Ranking personalized hotel search using machine learning

Gamma Andhika Octafarras$^{1[gammaandhika@gmail.com]}$

Vrije Universteit Amsterdam Univeristy of Amsterdam

1 Introduction

This report is based on a Kaggle competition inspired from Personalize Expedia Hotel Search - ICDM 2013 [7]. The competition aims to get the best ranking of hotels per search query based on the likelihood of the hotels being clicked or booked. This report gives a description of the task and the data sets, briefly explores the variables of the data sets and the relationship among the variables, describes the approaches taken and methods used during the competition and presents the results.

2 Business Understanding

The Expedia competition was organised in 2013 and the winners presented their approaches [2] in ICDM 2013. The winner Owen Zhang engineered over a 100 new features by creating composite features, averaging features over ids like prop_id, search_id, using rank based features per search and used an ensemble of two Gradient Boost Models. The approach was compute intensive requiring a 26GB machine with training time of models was a day.

The user with the highest score who was disqualified because of not meeting the competition requirements used a linear ensemble of LambdaMart [10], Neural Network, SGD and Gradient Boost validated with NDCG for 10% of the training data. The individual models were trained with different features - like LambdaMart used all numerical features and the mean, median and stdev for each prop_id of the combined train and test dataset, while SGD was trained with only the numerical features.

3 Data Understanding

Expedia Hotel Recommendation data set which contains the information on search query of the users, hotel characteristics, competitive information and for the training set the data also contains whether the user clicked and booked the hotel. In addition to clicking and booking information the features exclusive to training set includes *gross_booking_usd* which represents the value of the transaction and *position* which is the hotel's position on the Expedia website. The train and test data are given in a split form based on time. The training set contains the data from 2013 and 2014 while the test set has 2015's data. The training data has 4958347 rows and 53 features with 129113 unique properties, 2.79% of rows being booked, 4.47% being clicked. 62.3% of clicks resulted in a booking and 100% of bookings were clicked. The test data consists of 4959183 rows and 49 features of 129438 unique properties.

3.1 How some of variables relate to each other and the response variables

click_bool and *booking_bool* were analyzed through plots. Figure 1 depicts how clicking rate changes based on the hotel's price and star rating and 2 represents how clicking rate relates based on the hotel's price and review score. As expected as the star ratings and the review scores of the hotel's increase there is also an increase in the hotel prices. Also among the hotels with the same review scores and star ratings the hotel with the lower price is preferred to the expensive option.

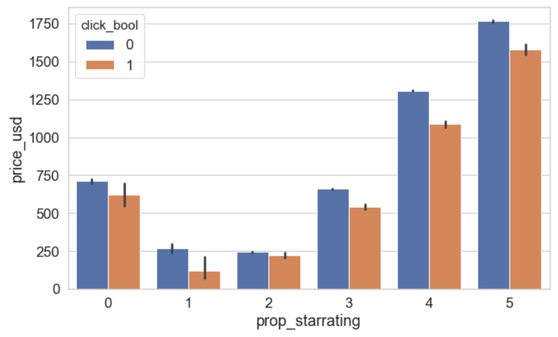


Fig. 1. Clicking Rate based on based on the Hotel price and star rating

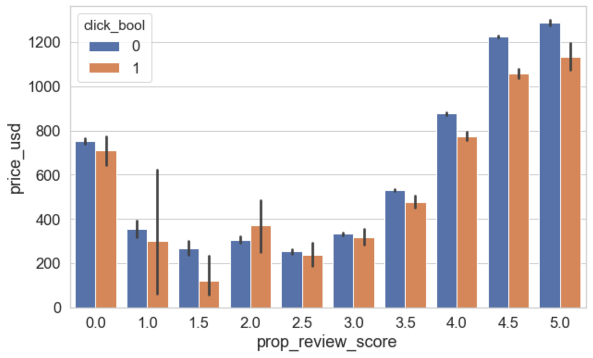
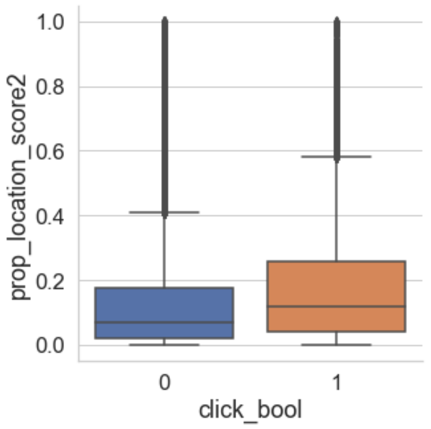


Fig. 2. Clicking Rate based on Hotel price and review score

The feature that has the most significant effect was found to be *prop_location_score2* when building the models. Whether the user clicks the hotel based on the hotel's location score2 is shown in Figure 3. As the figure also suggests the tendency of the user's to click on the hotel increases by a nonignorable amount as the *prop_location_score2* increases.



