Assignment 2 - Data Mining Techniques - Group18

Chenghan Song[1,2[2680951]], Haochen Wang[1,2[2698251]], and Li Zhong[1,2[2688794]]

[1] Group 18
[2] Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam

Abstract. With the development of information technology, we have come from the era of information scarcity to the era of information overload. It becomes difficult for users to quickly obtain useful knowledge from the massive amount of information. Therefore, in the hospitality industry, it is crucial for online travel agencies to optimize hotel query results to incentivize customers to make a purchase. In this assignment, we aim to build ranked hotel recommendations for users that are searching for a hotel to book.

Keywords: Data Mining · Recommendation Systems · LightGBM.

1 Introduction

Expedia [1] is one of the largest online travel companies that offer services including accommodation reservation, ticketing, and travel management. The online travel market is intensely competitive and crowded with an abundance of disparate agencies providing analogous products. Recommending personalized products is of vital importance in order to stand out from these competitors. The hotel recommendation system aims to model and provide a ranked list of hotels to predict the preference of users according to their query parameters. In this assignment, our task is to make ranked predictions of hotels that users are most likely to click on based on their particular search queries in the datasets offered by Expedia. We have tried multiple techniques to try to achieve the optimal solution. We first investigate the relevant researches to get a better business understanding. Then we conduct exploratory data analysis and data preprocessing, after which we engineer features. In the end, we build the model and fit the result. This report is structured in the order mentioned above.

2 Related Work

Zhang et al. [7] who attended the ICDM Challenge 2013, conducted some feature engineering work, including missing value imputation with statistical values, designing composite features, and balancing training data. They trained a combination of several models including Boosting Trees, Logistic Regression, SVMRank, and Random Forests to improve the performance. In their work,

they found that the price, rating, and location-related features are more important than other features. Besides, the listwise ensemble method, which combines each model's output, achieves the highest-ranking score of 0.53249 with z-score normalization being applied.

Another different approach was performed by Owen Zhang [9], who won the first place in the ICDM Challenge 2013. When handling missing values, he just imputed with negative values with the assumption that the missing itself has meanings. In the preprocessing stage, he also down-sampled negative instances, which causes the unbalancedness of the dataset. He proposed five groups of features, where the EXP features and estimated position features are novel and different from other participants in the competition. Owen chose an ensemble of gradient boosting machines as the ranking model and implemented two models with and without EXP features respectively.

3 Data understanding

3.1 Data Statistics

The datasets we use in this assignment are offered by Expedia. The training set contains 54 features and the test set carries 50 features, without 'position', 'booking_bool', 'click_bool', and 'gross_booking_usd', which are available in the training dataset. Each instance in the dataset contains information about the user, the search performed by the user, and one of all hotels belonging to this search. One query can have multiple records and one record correspond to one search.

About 44.7%percent of the training data and 46.3% percent of the test set is missing. Thus we group the attributes into five categories as Table 1 for further exploratory data analysis. From figure 1 we can see that there are around 29 attributes having more than 30% missing values. Among these attributes, a large quantity of competitor-related values is missing. We will conduct further analysis and process in the Section 3.2 and Section 4.1. In addition, only 4.47% of the hotels are clicked together with 2.28% of which getting booked and most users are from the same country which suggests that there may be a correlation between booking and clicking(verified in Figure 2). Also, one possible reason for this low conversion rate is that the training set might be imbalanced.

Table 1. Features exploration

Category	Attribute
User information	'vistor_' related features, srch_id, site_id
Search criteria	'srch_' related features, orig destination distance, random_bool
Hotel information	'prop_' related features, promotion_flag, price_usd
Reservation information	click_bool, booking_bool, gross_bookings_usd
Competitor information	'compX_' related features

