Data Mining Techniques - Assignment 2 - Gr185

A.J. Hazenberg[2649192], D. Pop[2618006], and T.J. Siebring[2683240]

Vrije Universiteit Amsterdam, 1081 HV Amsterdam, Netherlands

Abstract. The topic of this paper is in the area of Learning to Rank (LTR). More specifically, it is about predicting the hotel that a visitor is most likely to book. Using machine learning algorithms, companies like Expedia can then order the listings for a query in the most suitable way, with the most relevant on top. Leveraging the LightGBM module, we propose two methods of ranking properties based on queries, namely a pointwise approach with a regression loss and a listwise algorithm. The results that followed show that the first obtained an NDCG@5 of 0.333, while the second one scored 0.366. With the help of these models, Expedia could predict the preferences of their visitors over two times as good as random.

Keywords: Recommender systems · Offline Learn to Rank· LightGBM

1 Introduction

Due to the increasing use of the internet, more and more people order things online. One of the things that people order online nowadays are hotel rooms and apartments. It is very easy to insert your preferences and compare several hotels with just a few clicks. Because there are several different property options one can choose from, it can take hours before one finds the right hotel room to stay in. It is therefore important that sites where one can compare hotel rooms, apartments and boutiques, only show the visitor recommendations that fit his or her preferences and interests. Many travellers will make multiple visits to the website before making an actual booking. The repeating customers make up a significant part of the traffic to the website. The recently viewed properties and destinations are used such that the search time for repeating visitors is reduced by recommending properties that similar to the ones they have visited before. For new visitors, it is harder to determine recommendations since there is no historical data available [15]. Expedia, the company which made available the data for the project, is one of the many sites where visitors can compare hotels in countless cities around the world. They take a part of the hotels profit when someone books a hotel via their website. This is one of the main reasons why they are eager to show only the best customized recommendations for the visitor. Just as important, they also want to make the experience as pleasant as possible for the visitor. By participating in the Kaggle challenge '2nd Assignment DMT', the goal is to find a machine learning method that is capable of predicting suitable hotel recommendations for Expedia visitors looking for a property.

Content of the paper. The rest of the paper is organized as follows. Section 2 elaborates on the previous research and work that has already been done regarding this problem. After stating the previous work, exploratory data analysis is done on the Expedia data set in Section 3. Section 4 describes the different transformations applied to the dataset, and the information about the developed machine learning models and their performance can be found in Section 5. Lastly, Section 6 concludes the paper and provides recommendations for future work.

2 BUSINESS UNDERSTANDING

As the competition was held in 2013, many resources are available describing techniques used to perform predictions on this data set. Examples of these are [11], [16], and [7]. For this section however, we focus on analyzing the top performers of the competition, in particular the winner, Owen Zhang. He gave a short presentation on his solution which is used as the basis for this research [1]. In the following paragraph we elaborate on its content.

An important technique used is extensive feature engineering. Although other teams do this as well, the extent to which this is done matters. Besides all original features, extra features such as numerical features averaged over e.g. srch_id and prop_id, composite features such as price order were created. Furthermore, they transformed categorical features such that the overall influence of categorical values on the target variable was taken into account.

Outside the previously mentioned techniques, more subjective but not less important step is balancing of the data - down sampling negative instances to speed up learning and performance. Different approaches were used to handle missing values, each having its own rationale.

When it comes to modeling, an ensemble of gradient boosting machines (GBMs) is taken using the NDCG loss function. Two models are considered, one with all engineered features and one with part of it. In total, 26 separate modes were trained, each taking 20-30 hours on machine with 256GB RAM and 200GB swap space. Next, a weighted average is taken as the final result. Most participants try out many different techniques, whereas Owen has put significant effort and training time into one particular technique, GBMs, and optimized for this. The decision for GBMs makes sense as they have proven to be effective for ranking tasks. Aside from good preprocessing and feature engineering, we believe the focus on one particular suitable technique in combination with impressive hardware and training time was decisive to win.