!htb]]

Fig. 3. Clicking rate based on *prop_location_score2*

While analyzing the variables, significant outliers were observed in the variable *price_usd* which is shown in Figure 4. These outliers needed to be treated in order to not have biased predictions since *price_usd* was found to be an important variable in explaining the model. The treatment of the outliers will be mentioned in Section 3.

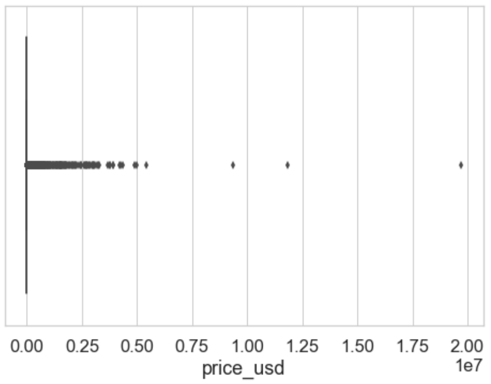


Fig. 4. Price of the Hotel

4 Data Preparation

Since the original data had many columns with missing values and outliers, the data had to be modified before fitting into a model in order to improve the prediction accuracy of the model.

The Top 5 Most Important features were identified using the Random Forest algorithm as it is shown in Figure 5. The features that are the most significant for our model is important to identify in order to pay special attention to those when filling the NaN values (not dropping that column but finding ways to fill the NaN's) and further making use of those features to create new ones.

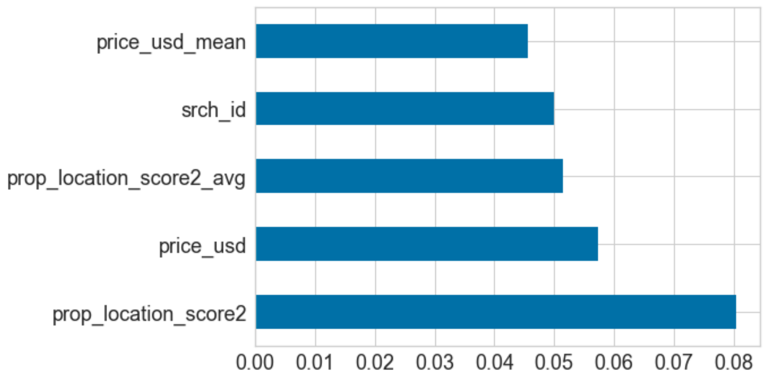


Fig. 5. Top 5 Most Important Features with Random Forest

4.1 Missing Values

Columns containing competitor data, *prop_location_score2*, *visitor_hist_starrating*, *visitor_hist_adr_usd*, *srch_query_affinity_score*, *orig_destination_distance* had NaNs in some columns more than 90% of the data was missing. An initial approach was to drop the columns with high missing values but that led to loss of information and a worse score than keeping the columns. The values were filled with 0 based on the assumption that users are less likely to click/book properties with lower information; as a result filling the missing values with mean or median might have biased the data towards the mean without any new information while putting values with 0 or -1 will penalise these rows for not having data. The values were initially filled with a negative value but filling them with 0s gave a better score.

4.2 Modified Features

price_usd was an important feature with high standard deviation and number of outliers. The values were bounded with a max value (1 std away - 2000.0), and all values above that value were set to the max value. The 3 features - *booking_bool*, *position*, *gross_bookings_usd* were removed as they were not present in the test set, and was not being used for any new features.

Default features and new features which will be discussed which are floats have been rounded to the closest .25, with the intent to reduce small variations.

4.3 New Features

New features were created from making certain changes in the current variables in order to improve the predictive performances of the models. *Date_time* variable was parsed into two new categorical features: *month* and *day* which were expected to capture the seasonality effect in the response variables. The popularity of certain hotels are expected to show an increase during certain months and certain days of the year (summer time, winter breaks, holiday seasons etc.) Using mean values for numerical features grouped over *prop_id* helped getting a better historical data of the property and was useful for smoothing out the variations. *price_rank* feature was to created to rank properties for each search id based on their *price_usd*

Count based features across *prop_id* were used for historical data across properties like

– *prop_click_count* - number of times property was clicked across the entire training set.
– *promotion_count* - number of times property had a promotion offer.
– *prop_count* - number of times property appeared across all queries.

