [3]. Note that the previous competition used $NDCG_{38}$ (Normalized Discounted Cumulative Gain at 38) as metric.

Michael Jahrer and Andreas Toescher achieved 1st place on the ranking, but were disqualified however. They created a model blend from LambdaMart, Gradient Boosted Decision Tree, Neural Network and Stochastic Gradient Descent models. This acquired a score of 0.5407 on the test set. Their best-performing

[1] Home page of Expedia: https://www.expedia.com/

single model was Lambdamart, reaching a score of 0.5338. They noted that the largest improvement was gained by adding the mean, standard deviation and median of all numeric features for each prop_id as new features.

On 2nd place came Owen Zhang with a score of 0.5398. He used an ensemble of Gradient Boosting Machines. He first preprocessed data by imputing missing values with negatives, bounding numerical values and down sampling negative instances for improved performance. He then added four new types of features: numerical features averaged over srch_id, prop_id and destination_id, composite features (difference between hotel and recent price, and order of the price within the same srch_id), estimated position and EXP features. The latter are numerical features converted from categorical features. He observed that position, price and location desirability were the most important features.

Jun Wang reached a score of 0.5384, claiming 3rd place. This score was achieved with a single LambdaMart model. He thought that market prices vary across cities and dates, and adopted feature normalization to remove those scaling factors. He normalized features with respect to different indicators (srch_id, prop_id, month, srch_booking_window, srch_destination_id, prop_country_id) and gained the largest improvement this way. Additionally, he imputed missing values on the prop_id features with worst case scenario. Here, the assumption was that users do not find hotels with missing values attractive. He also imputed missing competitor feature values with zeros. He extracted two new features:

$$starrating\_diff = |visitor\_hist\_starrating - prop\_starrating|$$

$$usd\_diff = |visitor\_hist\_adr\_usd - price\_usd|$$

Bing Xu et al. placed 5th with a score of 0.5310. They experimented with several types of ensembles, but found that the listwise ensemble method with LambdaMart performed best. They imputed missing values using the first quartile of all instances located in the same country. They also randomly sampled 10% of the dataset as training data to improve training time.

3 The dataset

The dataset is randomly split into a train and test set, each containing around 5 million rows. There are 199795 unique searches in the train, and 199549 in the test set with 25 search results on average. The train and test set contain 54 and 50 features, respectively. These can be categorized into five groups:

User search criteria: which contains date and time of search, destination id, length of stay in days, booking window in days, number of adults, number of children, number of rooms, and whether the stay included Saturday night.

Static hotel characteristics: which contains hotel id, hotel country id, star rating, user review score, whether it belongs to a hotel chain, two desirability scores of the location, and the mean historical price.

Dynamic hotel characteristics: which contains price, whether it had a sale promotion, and the probability of hotel being clicked on in Internet searches.

Visitor information: which contains visitor country id and distance between visitor and hotel.

Visitor's aggregate purchase history: which contains the visitor's historical mean star rating and mean price per night stay.

Competitor characteristics: which contains indicators of whether Expedia's price is higher or lower compared to eight competitors, whether Expedia had an advantage in hotel availability, and the percentage price difference.

Expedia information: which contains Expedia's site id, the id of search, and whether the showed hotel ranking was randomly ordered or not.

Additionally, the train set also contains binary indicators of hotel being clicked or booked, as well as the hotel rank on search results and the total booking fee. We use the click and book indicators to calculate the hotel *relevance*, which is the target variable in our task. The formula and scoring metric will later be explained in Section 7.1. We drop the other two variables because they are also not present in the test set.

4 Data exploration

We initially explored the dataset by visualizing interesting features and their statistics. These are presented in the following subsections.

4.1 Position bias

In our task, we aim to predict the relevancy of a hotel, which is given by whether it is booked or clicked. Since our data originates from a ranking problem, the target variables can contain position bias, introduced by the hotel's position in the presentation. Fig. 1 shows the distribution of clicks and bookings per position, for both the regularly and randomly ranked search results. Both ranking methods show an indication of position bias, but this is more clearly visible in the random ranking: even though higher positioned results are not better, they still receive more clicks and bookings. Additionally, we can see that randomly ranked results receive significantly less bookings. This inspired us to try and train two models separately based on normal and randomly ranked data, which we will describe in Section 8.1.