Besides Owen, we also analyzed the approaches of other top performing teams, as they also gave presentations on their approaches. Prominent predictors that were specifically mentioned by them: *price_usd*, *prop_starrating*, *prop_location score2*, *estimates_position*. Note that derived values, such as differences between them, are also implied. Besides this, composite features were created by combining multiple original features, such as *price_order*, indicating the order of *price* for a particular search. However, the effectiveness of these

is unclear. Popular models that were used in other approaches include: Gradient Boosting Machines, LambdaMART, (Pairwise) Logistic Regression, Random Forest and Deep Learning. The first two models, including ensembles thereof, yield the highest performance.

3 DATA UNDERSTANDING

To get a better feeling of the Expedia data set, exploratory data analysis was done on the data. The training set consists of 4958347 rows each corresponding to a suggested property for a visitor's query. For each instance, 54 information features are saved, among which the time of the query, the location of the property, the price of the property, the location of the visitor and the star rating of the property. The test set consists of 4959183 rows, also each corresponding to a suggested property for a visitor's query.

Features unavailable in the test set. In the test set, four features have been left out because of connections to the feature that is predicted, namely the id of the hotel that was suggested to the visitor. The first feature that is missing from the test set is the feature *position*, because it has a correlation with the probability of clicking. The higher the property is listed, the higher the chance of clicking on it. The second and third feature that are not in the test set are the features *click_bool* and *booking_bool*, because they show if the visitor clicked or booked the property. The last feature that is not in the test set is the feature *gross_booking_usd*, which is the total value of the transaction. This can be linked to the suggested properties, and thus it is left out of the test set.

Missing values. Variables that have a lot of missing values are not trustworthy to use because they are only stored for a small number of rows (visitors). In the Expedia data set, there are a large number of missing values. The features *visitor_hist_starrating* and *visitor_hist_adr_usd* miss almost 95% of their data. It could be that the visitor has never booked a hotel via Expedia before, and therefore there is no historical value (starrating/ hotel price) to compare with. The feature *srch_query_affinity_score*, indicating the log of the probability that a hotel will be clicked on in internet searches, is missing almost 94% of its data. This indicates that a lot of hotels do not register in internet searches. It can be noticed that the features corresponding to competitor scores also have a lot of missing values. This suggests that there is no competitive data available to compare to. Overall, it seems that 22 out of the 54 columns in total do not have any missing values. It can also be noticed that the train and the test set have missing values in the same columns.

Distributions and checks. For several features, histograms were plotted in order to observe their distributions. Besides checking their distribution, it is also checked if the features do not contain any values that are out of the ordinary, according to the table in the provided documentation. Figure 1 shows the distribution for the variable *visitor_hist_starrating*. This feature displays the mean

star rating of hotels that the visitor has previously purchased. Almost 95% of the data in this column was missing, because visitors did not previously book a hotel via Expedia. The distribution of the 5% that is present in the data, is visible in Figure 1 below. It can be seen that the distribution is slightly left-skewed, where most visitors booked hotels with at least 3.5 stars.

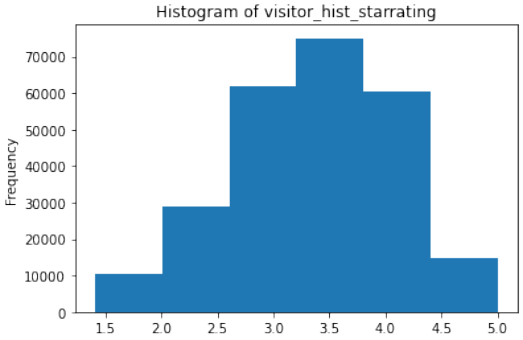

Fig. 1: Distribution of the historical visitor star rating

Secondly, a histogram was made of the feature *prop_starrating*. This feature shows the star rating of the hotel, ranging from 1 to 5 with increments of 1. A 0 indicates that the property has no stars, the star rating is not known or it cannot be publicized. The results can be found in Figure 2. This distribution is also left-skewed, where most properties listed on Expedia have at least 3 stars.

