Using Recommendation Algorithms to Promote Ranking of Hotels(G-23)*

Haohui Zhang(2722930)[1], Willem Van Der Spek(2607407)[1], and Yilin Li(2737659)[1]

1 Vrije Universiteit Amsterdam, Netherlands
{h17.zhang,f.w.e.van.der.spek,y45.li}@student.vu.nl

Abstract. In this work, machine learning methods and recommendation algorithms were evaluated for generating a ranking of recommended hotels based on a data gathered from user behaviour on Expedia. An extensive approach to data preprocessing, analysis, feature engineering and model selection has been described. Several machine learning methods were evaluated and bench-marked on their performance, demonstrating that the models outperform a naive benchmark.

Keywords: Machine Learning · Recommender System · Ranking Model

1 Introduction

Expedia[1], a travel shopping company that provides a variety of information such as hotels, maintains a huge amount of users' search and transaction information. In this assignment, we participated in a Kaggle competition for designing a recommendation system to give each user, who in the test set, a ranking of recommendation based on pre-processing and training on train set. We first implemented Engineering Data Analysis (EDA) to analyze the importance and attributes of the available data[2]. Some irrelevant features, such as the feature with too many Null values, will be discarded. After that, the implementation of feature engineering filled the default values in existed features and generate multiple new features in reasonable methods. Feature selection, which include analyzing the correlation between different features and using the Wrapper method to complete selection, is also an essential phase in data processing and has an important impact on the effectiveness of model training. In the model training and prediction phase, various methods such as Random Forest Algorithm[3], LightGBM[4], and Factorization Machine were applied to the prediction. During the training process, we used NDCG[5] as an evaluation metric to judge the merit of the trained model.

2 Literature Study and Related Work

Dolnicar[6]analyzed behaviors of users in using booking websites and found that hotel attributes such as location, rating, price and business model have a strong

* Supported by Data Mining Techniques, Vrije Universiteit Amsterdam

influence on user decisions. Bernardi et al.[7] have trained several models related to travelers' behaviors and preferences by analyzing and learning from behaviors of users and hotel characteristics to achieve intelligent recommendations for different users of Booking.com. In particular, Bernardi describes in detail the design of Traveller Preference Models and Traveller Context Models, which are built with ideas very similar to our task. Specifically, we generate rankings by extracting and generating valuable features associated with the binary `<user,hotel>`, and train the model using either collaborative filtering or content-based filtering methods as described in Thorat et al[8]. The output of the model is a float number representing the likelihood of a given user renting a hotel. The recommended hotels are ranked according to the magnitude of this float number.

The ranking of recommendations is divided into 3 steps: 1) implementation of feature analysis, generation and selection; 2) training of various model of prediction and ranking through analyzing NDCG and other metrics; 3) Evaluation and optimization of existed models. In detail, according to the research from David Wind[9], `Conversion of prices`, `Construction of new features` and `Removal of outliers` are main content of feature engineering. We generated over 20 new features, such as `prop_avg_price_country` from given data set. Next, multiple prediction models, such as LambdaMART[4], LightGBM[10] and Factorization Machine[11], will be imported and trained based on NDCG[5]. Then, several parameters will be modified to enhance the performance of ranking.

3 Data Exploration & Preprocessing

3.1 Data Understanding

Two datasets were given by the third party, with one being intend for training and the other solely for testing the results. Consequently, features playing a great role in prediction were removed from the test set, namely: `booking_bool`, `click_bool`, `position` and `gross_booking_usd`. In Table 1, basic descriptive statistics are reported. The dataset used for training contains a total of almost 5 million rows and its main groups are defined by the features `search_id`, denoting the search query performed by a user and `prop_id`, denoting a hotel that could possibly be offered by a search query. Some strinking observations include the strong imbalance between hotels being booked or clicked versus no further action. Regardless, all search queries contain at least one clicked hotel and all contain hotels that have a booking. When considering the hotels, there appears to be great variance in the prices and amount of bookings. Continuing, we observe the features that have missing values, which are roughly half of the initial features. The amount of missing values have been visualised in Figure 1.

