Data Mining Techniques - Assignment 2

Xinyu Fu 2656355 and Gongze Cao: 2656348

VU-DM-2020-118

1 Assignment Introduction

The assignment is detailed described on Kaggle:hotel search by Experdia. Short version: Your task is to predict what hotels properties listed as a result of a hotel search a user is most likely to click on. Of course, more people have worked on such predictions. Can you find some other people that have tried to make such predictions (e.g. from the Kaggle competition)? And what have they used as most prominent predictors? Have other people that participate in the competition mentioned anything about their approaches? Please spend a couple of paragraphs on this topic (i.e. related work) in your report.

2 Related Works(Business's understanding)

We used Backward Search and use k-fold to select features with considering the trade-off between accuracy and the number of features. [5] proposes to composite features which group important features by related field. For instance, 'price_usd' grouped by 'prop_id'. We borrowed this idea for our feature engineering.

For the model part, we noticed that most popular method before around 2015 is either random forest or Boosting method, some might resort to an ensemble of many models such as RF, Adaboost, Naive Bayes. More and more people start to use XGBoost or LightGBM, which include high-performance implementation of GBDT and built-in ranking loss. From a method angle, there are majorly 3 methods for ranking in recommendation: Pointwise, PairWise and ListWise.

- Poinwise method: This method proposed to tackle the ranking task as a prediction task. Many methods varies in the form, such as regression, classfication, multi-class classification, ordinal regression and so on.
- Pairwise method: [1] first proposed to use neural network to learn the preference function. [7] is based on a probablistic assumption that the preference between two candidates could be modelled with a logistic regression on the difference of two score. RankingSVM[2] adopt SVM and tackle the ranking problem with a classification framework.
- Listwise method: [8] adopted a probablistic framework to obtain a ranking distribution of all docs, then update the parameter by calculating a a soft rank loss. [9] Adarank use boosting to directly update any ranking metric, it offers ability to adaptly optimize any metric. LambdaMart[1] combines RankNet and MART[6] and construct a lambda gradient listwisely, make the update of the model focus on the top of the current list, therefore resulting a listwise model.

3 Data Understanding and exploring

3.1 Dataset statistics

The dataset contains training and test data. The size of the training data is 1.18GB. Training data comprises 4958347 rows and 54 attributes and 199795 unique search id. On the other hand, The size of the test data is 1.12GB. Test data contains 4959183 rows and 50 attributes (4 features less than the training data) and 199549 unique search id. Both of them are considered being extensive datasets. Thus some of the machine learning algorithms may not work on them appropriately.

There are three types of object in attributes: categorical attributes, continuous attributes, DateTime object. We separate them because some algorithm may have worse outcome without distinguishing them (i.e., regressor and mostly non-tree based algorithm)

Besides, we found four features only in training but not in the test data. They are "click_bool", "booking_bool", gross_bookings_usd","position'. Where we need to build a target column from them and consider the impact of the 'position' feature.

The 54 attributes in the training set could be grouped into four groups: visitor info, property info, competitor info, search info as the table shows below:

Table 1. Attributes containing missing values from random sample $n_rows = 100,000$

Group name	Attributes	Missing value in %	
Visitor information	attributes start with 'visitor_', 'srch_id', and 'site_id'	visitor_hist_starrating	0.94561
		visitor_hist_adr_usd	0.94542
Property information	attributes star with 'prop_', 'position' , and 'price_usd','promotion_flag'	prop_review_score	0.00141
		prop_location_score2	0.21728
Competitor information	Members starting with 'comp'	A big amount of missing values, attributes dropped	
Query info	attributes start with 'srch_','random_bool', 'orig_destination_distance'	srch_query_affinity_score	0.93674
		orig_destination_distance	0.32092
Evaluations	'click_bool', 'booking_bool', 'gross_bookings_usd'	gross_bookings_usd	0.97209

3.2 Plots and Remarks about data distributions

Before continuing visualizing the data, we decide to work with a sample of training data to save RAM and computational power. Thus we randomly select $nrows = 100,000$ from training data, which takes up 40 megabytes memory.

We performed a box plot to have a general view of the training data, as shown in figure Fig.1 (left). It also reveals the potential outliers, but the plot looks alright. We decide to leave it for now since some of features will be dropped later.

