DMT Assignment 2

Weiqiang Guo - 2748901, Lu Wang - 2754874, and Zheng Zhang - 2748843

Group 78

1 Introduction

The 2022 VU Data Mining Techniques Cup[1] requires the implementation of a hotel recommendation system that anticipates which hotels a user is most likely to book. We will reveal how we interact with data and build our model in this report.

2 Related Work

Since this assignment is based on another Kaggle competition ICDM 2013 held by Expedia, there are numerous exceptional solutions available that allow us to stand on the shoulders of giants.

The champion of this competition won with a score of 0.54074, they applied LambdaMART model and blender with GBDTs, NNs, SGD models. They took all the numerical features into account and process them in three ways: average, stander deviation, and median of numeric features per prop_id. According to the author, due to the great amount of data and multiple models being applied, the training time is horrible. Unfortunately, this team was disqualified for not full fill all the requirements of the competition even though they got the highest score.

The second place in this competition was taken by Owen Zhang, he used Ensemble of Gradient Boosting Machines (GBM) to achieve a 0.53984 score. The punchline of the work is an explicit distinction between the user's rational behavior and random noise. The most prominent predictors are Position, Price, Location desirability, and Estimated Position. Estimated Position takes great credit for the precise prediction, which is a new numerical feature based on the average of three attributes given by the dataset(prop_id/dest_id/target_month) and position of the same hotel in the same destination in the previous and next search. The approach of Downsampling was introduced to reduce the training time and improve model performance.

There are essentially three types of models that are appropriate for Learning to Rank(LTR) problems, Pointwise, Pairwise, and Listwise methods[7].

1. Pointwise: Pointwise LTR models optimize for predicting a key metric. All the standard regression and classification algorithms can be directly used for pointwise learning to rank. Generally, this model will convert a ranking problem into a prediction problem.

2. Pairwise: Pairwise approaches look at a pair of documents at a time in the loss function. Given a pair of documents, they try and come up with the optimal ordering for that pair and compare it to the ground truth. In certain circumstances, the Pairwise model has a better performance than the Pointwise model in piratical, because predicting relative order is closer to the nature of ranking than predicting class label or relevance score.
3. Listwise: Listwise takes document lists instead of document pairs as instances in learning. The cost function in Listwise is the correctness of lists contained in the dataset. LambdaMART is a Listwise type LTR algorithm that we applied in our solution, which is based on the LambdaRank algorithm and the MART (Multiple Additive Regression Tree) algorithms. By applying LambdaMART, the ranking problem based on the search engine results will convert to a regression decision tree problem.

3 Exploratory Data Analysis

3.1 Overview

The training set contains 4958347 instances and 54 features. Each instance in the dataset stands for a combination of a search query by a customer with one specific hotel property. The test set contains 4959183 instances and 50 attributes. Four attributes are included in training set only: position, click_bool, booking_bool, gross_booking_usd.

We classified the features into groups based on their characteristics, which are as follows:

Search-Related Features like srch_id, date_time, srch_booking_window, etc. These features depict the basic information about the searches, such as the time of the search, the website of the search, etc.

Visitor-Related Features like visitor_hist_starrating, visitor_hist_adr_usd, etc. These features describe the user's historical information, such as the mean star rating of hotels the customer has previously purchased, etc.

Static Hotel-Related Features like prop_id, prop_brand_bool, prop_review _score, etc. These features depict the basic information about the hotels, such as if the hotel is a part of a major hotel chain, the average customer review score the hotel, etc.

Dynamic Hotel-Related Features liker click_bool, booking_bool. These features describe the information about the interaction between the user and the hotels, such as if the user clicks the hotel, if the user books the hotel, etc.

Competitor-Related Features like comp2_inv, comp2_rate, etc. These features depict the difference between the hotel and its competitors, such as the price difference, etc.

3.2 Statistics of attributes

There are several attributes that can provide us with a general overview of the dataset, and we created a Table. 1. to allocate the summary statistics of these attributes.