We proceed to observe the *pearson correlation* (See Figure 2)between some selected features within the dataset and the amount of hits (results from search queries) as a function of time. From the pearson correlation we can observe that the only strong linear correlation in this dataset is between `booking_bool` and `click_bool`; people booking a hotel have to click the link to their website first (See Figure **??**). Finally, the click and booking probability for the hotels were

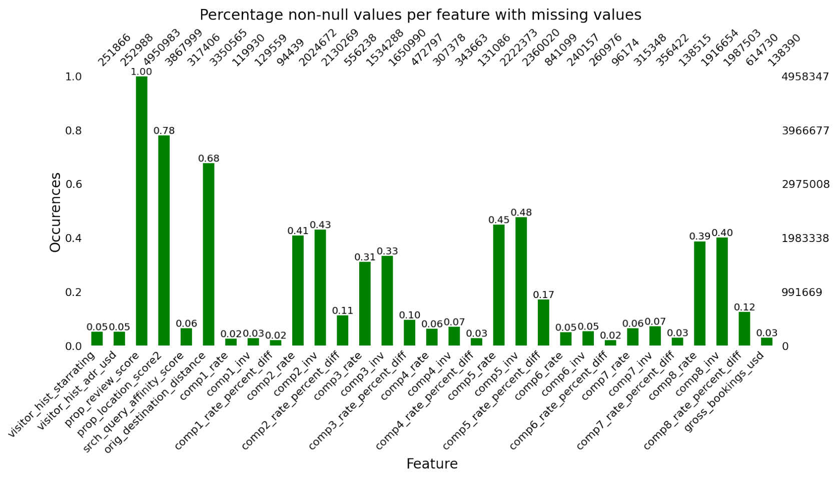

Fig. 1: It becomes obvious that there is a stark difference between the features in terms of how many of their values are missing. Missing values for each feature that contain missing values. Decisions need to be made on which of these values to retain through imputation and which features would require dropping due to too much noise being introduced by imputation. In practical, we will first process the existed features by analyzing the quality of each column and drop the column if the ratio of missing values more than **70%.**

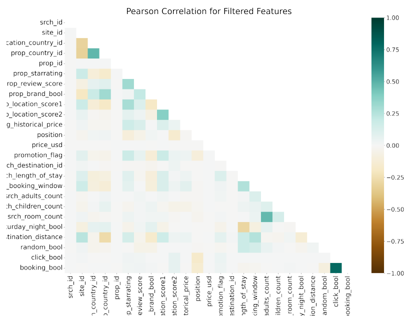

Fig. 2: The overview of Pearson Correlation for valid features, which measures the strength of the linear relationship between two features. In practical, all columns were remained in this step.

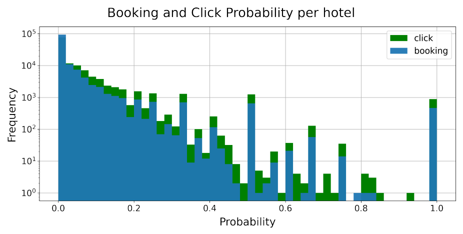

Fig. 3: Click and book probability.

Table 1: Basic Summarization for the dataset grouped by records, distinct searches and distinct hotels (properties).

Records	
Total	4958347
Booked (%)	2.79%
Clicked (%)	4.47%

Searches	
Total	199795
Mean Hits	24.82
Std. Hits	9.11
Booked (%)	69.27%
Clicked (%)	100%

Properties	
Total	129113
Mean Bookings	38.4
Std. Bookings	81.89
Mean Price	$276.55
Std. Price	$7254.48

