Data Mining Techniques - Assignment 2

Scientific Report: Expedia booking predictor

Group 190 - Caroline Hallmann, 2640914, Aneesh Makala 2730226, Nader Sobhi 2702248

Vrije Universiteit Amsterdam

1 Introduction

Expedia works with large databases generated through user engagement. They apply data mining techniques on large datasets to produce user specific content and accurate information. Expedia uses a Recommender system to produce a recommendation list of most relevant and personalised results to users. When users are provided with the correct recommendations and order of results, Expedia believes that it has a positive affect on booking behaviour. This report will evaluate different ranking models to predict what hotel a user is most likely to book by participating in the DMT2022 Kaggle competition [10]. The dataset originated from the 2013 Expedia competition[16] and contained roughly 5 million records. Information of a user's hotel search queries, resulted hotel properties and whether a user clicked and booked a hotel[11] was given. Relevance scores were used to calculate Normalized Discounted Cumulative Gain (NDCG)@5 of the models. The paper first discusses 2013 Expedia Kaggle winning solutions. Part 3 describes and explores the data through descriptive statistics. Part 4 looks at feature engineering and part 5, the model implementation and evaluation.

2 Business Understanding

To predict which hotel a user is most likely to book, one needs to investigate the user importance of hotel properties. To generate future predictions about user's hotel booking behaviour, previous data mining approaches from the 2013 Expedia Kaggle challenge were investigated. The top two winners were selected and the best predictions were evaluated. For data pre-processing, Owen Zhang (1st place), conducted missing value imputation by assigning negative value. Jun Wang (2nd place) replaced missing values with the value 0[13]. Furthermore, both used the normalisation of numeric values by search_id approach. Additionally, Zhang averaged the numerical features prob_id and destination_id. Wang normalised a larger set of features including search_id, prop_id, month, srch_booking_window, srch_destination_id, and prop_country_id.

The winning approaches included composition of features. Zhang analysed the difference between hotel price and recent price (price_diff_from_recent). He also evaluated the order of the price within the same srch_id (price_order). Wang, on the other hand, was interested in the user's history data and hotel quality.

He focused on the difference between past rankings and hotel ranking, and past price and hotel price. Hotel quality was estimated by the overall probability of being clicked or booked. Zhang implemented the GBM method. He added the EXP feature of position based on prop_id/dest_id/target_month. For the GBM implementation, he used two types of models: with EXP features and without EXP features. After running 26 GBM models with feature engineering, the most relevant feature was found to be estimated_position. The predictors which received 1st place in the competition were position, price, and location desirability. The models implemented by Wang were LambdaMART, Regression, SVM-Rank. LambdaMART was chosen as it is nonlinear and computationally efficient. A possible explanation to why this method ranked 2nd place is that it is score feature based. Some features such as time and locations do not work with LambdaMART.

3 Data Understanding

3.1 Data Analysis

For the competition, two datasets (training_set_VU_DMT, test_set_VU_DMT) and one submission set (submission_sample) to submit as a solution was used. The training set accounted for 4958347 entries and the test set of 4959183 entries. The training data included 54 columns with information about search characteristics, search related hotels, user platform interaction and the interrelationship between the Expedia search platform and its competitors. The test set included similar data of this nature with the exception of the position, click_bool, booking_bool and gross_booking feature.

To include all features and make accurate predictions, the training set will be used for the explorative data analysis (EDA). The data was imported in several data mining software, including RapidMiner[15], dataiku[8] and Knime[12]. The training data had a total of 7 nominal, 16 numeric, 28 ordinal and 1 time feature. The data was collected over a eight month time period (date_time), producing approximately 650000 data scores per month.

Missing values The data was first checked for missing values. Several features contained missing values. Majority of NULL values were derived from the search (srch_) and the competition (comp_) features (Fig.1).

The competition features had the highest percentage of missing values from all features in the training set. Next, the skeweness of the numeric data was evaluated and scores greater than 0.75 were changed with the log function to ensure normal distribution.