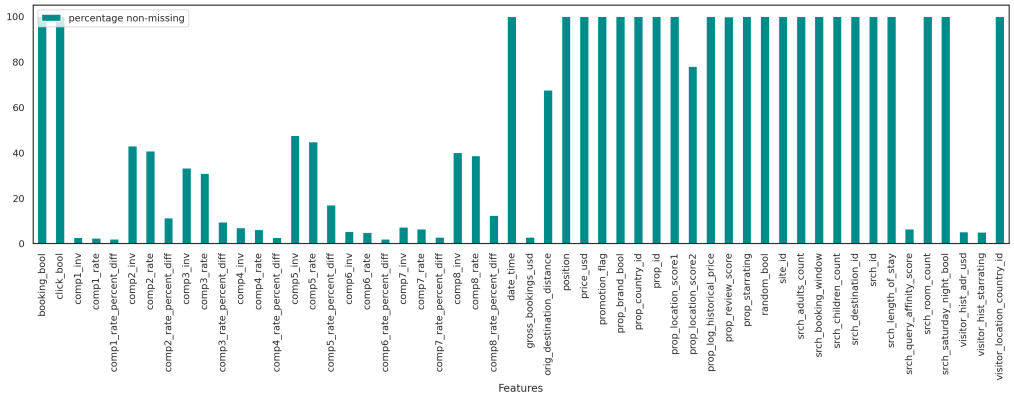

Fig. 1. Proportion of non-missing values for every attrubute.

To get a better understanding of different attributes, we performed a correlation analysis. In the figure 2, we discover that the historical mean price per night of the hotel the users have previously purchased has a positive association with the mean star rating of hotels they purchased before. It reveals the correlations among search attributes. A negative relationship between stays that includes Saturday night and the number of nights stayed suggests that the weekend trip is usually short. We also notice there are correlations between the star rating of the hotel and the mean customer review score as well as the desirability of its location. We dive a bit deeper into the hotel and reservation attributes. Figure 3 suggests that four-star rated hotels get clicked on most but only account for 5% of the hotel amount. Users who rate hotels one star are most discrete. Furthermore, We also notice that the promotion flag increases the possibility of users clicking on the hotel which is coherent to daily life.

Regarding the reservation information attributes, we find a correlation between clicking and position. Figure 4 shows that the times of user clicking decrease as the hotel position on the search results page getting larger whether it is random or not, hinting that users tend to click on the top results which in line with user psychology. On the normal sorted result page, the higher the hotel positions, the more likely users make the reservation. However, there seems to be no relationship of booking with hotel positions when they are arranged randomly.

3.2 Data Distributions

Missing Values We created a summary of the missing values in the training dataset and calculated their percentages in all records and booked records respectively.

From the table, we can see that the percentages of prop_location_score2 in all records are very different from the one in only booked records, therefore we