Since *price_usd* was an important feature containing high variability new features were obtained using this variable based on difference between the user historical price data and property's current and historical prices.

– *price_month_mean* - mean price per month.
– *price_usd_mean_diff* - difference of current price and the mean price of the property.

Another variable created was for the difference in the star ratings based on the historical data of the user and the property star rating. The new features obtained from Bing Xu's paper [12] are the following :

– *price_diff_1* = *visitor_hist_adr_usd* - *price_usd*
– *price_diff_2* = $e^{prop\_log\_historical\_price}$ - *price_usd*
– *star_diff* = *visitor_hist_starrating* - *prop_starrating*

A feature to define the average distance (*orig_destination_distance*) to other props in the same *search_id* was also added:

$$srch\_prop\_dist\_avg = \frac{query\_avg * total\_rows - current\_row\_index}{total\_rows - 1}$$

4.4 Data Splitting

Since the data on hand was too large in size (1.2Gb), it took a long time to execute the code. Initially the 10% of the training data was used to train the models for computational efficiency. Later, the initial training data was split into 5 datasets (train_i, i $\in [1, 5]$) linearly out of which train_2 was used as a training set (it had 1000000 rows with 2.79% booked and 4.46% clicked rows with 62.7% clicks resulting in bookings, a similar percentage to the overall training data and could be easily loaded into memory) while train_5 was used for validation as it contained the latest training searches. The test set was also split into 5 parts and loaded into memory linearly to avoid out of memory errors.

A smaller train dataset was also created containing 357765 rows by doing a random under-sampling based *srch_id*, in which a single positive row (having *click_bool* value 1) and another random negative row were taken as the sample for each search query from the training sets (train_i, i $\in [1, 4]$). train_5 was not used as it would bias the local evaluation results would become biased. This dataset was used for tree based models to avoid categorical bias. [9]

5 Modelling and Evaluation

Six machine learning algorithms were used in this project - 3 classifiers, 1 regression model and 1 learning to rank algorithm. When training and evaluating the models, it was observed that no single set of features worked best across all the models. The type of dataset that was used was also a major factor to how well a model performed.

5.1 Target Variables

Since properties had to be sorted based on likelihood of being clicked and booked, an approach was to try it as a multi-classification problem for the variables *booking_bool* and *click_bool* with a weighted average of their predicted probabilities as:

$$target = 1 * Prob(click) + 5 * Prob(book)$$

But this gave worse scores than doing a single classification of target variable *click_bool* (0.17 vs 0.26) .

Another approach was to try two separate random forest models for a single classification: one for *booking_bool* and *click_bool.* Classifying according to a weighted combination of each probability score gave slightly lower results than sorting them as a single classification by *click_bool.*

This was probably because of a higher weight like 5 biasing the results towards the booking probability and booking had 1/3rd lesser rows than clicked leading to a smaller sample size. A booking directly implied a click, so measuring the target variable as *click_bool* for the classification algorithms worked. Since the evaluation metric is NDCG@5, a correct prediction for click would keep the booked row amongst the top5 rows of result set ensuring that the penalty for using *click_bool* was relatively low.

5.2 Internal Evaluation

Since submissions on Kaggle were limited to 2 per day, it was easier to test out NDCG score locally[6]. Train_5 dataset was used for local evaluation with the rows for each *srch_id* sorted on the sum of *click_bool* and *booking_bool* in descending order. The local validation score gave a baseline to locally test out the changes as it was linearly related to the kaggle score. However, the use of different types of datasets and models would have different correlations between the Local and Kaggle NDCG score. The workaround to this was to get a baseline score by submitting to Kaggle for each new model and dataset combination.

5.3 Random Forest

Random Forest was used as it is a popular classification algorithm and the winners had used it as part of their ensembles. For each search query, the rows were sorted on the descending order of the prediction probability. Random Forest was used on the train_2 dataset with features with 90% missing values dropped to get a baseline score which was better than random.csv. With the addition of numerical features averaged over *prop_id*, diff features, day and month improved the score further. Parameters for the model were 'n_estimators': 400, 'min_samples_split': 100, 'min_samples_leaf': 4 Since the data was highly unbalanced, only 4.4% of the rows belonged to clicked category, the balanced dataset gave a much better baseline score which could be further improved with feature engineering and cross validation.