Table 1. Summary statistics of training set

Attribute	Summary statistics
click_bool	4.47% of the whole instances were clicked.(click rate)
booking_bool	2.79% of the whole instances were booked.(booking rate)
srch_id	Contains 199795 unique ID. Each ID appears from 5 to 38 times.
date_time	From 01/11/2012 to 30/06/2013.
prop_id	Contains 129113 unique hotel ID.
prop_brand	63.47% of the hotel is part of the hotel chain.
	36.53% of the hotel is an independent hotel.
	2.92% of the booking rate from a major hotel chain, while 2.57% not.
	4.49% of the click rate from a major hotel chain, while 4.45% not.
promotion_flag	21.56% of the instances had a promotion flag.
	2.48% of the booking without promotion flag, and 3.92% with flag.
	4.04% of the click without promotion flag, and 6.03% with flag.
random_bool	29.60% of the instances was sorted in random.
	0.53% of the booking with random sort, 3.74% with normal sort.
	4.66% of the click with random sort, 4.40% with normal sort.

3.3 Correlations

In this part, we calculate the correlation between each variable and click_bool, booking_bool. It turns out that several features, like srch_query_affinity_score, prop_location_score, position, prop_review_score, prop_starrating, etc, have a strong correlation with click_bool, booking_bool. This observation guides us to pay more attention on these features in the feature engineering part.

3.4 Missing values

There are 54 attributes in the training dataset, and 31 of them contains missing values, shown as Fig.1. In particular, 28 attributes have more than 40% of missing values, and most of them come from the competitor's data. Two historical customer data(visitor_hist_starrating and visitor_hist_adr_usd) also contains missing as there is no purchase history on the customer, while the missing value in srch_query_affinity_score shows a hotel did not register in any searches.

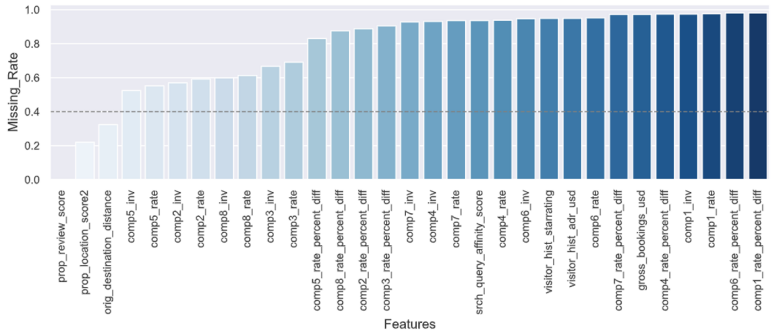

Fig. 1. Missing value in the training set

Outliers The original training dataset contains a large number of outliers, which has not been preprocessed. The Boxplot is commonly used to analyze the outliers. We take attribute of price_usd and srch_length_of_stay for instance, shown as Fig.2. There are widely distributed outliers in these attributes. For example, the price of a hotel ranges from 0.01 to 10000000, and the number of nights stay ranges from 1 to 57. There apperently are a lot of outliers. Outliers act as noise, which might have a negative effect on our model, for example, our model might learn some wrong patterns from these outliers, which results in poor performance.

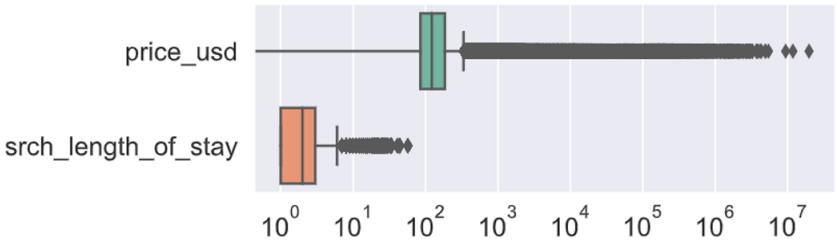

Fig. 2. Boxplot of two features for example

3.5 Numerical data distributions

Position We noticed that the Expedia website displayed search results either in random order or in normal sort order, the value of 1 stand for random search in random_bool. There is an dependent relationship between the value of position and the number of bookings and clicks, shown as Fig.3. The value of a position is inversely proportional to the number of clicks and bookings.