4.2 Hotel price

One of the main deciding factors for booking a hotel is the price. We visualized the hotel prices for each hotel star rating. Fig. 2 shows that the distribution of hotel prices of low and medium ratings (1-3) are very similar to normal distributions. On the other hand, the price distribution of 4 and 5 star hotels tends to be right-skewed. The same applies to hotels with no star rating (0). Having no rating indicates that the hotel is new on Expedia, and thus, hotels of any quality can be grouped here. This could explain the positive skewness of the 0 rating distribution. A more general reason for the skewness visible across star ratings, is that the price sometimes encompasses the whole stay instead of a single night.

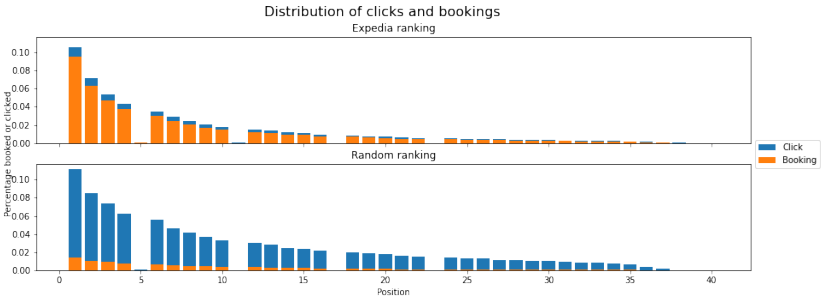

Fig. 1: Distribution of bookings and clicks in the training data, for searches ranked by Expedia and randomly ranked searches.

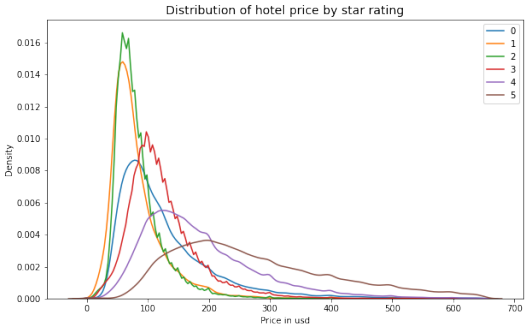

Fig. 2: Visualization of hotel prices for hotels within each star rating. A star rating of 0 indicates that the rating is missing or has not been given yet.

4.3 Outliers

Oftentimes in real-world datasets, the acquired data can be noisy and contain outliers. Fig. 3 visualizes the distributions of a subset of the numerical features using a box-plot. The most interesting distributions are *price_usd* and the competitor rate difference features. The *price_usd* feature ranges from 0 to 20 million USD, which indicates that outliers are present. Moreover, 75% of the values are below 185 USD. The competitor difference features contain percentage values ranging from 2 to 1 million, which is an unrealistic range for percentages. These outliers will be dealt with in Section 5.1.

The other variables, such as *orig_destination_distance* and *visitor_hist_adr_usd* have sensible ranges and do not contain values significantly higher than the 75th percentile (which could be classified as outliers). For example, the former has distances ranging from 0.01 to 11667 km, which is still sensible when traveling across the globe. Compared to the *price_usd* feature, *visitor_hist_adr_usd* has a much smaller range, with values between 0 and 2000 USD.

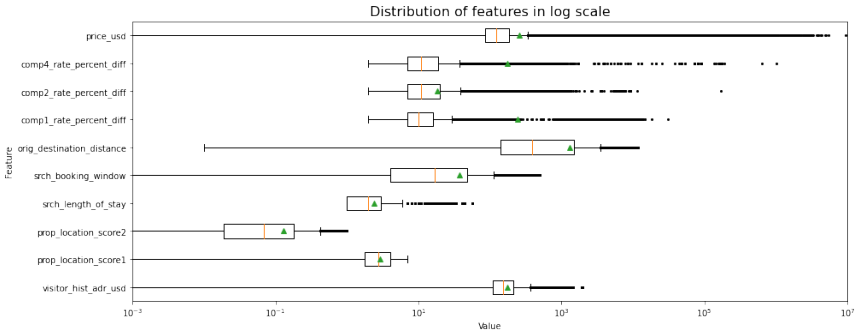

Fig. 3: Distribution of some numerical features in log scale. The mean is shown as a green triangle and values above the 75th percentile are shown as dots.