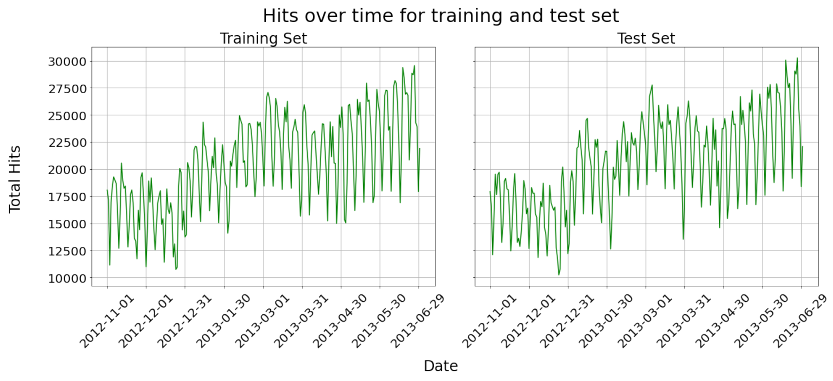

Fig. 4: As for the time series analysis, it is worth noting that the test set and training set are in the same timeframe, which tells us that the split was most likely performed irregardless of time and thus we are able to extract time-sensitive features from the training test, such as the amount of bookings per hotel or the average position of the hotel in search results. Furthermore, the time series clearly show that the amount of hits has risen steadily over time and that this pattern is similar for both training and test.

analysed (See Figure 3). The vertical axis has been plotted to logarithmic scale in order to properly see the distribution. Naturally, there is a lot of skew in this distribution where most hotels have a very low probability of being booked due most hotels not being booked by users. Therefore, we hypothesise that there are a few distinctive popular hotels as opposed to the majority being unpopular. The distribution coming from clicks has slightly more data for the higher probabilities, while having less for the probability being 0. Interestingly, there are two deviant peaks at the probability being 0.5 and 1.0. We suspect the latter one to be a resultant of hotels appearing only once in all records and being booked coincidentally. The probability peak at 0.5 are hypothesised to be relatively popular hotels that have a good balance between appearing in search results and being booked.

3.2 Data Preprocessing

In the preprocessing part, the most important task is to handle the missing value. Based on the previous analysis, we first dropped the columns which the ratio of missing values more than **70%** excluding the `srch_query_affinity_score` variable. We conducted our initial test on the baseline (random forest model) using all the raw data while filling all the missing value with 0. Categorical ID variables are not used because they were not found to provide distinguishing information. After analyzed the feature importance learnt from the random forest model, we found some variables including `prop_location_score2`, `price_usd`, `position` have tremendous relation with the targets. Therefore, we determined to use different techniques to impute these variables instead of filling zero.

1) Data set cleaning and supplementation: `price_usd`, one of the most important features, has different conventions in different countries, therefore, the value may be per night or for the whole stay. What we should do is to standardize it. In detail, train set and test set will be aggregated and grouped by `country_id` first, the mean of `price_usd` in all set will compare with the mean of price in each country. Consequently, a set of countries' `price_usd` will be expressed as Equation 1 (i.e., N express the total number of records in data set). Besides, there is some obvious outliers which are some extremely high hotel prices in the dataset. By manually looking and statistically analyzing the data, we decided to remove all training samples' price over 100,000 or equals to 0.

$$price\_usd' = \frac{price\_usd}{stay\_of\_length} if \frac{\sum_{i=1}^{n} price\_usd^{(i)}}{n} > \frac{\sum_{i=1}^{N} price\_usd^{(i)}}{N} + std \tag{1}$$

2) Imputation of missing value: Considering `orig_destination_distance` expresses the physical distance between the hotel and the customer at the time of search, we believe it is also necessary to remain this feature even if it contains about 30% missing value. The imputation of this feature can be represented as Equation 2. What's more, missing value in some features, like `prop_location_score2`, can be filled by training a Linear Regression[9] to predict the value of score based on relevant features (e.g., `price_usd`, `prop_loc_score1`)

$$Dist(srch\_id, dest\_id) \approx Dist(country\_id(srch\_id), country\_id(dest\_id)) \tag{2}$$

In addition, `position` which represents the ranking position of a particular hotel in Expedia is also a feature that cannot be ignored. As it only presents in the training set, we can calculate the average, median and standard deviation of position of a hotel to represent its popularity and assign it to the test set. Since a fraction of the search results are randomly ordered, we will only use the data when `random_bool` equals to 0 to exclude the noise. A KNN Imputer[9] will be introduced and trained to fill in the missing values of the above three variables as well as `srch_query_affinity_score`.

After analyzing the scenario, we proposed a hypothesis that users don't like missing values, hence, the rest of missing value will all simply be set to zero.