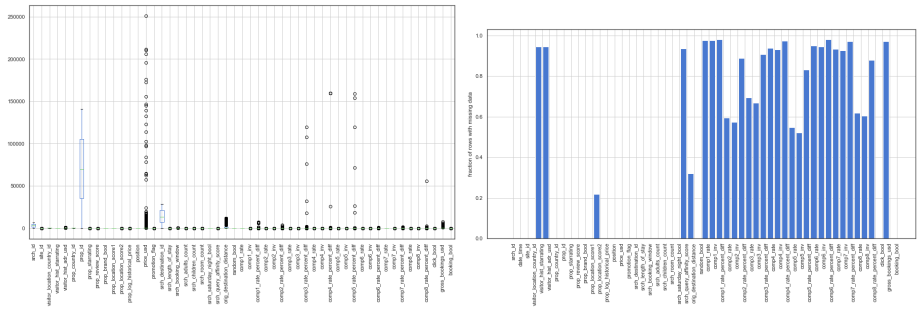

Fig. 1. Box plot of training sample (left) Bar plot of missing values (right)

As shown in figure Fig.1 (right), there are tons of missing values in many attributes. It is needed to decide which features shall be dropped or imputed.

We firstly dropped competitor features with large (60%) missing values and performed an initial importance rank of features by XGBoost. So that we know which feature we should pay more attention to. The reason we select XGBoost is 1. XGBoost is commonly used in the L2R algorithm. 2. It can ignore the missing values (Null) 3. It deals with categorical values properly since it is a tree-based algorithm.

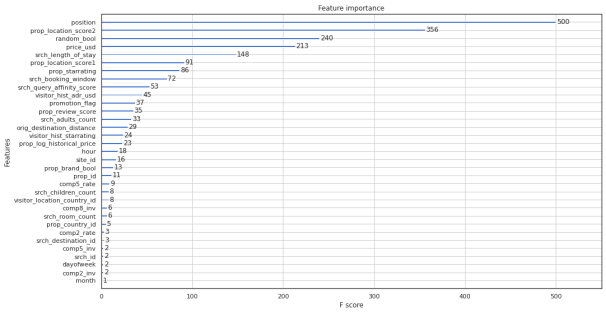

Fig. 2. Initial importance rank of features after competitor features with large (60%) missing values dropped (left) (right)

Also, we noticed some continuous features are not distributed as a normal distribution. For example, the 'price_usd'(from Fig.1) looks skewed to small values and ori_des_distance as well (from Fig.3). We picked these two features because we found they are important features after the initial importance rank by XGBoost.

Also, we knew that 'position' is an essential feature from the important rank. And since we found a paper saying position bias is a common problem in Learning to Rank problem [3]. So we plotted the mean probability of clicking as shown

in the Fig.3 (right), it shows in some positions, the clicking probability is significantly higher than others, and we should take it into account before modeling.

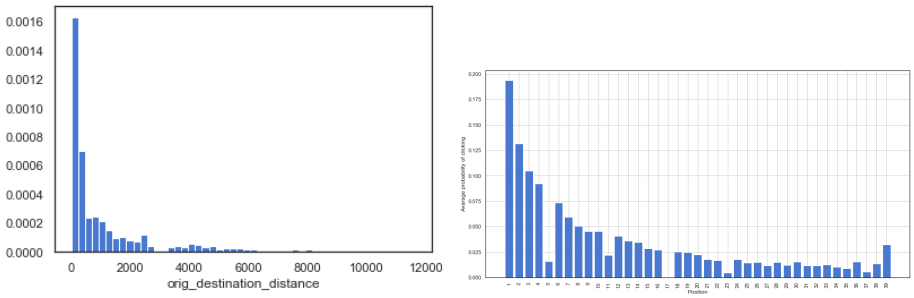

Fig. 3. Distribution of ori_des_distance feature (left) (right)

4 Data preparation and engineering

4.1 Data pre-processing

Missing values We firstly dropped the features containing more than 60% of values being Null except the following features: "visitor_hist_adr_usd", "visitor_hist_starrating", "srch_query_affinity_score", "prop_location_score2". Because from initial important rank, we knew they are informative.

So far, we are not sure if to impute the rest of the missing values since we don't know our algorithm yet. We decide to leave them. And it turns out if we use LambdaRank or XGBoost, we can leave the missing values Null.

Datetime object Values from DateTime object is divided into the hour, day of week, and month because DateTime object can not be seen as input to machine learning ranker (MLR) models such as XGBoost and LGBMranker.

Build Target Column The target column is defined as the following: $5 = booked, 1 = clicked, 0 = others$. So we joined the 'click_bool' and 'booking_bool' with weight 1 and 4 respectively to get the label as

$$target = 1 * click\_bool + 4 * booking\_bool$$

Feature Normalisation We logarithmiced the continues features: 'price_usd' and 'orig_destination_distance'. Fig.**??** shows the qqnorm after the log. It turns out they are close to a normal distrbution (we see the potentional outliers in 'price_*usd*' again, but we decide to leave them because they are many values).