4.4 Missing values

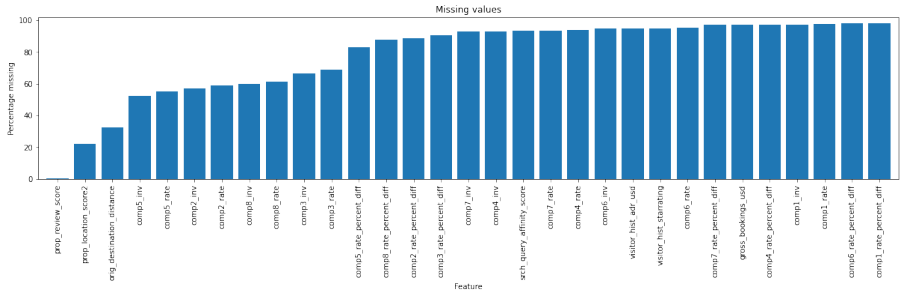

Fig. 4: Percentage of missing values.

Real-world datasets can have missing data for various reasons, one of which could be data collection problems, or the data did not exist in the first place. Our dataset contains incomplete features such as *visitor_hist_starrating* and *visitor_hist_usd*, which aggregate the user's purchase history. This will immediately be missing if a user has not made any bookings before. All features with missing values are highlighted in Fig. 4. Most of these features have a significant amount of missing values (>20%), which can become a problem for model training. Most of these features originate from competitor information, from which the values automatically becomes missing if competitor information was unavailable at search. Three other features are *prop_review_score*, *prop_location_score2* and *orig_destination_distance*, which have the relatively lowest percentage of missing values. Missing values will be imputed at a later stage for some features, while other features with too many missing values will be dropped.

4.5 Correlations

Previous competitors mentioned that the location score and prices are the important features [3]. We wanted to gather insights on the preliminary usefulness of features before pre-processing. In Figure 5, we reported the Pearson correlation coefficients between features and target relevance. *prop_location_score2* showed the highest correlation, which agrees with the previous statement. On the other hand, the price feature shows a more average correlation. We note that even the two highest correlated variables do not show a significant correlation, which indicates that our prediction task is rather complex.

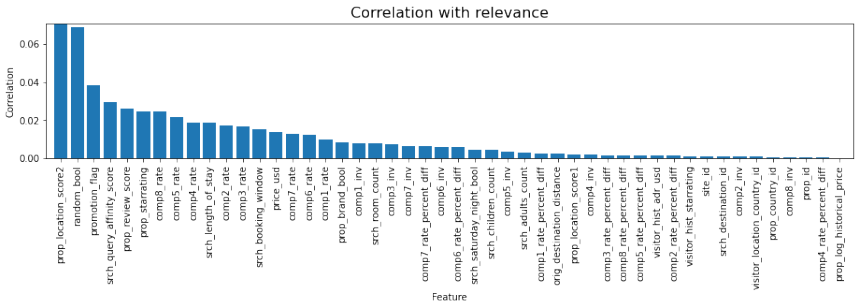

Fig. 5: Correlation coefficients between original features and the target, relevance.

5 Dataset pre-processing

The initial dataset was not optimally processed yet for model training, as it contained outliers, noise and had many missing values, as previously observed. We pre-processed this data and added extra features to improve our model's predictive performance. The improvement (or decrease) was manually checked on a held-out validation set when new features were added. For this, we employed an 80/20 split for the training and validation set coming from our labeled data.

5.1 Outlier removal

As seen in Fig. 3, some numerical features have extreme outliers. One strategy to improve the *price_usd* feature is to simply remove datapoints with a price higher than 10.000, suggested by David Wind [4]. We used a slightly different approach and bounded price values between 0 and 10.000, which reduces the effect of outliers on model training. For the competitor features, we capped percentage values at 100%.

5.2 Missing value imputation

Our dataset also contained many missing values. We imputed missing *prop_location_score2* values using the minimum score given to other hotels in the same area by checking if they shared the same destination id. This was suggested by a former Expedia employee on the Kaggle forums[8]. We filled in missing *prop_review_scores* with zeros. For *orig_destination_distance*, we calculated the mean distances between each visitor's country and their destination, and used this mapping to fill in empty values.

We then dropped all features with over 75% missing, but only after feature aggregation was carried out. This is described in the following subsection. The 75% was empirically chosen as the best performing cutoff. Missing values for the remaining competitor features were imputed with zeros, indicating that there was no difference between Expedia and their competitors.