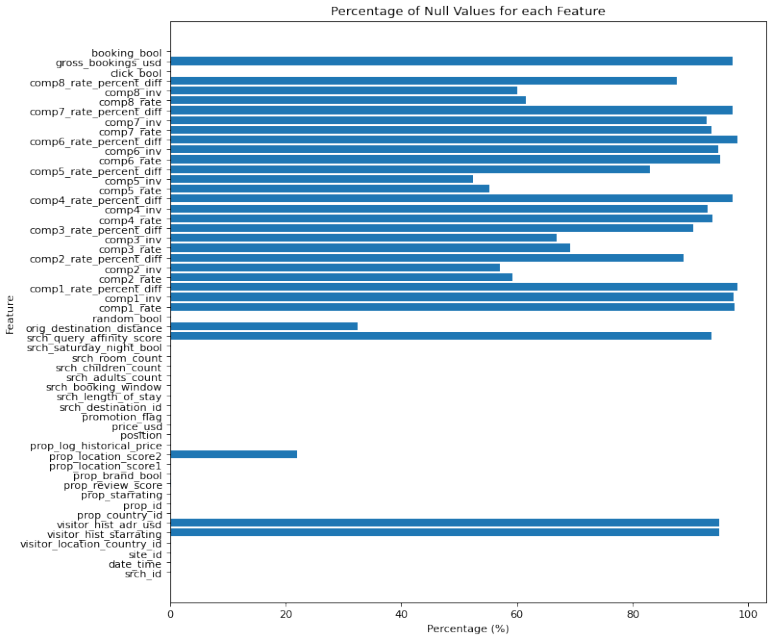

Fig. 1. Percentage of null values per feature

3.2 Correlation Analysis

To gain better understanding of users clicking and booking behaviour, it is important to analyze the relationships of the given variables. A Pearson Correlation Coefficient (PCC) matrix was produced on all 52 features. However, the matrix was later adjusted and grouped into feature types. The main types of features were 1)competition (comp_), 2) search (srch_) and 3) hotel (prop_) features. Due to the high missing values in competition features, a separate correlation matrix of search and hotel features was created (Fig.2).

The highest correlation was between click_bool and booking_bool with value 0.782. This result shows that users need to click on the property in order to book it. Without clicking, the user can not book.

For the search features, the highest value (0.473) was between srch_room_count and srch_adult_count. A positive correlation was also seen for booking window and length of stay with value 0.267.

Amongst the hotel features, prop_location_score1 and prop_location_score2 had the highest correlation (0.433). There is a positive correlation they share the same attribute and have similar distribution. However, some differences were seen between the two when comparing to prop_starrating. Prop_location_score1 had a significantly higher positive correlation of 0.289 compared to prop_location _score2 with a 0.084 PCC. The difference might be due to the missing values for score2. Another interesting PCC was of prop_review_score and prop_starrating

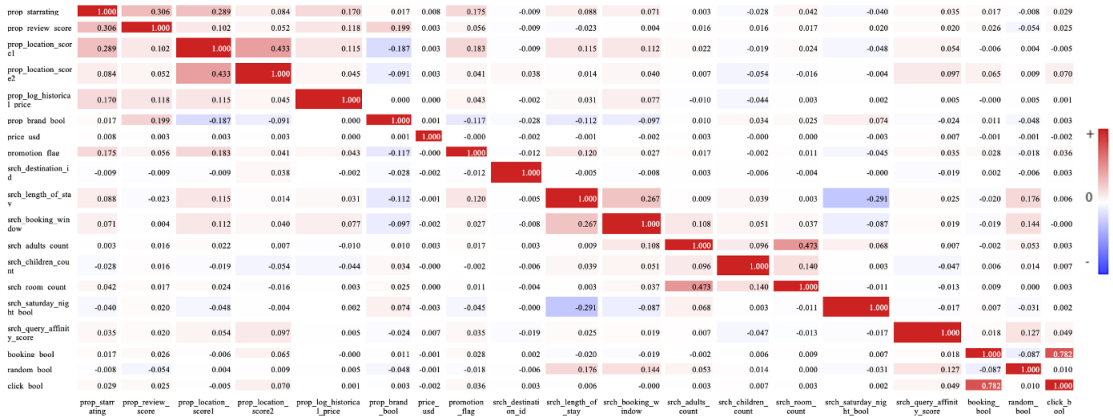

Fig. 2. Pearson correlation matrix of search and hotel features

with value 0.306, showing that star rating and review scores are sometimes given together. The higher the review score, the higher the star rating.