Thus, we normalised them to standard normal distribution. For the rest of continues features, we standardised them from 0 to 1 by MinMaxScaler from sklearn:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

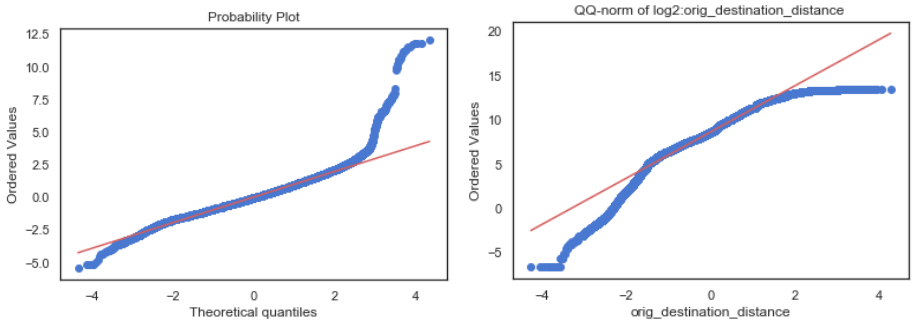

Fig. 4. Distribution of ori_des_distance feature (left) (right)

4.2 Feature Selection/Variables analysis

Normalisation concerning dependent feature Features like "price_usd" may vary from different cities, time, search id. Then We decided to normalize the top important features with concerned dependent variables. For instance, we normalise the prop_location_score2 with respect to "srch_destination_id", "price_usd" with respect to "prop_id" and so forth. This idea is borrowed from [5] and the corresponding GitHub repository.

As a result, we found the importance of some features grows after normalization. For instance, as shown in the Fig.5, the prop_location_score2 with important fator, 803 is less than the importance of prop_location_score2_norm with 943 and so forth.

feature selection To select the important features, we performed the XGBoost for the "engineered" features again. As shown in Fig.5

Besides, we performed the correlation map among all available features so that we dropped the features: 1. less important features, according to Fig.5 2. Features with strong correlation (0.85 or more) to another one but has less importance(i.e. 'price_usd' is less important to target than 'price_usd_norm_by_propid'.

Also, we dropped the "position" feature, because we failed our test to predict the "position" in the test data (more details in the last section: Failiues and learned).

Eventually, we select the top 17 features from importance rank (Fig.5) excluding "prop_location_score2", "price_usd", "position" as shown in the Fig.5

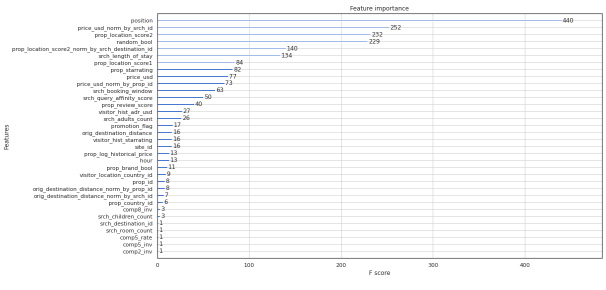

Fig. 5. Important rank for "fixed" features.

5 Models and evaluation

5.1 Task briefing

The user i has queries noted as q_i, the candidate set of documents is $c_{i,1}, c_{i,2}...c_{i,N}$, the correct result is assumed to be sorted as noted as $c_{i,1}, c_{i,2}...c_{i,K}$. The candidate document set is generally larger than the final returned set, therefore $N >= K$. For example, given a question query, choose the best 5 answers from 10 candidate sentences, and the returned list should also be a list of document of length 5, sorted by their correlation with the queries.

5.2 Methods

There have already been many successful model for LTR(Learning to Rank) task and they could be roughly classified to 3 categories: Pointwise, Pairwise, Listwise. The detailed explaination is as following:

Pointwise Pointwise treats the ranking problem as a binary classification problem, and the training samples are organized into a triple $(q_i, c_{i,j}, y_{i,j})$. $y_{i,j}$ is a binary value that indicates whether $c_{i,j}$ is a correct answer to the q_i. We can train a binary classification model:$h_\theta(q_i, c_{i,j}) \rightarrow y_{i,j}$, $0 \leq y_{i,j} \leq 1$. The objective of the model is to minimize the cross entropy of all the query and candidate pairs in the dataset.

In the prediction stage, the scores obtained from binary classification model h_θ is used to sort all candidate sentences, then top-k scored sentences are selected as the top-k answers.

Common Pointwise methods include McRank[4] and so on.