4 Data Preparation

4.1 Feature Engineering and Generation

To devise new features used for training, a mixture between *composite* and *aggregated features*. The former of which we define as a function of multiple existing features within the dataset, and the latter as taking some aggregative function to compute features belonging to a specific search or property id, were used.

1) Composite features:

Feature Name	Function
click_prob	$= P(click\_bool \mid H^{(i)}) = \frac{\sum_{i=1}^{n} click\_bool^{(i)}}{\lVert H^i \rVert}$
booking_prob1	$= P(booking\_bool \mid H^{(i)}) = \frac{\sum_{i=1}^{n} booking\_bool^{(i)}}{\lVert H^i \rVert}$
booking_prob2	$= P(booking\_bool \mid click\_bool) = \frac{\sum_{i=1}^{n} booking\_bool^{(i)}}{\sum_{i=1}^{n} click\_bool^{(i)}}$
prop_price_diff	$= e^{(prop\_log\_historical\_price)} - price\_usd$
country_price_diff	$= prop\_price\_per\_country - price\_usd$
visitor_price_diff	$= visitor\_hist\_adr\_usd - price\_usd$
visitor_starrating_diff	$= visitor\_hist\_starrating - prop\_starrating$
per_fee	$= \frac{price\_usd \times srch\_room\_count}{srch\_adults\_count + srch\_children\_count}$
total_fee	$= price\_usd \times srch\_room\_count$
score2ma	$= prop\_location\_score2 \times srch\_query\_affinity\_score$
score1d2	$= \frac{prop\_location\_score2 + 0.0001}{prop\_location\_score1 + 0.0001}$
comp_rate_sum	$= \sum_{i=1}^{8} comp\_rate^{(i)}$
comp_inv_sum	$= \sum_{i=1}^{8} comp\_inv^{(i)}$

Here, $H^{(i)}$ represents the count of feature for hotel H in results list i. Moreover,for each hotel, we created new variables to indicate its relationship to other hotels by calculating the absolute and relative differences ($relative\_diff = \frac{|x-y| \times 2}{|x|+|y|}$) between hotels and mean of hotels on numerical variables in the same search. What's more, we ranked hotel sequence for the some numerical features in one search. Finally, we added a variable fm_score which is the rank score predicted by simple Factorization Machine (Alternating Least Square Matrix Factorization) using only visitors and hotels interaction features (click_bool).

2) Aggregated features: The first two aggregated features of booked times and clicked times per `prop_id` are counted in the training dataset. The rest aggregated features are estimated on the concatenation of both training and test dataset. A typical one is calculating the proportion of occurrences of a particular hotel to the total number of searches. This ratio represents the probability of a hotel being recommended in Expedia when searching for a certain area. For the reason of lowering the noise in the data, we made some new variables which were mean, median and standard deviation of numerical features over the `prop_id`. We also proposed to aggregate and group the data based on the `destination_id` and `country_id` of the full data set, we counted the frequency of occurrence of `prop_id` in the two groups and also calculated the mean, median and standard deviation of all numerical features. We transformed `Datetime` into unix timestamp and split the `Datetime` into four variables: hour, day, month and weekday

and generated multiple valid features based on them. As an example, we counted the order of clicks on a particular hotel per month, day or even per hour. These features represent the level of attention a hotel has received during a certain time period. Similarly, we generated the rate of clicking for a hotel in a give month, or day. Based on the analysis of 4, we added the final feature called srch_hist_per_timestamp which is an actual historical feature interpreting as for hotel i at timestamp j there have been k searches.

At last, the amount of our features is expanded to 140.