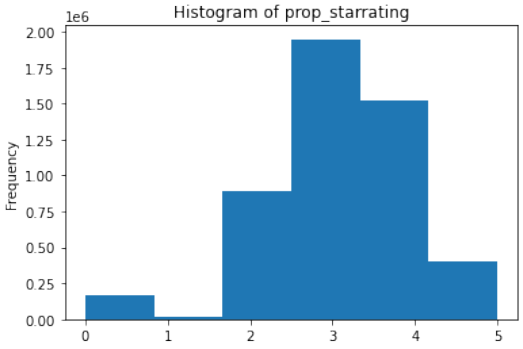

Fig. 2: Distribution of the property star rating

The last feature that is investigated is the feature *prop_review_score*. This feature displays the mean customer review score for the hotel on a scale out of 5, rounded

to 0.5 increments. A 0 means there are no reviews. Figure 3 shows that the distribution is again left-skewed and one can see that most properties score at least a 3.5 or higher out of 5.

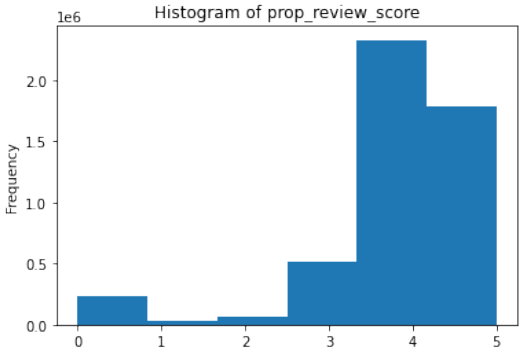

Fig. 3: Distribution of the property review rating

To make sure that the Expedia data can be used in the modeling part and that no outstanding numbers are present, small, common sense checks are done. For every record is checked if the number of adults inserted, the length of stay, the price of the property, and the distance between this and the visitor are always positive. Negative values would indicate wrongly inserted data, which could cause trouble when using the data for modeling, but would also not make sense from a conceptual point of view. Investigations show that all of those features have strictly positive values and that the numeric features can thus be used for modeling.

4 DATA PREPARATION

In order to prepare the data for modeling, several transformations were applied as it follows in the next paragraphs. Essentially, we derived additional variables and filled in missing values to facilitate the learning process of the algorithm, we removed the features we deemed problematic, and we split the data in train and validation sets.

Filling in missing values. Prior to inputting missing values, the features which are relevant when reserving a property, but have a large proportion of unavailable entries (*visitor_hist_starrating, prop_starrating, prop_ review_score* and *prop_log_historical_price*) were inspected. For these, a new feature was derived for each whenever the respective variable has an "null" value. This was done to indicate that the missing values have a specific meaning, which is most likely

that the visitor/property is new on the website. Allegedly, these could indicate the model certain patterns which occur only for new visitors/properties.

Then, the missing entries from the variables *visitor_hist_starrating, visitor_hist_adr_usd, prop_review_score*, and *prop_location_score2* were filled in with the average of the column because they were deemed relevant in securing a booking according to both our intuition and the results of the competition winning strategies studied. The mean was preferred over other aggregates because there are no outliers in any of the variables which could have a significant impact.

When it comes to the features concerning the competitors of each property, namely *compX_inv* and *compX_rate*, we decided to fill in the missing entries with "1". The reason behind this decision is that if data about competitors is missing it most likely means that there are no competing properties, in which case the respective estate should have an advantage when it comes to booking, as it is literally the leader in the respective field. Furthermore, if there are no competitors, there are no available competing places, while retrieving the location for the search implies that there are available places, hence the value "1" for *compX_inv*.

Deriving additional features. Initially, the *date_time* is converted to a Unix format. Unix time is a system of representing the date and time with the number of seconds that have passed since the the first of January 1970, 00:00:00 UTC [17]. The reasoning behind this transformation is that the previous format cannot be used by a machine learning algorithm. Additionally, three other features were derived from the date, namely the day of the week, the month, and whether the day fell on a weekend, because we considered the potential weekly and yearly seasonality in the likelihood of booking a property. For instance, a hotel close to the beach is more likely to be booked around the summer months, in which case the model would benefit from knowing the month when the search takes place.