Disadvantages:

- The pointwise method does not consider the internal dependencies between the docs corresponding to the same query. On the one hand, the samples in the input space are not IID, which violates the basic assumption of ML. On the other hand, the structure between the samples is not fully utilized.

Moreover, when different queries have different numbers of docs, the overall loss will be dominated by the query group that has larger number of docs. As mentioned earlier, each group of queries should be equivalent.
- The loss function also has no position information from model to prediction ranking. Therefore, the loss function may inadvertently overemphasize those unimportant entries, that is, those entries that has a small impact on the user experience.

Pairwise In pairwise ranking model approach h_θ allows the correct answer scored significantly higher than the wrong candidate answer. To give a query, pairwise gives a pair of candidate answers to learn and predict which sentence is the best answer to the query. Sample training for (q_i, c_i^+, c_i^-), in which q_i is a query, c_i^+ is the correct candidate, c_i^- is a wrong candidate.

The loss function is the hinge loss function:

$$loss = max\left\{0, m - h_\theta(q_i, c_i^+) + h_\theta(q_i, c_i^-)\right\} \tag{1}$$

where m is the threshold. If $h_\theta(q_i, c_i^+) - h_\theta(q_i, c_i^-) < m$ and the loss is greater than zero, this means that the the incorrect entry could come above the correct result; if the loss equals to 0, the model correctly result came in above the incorrect result. The purpose of the hinge loss function is to make the score of the correct result greater than the score of the wrong result by a margin of m.

Pairwise method has many implementations, such as Ranking SVM, RankNet, Frank, RankBoost, etc.

Disadvantages:
- The number of candidate pairs will be twice the number of candidates, so the issues of imbalance of the number of doc between queries are worsened in the pairwise class method.
- The pairwise method is more sensitive to the noise in label than the pointwise method, that is, one wrong label can cause multiple doc pair label errors.
- The pairwise method only considers the relative position of the candidate pair, and the loss function still does not include any position information.
- The pairwise method also does not consider the internal dependencies between the candidate pairs corresponding to the same query.

Listwise What Pairwise and Pointwise method do is actually factorize the whole sorted series to multiple compositions, and model each composition using auxiliary loss function. The reason is that normally the metrics for a whole query-result pair is not differentiable with respect to parameters. But still many methods are developed to train such model with comprehensive target loss like NDCG or MAP.

The Listwise method is a much more direct method than pariwise and pointwise, as it optimize the metrics directly. It focuses on its own goals and tasks and directly optimizes the document sorting results, so it is often the best.

Listwise method also has many implementations, such as AdaRank, SoftRank, LambdaMART.

5.3 Metrics

We used the NDCG@5 as the metric for cross validation, as it is also the test score. NDCG stands for Normalized Discounted Cumulative Gain. It is a common metric for sort and recommendation evaluation. The recommendation system usually returns an item list for a user, assuming that the list length is K. In this case NDCG@K can be used to evaluate the gap between the sorted list and the user's real interaction list. The calculation of NDCG is showing as following:

- Gain: Relevance score of each item in the list

$$Gain = r(i) \tag{2}$$

- Cumulative Gain: an accumulated Gain along the K items list

$$CG@K = \sum_{i}^{K} r(i) \tag{3}$$

- Discounted Cumulative Gain: Considering the immportance of the position, a higher-correlated result has higher impact on the result, and the lower-ranking items have discounted impact:

$$DCG@K = \sum_{i}^{K} \frac{r(i)}{\log_2(i+1)} \tag{4}$$

- DCG can evaluate the recommendation list of one user. If you want to use this metric to evaluate a recommendation algorithm, you need to evaluate the recommendation list of all users. But the DCG between different users does not have clear interpretations. So the natural idea is to calculate the DCG score of an ideal situation (or obtained from real-life simulation) for each user, expressed by IDCG, and then use the ratio of DCG and IDCG of each user as the score for the specific user. Averaging the score among all users will give the final NDCG.

$$NDCG_u@K = \frac{DCG_u@K}{IDCG_u} \tag{5}$$

$$NDCG@K = \frac{NDCG_u@K}{|u|} \tag{6}$$

5.4 Experiments

We conducted the experiments with two different framework, XGBoost and LightGBM. Below is the detailed configuration and cross validation performance.