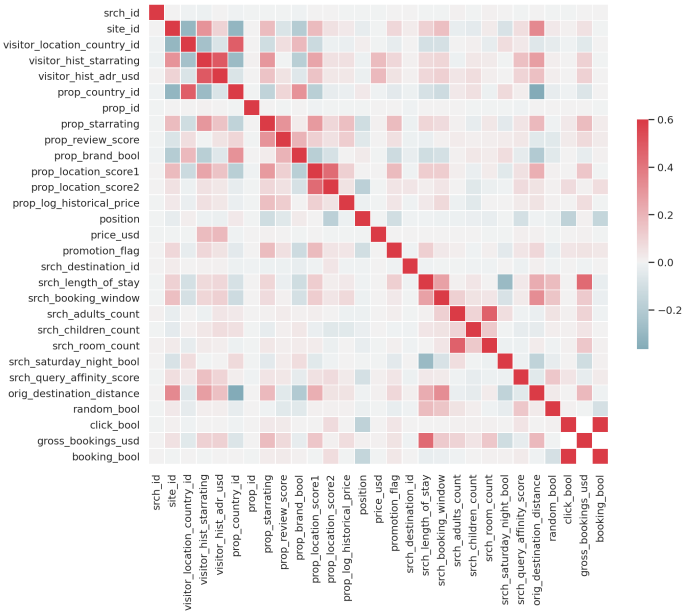

Fig. 2. Correlation analysis of attributes

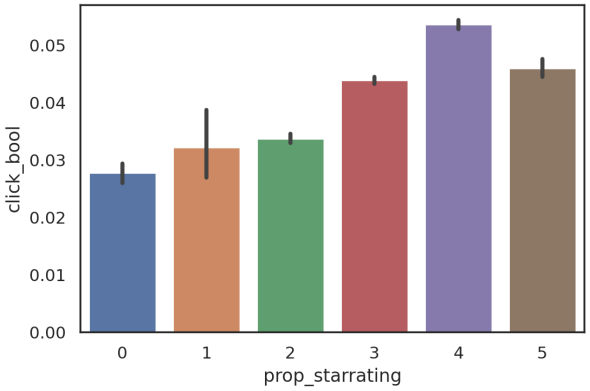

Fig. 3. Hotel stars and clicking

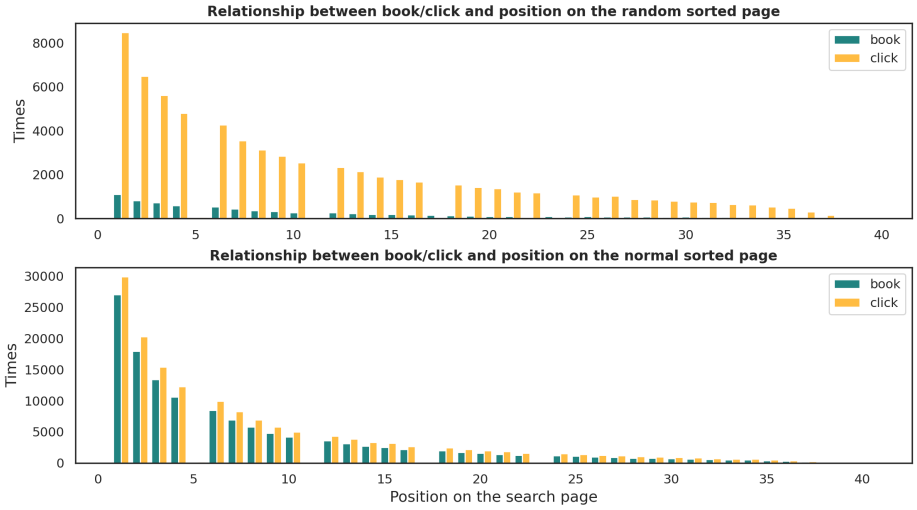

Fig. 4. Relationship between book/click and position

perform the Exact Fisher's exact test [8] to see if these two values are significantly different. The p-value of Fisher's exact test is less than 0.05, which means that the missing percentage of prop_location_score2 is different between these two

Table 2. Missing values and corresponding percentages

Attributes	All records	Booked records
$comp\_rate_{1\_8}$	78.19%	77.15%
$comp\_inv_{1\_8}$	76.72%	75.23%
$comp\_rate\_percent_{1\_8}$	92.6%	91.6%
visitor_hist_starrating	94.920%	93.433%
visitor_hist_adr_usd	94.898%	93.400%
prop_review_score	0.149%	0.084%
prop_location_score2	21.99%	10.459%
orig_destination_distance	32.426%	32.292%
srch_query_affinity_score	93.599%	92.135%
gross_bookings_usd	97.209%	0.0%

populations, thus the missing attribute prop_location_score2 has some meaning itself.

Although the gross_bookings_usd is highly related to the action of bool, it can only be precisely calculated after booking and it is not available in the training dataset, therefore it provides very limited meaning for the ranking.

The missing remaining attributes do not show the obvious relationship with the booking action, which means the missing of these attributes has no meaning itself and is missed randomly.

Outliers From the descriptive statistics of training data of each feature, we found some strange features with large gaps between their quartiles. In order to find out if the gap was caused by their skewed distributions or the outliers, we created boxplots for these features respectively, which are shown in Fig. 5. From the boxplots, we can see that the large gap of some features, such as srch_adults_count, srch_children_count, is caused by their skewed distributions and there are no significant outliers in these features.

According to the analysis of missing values above, the outliers detected by boxplots in features visitor_hist_adr_usd and prop_location_score2 might be caused by the great number of missing values, and we should deal with these two features when handling missing values.

As for outliers in the remaining features, we calculate the percentages of them among the whole data. Considering that the outliers account for a large part of the total data and hotel booking is a complicated decision scenario, which is influenced by many factors and involves an extremely diverse population, it will lose valuable information if we just exclude them during the data processing stage.