Numerical feature averages like *prop_star_rating_avg*, *prop_location_score2_avg* and count feature *prop_click_count* helped improve the score with train_2 dataset from the baseline. But the averages weren't used with the balanced dataset as the number of datapoints per property id were a lot less.

For Cross Validation, Radom Forest doesn't provide the NDCG function so F1 score and AUC score were tried for parameter tuning during cross validation and the parameters of the best estimator helped improve the kaggle score.

The best score with the balanced dataset was 0.357 on Kaggle while with the unbalanced dataset was 0.334. Parameters used with balanced dataset were 'n_estimators': 800, 'min_samples_split': 50, 'min_samples_leaf': 2

The advantage of using Random Forest was familiarity and ease of use it had been used in an earlier assignment. But the best parameters involved a high value of n_estimators which made the evaluation of the model slow.

5.4 Ensemble Model with Logistic Regression and Random Forest

This method is often known as Voting Classifier with is a useful technique for classification type problems. The main idea of this method is to create a better classifier by aggregating the predictions of the classifiers combined and give predictions for the classifier that has he most votes. The Voting classifier can be a strong learner even when the methods combined are weak learners. This statement from [1] suggests that the ensemble method can achieve higher scores

than the best classifier method in the ensemble.
The classifiers chosen to create the Voting Classifier were Logistic Regression and Random Forest. These models were chosen for the ensemble model because of their computational efficiency. Also since their individual performances were previously tested and the performance of the combination was of interest. Training on the dataset that has 10% of the original training data w,th l-fold cross validation with 5 folds the NDCG score obtained was 0.311 which is higher than both of the classifier's scores individually used on the same training data.(Logistic Regression 0.21 and Random Forest: 0.3) Using the balanced dataset again with hyperparameter tuning via k-fold cross validation with the same number of folds improved the methods score to 0.366.

5.5 Gradient Boosting Classifier

As the gradient boosting was an ensemble method that was often used by the high scorers of the Expedia competition organized in 2013 it was tried out. It is considered to be one of the most powerful techniques when constructing predictive models. The Gradient Boosting Algorithm mainly works with a loss function, a weak learner and finally an additive model to add the weak learners. The weak learners of the gradient boosting algorithm are the decision trees and the additive model is used to add the trees one by one. For this classifier the gradient descent algorithm is used when adding the decision trees meaning the tree that reduces the loss based on the gradient descent is added to the model each time.[3]
The Gradient Boosting Classifier was first used on the 10% of the training data with it's default parameters and without adding the new features which gave a NDCG score of 0.33. Later using the same training set the model's hyperparameters were tuned by k-fold cross validation with 5 folds and grid search over it's hyperparameters. The NDCG score improved to 0.336. The Gradient Boosting Classifier was also used on the balanced dataset by tuning the hyperparamaters of the classifier based on the new training data set. the score obtained with the new dataset improved to 0.3526.

5.6 XGBoost

XGBoost is one of the most frequently used machine learning algorithm for Kaggle competitions. It can do both regression and classification. It is known for it's execution speed and high model performance. XGBoost implements the gradient boosting decision tree algorithm that uses more accurate approximations in order to get the best tree. The advances of this model comes from its mechanism computing the second order gradients and the advanced regularization structure that improves the model performance.

Initially the model was used on 10% of the training data with its default parameters, which received a score of .279. After the addition and modification of features, an score of .3 was achieved which further improved to .358 after tuning the parameters of the model. With the use of the balanced dataset, the model achieved scores above .36. However, a bottleneck became apparent. New features that have improved the result previously on the normal dataset caused the score to lower.

The parameters of the model was tuned by doing cross-validation using RandomizedSearchCV from scikit[1]. With the main focus of tuning the params *learning_rate* and *n_estimators*, with the scoring used being *roc_auc*. The best XGBoost model used the parameters -
'max_depth': 5, 'subsample': 1, 'min_child_weight': 1, 'gamma': 1.25, 'colsample_bytree': 0.8, 'learning_rate': 0.135, 'n_estimators': 140

Though this worked whilst using normal dataset, whilst using on the balanced dataset, the CV would give unfavourable parameters. The approach that was taken for the balanced dataset was to use the best params from the CV of the normal dataset, which was then further optimized by creating a loop to check different params and taking the one with the best local NDCG score. which was - 0.135 *learning_rate* and 140 *n_estimators*.