Thus, the features from Fig. 4 that remain are the review score, location score, destination distance, price rate and hotel availability for competitors 2, 3, 5 and 8.

5.3 Feature engineering

We created new features, transformed from the existing ones, that were either more compatible with our learning to rank problem, or simply more discriminative.

The date-time feature had a very high cardinality (many unique values), and would be useless for model training. However, we hypothesized that bookings are influenced by the day and month of search, and thus extracted the monthly and daily components from this feature before discarding it.

Composite features We proposed some composite features that have had successes in previous approaches. We first implemented two features previously adopted by Bing Xu et al. (5th place) [5]:

$$count\_window = srch\_room\_count \cdot max(srch\_booking\_window) + srch\_booking\_window$$

$$historical\_price\_diff = exp(prop\_log\_historical\_price) - price\_usd$$

The historical feature highlights differences between the hotels current and previous prices, which can tell us how much a hotel is discounted. The *count_window* feature was not explained by Bing Xu et al., but we think it helps differentiate booking behaviour between smaller and larger visitor groups. Moreover, this feature improved model performance.

We also adapted the *starrating_diff* and *usd_diff* features proposed by Jung Wang (2nd place), outlined in Section 2. The rationale for these features is that it highlights the matching and mismatching between a visitor's historical data and the hotel data.

Transformations Within one feature in the original dataset, values can differ highly between datapoints. For example, when the user searches for five-star hotels, the prices will be relatively higher compared to a search for two-star hotels. To make comparison between different groups less biased, we normalize the numerical features that describe the hotel quality (star rating, review score, location scores and the current and historical price) and add these to the original features. Normalization is carried out within the given search (group of same *srch_id*) and also for all listings of a hotel (all points with same *prop_id*). The former helps in-list comparison between the search results, while the latter benefits comparison between different hotels.

Aggregations We created new aggregate features which were the mean, standard deviation and median of the numerical features. These were aggregated separately for each *prop_id*. These features have benefits compared to the original features: they have increased robustness to noise in the data, and they can capture changes in value over time. Moreover, the median can represent a more general average that is not affected by possible outliers. This approach was also suggested by Michael Jahrer and Andreas Toescher (1st place).

Additionally, we also aggregated the three competitor features across all competitors since the original features were very sparse. We took a sum over all competitors for the binary indicator features, and for the price difference feature we computed the average.

6 Model application

There are a multiple of ways to tackle our task. We could, for example, see it as a relevance prediction task and turn to regression models. However, a more intuitive method would be to tackle it as a ranking problem, since we are optimizing hotel recommendation where mainly the first few results matters. For this, we employed LambdaMART, which has previously been used by some of the winning competitors.

6.1 LambdaMART

LambdaMART is an algorithm applied for solving ranking problems. This algorithm is based on LambdaRank, which is again based on RankNet. RankNet optimizes for the number of pairwise errors when learning to rank. This does not match well with some information retrieval measures like NDCG. These measures emphasize the top few results, because the top results have more impact on the clicks these are more important when learning to rank. Pairwise errors do not emphasize the top results, thus the concept of writing down the desired gradients directly is the main idea of LambdaRank which shows improvements in both accuracy and speed [6].

Additionally a multiple additive regression tree (MART) is a boosted tree model which outputs a linear combination of the outputs of a set of regression

trees. It improves its predictions by training a new tree to predict the errors of earlier trees, this is known as a gradient boosting forest. The sum of all estimations made by the generated trees results in the final prediction [7].

LambdaMART is a combination of LambdaRank and MART which is used to perform list-wise or pairwise ranking. Instead of using boosted decision trees for predictions like MART, it uses gradient boosted decision trees and a cost function from LambdaRank for learning to rank. The LambdaMART algorithm is flexible when it comes to data input, can meaningfully rank based on utility, results in interpretable predictions, and is a powerful tool to approximate functions [7]. It showed better results compared to the original RankNet and LambdaRank [6].

We implemented LambdaMart using XGBoost[2] in Python 3.8. LambdaMART can be trained using a pairwise or listwise ranking loss to minimize the pairwise loss or maximize the NDCG, respectively. We opted for the former because this showed improved results over the listwise loss.

7 Evaluation

To evaluate the performance of our models the training set was randomly split into a training and validation set with a 80/20 ratio. This was done to be able to quickly evaluate the model without the upload restriction of Kaggle. This made it possible to efficiently test changes to the dataset and model throughout the evaluation process.