Then, the feature engineering part was continued at multivariate level. The first extra variable to add was an overall location desirability by adding up the *prop_location_score1* and *prop_location_score2*. Secondly, we subtracted the historical price of a property from the current price to get the difference, which should indicate whether the property is being rented at a discount or at a premium. Theoretically, a property at discount would have more chances to be booked. Next, another feature summing up the total number of individuals searched was computed summing up *srch_adults_count* and *srch_children_count*, as well as a feature dividing this number by the number of rooms. Finally, the price per person was calculated and the destination id where the search was performed was compared with the id of the visitor's location to see whether the search was done abroad.

Dropping columns. As some of the features have a large number of values missing and no feasible inputting strategy was figured, they were eliminated altogether. These variables are: *srch_query_affinity_score, gross_bookings_usd, orig_ destination_distance*, as well as the percentage difference of the property and all its competitors.

Preparing for a learning tasks. As most of the machine learning algorithms cannot deal with categorical features, it was decided to use the one-hot-encoding technique on these and create a new binary column for each category in all the categorical variables. Additionally, 10% of the data was held out for validation purposes, while the rest was used for training. This was done, instead of using the submissions on Kaggle as test set, to avoid overfitting the data on it. Since it is important that the search ids groups are never divided, a grouped shuffle split [6] was used instead of the basic shuffle split. Essentially, this method considers a group as an un-divisible instance and it ensures the train and test sets contain mutually exclusive groups.

5 MODELING AND EVALUATION

Moving on to the modeling part, we decided to leverage the LightGBM module [13] for a variety of reasons. First of all, gradient boosting models performed the best according to the winners of the competition. Secondly, among the gradient boosting algorithms available, LightGBM offers the best trade-off between speed of training and computational performance, which is especially important given that the train data consists of almost five million records. Furthermore, the library offers a selection of estimators, including a regressor and a ranker, which would make possible the implementation of both a pointwise and listwise Learn to Rank approaches. Finally, we also have experience using this framework, making it easier for us to develop a suitable model for this assignment.

LightGBM is fast and accurate thanks to leaf-wise tree growth, which allows the tree to grow by leaf in the direction that yields the largest decrease in loss, regardless of the level of the leaf as opposed to the other boosted tree algorithms. In other words, the level-wise growth constrains the tree growth to take place level-by-level, as opposed to leaf-wise growth that only considers the best-first, that is the leaf with maximum delta loss [12]. Another feature specific to the model is the Gradient-Based One-Side Sampling (GOSS) that suggests a sampling method based on the gradient to avoid searching the entire space. Instead, the method selects the instances with large gradients and randomly samples among the ones with small gradients. Accordingly, the information gain is enhanced as it was proven by Ke et al. (2017) [8] that the larger gradients play a more important role in the computation of the information gain. Lastly, there is Exclusive Feature Bundling (EFB) which is a functionality that is best applied when the number of dimensions is large and the features itself are sparse and, more importantly, mutually exclusive (e.g. One-Hot Encoded data). EFB merges together the sparse variables, reducing dimensionality, which leads to better speed and computational performance of the model.

Learn to Rank (LTR). Learing to Rank (LTR) is a process defined by automatically constructing a ranking of elements by their relevance after learning from training data [9]. Essentially, given a user's query and a collection of documents, a ranking system provides to the user the ranking of documents according to

their relevance on the query. This relevance is computed through a machine learning algorithm that is trained on a set of query and document features and their historical relevance, outputting a score for each document-query combination so that relevant documents are placed higher. In our case, the documents are the property listings on Expedia. Given that the learning happens on annotated data, the task at hand is classified as offline or supervised LTR.

LTR approaches. There are three LTR approaches, namely pointwise, pairwise, and listwise. As the name suggests, the pointwise method considers the lines of the data individually, predicting a relevance score for each. In other words, a pointwise model would simply be trained on a number of records, predicting an individual score for each record, ignoring the interaction with other rows. Although this method is simple and easy to scale, it does not optimize directly the ranking quality because it does not take into account the ordering of the items. Therefore, its performance compared to the others is lower.

The pairwise method is based on document pairs and its loss function tries to minimize the number of correct inversions in ranking [5]. There are several versions of the loss function for this apporach, however the most popular is RankNet [14]. The problem with this technique is that it assigns every pair an equal importance, whereas, in reality, this is not the case because the order in the top documents is much more important than the order of the documents in the second part of the ranking.