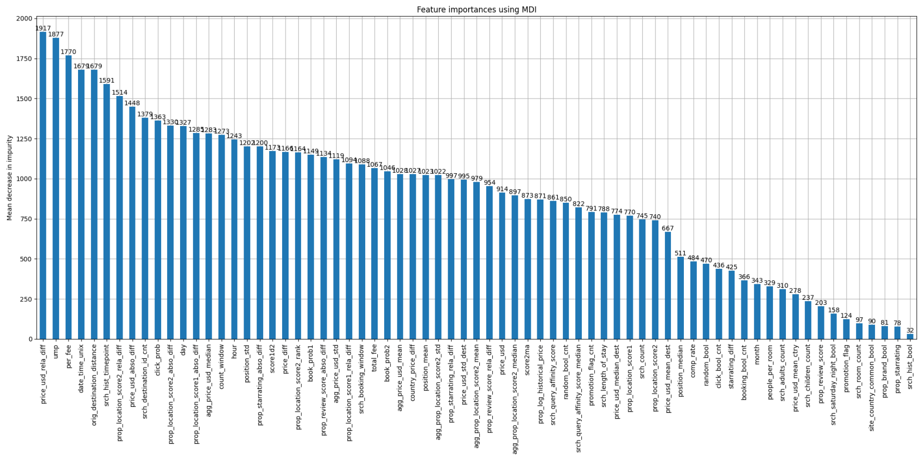

Fig. 5: We used the most typical Recursive feature elimination (RFE) as the kernel of Wrapper. In fact, we take the concatenation of the results of multiple tests as the result of feature selection.

4.2 Feature Selection

We have not only used Pearson Correlation to evaluate the importance of each feature, but also used the mutual information test method for feature testing as well. It is a filtering method used to capture any relationship (i.e., both linear and non-linear) between each feature and the label. The mutual information method can find non-linear relationships between features. It returns "an estimate of the amount of mutual information between each feature and the target", which takes values between [0,1], where 0 means that the two features are independent, and 1 means that the two features are totally correlated. In addition, we also used the Wrapper Feature Selection[12] to recursively add features and determine whether to keep a particular feature by continuously evaluating the effect of the model or not. The result of feature selection is shown in Figure 5. After feature selection, we shrink the amount of the features to 64.

4.3 Train-test Split

In order to split the dataset in an unbiased manner, we chose to implement *stratified splitting* and obtained four different splits: *training*, *validation*, *development* and *test*. Stratified splitting works by taking the unique values for the search id column, i.e. the strata for the dataset. The splitting is done by randomly selecting strata from the unique list of search id's and using all data corresponding to this search id in the desired split. The original training dataset is split up as 70% training, 21% validation and 9% development. The purpose of the development set is to review the goodness-of-fit as a result of different hyper parameter settings, with the validation set serving as a means for goodness-of-fit for the data itself. Finally, an additional test set is provided to score the overall performance of the model with the split being done by an third party.

5 Algorithm & Model

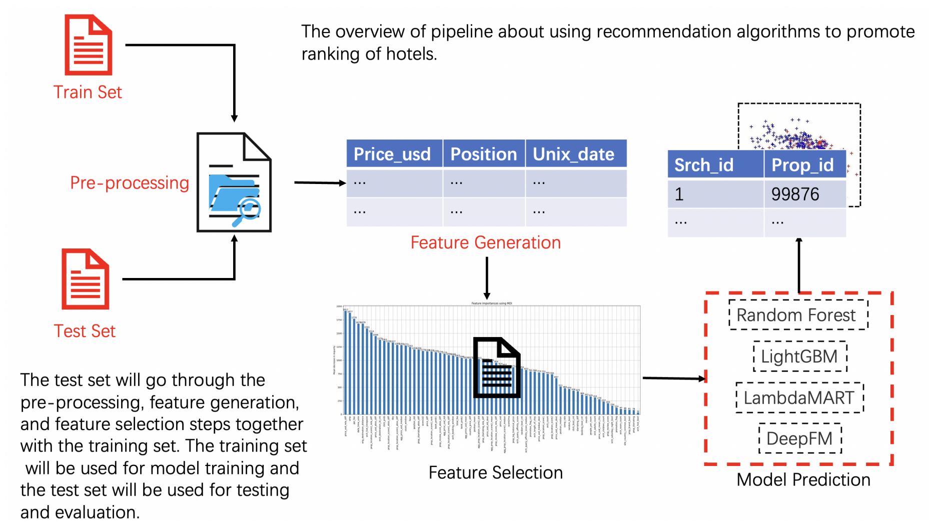
The overview of pipeline about using recommendation algorithms to promote ranking of hotels.
Train Set
Pre-processing
Test Set
Price_usd
Position
Unix_date
Feature Generation
Srch_id
Prop_id
1
99876
Feature Selection
Random Forest
LightGBM
LambdaMART
DeepFM
Model Prediction
The test set will go through the pre-processing, feature generation, and feature selection steps together with the training set. The training set will be used for model training and the test set will be used for testing and evaluation.