The evaluation process consisted of using the XGBoost model with a learning rate of 0.1, a max tree depth of 6, a subsample ratio of 0.9, and a total of 500 estimators. This model was used as a baseline to explore various types of data exploration and feature engineering. For each setup the training was ran with 10 seeds and the average score taken to determine if a change improved or worsened the model.

In the first setup no further data processing was applied past missing value imputation and the removal of the features *srch_id*, *prop_id*, as these features do not contain information that translates to other data sets, and as such should not be used. This gave us a benchmark to which we could compare the effect of the feature engineering explained in Section 5.3.

The first features that were added to the model were the composite and extracted features. Next the features containing data concerning the competitor websites were aggregated and the original columns dropped due to their sparsity. Then the numerical features were normalized, and lastly the hyperparameters of the model were manually explored to find the optimal model setup. These are described in the final model description, Section 9.

7.1 Metric

In information retrieval, a widely used metric in learning to rank problems is the Normalized Discounted Cumulative Gain (NDCG). This is a measure of

[2] The main parameters of XGBoost can be found here: https://xgboost.readthedocs.io/en/latest/parameter.html

ranking quality that assigns graded relevance to documents (hotels in our case) and determines their usefulness, or gain, based on its rank in the list. It is a normalized version of DCG. The latter is defined as:

$$DCG_r = \sum_{i=1}^{r} \frac{2^{rel\_i} - 1}{log_2(i+1)}$$

where r is the cutoff rank and rel_i is the relevance of the result at rank i. The relevance is 5 if the hotel has been booked, 1 if clicked, and otherwise 0. DCG is based on the assumption that highly relevant hotels are more useful to users when they are ranked higher, followed by less relevant and non-relevant hotels. This is realised by the logarithmic decay of relevance grade that is proportional to the rank of the result.

DCG scores depend, however, on the length of the list and the number of relevant results. To normalize this, DCG is divided by the ideal discounted cumulative gain (IDCG) which is the highest attainable score. The latter is computed by first sorting the results based on relevance and then calculating the DCG. NDCGr is then computed by dividing DCGr with IDCGr.

In our learning to rank task, we measure performance using NDCG_5 to focus on the top five results.

7.2 Baselines

Two baselines were set to compare against our model performances. The first one is a random ranking, which was created by randomly re-ranking the provided search results. The second one is the Expedia ranking: the training dataset contained a *position* variable that represents the hotel's rank Expedia's search results. We evaluated both baselines on the validation set.

8 Results

Table 1: Comparison of NDCG5 scores for model configurations.

Model Configuration	Validation	Test
Value imputation only	0.3855	0.3719
Added composite and day/month features	0.3937	0.3867
Added competitor aggregate, drop missing columns	0.4031	0.3916
Added normalized features	0.4072	0.3968
With model fine-tuning	0.4101	0.4031
Two separate models for *random_bool*	0.3971	0.3939
Expedia ranking	0.3913	N/A
Random ranking	0.1581	N/A

During the evaluation process, each model was used to predict the rankings on validation set and the resulting rankings were uploaded to Kaggle to give a

test score. The first model resulted in an $NDCG_5$ test score of 0.3719. Adding the composite and extracted features increased the score to 0.3867. Then aggregating the competitor data resulted in the score 0.3916. The final additions were the normalized features which gave a score of 0.3968. Lastly, by tuning the hyperparameters of the model to their optimal values, a final score of 0.4031 was obtained. Compared to the baseline of the Expedia rankings the final model was able to obtain a higher $NDCG_5$ score and thus it is able to provide improved property rankings for search queries.

8.1 What did not work

Some changes during the evaluation process resulted in a lower validation score and as such were not used to obtain a final test score as this was not deemed necessary given the limited Kaggle submission attempts. Some of these features that were tried were the list-wise features proposed by Bing Xu et al. [5]. These features had little to not effect on the performance of our model, and as such were left out. The features obtained by normalizing per destination_id resulted in a worse validation score therefore these were also left out. Leaving in the original columns containing the competitor website data also resulted in a lower validation score.

Another idea that was explored was the use of two separate models trained on two datasets obtained by splitting the data by the nature of the search query ranking. Some of the rankings were randomized while others already had a ranking method applied. It was hypothesized that the user behaviour on these types of rankings would be vary enough to warrant separate models. Since the validation scores alone did not give enough insight on the performance of this setup was used to process the test data and create a Kaggle submission. Using two separate models resulted in a score lower of 0.3939 compared to the then best score of 0.3968.