The listwise approach addresses the fundamental issue of the other two, namely optimizing the ranking quality directly. It does this by considering the impact of documents pairs as gradient in an algorithm called LambdaRank [4]. More precisely, these gradiends are multiplied with the change in loss by swapping the rank positions of the two documents.

Experiments and results. In this project, we decided to start with a pointwise model having a regression loss, due to its simplicity from an implementation standpoint, and then program a listwise algorithm to observe the difference in performance. One would expect the second to return better results as it is a method introduced specifically to solve the fundamental problem of the pointwise approach, namely disregarding the ordering of the instances.

Accordingly, the first model designed was a LGBMRegressor. When it comes to its implementation, we experimented with different target variables, two encoding techniques for categorical features, and multiple combinations of hyperparameters. Initially, we used the *position* as the target variable of the model because we thought this would be the most important factor in deciding whether a booking is made, more significant than the clicks and bookings done by other users. Then, we tried the relevance score, defined as $5 * booking\_bool + click\_bool$, as used by the third place in the competition [10]. Finally, we used the feature *click_bool* as inspired from this approach [3].

In the following table, one can see the NDCG@5 for the three options on the held-out test data, after fitting the default LGBMRegressor on the other 90% of the train dataset. Surprisingly, the click variable performed the best. For this

reason, we decided to continue using the *click_bool* as the target variable for the rest of the modeling.

Target variable	NDCG@5
Position	0.279
Relevance score	0.325
Click	0.335

Table 1: NDCG@5 for different target variables

Our next experiment addressed the encoding of the categorical features. As the number of features resulted after one-hot-encoding is very large, we decoded back the multi-categorical features (i.e. the variables with more than two categories) and used the built-in functionality of the LightGBM module, called cat_features. The next experiment involved doing the same steps, this time for all the categorical variables, including the binary ones. As shown in table 2, when one-hot-encoding is kept on the binary features, the NDCG@5 on the validation set is larger, hence, the data was used using this encoding in the following tests.

cat_features use on	NDCG@5
All variables	0.339
Only multi-categorical variables	0.344

Table 2: NDCG@5 for different categorical features encoding techniques

The last step we took in optimizing the score of the pointwise algorithm was to tune its hyperparameters. Similarly with the train-test split, the train-validation splits done for a 4-fold Cross-Validation method was done with respect to the *srch_id* groups. Then, a random search procedure was implemented to try out multiple combinations of parameters and aggregate their performance of the validation sets. Consequently, the best combination of hyperparaters found was: $boosting\_type = \text{"}dart\text{"}, learning\_rate = 0.1, max\_bin = 300, min\_child\_samples = 23, n\_estimators = 120, num\_leaves = 34, sub-sample = 1$.

The score of this model on the held-out dataset increased from 0.344, when the default parameters were used, to 0.350. Unexpectedly, the enhancement is minimal. Moreover, after fitting the entire train data to this model (i.e. both train and validation sets) and evaluated on the Kaggle test set, the model only scored 0.333. This could be justified by two arguments. First, the test data consists of some instances whose pattern was not observable in the train data. Secondly, there is a small overfit on the held-out test set selected from the train data.

Ultimately, we designed a listwise approach in the form of a LGBMRanker with LambdaRank objective. For this model, we integrated two versions, one with the default parameters, and one with the hyperparameters optimized after a random search method with 4-fold Cross-Validation. The results of the two experiments can be found in table 3. One more time, the impact of tuning the parameters is small.

Model parameters	NDCG@5
Default	0.372
Optimized	0.375

Table 3: NDCG@5 for different categorical features encoding techniques

Final model evaluation. As the model with tuned parameters is better on the validation data, this is used to make the final submission on Kaggle, but not before fitting it another time using 100% of the train data, including the held-out set, which gives the model an increased chance of learning the patterns in the data. The resulted submission generated an NDCG@5 of 0.366, suggesting a slight overfit on the train data as the score is larger than the one achieved during the tryout. The score of 0.366 can be interpreted as over two times as good as the random baseline ranking which has an NDCG@5 of 0.156. This means that, using the developed model to rank listings for a given query, Expedia would satisfy their customers twice as much by placing relevant properties higher in the page in comparison with posting the results randomly.