3.3 Data statistics

To further investigate whether a user is likely to book a hotel, several features were chosen and evaluated with booking_bool. Statistical values were calculated in the tables below.

properties	avg_price	avg_booking_price	avg_user_hist_price	booking_bool	click_bool
4736468	252.44	NaN	175.933587	0	0
83489	343.95	NaN	184.963617	0	1
138390	260.51	386.283316	176.356201	1	1

When evaluating the price to bookings, several differences were detected. When users do not click and book, the average price stayed 252.44 and the the hist_price 175.93. When users clicked but not booked, the price increased to 343.95 and had a hist_price of 184.96. When users book and click, the average price went to 260.51 with a hist_price of 176.36. When users did not book the

searches	booked_same_country	median_price	median_booking	avg_price	avg_booking_price
48013	False	128.00	317.74	411.239718	555.393509
90377	True	107.48	185.45	180.440704	296.443117

same country, the average booking price was 555.39, compared to booking the

same country which had an average booking price of 296.44. More users book the same country because they pay less than compared to booking another country where they pay almost double as much.

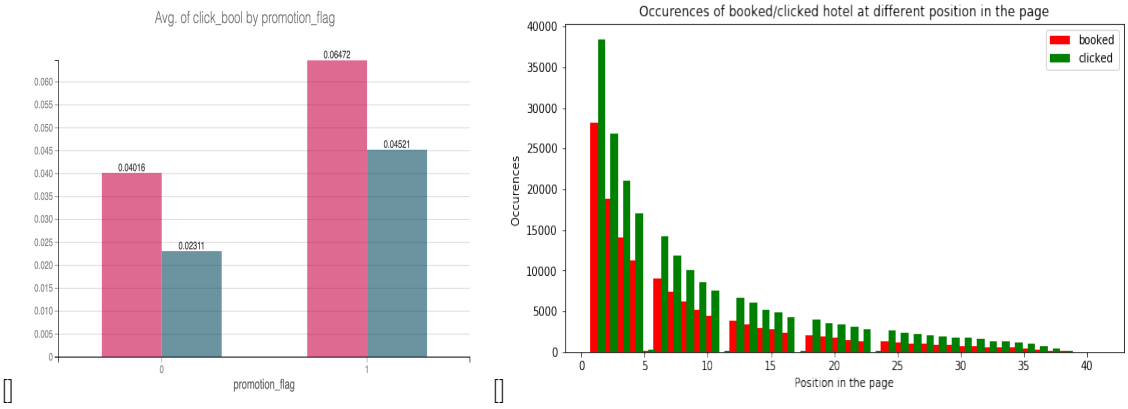

Fig. 3. (a) Average of clicks by promotion flag (pink: click, blue: book) (b) Position on booking and clicking behaviour

Figure 3 (a) shows that promotion flag has a positive affect on booking and clicking behaviour. When no promotion flag was present, an average of around 33 percent of bookings were processed. With the promotion flag present, the booking rate doubled to roughly 66 percent, showing a positive correlation between booking, clicking and promotion flag.

Figure 3 (b) shows that the positioning of the hotel on Expedia influences booking and clicking behaviour. Users book and click on properties that are ranked higher in position than properties that are ranked lower.

properties	median_review_score	median_star_rating	avg_review_score	avg_star_rating	booking_bool	click_bool
4736468	4.0	3.0	3.772451	3.173522	0	0
83489	4.0	3.0	3.814749	3.359820	0	1
138390	4.0	3.0	3.937649	3.312060	1	1

As review score and star rating are positively correlated, the table shows how they are related to clicking and booking behaviour. More users click and book properties with review scores and star ratings. 138390 properties were booked and clicked with a 3.94 review score and a 3.31 star rating.