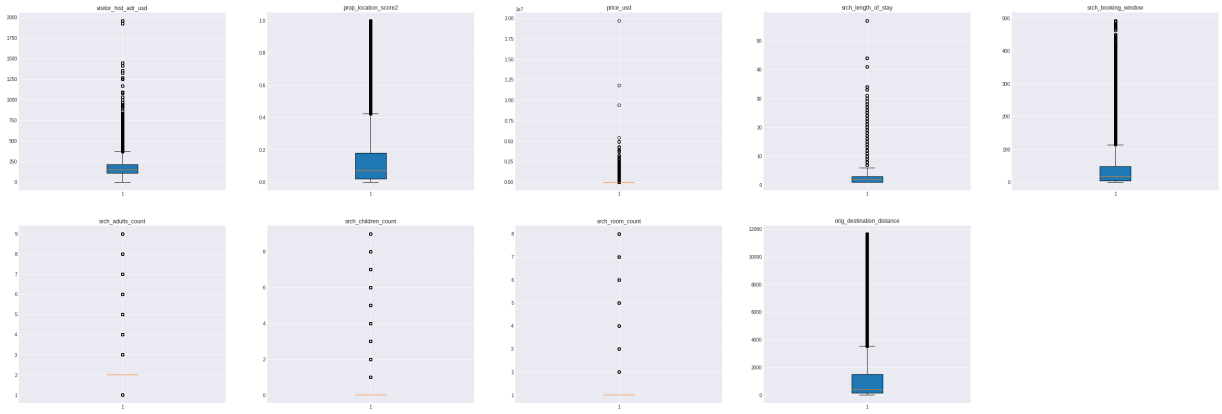

Fig. 5. Boxplots of features with strange distributions

Table 3. Percentages of outliers in all data

Features	Percentages
orig_destination_distance	7.9%
srch_length_of_stay	2.3%
srch_booking_window	9.8%

4 Data process

4.1 Missing values and Outliers

Missing values As discussed previously, there are a few numerical features with huge missing values, thus it will be a tremendous waste of information in the dataset if we just remove those instances. Besides, we also found that the missing value itself of some features influences the booking decision and has valuable information. The methods we adopted to handle missing values are presented in Table 4.

From the previous investigation, we found that missing values of prop_location_score2 are related to the target value, booking_bool, thus we should keep them missed with the imputation of negative values, which is out of its rational range. Considering that there is no significant hotel price difference between Expedia and other competitors, we fill the missing values of all the competitor variables with zero, which is neutral and provides convenience for our feature engineering later on.

As for the orig_destination_distance, we first fill in the missing values from other records with the same visitor_location_country_id and same prop_country_id, and impute the remaining with a median of all data, since the distribution of orig_destination_distance is right-skewed and median can reduce the impact of

larger values and outliers. Distributions of prop_review_score and visitor_hist_adr_usd are also skewed, thus we use the median to replace missing values, while distributions of srch_query_affinity_score and visitor_hist_starrating are almost normal, therefore we replace with mean values, which carries more information about the distribution. The gross_bookings_usd is not available in the testing dataset and might include noise when training, thus we remove it from the dataset.

Table 4. Handling Missing values

Attributes	**Method**
prop_location_score2	Replace with negative values
comp_rate_{1_8}	Replace 0
comp_inv_{1_8}	Replace 0
$\text{comp\_rate\_percent}_{1\_8}$	Replace 0
orig_destination_distance	Replace with matching and median
prop_review_score	Replace with median
visitor_hist_adr_usd	Replace with median
srch_query_affinity_score	Replace with mean
visitor_hist_starrating	Replace with mean
gross_bookings_usd	Remove

Outliers From the discussion in section3.2, just excluding outliers will lose valuable information, thus we only decrease the weights(to-do:add-ref) of instances with outliers to reduce the impact of outliers. We set lower weights to instances with outliers when feeding data to our ranking model in the training process.

4.2 Feature Engineering

To help the model to find latent patterns among different features, we design several kinds of features. First of all, we aggregated some features to obtain the overall information of that feature and utilize the difference between an overall distribution and one certain instance. For example, we calculated aggregated_prop_review_score by computing the mean value of prop_review_score with each search to provide the distribution information of that feature. Meanwhile, we also created features with order information. such as the price_rank, which expresses the position of that price in the hotel queue. Besides, we designed some composite features, where multiple features are combined to extract the interaction among them, such as the price_diff and rating_diff. Finally, we created 24 new features and removed three features, resulting in 75 features in total, the methods of calculating some features are shown in Table 5.