When it comes to its hyperparameters, these are: $objective = "lambdarank"$, $metric = "ndcg"$, $boosting\_type = "gbdt"$, $learning\_rate = 0.1$, $max\_bin = 255$, $min\_child\_samples = 25$, $n\_estimators = 200$, $num\_leaves = 33$, $subsample = 1$.

Looking at Figure 4, one can see the feature importance of the final model. As shown, the most important variable is by far the property id, implying that, most of the times, if specific properties are retrieved by the query, they are generally more likely to be booked. The second most important feature is the destination where the search takes place, which is surprising. Finally, the desirability of the location, the price difference between the current and the previous prices, and the star rating play an important role in helping the model rank the listings.

6 CONCLUSION AND DISCUSSION

The goal of this paper was to set up a machine learning model capable of ranking suitable property recommendations for Expedia visitors. Suggesting suitable properties results in more satisfied customers and eventually higher profits, since the recommender website gets a fee for every property that was booked through

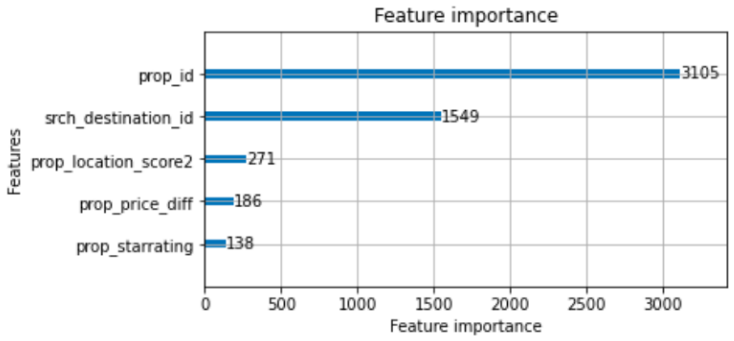

Fig. 4: Feature importance of the final model

their website. After investigating the data and checking for strange values, the data was prepared such that it could be used for modelling. Missing values were filled in were possible, extra features were derived and columns that did contain a lot of missing values, but no strategy to fill them in, were removed from the data set. Lastly, the categorical features were one-hot encoded and the training set was split up into a training and validation set to avoid overfitting. The data is split such that there is no 'look-ahead-bias'. After the data was prepared such that it could be used for modelling, two LightGBM algorithms were investigated. LightGBM appeared to be a suitable module since the current dataset is quite large and LightGBM has a high training speed and high efficiency. The LGBMRanker estimator obtained the highest NDCG@5 of 0.366. Using this knowledge and the corresponding predictions, Expedia can now give better-than-random recommendations to its visitors.

Lessons learned. The current project, regarding recommender websites suggesting interesting properties to Expedia visitors, is not something that we have done in any other project before. Some team members had more knowledge on machine learning methods already than others, which made it interesting to see what result we could achieve. The ranker estimator used in the project was a new algorithm to all of us, which made it difficult and interesting to work with. We started using a point-wise ranking model, after which the list-wise model was implemented. The listwise model gave a better result, which was expected because of the fundamental issue of pointwise algorithms, namely not taking into account the ordering of the records. One difficulty that we encountered was the hyperparameter tuning. The combination of the NDCG@5 evaluation metric and the grouped splitting for Cross-Validation made it very difficult for us to implement the tuning automatically with the help of sklearn's RandomizedSearchCV. Instead, we had to split the data using a group shuffle split and then manually fit the train data, score the validation set, and average all the scores for all the iterations, for different combinations of hyperparameters.

Future work. To improve the performance of the current LightGBM model, more feature engineering can be done. One idea can be taken from the winning solution, which included averaging the numerical features by *prop_id* and *destination_id*. Then, other imputation techniques could pass more information to the model. For example, one idea is that the missing values from property review and star columns could be filled in by designing a KNN model that considers the price and location of other properties. Additionally, it is well known that click annotations suffer from position bias. In other words, one is more likely to click and book a property if shown at the top of the page. In this case, the *position* variable is very important. Therefore, another solution with potential would be to predict the position for the test data and use that as a predictor feature when ranking the likelihood of booking.