Loc2_score consists of missing values as opposed of loc1_score, which explains the difference of average score between them. However, an upward trend can be detected when users click and book. The avg_loc1_score was highest with 0 booking but with 1 click and for avg_loc2_score, highest when users both clicked and booked. Hence, prop_loc2_score has higher predictive power than loc1_score.

properties	median_loc1_score	median_loc2_score	avg_loc1_score	avg_loc2_score	booking_bool	click_bool
4736468	2.77	0.0668	2.871865	0.127672	0	0
83489	2.89	0.1019	2.962662	0.169001	0	1
138390	2.71	0.1276	2.843014	0.188592	1	1

4 Data Preparation

4.1 Target Variable

As described in the evaluation section of the competition, we computed a relevance score for each property (5 - if booked, 1 - if clicked, 0 - otherwise). We used this column as the target prediction variable.

4.2 Feature Engineering

After exploring and investigating the data, we sought to create more features that would aid the ranking process.

Z-score normalization across properties in the search The column price_usd was not usable in its original form, given that it was not clear whether the price was per night or an aggregate for the entire booking. Therefore, we decided to normalize by computing the z-score of the price wrt all the other properties in the search query. We then decided to normalize the other property-related numerical columns such as prop_starrating, prop_review_score, prop_location_scorex, etc as it would be easier for the model to deal with z-score as opposed to raw values.

Given that the mean is very sensitive to outliers, we decided to also compute a modified z-score (with respect to the median as opposed to the mean). [14]

Summary of property values across searches It was noted in the discussion forum on the original kaggle competition [9] that a high score was achieved by simply computing the average, std, median of the numerical property features across searches. This, we believed, would give a sense of the average score/price/etc of the property, independent of the search parameters. Therefore, we computed the same for all the numerical property features. In addition, we also normalized the raw values using z-score, in order to get a sense of how much the search-specific property features were deviating from their averages.

Time features We extracted the month and weekday of search as well as the booking date with the intention of potentially exposing some patterns with respect to seasonality or holidays.

Deltas (price, rating) The visitor_hist_starrating and visitor_hist_adr_usd columns indicate the visitor's historical information about the rating and prices of the hotels that they had previously booked. We computed their respective deltas with the property rating, price, with the assumption that the budget ranges of people seldom changes.

Combining Competitor Information There were few features comparing the property listed on Expedia with the listings on competitor booking websites. We decided to combine them as they were so many columns (involving a lot of NaNs) which might contribute to noise. We did this by coming up with an aggregate column which encapsulated all the relevant information. We took a weighted sum of comp_rate(whether Expedia had a lower price) and comp_inv(whether competitor had no availability) multiplied by the absolute comp_rate_diff. We computed this value for all the competitors, and added them to come up with an all-encompassing all_comp_rate. We further treated this column as another numerical property value and computed the z-score normalization and summary attributes as described previously.

Removing Outliers Finally, we decided to remove outliers in the data, which we believed would add noise to the model, if trained with. To achieve this, we removed rows for which the value exceeded 3 standard deviations away from the mean of the column. This removed about 7.5% of training data.

4.3 Feature Importance and Selection

After all the computations described above, we ended up with 10 features. We reduced this feature list (as we deemed it to be too large) by eliminating the features that scored low on feature importance as output by the models we trained. We then retrained the model with the updated feature list. It was interesting to note that some of the features that we thought would be important such as the date time extractions, srch_saturday_night_bool, deltas in price, rating, etc turned out to not be that important. The top 10 features are described in the next section

4.4 Train-test data split

In order to test our models locally we split our training data, 80% was used to train the model and 20% was used to evaluate the models accuracy using the NCDG metric. This will be described in Section 5.3.