Some features are normalized to reduce the scale of data and build a robust model. Firstly, we perform log function on price_usd to decrease the large

scale of that feature. Besides, we perform the normalization with the mean value and standard deviation on some features within the same srch_id, such as prop_starrating and price_usd. The label (i.e. target value) of instances was set to be one if either the click_bool or booking_bool is true since the click action is highly related to the booking decision and large difference between click and booking will bring the noise to the ranking model.

Table 5. Part of Feature Engineering

Features	Method
aggregated_price_usd	The mean price_usd of each hotel
aggregated_dest_price_usd	The mean price_usd of each srch_destination_id
aggregated_prop_review_score	The mean prop_review_score of each search
weekday	The weekday of the date when search was performed
week_of_year	The week number within the year
early_night	If the search was performed very early of that day
price_rank	The price rank of this hotel within one search
star_rank	The starrating rank of this hotel within one search
price_diff	price_usd - visitor_hist_adr_usd
rating_diff	prop_starrating - visitor_hist_starrating
target	0 if click_bool 1 and booking_bool is 2

4.3 Feature Selection

In order to filter out important features and prevent our model from overfitting, we utilized the feature importance function provided by LGBMRanker to measure the importance of all the features. After obtaining all the importance information, we calculated the quartile of the importance data and we removed those features, whose importance score is below the first quartile. After the selection, we only have 55 features remained and their importance scores are all greater than 15.

5 Model

5.1 Algorithm

LearningToRank(LTR) algorithm is employed in our project, which is one algorithm of Boosting in ranking tasks, or recommendation systems. We try to implement various types of Boosting using LightGBM. LightGBM is a gradient boosting framework that uses tree-based learning algorithms [2]. In detail, **LambdaMART** is one of the most common pairwise algorithms. From the name, it obviously consists of two parts, one is Lambda(gradient), the other is

MART(Multiple Additive Regression Tree). Essentially, MART is GBDT with gradient addition under the idea of Boosting [6].

Lambda stands for LambdaRank. It is a gradient instructing the orientation and intensity of optimization for the next iteration. The derivation has been raised by Burges, and we can have the value λ in Equation 1 [4]. Where i and j are the two hotels, s is the value given by the model. ΔZ_{ij} is the indicator of evaluation, in our project is NDCG, which will be discussed in Section 5.3. Therefore, the optimization is not to first define the loss function but to directly define the gradient according to the meaning of ranking. Furthermore, we can derive the loss function in Equation 2.

$$\lambda_{ij} = -\frac{1}{1 + exp(s_i - s_j)} \cdot |\Delta Z_{ij}| \tag{1}$$

$$L_{ij} = log\{1 + exp(s_i - s_j)\} \cdot |\Delta Z_{ij}| \tag{2}$$

The LambdaMART model consists of many decision trees integrated by boosting. The optimization goal of every tree is the gradient of the loss function, Equation 2, using Lambda method, Equation 1.

For building Decision Trees, LambdaMART uses the simple least square method. The least-square error, LSE is the indicator to break the current leaf and generate new leaves. The LSE of Lambda of the two leaves is calculated as the cost. And further one best dividing node can be discovered with the lowest cost. Then, one ideal Decision Tree containing L leaves is generated.

Last part of the model, for the generated Decision Tree, we calculate the value of each leaves using Newton Step. The value is right the number of hotels which is fitted for the condition. Then, the model will be updated with the Decision Tree. The learning rate also is the parameter of regularization.

5.2 Parameters Setting

In this subsection, the final model and its parameters in training will be discussed. The tuning and choosing of the final parameters will also be clarified.

The pipeline of the final model is shown in Figure 6. After handling missing data and outliers, we do feature engineering to aggregate and add some features. When preparing data to train and test, we first split the training set and the validating set. We do the same pre-processing on the testing set. Then the processed data is sent to the LGBMRanker. To keep track of building model and shrink the memory, we save the model during training. Last, we use it to predict and sort by score.