Besides this, another improvement is likely if a more extensive hyperparameter tuning is performed. Theoretically, with more computational resources, significantly more combinations of parameters can be tried, which is likely to ultimately increase the NDCG on the test data.

Lastly, in this project, only LightGBM models have been investigated, nevertheless, implementing other models such as CatBoostRanker [18] and LambdaMart from RankLib [2], could be beneficial. From out experience with classifiers and regressors, CatBoost generally performs better than LightGBM. Hence, we expect the ranking estimator to also be better. However, CatBoost is also much slower when it comes to training, which would be an impediment given the large size of the data. Concerning RankLib, this tool was used by the winners of the competition, thus the reason to try it out and compare its performance with the algorithms developed in the since 2013.

References

1. Personalized expedia hotel searches – 1st place. ICDM 2013 – Owen Zhang
2. Ranklib library. https://github.com/jobandtalent/RankLib, accessed: 26-05-2022
3. Ambuj Agarwal, Lanqiu Mei, Y.G.Y.L.: Slides personalize expedia hotel searches. https://www.slideshare.net/lethalamby/personalize-expedia-hotel-searches-2, accessed: 26-05-2022
4. Burges, C., Ragno, R., Le, Q.: Learning to rank with nonsmooth cost functions. In: Schölkopf, B., Platt, J., Hoffman, T. (eds.) Advances in Neural Information Processing Systems. vol. 19. MIT Press (2006), https://proceedings.neurips.cc/paper/2006/file/af44c4c56f385c43f2529f9b1b018f6a-Paper.pdf
5. Chen, W., Liu, T.Y., Lan, Y., Ma, Z., Li, H.: Ranking measures and loss functions in learning to rank. p. 315–323. NIPS'09, Curran Associates Inc., Red Hook, NY, USA (2009)
6. scikit-learn documentation: Group shuffle split. https://scikit-learn.org/stable/modules/cross_validation.html#group-shuffle-split, accessed: 26-05-2022
7. Gourav G. Shenoy, Mangirish A. Wagle, A.S.: Kaggle competition: Expedia hotel recommendations, https://arxiv.org/pdf/1703.02915.pdf
8. 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: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017), https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf
9. Liu, T.Y.: Learning to rank for information retrieval. Found. Trends Inf. Retr. **3**(3), 225–331 (mar 2009). https://doi.org/10.1561/1500000016, https://doi.org/10.1561/1500000016
10. 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 (2013). https://doi.org/10.48550/ARXIV.1311.7679, https://arxiv.org/abs/1311.7679
11. Michael Arruza, John Pericich, M.S.: The automated travel agent: Hotel recommendations using machine learning (2016), http://cs229.stanford.edu/proj2016spr/report/017.pdf
12. Microsoft: Leaf-wise (best-first) tree growth. https://lightgbm.readthedocs.io/en/latest/Features.html?highlight=leaf%20growth#leaf-wise-best-first-tree-growth, accessed: 26-05-2022
13. Microsoft: Lightgbm documentation. https://lightgbm.readthedocs.io/en/latest/index.html, accessed: 26-05-2022
14. Microsoft: Ranknet: A ranking retrospective. https://www.microsoft.com/en-us/research/blog/ranknet-a-ranking-retrospective/, accessed: 26-05-2022
15. Mitsoulis-Ntompos, P., Varelas, D., Brady, T., Landry, J., Dickerson, R.F., Renner, T., Harris, C., Ye, S., Amirabadi, A., Jones, L., Cardo, J.L.: Landing page personalization at expedia group (2020)
16. Ramzan, B., Bajwa, I.S., Jamil, N., Amin, R.U., Ramzan, S., Mirza, F., Sarwar, N.: An intelligent data analysis for recommendation systems using machine learning. Scientific Programming **2019** (2019)

17. Wikipedia: Unix time. https://en.wikipedia.org/wiki/Unix_time, accessed: 26-05-2022
18. Yandex: Catboost ranking. https://catboost.ai/en/docs/concepts/loss-functions-ranking, accessed: 26-05-2022