5 Model and evaluation

5.1 Introduction

Once we had identified and selected the features we thought would be useful, we needed to select models that would be able to make use of them to provide accurate predictions for the ranks of the various properties. Due to the sheer number of total features and their variety we decided to go for two different approaches, to try and cover a large basis. The first was to use implementations of the Lambdamart algorithms from the LightGBM and XGBoost libraries. The second was to train two binary classifiers, one that predicts whether a property has been clicked and another that predicts whether the property has been booked.

5.2 Learning to rank algorithms

There are a set of algorithms that are designed with ranking problems in mind. Within this set of algorithms are three main approaches of predicting how suitable a given answer is to a query. The three approaches are Pointwise, Pairwise, and Listwise [7].

Pointwise This approach looks at how suitable a single answer is for a given query, independent of other answers. In this case the problem of ranking essentially becomes a regression problem. As the algorithm will be trained in such a way that for a given query-answer it predicts the relevance. With many examples of query answers the model can be used to predict the relevance for a list of answers to a query and then the answers can be ranked based on the models output. Most regression models can be used to accomplish this.

Pairwise A pairwise approach can be accomplished with classification models where the target is to predict whether an answer is relevant to a query. The probabilities of whether the various answers to a query are relevant can then be used to rank the answers.

Listwise Listwise approaches try to minimize the error in the above approaches across the entire training dataset, as in they take either a pointwise or pairwise approach and try to correct the error in the model parameters that create errors in the ranking over the entire training set.

5.3 Our approach

We implemented a pairwise approach - LambdaRANK. The LambdaRANK function is a gradient formulation that aims to minimize the ranking loss on a pairwise level. We utilised the LightGBM and XGBoost libraries that implement gradient boosting (of the lambdarank gradient in this case) using decision tree learners. [4] [2].

We also implemented a pointwise approach using a pair of binary classifiers that we then took the probabilities of and ranked the results for a given query.

Pairwise LambdaMART - LightGBM

This model is a gradient boosted model, that uses an ensemble of decision trees. The LightGBM library provides a SKlearn like API for training a ranking model that takes a listwise approach while allowing for the use of different kinds of trees to create the model. The parameters for this model are as follows:

- **boosting_type**: There are three different types of ensemble decision trees that we experimented with: GBDT, Gradient Boosting Decision Tree. DART, Dropouts meet Multiple Additive Regression Trees. GOSS, Gradient-based One-Side Sampling. We settled on using GOSS as we saw the best performance using that, with DART being the worst.

- **objective**: The library provides an objective function which aims at minimizing the loss of the prediction of the relevance for a given model.
- **learning_rate**: Controls the rate at which the training will complete an important part of the gradient boosted methods as they are quick to learn and can tend to overfit.
- **max_depth**: Determines the maximum depth that each of the trees of the decision tree ensemble model can have.
- **n_estimators**: The number of gradient boosted decision trees to train that are part of this model.
- **subsample**: The randomly selected percentage of the training data that will be used when fitting each tree. The value of this parameters is also important to avoid overfitting.
- **colsample_bytree**: The same as the subsample except this refers to the percentage of features that will be used in training the models.
- **metric**: This is the metric for which we are trying to minimize the loss, this metric is the one we will be evaluated on. We used the Normalized Discounted Cumulative Gain, NDCG for the top 5 elements as this was the evaluation metric that would be used in the competition scoring [6]. This metric will prefer a higher relevance for items that are actually ranked higher, for only the top 5 elements.

$$DCG_p = \sum_{i=1}^{k} \frac{2_i^{rel}}{\log_2(i+1)} \quad (1) \qquad IDCG_p = \sum_{i=1}^{|REL_p|} \frac{rel_i}{\log_2(i+1)} \quad (2)$$

$$nDCG_p = \frac{DCG_p}{IDCG_p} \quad (3)$$

Where p is 5 in our case.