9 Final model description

Our best performing model was LambdaMART with the following hyperparameters: a learning rate of 0.05, a max tree depth of 8, column sample ratios of 0.8, 0.9, 0.9 per tree, level, and node, respectively, and finally a data row sample ratio of 0.95. The model was trained using a pairwise loss for 2000 rounds (estimators) with early stopping if the validation score did not improve in the last 100 rounds. During training, the best performing epoch was saved as the final model.

For features, it used the extracted day and month, all composite features, the aggregated features, all normalized features and the original features with less than 25% missing values. From the original features, *srch_id*, *prop_id* and *date_time* were also dropped. The final model contained 81 unique features.

According to the feature importance in figure 6. The most important features were the already predictive features in the dataset together with their normalized

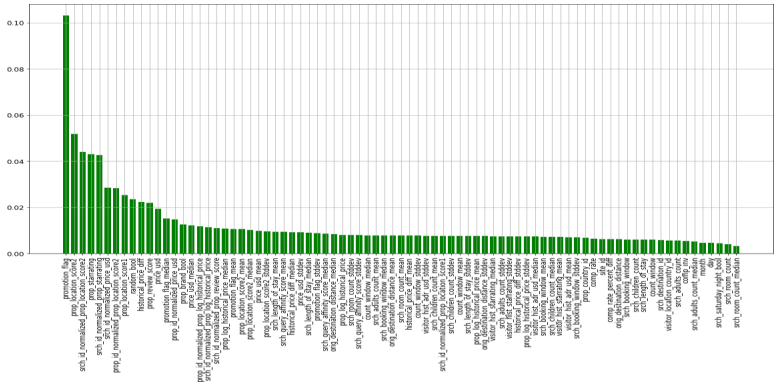

Fig. 6: The importance of each feature in the final model.

counterparts. For the more precise distinctions in the data the features' medians and standard deviations were mostly used with at the bottom the less predictive features of the original dataset.

10 Discussion

In this challenge, we tackled a large real-world dataset while optimizing it for a learning to rank task. We first explored the data, plotting interesting observations, and then followed up with data pre-processing. We successfully trained a model that is able to outperform the Expedia benchmark, and were able to reach the 22nd rank on the in-class Kaggle competition.

10.1 What did we learn

During this challenge, we learned that using the mean, median and standard deviation is useful to oppose noise in datasets. Another method to oppose this is normalizing features to better compare within groups. We also gained experience with mining real-world datasets and explore different techniques to use. We learned what method works and what does not work in a situation with recommender systems. We also gained experience with an application of data mining techniques for such recommender systems by participating in a Kaggle competition. The final and most important lesson we learned during this project is the fact that there is no one solution when it comes to data mining. Every real world data set is unique and requires its own routine of exploratory data analysis and preparation. Some techniques can be applied to multiple datasets but every data mining problem is unique.

References

1. Peter I. Hofgesang: Relevance of Time Spent on Web Pages. In: 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006) (2006)
2. Expedia Homepage, https://www.expedia.com/. Last accessed 21 May 2021
3. Personalized Expedia Hotel Searches, https://www.dropbox.com/sh/5kedakjizgrog0y/_LE_DFCA7J/ICDM_2013. Last accessed 21 May 2021
4. Wind, D.: Concepts in predictive machine learning. Technical University of Denmark, Denmark.
5. Xudong Liu, Bing Xu, Yuyu Zhang, Qiang Yan, Liang Pang, Qiang Li, Hanxiao Sun, Bin Wang: Combination of Diverse Ranking Models for Personalized Expedia Hotel Searches. CoRR abs/1311.7679 (2013)
6. Christopher J.C. Burges: From RankNet to LambdaRank to LambdaMART: An Overview. In: Microsoft Research Technical Report MSR-TR-2010-82 (2010)
7. Ziniu Hu, Yang Wang, Qu Peng, Hang Li: Unbiased LambdaMART: An Unbiased Pairwise Learning-to-Rank Algorithm. (2019)
8. Personalized Expedia Hotel Searches discussion, https://www.kaggle.com/c/expedia-personalized-sort/discussion. Last accessed 21 May 2021.