LGBMRanker has many parameters to build a suitable model. Through that, we focus on the following key parameters shown in Table 6. Some of them should be fixed specifically for the reason of our algorithm and basic setting. Inspired by the grid search [3], we try to build models under every combination and find out the best one.

The parameters which should be tuned are shown in the Table 7 together with their candidate values. Here boosting type is 'dart'(Dropouts meet Multiple

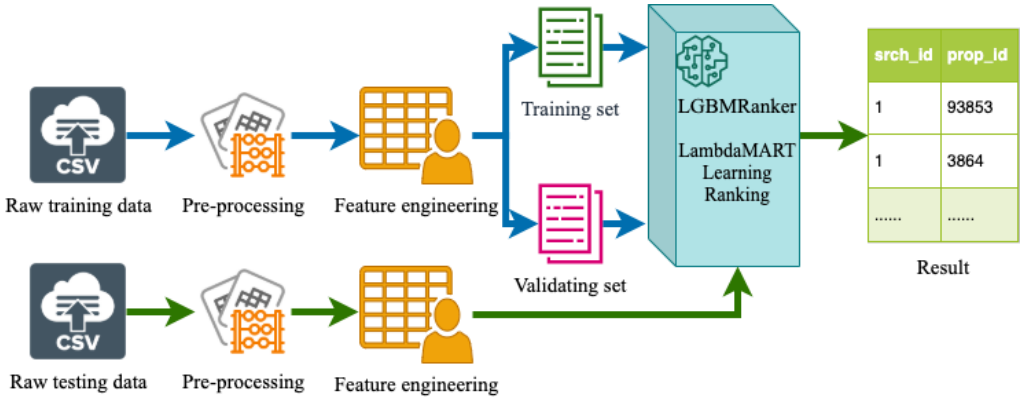

Fig. 6. Pipeline of the final model

Table 6. Parameters choosing

Parameter	Description	Value
boosting_type	Type of boosting method in training	'dart'
num_leaves	Maximum tree leaves for base learners	32
max_depth	Maximum tree depth for base learners	5
learning_rate	Boosting learning rate	0.12
n_estimators	Number of boosted trees to fit	1500
label_gain	Relevant gain for labels	[0, 1, 5]
objective	Specify the learning task	'lambdarank'
random_state	Random number seed	47
eval_metric	Customize evaluation metric	'ndcg'
eval_at	Evaluation positions of the specified metric	5
categorical_feature	Categorical features	*8 features*

Additive Regression Trees) for our algorithm, we also deploy 'gbdt'(traditional Gradient Boosting Decision Tree) which is the default method as a reference or benchmark. The candidates in the categorical feature are the number of features adopted. With several times attempts, under the validation of 8 folds by NDGC@5, the best combination considering score and time is illustrated in Table 7. We finally decide to adopt the setting of the parameters in Table 6. Note that: the categorical features are ['dayofweek', 'month', 'prop_country_id', 'site_id', 'srch_id', 'prop_id', 'visitor_location_country_id', 'prop_country_id']

5.3 Evaluation

In this subsection, we will describe the evaluation metric, the validation setting, and the evaluation of our final model compared to some models during tuning parameters.

Table 7. Tuning parameters

Parameter	Candidates	Ideal value
boosting_type	['dart', 'gbdt']	'gbdt'
num_leaves	[16, 32]	32
max_depth	[5, 10]	5
learning_rate	[0.10, 0.12, 0.15, 0.18]	0.12
n_estimators	[1000, 1500, 2000, 3000, 5000]	1500
categorical_feature	[[6],[7],[8],[9]]	*8 features*

Metric As mentioned before, the competition requires the metric of **NDGC@5**. NDGC, Normalized Discounted Cumulative Gain, is used to evaluate the algorithm of searching or ranking. The derivation of NDGC and validity on LTR task have been discussed in [5]. We simply explain how NDCG calculates in Equation 3, 4, and 5. Where k is the assigned position in our task is 5, r is the relevancy on the k, R is the descending rank of relevancy, and taking the first k sets.

$$DCG_k = \sum_{i=1}^{k} \frac{2^{r_i} - 1}{log_2(i+1)} \tag{3}$$

$$IDCG_k = \sum_{i=1}^{|R|} \frac{2^{r_i} - 1}{log_2(i+1)} \tag{4}$$

$$NDCG_k = \frac{DCG_k}{IDCG_k} \tag{5}$$

NDCG has outstanding characters, the high relevancy results have more influence on the score. And the score will increase when they lie in the front of the rank. This metric is based on query and position. Based on the query, even the worst hotel ranking of one query by a user, it will not heavily influence the evaluation procedure as a whole. The reason is that every pair of user_query-hotels contributes the same to the average. Based on position, NDCG explicitly uses the information of the position in the ranking list. The side-effect is NDCG metric is discrete.