Pairwise LambdaRANK - XGBoost

This model is also a gradient boosted model that uses an ensemble of decision trees with the main difference being in how the gradient boosting is implemented. We had no reason to believe one was better than the other, so trial and error through tuning the parameters and trying different features was to used to maximise our score. The parameters for this model are as follows:

- **tree_method='approx'**: The split finding algorithm to use when fitting the decision trees, we use approx since it is more computationally efficient and we have a fairly large amount of data, as well as due to the fact that we have lots of features and we want to avoid overfitting to our training data.
- **booster='gbtree'**: Gradient boosted trees are what we used drawing on our experiences from the LightGBM ranking model. This is because using DART, Dropouts meet Multiple Additive Regression Trees did not give us good results.

– **objective='rank:pairwise'.**: Use LambdaMART to perform pairwise ranking where the pairwise loss is minimized.

We believed that a pairwise approach using these two libraries was the most appropriate given that the nature of the ranking problem in this case was usually dealt with in a hybrid way where pairwise comparisons are often the deciding factor [7]. Consider for instance the process of selecting a hotel to book from a list of option, you might initially scan for the few properties that match your desired attribute(s). Once you have done that you are likely to then compare your narrowed down list on a hotel-to-hotel basis.

Pointwise approach - Two Binary Classifiers

Our other approach was to attempt to just use the probability output of a pair of binary classifiers and use that to rank the results of given search. Our reasoning for this is that whether a result is relevant or not (clicked or booked in our dataset) is something can be seen on an independent basis as classification of relevance between a query-answer.

If the model is able to learn what makes an answer appropriate for its query or not well enough then the probability can be seen as a measure of the likelihood of its relevance to the query. This enables ranking a list of answers to a query based on this probability.

In our scenario we have two variables that we focused on that indicate relevance that we then combined as follows.

$$rel = ((c = 1 \rightarrow 1) \wedge (c = 0 \rightarrow 0)) + ((b = 1 \rightarrow 5) \wedge (b = 0 \rightarrow 0))$$

Where c is 1 when the property has been clicked and 0 otherwise, similarly for b but for booking

With the output from our two classifiers we are able to do a similar calculation except we multiply probabilities with their respective values had they been true. This is formulated as follows

$$rel_p = c_p.1 + b_p.5$$

Where c_p is the output predicted probability that a property is clicked by model trained to predict clicks and b_p is probability that a property is booked from the model predicting bookings.

With this setup we can compare the predictions of the model based on the same NCDG metric described before if we rank them based on the predicted relevance.

We tried a few different binary classifiers, DecisionTreeClassifiers, KNN, and Gaussian Naive Bayes classifier from sklearn as well as the LGBMClassifier from LightGBM. We settled on taking forward and optimizing the LGBMClassifier since it is an ensemble classifier and we had promising results with the LGBMRanker model.

This makes this approach a pointwise approach, since we train and predict on an individual query-answer basis. It could be argued that the combination two models makes this a hybrid pointwise-listwise approach since the relevance is calculated across multiple answers to a query. The parameters for both our binary classifiers are: 1. **n_estimators**=500 2. **colsample_bytree**=0.75, 3. **subsample**=0.75, 4. **max_depth**=75
These all have the same meaning and utility as described previously in the LGBMRanker model.

5.4 Hypertuning

Initially, we chose the parameters based on defaults or on manual trial and error and observation of which parameters were leading to the best performance. We then decided to hypertune them using the hyperopt library. We ran about 100 iterations of the minimization optimization function with a sequential search of the hyperparameter space. For the XGBoost model, the following parameters resulted in the best score: 1. **colsample_bytree**: 0.67 2. **learning_rate**: 0.095 3. **max_depth**: 12.0 4. **n_estimators**: 20 5. **subsample**: 0.82 For the LightGBM model, hypertuning did not yield results that were better than the defaults.

5.5 Evaluation

A comparison of the performance of various models we trained can be found in table 1

Model	NDCG@5 (training data test split)	NDCG@5 (Kaggle test data)
XGBoost	0.365	0.342
LightGBM	0.396	0.389
Two Binary Classifiers	0.382	0.372

Table 1. Comparison of different models

The approach of using two binary classifiers is a simple implementation, and it is worth noting that it comes quite close in performance to the LambdaMART-based models.