Fig. 6: The overview of the whole processes.

5.1 Benchmark and Baseline

According to the analysis in Winther's work, the Top-5 recommendation of hotels for each user will be selected in completely random. Besides, the establishment of another benchmark for prediction is based on summed clicks per hotel. For example, user A's results consist of the five hotels with the most hits in A's search history. The result of implementing Random Forest will be treated as the baseline of experiment. Table 2 (See bottom panel) summarized the fundamental information of the benchmark and baseline. Considering that random forest[13] can not only judge the importance of features, but also has the characteristics of not easily overfitting and can balance the error for unbalanced data sets, we believe that using it as a baseline can effectively evaluate the fitness of the model and the effect of training. This is because our dataset is characterized by unbalanced and complex feature relationships, and the advantages of random forest can offset the defects of the data set itself. Besides, the property of preventing overfitting can also help to evaluate the training effectiveness of other models.

5.2 Multiple Predictive Models

XGBoost[14], as a classification model consisting of multiple tree structures, the algorithmic idea is to recursively add trees (e.g., decision tree) and continuously perform feature splitting to grow trees. Each time a tree is added, a new loss function is learned to fit the residuals of the last iteration. The score of the sample is the sum of one score corresponding to each leaf node, which can be express as Equation 3. Unlike XGBoost, LightGBM[10] adopts a completely different storage algorithm (i.e., Histogram Algorithm) to accomplish parallel computation for each feature. In addition, LightGBM uses the GOSS algorithm to complete the sampling to speed up the execution of training. Finally, the features with good results are combined into a set of features through utilizing the Exclusive Feature Bundling (EFB), and multiple subtrees are fused according to the Leaf-wise (Best-first) Tree Growth principle to obtain the final prediction results.

$$y' = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), where f(x) = w_{q(x)}(q : R^m \Rightarrow T, w \in R^T) \tag{3}$$

Our project employs the LearningToRank (LTR) algorithm, which is a set of three main techniques for applying supervised machine learning algorithms to solve various ranking problems. LambdaMART is one of the LTR algorithms normally used for converting rankings into pairwise classification or regression problems. It is a combination of LambdaRank and MART (Multiple Additive Regression Tree). MART uses gradient boosted decision trees for prediction tasks. LambdaMART improves on this by using gradient boosted decision trees and a cost function derived from LambdaRank to rank any ranking situation[4] which fits perfectly with the goal of our task.

Factorization Machine[11] is another widely used ranking algorithm, whose main principle is to build a sparse matrix based on each feature. After constructing the training matrix, we will train the parameters of the matrix and the loss function can be chosen as Stochastic Gradient Descent. The predicted values can be represented as Equation 4. For this Ranking problem, the vector X_i is sorted by the score of $y'(X_i)$ and then the ranking for each user in test set will be generated.

$$y'(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < v_i, v_j > x_i x_j \tag{4}$$

Finally, we also implemented Singular Value Decomposition (SVD) [15] to decompose the feature matrix and discover potential patterns in the vectors. The main goal of SVD is to obtain feature matrices ϵ (i.e., $A = U\epsilon V^T$) containing singular values and to make recommendations by similarity.

5.3 Final Model and Hyperparameter

Table 2: The performance of benchmark, baseline, multiple methods and final model we tested. Note: NDCG means NDCG@5; 25 features means we tested without using result in feature generation; 64 features means we imported new features selected by us; 120 features means we imported all new features.