XGBoost We use the provided Scikit-Learn api called "XBGRanker", and set **learning_rate=0,1**, **subsample_ratio=0.5**, then perform grid search on **objective=rank:pairwise, rank:ndcg**, **n_estimators:500, 1000**, **max_depth:[3,5,10]**. We found out:

- Setting **max_depth** to 10 will lead to severe overfitting, could result up to a 0.2 gap in terms of ndcg@5 between train split and validation split.
- It is generally beneficial to use more steps(estimators) and set a reasonable early stop bound.
- The local best cross-validation ndcg@5- among all configurations is 0.1894, which has params as **objective=rank:pairwise**, **n_estimators:1000**, **max_depth:5**. The kaggle test result is 0.15780, which indicates we might need to improve either feature engineering or explore more configurations.

LightGBM For LightGBM we use **objective='lambdamart', metric='ndcg', max_position=5, label_gain=[0,1,2]**, and did a grid search on several parameters: **n_estimators=[400, 1000], boosting=['gbdt', 'dart'], max_depth:[-1.5]**. The result we have is following:

- Setting **max_depth** as default -1 can easily lead to an ever rising metric on training set, meaning the model can easily overfit.
- We found dart get easily overfitted, leading to as much as 0.15 gap between training and validation set. Whild gbdt is less vulnerable. This might relate to the fact dart does not allow early stopping.
- We have the best cross validation ndcg@5 of 0.4387, coming from the configuration **n_estimators=1000, boosting='dart', max_depth:-1**. And the final test score according to kaggle is:0.40401.

Machine specs We run most of the experiments on a machine with Xeon Gold 5118 with 12 cores, a GeForce 1080Ti and 60G RAM.

6 Failures and Lessons

First, we have a deeper understanding about how to do the feature engineering. We learned how to use EDA to mine the relationship between features to find the way of improvement, and read the papers about Dimension reduction, for example PCA(we didn't use since after feature merging, the dimension was reduced already), and about feature importance ranking, for instance GBM and XGBoost feature ranking. Second, LeToR is a completely new field to us. We learned the basic rules of LeToR, having access to Pointwise, Pairwise, and Listwise, and knew the root different of them. We learned to use lightgbm and XGBoost as well, which are two critically useful tool for data science.

From a practical side, We put many efforts at first to figure out how integrate both XGBoost, LightGBM and Scikit-learn to make the full of use of all Scikit-Learn utility function. We write some hack class and functions for this use,

but at last still can not manage to perform the cross validation in Scikit-learn way. In turn we wrote ourself a cross-validation class to use. We also met many technical issues that is not well documented, such as we experienced several abrupt breakdown while using XGBoost. At first we suspect it to be a memory issue, so we tried scaling our machine up, adding more ram, but the bug is still there. Then we found out a key parameter which could be set to run the xgb-hist on gpu instead that fixed this issue. We learned from it that a clean design beforehand and abundent experience dealing with data would be the key to success in a data mining competition.

While in feature engineering, two failures we confronted are the follows:

Importance ranking by decision-tree based method Before using XGBoost for importance rank, the first method we choose to is the Extra Trees Classifier provided by sklearn. However, the result does not match our business understanding at all. The reason could be 1. decision-tree-like model requires 0 missing values, so we filled missing values with imputed mean among the whole feature values, which may be not appropriate at all.

Position estimation Before we dropped the "position" feature, We tried to fill in the "position" in the test data. We employed a quick random forest regression/classifier to predict the position values in the training data. However, we got 3% accuracy. We believe the regression is not working because the "position" is discrete numbers. But we are not sure why the random-forest classifier is not working at all.

References

1. Burges, C.J.: From ranknet to lambdarank to lambdamart: An overview. Learning **11**(23-581), 81 (2010)
2. Cao, Y., Xu, J., Liu, T.Y., Li, H., Huang, Y., Hon, H.W.: Adapting ranking svm to document retrieval. In: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 186–193 (2006)
3. Collins, A., Tkaczyk, D., Aizawa, A., Beel, J.: Position bias in recommender systems for digital libraries. In: International Conference on Information. pp. 335–344. Springer (2018)
4. Li, P., Wu, Q., Burges, C.J.: Mcrank: Learning to rank using multiple classification and gradient boosting. In: Advances in neural information processing systems. pp. 897–904 (2008)
5. 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)
6. Rashmi, K.V., Gilad-Bachrach, R.: Dart: Dropouts meet multiple additive regression trees. (2015)
7. Song, Y., Wang, H., He, X.: Adapting deep ranknet for personalized search. In: Proceedings of the 7th ACM international conference on Web search and data mining. pp. 83–92 (2014)

8. Taylor, M., Guiver, J., Robertson, S., Minka, T.: Softrank: optimizing non-smooth rank metrics. In: Proceedings of the 2008 International Conference on Web Search and Data Mining. pp. 77–86 (2008)
9. Xu, J., Li, H.: Adarank: a boosting algorithm for information retrieval. In: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 391–398 (2007)