Validation For the validation setting, we try two ways of splitting the training set into 8 folds. That is 87.5% of the dataset for training and 12.5% for validating. The first way is splitting in a random state, while the other is in the sequence of 12.5%. After validating with both ways, we realize that random sampling is not applicable in the ranking task. Under the same configuration and setting, the NDCG@5 of random splitting on the testing set is only 0.35246, while the score of sequential splitting can reach 0.39900. Therefore, we decide to use 12.5% sequential records of the training set as validating set and the others are trained the model.

During tuning the model, we get many results with different parameter combinations. Here, we only show the final ideal parameter setting of the model and

the benchmark model using the traditional Gradient Boosting Decision Tree. The tasks all run in Google Colab. The results are shown in Table 8. The benchmark is the best score, for the random state of generating Decision Trees. The worst validating score of the benchmark is 0.25044. We can know that the benchmark model may be a little overfit. The final model on validating set can be 0.42103, during training it fluctuates around 0.42, however, the score only is 0.40283 testing the output on Kaggle.

And the training procedure is time-consuming. Actually, when processing the data including feature engineering, it needs about 20 minutes to finish, Although the benchmark trains for less time, the overfitting seems much heavier.

Table 8. Model evaluation

Model	**Validating NDCG@5**	**Training time**	**Testing NDCG@5**
Final model	0.42103	192min	0.40283
Benchmark	0.49692	56min	0.26869

6 Conclusion

To conclude, we build the pipeline of a recommendation system for hotel data by Expedia. Using LGBMRanker as the core model, we can predict hotels which the users may most probably click into. The main challenge is still missing values of many features, which is usually in actual experience. We can release this by pre-processing and feature engineering.

From the Kaggle competition, we learned more structural data mining methods. Especially, we gain practical acknowledgment of the recommendation system and some different algorithms with implementation. We also start to know the metrics evaluating a ranking model, like NDCG. Through the training, we realize that feature engineering is a very important part of data mining. Therefore, it is vital to do exploratory data analysis. Aggregating features can improve the quality of the data. Facing the problem of low scores, we can add more features according to the inner logic of the data itself.

Finally, we find it really efficient to report individual progress during the project and necessary to have a meeting every three days. We can exchange our problems and make a group decision on our strategy, which is really helpful for the improvement. We can have a division of tasks for members sequentially and conquer the work individually, then merge into the whole. We regard it as a good developing method.

References

1. Expedia group, https://www.expediagroup.com/home/default.aspx. Last Accessed May 20, 2021
2. Lightgbm, https://lightgbm.readthedocs.io/en/latest/index.html. Last Accessed May 20, 2021
3. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of machine learning research **13**(2) (2012)
4. Burges, C.J.: From ranknet to lambdarank to lambdamart: An overview. Learning **11**(23-581), 81 (2010)
5. Busa-Fekete, R., Szarvas, G., Elteto, T., Kégl, B.: An apple-to-apple comparison of learning-to-rank algorithms in terms of normalized discounted cumulative gain. In: ECAI 2012-20th European Conference on Artificial Intelligence: Preference Learning: Problems and Applications in AI Workshop. vol. 242. Ios Press (2012)
6. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of statistics pp. 1189–1232 (2001)
7. Liu, X., Xu, B., Zhang, Y., Yan, Q., Pang, L., Li, Q., Sun, H., Wang, B.: Combination of diverse ranking models for personalized expedia hotel searches. ArXiv **abs/1311.7679** (2013)
8. Wikipedia: Fisher's exact test. [EB/OL] (2020), https://en.wikipedia.org/wiki/Fisher%27s_exact_test
9. Zhang, O.: Personalized expedia hotel searches, 1st place, https://www.kaggle.com/owenzhang1