Benchmark	Inputted Features	Hyper-parameters	Score	NDCG
Random Selection	25 features	N/A	0.16	N/A
Sorting by Clicks	25 features	N/A	0.26	N/A
Baseline	Inputted Features	Hyper-parameters	Score	NDCG
Random Forest	64 features	criterion='entropy' max_depth=20 n_jobs=-1 verbose=2 random_state=42 n_estimators=1000	0.3829	0.4083
Multiple Methods	Inputted Features	Hyper-parameters	Score	NDCG
LambdaMART	25 features	force_col_wise=True learning_rate=0.05 verbosity=2 max_depth=20 metric='ndcg' num_leaves=255 objective='lambdarank	0.35	0.4521
LambdaMART after feature selection	64 features	force_col_wise=True n_estimators=2000 learning_rate=0.1 verbosity=2 max_depth=12 metric='ndcg' num_leaves=32 objective='lambdarank	0.3838	0.4831
LambdaMART without feature selection	120 features	force_col_wise=True n_estimators=2000 learning_rate=0.1 verbosity=2 max_depth=15 metric='ndcg' num_leaves=255 objective='lambdarank	0.2749	0.4650
LambdaMART FM as features	64 + 1 features	force_col_wise=True n_estimators=2000 learning_rate=0.1 verbosity=2 max_depth=12 metric='ndcg' num_leaves=32 objective='lambdarank	0.3807	0.4744
LambdaMART FM as features	64 + 1 features	force_col_wise=True n_estimators=5000 learning_rate=0.05 verbosity=2 max_depth=15 metric='ndcg' num_leaves=10 objective='lambdarank	0.3829	0.4924

The pipeline of generating ranking for uesrs is summarized in Figure 6. After completing feature engineering, generation and selection, we totally implemented seven different models for testing. The reason about why we treated FM and SVD as a feature will be explained in Section 6. The results and hyper-parameters are shown in Table 2. According to the result in Table, the**final model** we utilized for competition is LambdaMART with 64 features (hyper-parameters show in the fourth panel) and its score reach 0.38379 on Leader-board.

6 Results and Evaluation

6.1 NDCG@5 for Evaluation

With the aiming of evaluating the performance of ranking in various models, the Normalized Discounted Cumulative Gain (NDCG) is imported as the metric to evaluate the consistency of ranking for each search result[5]. Specifically, we need to measure the goodness of ranking on two aspects. On the one hand, we want the most relevant results to appear at the top of the ranking. On the other hand, we need all the content in the ranking to be as relevant as possible to the user's characteristics. Equation 5 shows that NDCG meets our needs in evaluating ranking, where the weights represent the relationship between position and relevance, and where the summation operation shows that NDCG also values the relevance of all content to user characteristics. Note that the k in equation should be set in "5", consider the Leader-board only evaluates the correctness of the top 5 result in ranking.

$$NDCG(k) = \frac{DCG_k}{IDCG_k} \in [0,1], whereDCG_k = \sum_{i=1}^{k} \frac{rel_i}{log_2(i+1)} \tag{5}$$

6.2 Model Results, Comparison and Evaluation

(1) Results and Comparison: All results of ranking with different methods and hyper-parameters are summarized in Table 2. In detail, the score of two benchmarks (i.e., selects randomly and sorting by clicks) can reach 0.16 and 0.26 respectively. Then, the baseline, which generated by using Random Forest with suitable parameters, achieved a score of 0.3829. Its result shows the basic effectiveness of recommendation algorithm. Besides, multiple generated features and several advanced models also aids for the process of training and ranking. Specifically, LambdaMART without generated features achieves the average of 0.35 for score in Leader-board. Comparison with the previous one, the score of LambdaMART with new features (or FM, SVD feature) improved by 3 percentage points to about 0.38.

(2) Evaluation: By comparing the performance of a range of models with different inputs, we find that using unbalanced data sets (i.e., sampling) has a negative impact on ranking. Also, in order to achieve good results for LambdaMART and Random Forest, the parameter `n_estimators` in the model should

be adjusted as large as possible, while the depth of the tree should be adjusted as small as possible. This is because too large `max_depth` can lead to overfitting of the model and thus affect the prediction results. In addition, the reason for not being able to use FM with SVD is that these two models cannot predict `srch_id` that does not exist in the train set. Therefore, we turn them into features for training.