LightGBM's implementation scored better as compared to XGBoost. We believe this is due to the fact that LightGBM's faster training times allowed us to experiment with a much higher number of estimators (500) as compared to the number of estimators used for XGBoost (20).

The scores on the Kaggle test dataset were consistently lower but only slightly lower than the scores run on the training data split locally. This is an indication that our model did not overfit the training data to a large extent.

5.6 Final Model

The model that we finallly settled with, which gave us the highest score on Kaggle was the LightGBM implementation of gradient-boosted decision trees of the lambdarank gradient with the following parameters:

- boosting_type=GOSS
- objective=LambdaRANK
- learning_rate=0.1
- max_depth=75
- n_estimators=500
- subsample=0.75
- colsample_bytree=0.75
- metric="ndcg@5"

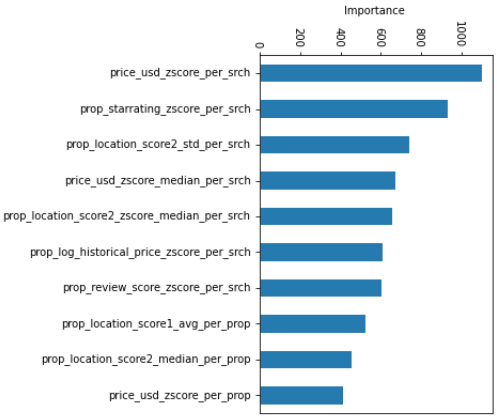

Fig. 4. Feature Importance

The top 10 important features as calculated by the model are in Figure 4.

Learnings

When you have a lot of data have to be smart about how to process it to get something you can train a model on.

Ranking is a hard problem to predict for since you have a target variable that is not really perfect or indicative of the desired result.

Feature engineering takes a lot of time and requires insight into the meaning of the various features to extract or create meaningful representation of the various states and outcomes.

Model parameters have a large effect on the performance of the model and often comparing the setup of winning solutions is necessary to maximise performance.

Trial and error is a very important part of the process of creating machine learning models, as each iterative attempt will guide in the direction that needs to be taken.

References

1. Qiang W., Chris J.C. B., Krysta M. S. and Jianfeng G.: Ranking, Boosting, and Model Adaptation. Microsoft Research Technical Report **MSR-TR-2008-109** (2008)
2. Chris J.C. B.: From RankNet to LambdaRank to LambdaMART: An Overview. Microsoft Research Technical Report **MSR-TR-2010-82** (2010)
3. Christopher J.C. B., Robert R., Quoc Viet L.: Learning to Rank with Nonsmooth Cost Functions. Advances in Neural Information Processing Systems **19**(2007)
4. The inner workings of the lambdarank objective in LightGBM, https://ffineis.github.io/blog/2021/05/01/lambdarank-lightgbm.html. Last accessed 24 May 2022
5. Stackoverflow - Sample parameters, ttps://stackoverflow.com/questions/51022822/subsample-colsample-bytree-colsample-bylevel-in-xgbclassifier-python-3-x. Last accessed 24 May 2022
6. Wikipedia - Discounted Cumulative Gain, https://en.wikipedia.org/wiki/Discounted_cumulative_gain. Last accessed 23 May 2022
7. Wikipedia - Learning to Rank, https://en.wikipedia.org/wiki/Learning_to_rank. Last accessed 25 May 2022
8. https://www.dataiku.com/
9. Expedia Personalized Sort https://www.kaggle.com/competitions/expedia-personalized-sort/discussion/6228
10. "2nd Assignment DMT." Kaggle, www.kaggle.com/competitions/2nd-assignment-dmt2022/overview/evaluation.
11. https://www.kaggle.com/c/expedia-personalized-sort/data
12. https://www.knime.com/
13. https://www.dropbox.com/sh/5kedakjizgrog0y/_LE_DFCA7J/ICDM_2013
14. Modified Z-Score https://www.statology.org/modified-z-score/
15. https://rapidminer.com/platform/
16. https://www.kaggle.com/c/expedia-personalized-sort/overview