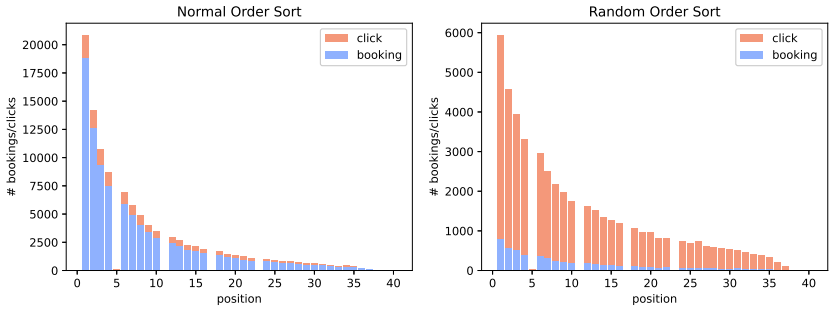

Fig. 3. Booking and click rate in different position group by random search

Date_time The original format of date_time contains calendar time is yyyy-mm-dd hh:mm:ss, which can not be used directly, and we use Python's panda library to solve this problem. DatetimeIndex() splits the date_time into year, month, week and day. Moreover, the amount of booking and click are related to the daily trend, show in Fig.4. Apart from the missing period, we find that the first half(spring and summer) of the year is significantly higher than the rest half(winter). Therefore, we consider introducing seasonal attribute into the model.

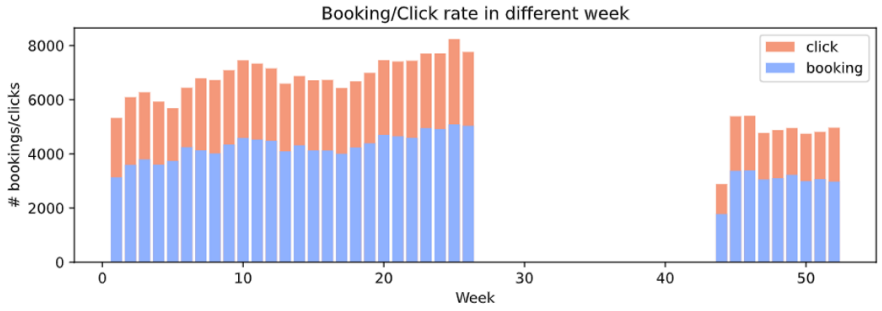

Fig. 4. Booking and click number in different weeks among one year

4 Feature Engineering

4.1 Dealing with missing values

First, we remove the features with more than 40% missing values. The first reason is that these features do not have high correlation with the target. The second reason is that they have too many missing values which means that they

only carry negligible information. There is also a positive consequence in that we can lower the model's running time and exclude noises from our training.

After removing the features above, we still have some features with missing values, such as prop_review_score, prop_starrating, etc. Generally, people normally do not like to make their purchase with incomplete information. Hence, we fill the missing values with the worst case, that is -1.

4.2 Date time feature

In terms of date time feature, we first divide it into four seasons, which are, spring, summer, autumn, and winter. From our experience, we believe that many people prefer to travel in a certain season, like spring. This trait, however, does not appear to have a strong relationship with the target.

Furthermore, we find that the original date time refers to the time of searching rather than check-in. We think that check-in date time is a more convincing choice since people have different preferences when comes to booking a hotel. For example, some people prefer to book a hotel in advance, but some people are more procrastinated. Hence, we transform the original date time into the check-in date time. However, it still does not have a big impact on the target.

4.3 Position

As we can see from Fig.3, there is a clear correlation between position and click, booking. Inspired by paper[8], we decide compute the average position for every combination of prop_id and srch_destination_id. At the same time, we add this feature to the test dataset. It turns out that this feature have a positive effect on our result as we expect.

4.4 Add normalized features

We notice that feature price_usd has different displaying format. The displaying fees may be per night or for the whole stay. Hence, we decide to normalize price_usd concerning prop_id using the following formula.

$$normalized\_price\_usd = \frac{price\_usd - mean(price\_usd)\ w.r.t.\ prop\_id}{std(price\_usd\ w.r.t.\ prop\_id}$$

Furthermore, we also add normalized numeric features with respect to prop_id, srch_id and srch_destination_id, such as prop_review_score, prop_starrating, etc, which is inspired by one of submissions in the 2013 Expedia competition[8]. It turns out that these features have a significant effect on the result.

4.5 Add Listwise features

In learning to rank problems, what we care about is the ordering of instances in the corresponding group. For pointwise/pairwise methods, they are not able

to leverage listwise information which should be useful. Considering that we are using pairwise method LambdaMart, we decide to add some listwise features which bring listwise information to our model.