7 Discussion

Going through all steps of pipeline, except for selecting a correct efficient model, the feature engineering steps was regarded as the most important part of our algorithm. We spent a lot of time on manually creating new features including composite features and aggregated features. However, after increasing our features using various techniques to a considerable number, the performance of our model unexpected become worse. Based on our reasonable analysis, we believed that the reason arises from some unsatisfactory features with weight will decrease the correctness of prediction, especially for random forest. Therefore, we tried to use different approaches to select the features. We firstly tried the variance threshold approach to select the robust features. However, when reviewing the selection results, we observed that this approach removes features that have higher weight of importance in the training model such as click_prob. We then tried three other methods separately correlation coefficient approach, mutual information-based approach and RFE approach, we then combined these three methods and found the 64 most effective features.

As for imputing missing values, we followed the approach suggested by some papers, filling values with zeros. What our improvement is we used the distance matrix, linear regression and KNN imputer to impute some very important variables such as prop_location_score2 and position and yield reasonable results.

The main difficulties what we have met is that the improvement in evaluation of our development set doesn't mean the same improvement in the leaderboard. Therefore, we don't a good reference locally, and we cannot conduct enough experiments to induct the best features and hyperparameters efficiently.

8 What we learnt

In this competition, we built a pipeline of a recommender system using LambdaMART as the final model to predict hotels which the users may most probably book and click into. The main lessons from this competition are about missing values, feature engineering, and predicting the right things. Before rushing to the experiments, what we first need is to do the EDA sufficiently and to read some relevant materials to find the implementable models to the task. Instead of building wheels and explored different models by ourselves, determining the best fit model according to previous solutions can let us avoid taking more roundabout courses. In addition, the ranking of the leaderboard also continuously stimulates and motivates us, pushing us to continuously optimize and improve our model.

Bibliography

[1] H. ee, B. Denizci Guillet, and R. Law, "An examination of the relationship between online travel agents and hotels: A case study of choice hotels international and expedia. com," *Cornell Hospitality Quarterly*, vol. 54, no. 1, pp. 95–107, 2013.

[2] S. Dolnicar, "Business travellers' hotel expectations and disappointments: A different perspective to hotel attribute importance investigation," *Asia Pacific Journal of Tourism Research*, vol. 7, no. 1, pp. 29–35, 2002.

[3] X. Liu, B. Xu, Y. Zhang, Q. Yan, L. Pang, Q. Li, H. Sun, and B. Wang, "Combination of diverse ranking models for personalized expedia hotel searches," *arXiv preprint arXiv:1311.7679*, 2013.

[4] C. J. Burges, "From ranknet to lambdarank to lambdamart: An overview," *Learning*, vol. 11, no. 23-581, p. 81, 2010.

[5] Y. Wang, L. Wang, Y. Li, D. He, W. Chen, and T.-Y. Liu, "A theoretical analysis of ndcg ranking measures," in *Proceedings of the 26th annual conference on learning theory (COLT 2013)*, vol. 8. Citeseer, 2013, p. 6.

[6] S. Dolnicar and T. Otter, "Which hotel attributes matter," *A review of previous and a framework for future research*, pp. 176–188, 2003.

[7] L. Bernardi, T. Mavridis, and P. Estevez, "150 successful machine learning models: 6 lessons learned at booking. com," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 1743–1751.

[8] P. B. Thorat, R. Goudar, and S. Barve, "Survey on collaborative filtering, content-based filtering and hybrid recommendation system," *International Journal of Computer Applications*, vol. 110, no. 4, pp. 31–36, 2015.

[9] O. Winther, "Concepts in predictive machine learning."

[10] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[11] S. Rendle, "Factorization machines," in *2010 IEEE International conference on data mining*. IEEE, 2010, pp. 995–1000.

[12] A. Jović, K. Brkić, and N. Bogunović, "A review of feature selection methods with applications," in *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)*. Ieee, 2015, pp. 1200–1205.

[13] G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, no. 2, pp. 197–227, 2016.

[14] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[15] F. Kleibergen and R. Paap, "Generalized reduced rank tests using the singular value decomposition," *Journal of econometrics*, vol. 133, no. 1, pp. 97–126, 2006.