5.7 LGBM Ranker

LGBM is a tree based algorithm that can perform both classification, regression and ranking. It has certain function created to perform learning-to-rank algorithms. Lambdarank objective function is used when doing the ranking and the NCDG score is used to evaluate the ranking of the model which comes with LGBM Ranker based on [4]. The model itself has the fastest model time among the models that have been used for this project, and is widely known to be very fast in comparison to other GBM models [11].

Unlike the other classification models, the LGBM Ranker worked best without using the a balanced set. As a comparison, using the default features with the same model parameters would give the balanced dataset a score of 0.19, whilst the normal dataset gave a score of 0.369. This is because the ranker is grouped per search id and the number of rows per search id are a lot less(only 2) in the balanced dataset than train_2. The accuracy of the model also scaled with the amount of training data used.

The baseline score the LGBM Ranker with LambdaRank as the objective was better than the other approaches and adding average, diff features helped improve the score further. The top 5 features for the mode is illustrated in fig 6.

[1] https://scikit-learn.org/stable/

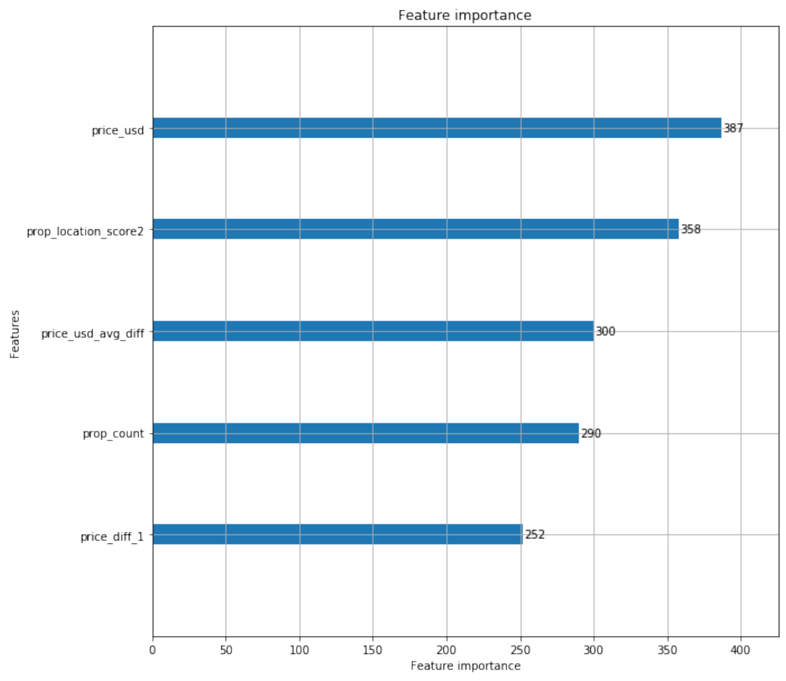


Fig. 6. Top 5 Most Important Features best model

The parameters[5] used for the model were the parameters that the best scoring XGBoost model used, 0.135 *learning_rate* and 140 *n_estimators*, which was found to also have good result on the LGBM Ranker. After a tuning based on the XGBoost parameters, the parameters that was used for the final mode was:

- n_estimators: 150
- learning_rate: 0.13
- colsample_bytree: .8
- objective: ”lambdarank”
- metric: ”ndcg”
- eval_at: 5

As *n_estimators* is the amount of trees that will be fit and the amount is low, the time it takes to train is relatively fast - 1 minute on a laptop with 16 GB of RAM and 6 cores. The LGBM Ranker achieved the best score amongst the models that were used, with a score of 0.39.

6 Unsuccessful Methods/Attempts

Going through the discussions of the Expedia competition, the competition admin had suggested removing position bias[8]. In the models, the position feature was not dropped and position in the training set was set to -1, for it to have no effect for model prediction. Position was an important feature in the model fitting but it reduced the importance of the other features which ended up reducing the score for the models, which is why this approach was discarded.

Rounding the float features improved our accuracy for most of the models, however rounding the feature *price_usd* constantly gave lower scores across the models. Since *prop_location_score1* and *prop_location_score2* was high on the feature importance list across all the models, a new feature - *prop_location_score_ratio* was added. The new feature was high on the feature importance list, but was a disruptive feature in which it did not give a better accuracy and pushed other important features down.