The listwise features we use are shown below, some of them are inspired by the paper[8]

$$hist\_price\_diff = prop\_log\_historical\_price - price\_usd$$
$$price\_diff = visitor\_hist\_adr\_usd - price\_usd$$
$$prop\_starrating\_diff = visitor\_hist\_starrating - prop\_starrating$$

Apart from that, we also add rank information and statistics information(like mean, median and standard deviation) with respect to prop_id.

It turns out that these features have a positive impact on our model. For our problem, we have a clean dataset and mature algorithm. As a result, we should concentrate more on feature engineering, which will extract as much useful information from the data as possible. This procedure requires a good understanding of the data. In our opinion, the most important aspect of this competition is to perform data analysis and extract useful information from the data. Adding new features or transforming the origin features from dataset, both of these methods are frequently used in feature engineering.

5 Model and Evaluation

5.1 Evaluation metric

The Evaluation metric for this competition is Normalized Discounted Cumulative Gain(NDCG) that is widely used in the learning to rank problems[5]. The relevance score is a part of NDCG. In our case, the relevance score of hotels for each user query is either set to 5(if the hotel is booked), 1(if the hotel is clicked), or 0(if the hotel is neither clicked nor booked). Based on the relevance score, the definition of Cumulative Gain(DCG) for each query is given as follows:

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{1}$$

where rel_i the relevance score of the hotel in the position i, p the maximum rank of corresponding query. Next, the definition of Ideal Discounted Cumulative Gain is given:

$$IDCG_p = \sum_{i=1}^{|REL_p|} \frac{rel_i}{log_2(i+1)} \tag{2}$$

where REL_p represents the list of relevant documents (ordered by their relevance) in the corpus up to position p.

Based on the DCG and IDCG, the definition of NDCG for each query can be given as follows:

$$NDCG_p = \frac{DCG_p}{IDCG_p} \tag{3}$$

Averaging $NDCG_p$ values for all queries(even in the case that queries have different length) can be a good measure of the ranking algorithm performance. It ranges from 0 to 1.

5.2 Algorithm

The main model we used in this competition is LambdaMART[10] due to the fact that it turns out to be a very successful algorithm for solving ranking problem. For example, The Champion of 2010 Yahoo! Learning To Rank Challenge used an emseble of LambdaMART[3].

LambdaMART orignated from LambdaRank and MART[4]. MART uses "Gradient" to improve the model. LambdaRank[2] specifies the "Gradient" at any point during training. We can say that LambdaMART is the marriage of the two. Briefly speaking, LamdaMART provides MART with Gradient(called as Lambda) and uses MART to predict the relevance score which we mentioned above. Then, we order the instances within each query according to their relevance sore. The experiments have shown that LambdaRank and LamdaMART can optimize NDCG directly[11]. Hence, it is a reasonable choice to use it to solve our ranking problem.

5.3 Data Split

The data was split into training set and validation set using GroupShuffleSplit() function, which can generate a user-determined number of random stratified splits. We take 70% of the data as the training set and 30% as the validation set. This is called as the hold out method. In addition, we guarantee that the proportion of click and booking is same in both of sub-dataset.

There are two reasons to explain why we split the dataset in such a way. First, our dataset is quite large, using cross validation will take much time, which is infeasible for us. Second, it turns out that the validation set got from the hold out method can measure the generalization error of the model very well[9]. In this case, we chose it to shorten our training time without sacrificing generalization error accuracy.

5.4 Parameter Tuning

Parameters also play a crucial role in training model. A good combination of parameters can not only improve our evaluation score, but also avoid overfitting. In our case, we use LGBMRanker() function of LightGBM library[6] to build our model. LGBMRanker() provides a lot of parameters, however, we have to limit parameter tuning to six parameters which we think most important due to time

constraints. They are boosting_type, n_estimators, max_depth, num_leaves, learning_rate and reg_alpha.

Unfortunately, scikit-learn and lightgbm libraries do not provide well-established APIs to finish parameter tuning for learning to rank problems when using NDCG as the evaluation metric. Hence, we decide to manually tune parameters based on the idea of Grid Search method.

Based on the the results of a lot of experiments, we find that the combination that

- boosting_type="gbdt"
- num_leaves=255
- n_estimators= 5000
- max_depth=3
- learning_rate=0.1
- reg_alpha=20

is a good choice, which gives a score 0.39351 in Kaggle.