Going through the past reports of people that competed in the previous Expedia competition PCA (Principle Component Analysis) was also a method that was tried by many groups. With 53 features in the data set reducing the dimensionality by using PCA seemed to be a good way to reduce the running time while still achieving good results for the prediction. The features were grouped via this method making 5 columns in total and Random Forest algorithm was run. However this using method showed a significant decrease from the Random Forest with the original 53 columns therefore it was decided to not go with this method eventually. The reason that the PCA might not have delivered expected results could be that it might be leading a loss of valuable information in the variables.

7 Conclusion

Multiple models and variations of the dataset were used in our approach for the Expedia Inclass Kaggle competion. A total of 20 new features were added, and 3 features were removed. Which brought the feature list used in the best model to a size of 70. Several features that worked well with previous models were also not used in the best model. These features on the LGBM Ranker achieved our best score of 0.39. The results obtained from using the different types of data sets (in Section 4.4) before and after adding new features (in Section 4.3) created for different types of models are shown in Table 1

Ranking wise model was better suited for this problem because of the NDCG metric.

7.1 Future Work

A custom NDCG scoring function could have been used when cross validating the models (Random Forest, XGboost) to get the best parameters based on a score that is in line with the one used in the competition. Furthermore, cross validating the LGBM Ranker with the previously mentioned scoring function can be implemented to get a better representation of the possible parameter combinations.

Table 1. NDCG scores from Models used

Model	Data set used	NDCG - default features	NDCG - new features
Random Forest	train_2	0.25	0.332
Random Forest	Balanced	0.31	0.365
Ensemble Model with Logistic Reg. and Random Forest	train_2	0.261	0.336
Ensemble Model with Logistic Reg. and Random Forest	Balanced	0.311	0.365
Gradient Boosting Classifier	Normal (10%)	0.330	0.336
Gradient Boosting Classifier	Balanced	0.348	0.352
XGBoost	Normal (10%)	0.279	0.348
XGBoost	Balanced	0.339	0.367
LGBM Ranker	Normal	-	0.39
LGBM Ranker	Balanced	-	0.19

An ensemble combining the LGBM Ranker model with others may boost accuracy scores, this was not tried as it was found that different sets of features worked with different models. A feature alignment function can be made to overcome this, though results of individual models might decrease.

8 Learnings

By building these algorithms and trying out different data sets and feature engineering we found out that different sets of features work better with different models. Using the balanced data set with decision tree type models gives better performance. Sampling a large dataset can lead to comparable results and should be definitely be done if there are memory constraints. Dropping columns even with 90% of empty data might not be a wise choice as it leads to loss of information. Filling the columns with 0 or -1 values gave better performance than dropping the columns. When running cross validation, the scoring method is a major factor. Using a scoring method that is not reflective of the one used for the official evaluation can result in unsuitable parameters. Ranking wise model is quite sensitive to parameter changes.

References

1. Enhancing the performance measures by voting classifier in ml, `https://medium.com/analytics-vidhya/voting-classifier-in-machine-learning-9534504eba39`
2. Expedia icdm workshop. `https://www.dropbox.com/sh/5kedakjizgrog0y/\_LE\_DFCA7J/ICDM\_2013`
3. A gentle introduction to the gradient boosting algorithm for machine learning, `https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/`
4. Learning-to-rank with lightgbm (code example in python), `https://medium.com/@tacucumides/learning-to-rank-with-lightgbm-code-example-in-python-843bd7b44574`
5. Lightgbm ranker metrics, `https://lightgbm.readthedocs.io/en/latest/Parameters.html#metric-parameters`
6. Ndcg scorer, `https://www.kaggle.com/davidgasquez/ndcg-scorer`
7. Personalize expedia hotel searches - icdm 2013, `https://www.kaggle.com/c/expedia-personalized-sort/data`
8. Position bias, `https://www.kaggle.com/c/expedia-personalized-sort/discussion/5989`
9. Resampling strategies for imbalanced dataset, `https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets`
10. Burges, C.J.: From ranknet to lambdarank to lambdamart: An overview. Tech. Rep. MSR-TR-2010-82 (June 2010), `https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/`
11. 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. In: Advances in neural information processing systems. pp. 3146–3154 (2017)
12. Liu, X., Xu, B., Yuyu, Z., Yan, Q., Pang, L., Li, Q., Sun, H., Wang, B.: Combination of diverse ranking models for personalized expedia hotel searches (11 2013)