5.5 Final Model

With 70% of the data for training, we gained a score of 0.443101 in the training set, 0.399887 in the validation set, and 0.39351 in Kaggle with LamdaMART. It turns out that the chosen model has good scores on these three datasets, which proves that our model have a great generalization ability. Based on this observation, it is a reasonable option for us to train our model in the entire data since it makes most use of the data while avoiding overfitting. .

In the end, we input the combination of parameters mentioned above to final model, and the model trained with complete data. Finally, we get a 0.39720(our best submission) score on Kaggle.

6 Conclusion and Future work

6.1 Conlusions

In this report, we depict our approach, which incorporates ideas from several winning solutions of Expedia 2013 competition. We first dive into the related work, which helps us understand the question and find the right direction quickly. Then, we analyze the data deeply, such as missing values, data distributions and other statistics about the data. Our data analysis and related work guide us the right direction of the feature engineering. We removed some features that had far too many missing values, filled in the missing values of features with high correlation, and created new features based on the dataset. Next, we take 70% of the entire data as our training data and 30% as our validation data. It turns out that it is feasible to split data in such a way due to the fact that we have a quite large dataset. Furthermore, we use LambdaMART model to learn from data and tune its parameters. After got a good model which has a good

generalization ability, we build our model leveraging information of the entire data. Finally, we get a score 0.39720 in Kaggle, which turns out to be a quite satisfying result.

6.2 Future work

Although we have a good result based on work we mentioned above, there is plenty of room for improvement.

First, we could do more feature engineering which plays a vital role in the competition. For example, we do not use the information in the competitor features since they have too many missing values, but we still can try to extract some useful information from them. e can combine them to create a feature which summarize the information about competitors.

Second, we could build other models, such as logistic regression, random forests, and deep learning algorithms. Then, we could dive more into ensemble methods, such as stacking, blending, which are commonly used in machine learning competition. It turns out that they usually outperform the single model. In fact, we have tried blending method to ensemble several models, however, its results are not satisfying.

Third, we could try to transform our origin dataset which is imbalanced into a balanced dataset. This method might help us reduce training time and improve the predictive result.

7 Gains from Assignment

The two most prominent gains we made in this assignment were in the EDA and feature engineering sections. Our graphing skills were enhanced, and we learned how to extract useful information from these graphs in order to make sound decisions. We learned quite a bit about feature engineering by studying previous high-scoring projects, such as normalizing numerical data and creating new features.

We actually tried two different approaches for this assignment, one is the LambdaMART, and another is blending models based on several linear regression models. This allows us have a firsthand experience of the differences caused by the various model when dealing with the LTR problems.

Furthermore, the large-scale dataset involved in this assignment forced us to optimize the way we processed the data and made predictions, otherwise, training time would be horrendous and our PC hardware would reach its limit. For example, we chose to remove some attributes that had a lot of missing values with low correlation, which is one of many effective ways to reduce running time and obtain a more clear dataset. Also, we applied vectorization techniques instead of using for-loops or while-loops to avoid unnecessary time consumption.

References

1. Vu data mining techniques cup (2022), https://www.kaggle.com/competitions/2nd-assignment-dmt2022/
2. Burges, C., Ragno, R., Le, Q.: Learning to rank with nonsmooth cost functions. Advances in neural information processing systems **19** (2006)
3. Chapelle, O., Chang, Y., Liu, T.: The yahoo! learning to rank challenge (2010)
4. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of statistics pp. 1189–1232 (2001)
5. Järvelin, K., Kekäläinen, J.: Ir evaluation methods for retrieving highly relevant documents. In: ACM SIGIR Forum. vol. 51, pp. 243–250. ACM New York, NY, USA (2017)
6. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems **30** (2017)
7. Liu, T.Y., et al.: Learning to rank for information retrieval. Foundations and Trends® in Information Retrieval **3**(3), 225–331 (2009)
8. 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 preprint arXiv:1311.7679 (2013)
9. Raschka, S.: Model evaluation, model selection, and algorithm selection in machine learning. arXiv preprint arXiv:1811.12808 (2018)
10. Wu, Q., Burges, C.J., Svore, K.M., Gao, J.: Adapting boosting for information retrieval measures. Information Retrieval **13**(3), 254–270 (2010)
11. Yue, Y., Burges, C.: On using simultaneous perturbation stochastic approximation for learning to rank, and the empirical optimality of lambdarank. Microsoft Res., Redmond, WA, USA, Tech. Rep. MST-TR-2007-115 (2007)