Ranking Hotel Search Queries for optimal Purchase Rates

Luisa Ebner, Frederic Chamot & Philipp Lintl

1 Introduction

Ranking is an important class of machine learning problems, which focuses on finding an optimal order to a set of inputs in accordance with the preference of an individual. This report concerns ranking hotel search data provided by the online travel agency Expedia for the 2013 Kaggle Competition. On Expedia.com, users can input a number of criteria to perform a query for a hotel. The website then returns a ranked list of hotels. Finally, the user can click a hotel to gain more information, actually book a hotel room or do neither of both. As the online tourism is highly competitive, optimally matching their hotel inventory to their customers lifts their chances in actually bringing about a sale.

Business Understanding The objective target of this project is to develop a machine learning model, that successfully outputs a list of hotels ranked by highest likelihood of being booked by a customer on the basis of limited information about customer preferences. Detailed descriptions of the approaches and models used by the contestants of the 2013 competition are scarce, but some reports exist. Liu and colleagues [1] achieved a fifth place overall by combining a diverse array of individual ranking models into an ensemble that achieved a NDCG@38 (metric discussed in section **2.5**) of 0.53102. Remarkably, a single pairwise logistic regression model using seven simple features attained a score of 0.51273 [2], not far off the final, and much more complex model. Generally, crucial steps on the way to a pointed ranking result seemed to have been extensive feature engineering of hotel and customer characteristics, the predictive imputation of missing values and the use of the state-of-the-art ensemble learner LambdaMART. As a combination of the Lambda and the MART algorithm, LambdaMART belongs to the family of pairwise *Learning to Rank* algorithms, where a final ordering is achieved by iteratively comparing the relative relevance between pairs of hotels.

2 Explanatory Data Anaysis

2.1 Data Description

After the data set was posed by Expedia as the working basis of the 2013 Kaggle Competition, the competition organizers split the data randomly into test and training data. Overall, the data consists of a list of hotel search queries,

each of which are associated with a list of different hotels. Each search query consists of a list of 1 to 38 candidate hotels. The 53 given feature vectors comprise categorical, binary, date as well as numerical features, in turn comprising information on criteria of the search query, static and dynamic hotel characteristics, user's aggregate purchase histories, visitor information and competitive OTA information. Furthermore, we are given a 3-dimensional target vector of the form $[booking\_bool, clicking\_bool, transaction\_value]$, indicating whether a hotel was booked, clicked and what the sale price of a potential transaction was.

2.2 Data Exploration

Size of the data set It is to highlight that the entire data set is enormous in size: The training set comprises almost 400 thousand unique search queries conforming to nearly 10 million hotel lines. For the test set, we count 6 622 629 hotel lines representing 266 230 search queries. To save memory and computation time, we explored and plotted the training data on the first 100 000 lines only.

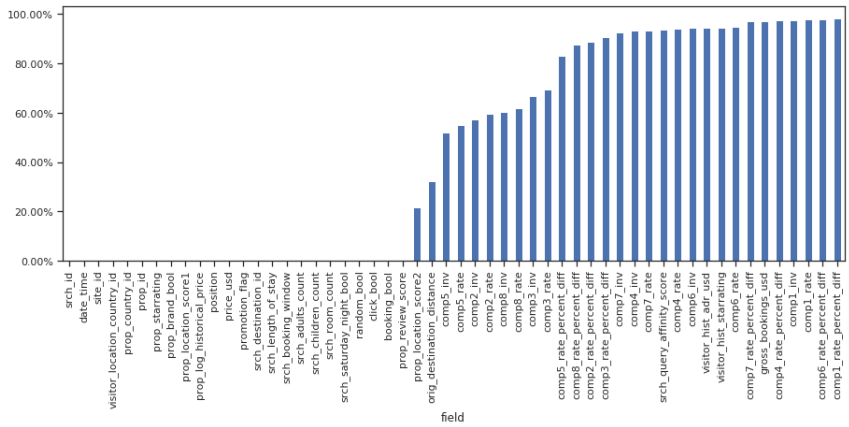


Figure 1: Percentage of missing values per feature.

Missing values Initially, we found that one great challenge within this ranking task will consist in handling the large amount of missing values. All 24 competitor information features contained empty input values. Furthermore, the visitors' purchase or star rating histories, record a fraction of more than 90% missing values (Figure 1).

Outliers Other than that, hotel price features and the competitor related features were found to include a number of erroneous outliers towards the top of their value range. The extremest values regarded hotel booking prices (see Figure 2).

Position Bias The hotels offered per search query are returned in a list. In 70 % of the cases this list appears sorted according to Expedia's current ranking algorithm. Only 30% are listed in random order to (partially) counteract the position bias.
As can be seen in Figure 3, Expedia's ranking clearly performs better than the random one. This introduces a bias of unknown size to the `position` feature in our data set.

Figure 2: Outliers according to boxplots.

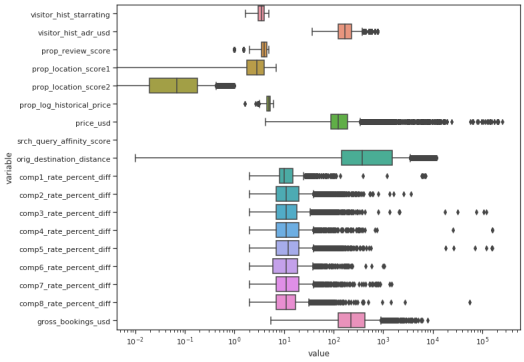
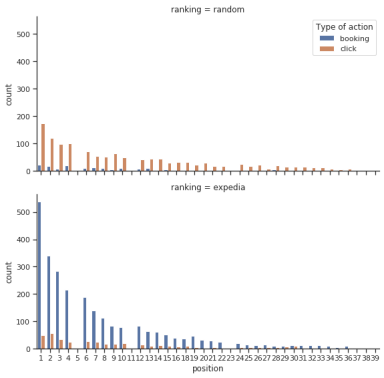


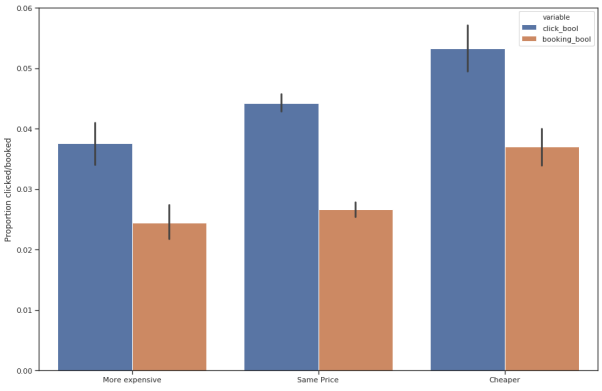
Figure 3: Position bias



Target Imbalance In each query, a customer books at most one hotel. Due to this nature of customer's online activity most instances per query are negative. In fact, only 2.8% of all hotel entries are positive in the sense of actually being purchased. 4.5% were clicked. Therewith, we are confronted with highly imbalanced classes, where only a small minority of training instances were booked.

Prize comparisons One might initially assume that competitor prices do not play a role in a customers booking behavior. With a look at Figure 4 however, we refuted this presumption. In fact, we see a monotonic increase in both the booking and the clicking behavior as Expedia's offers vary from *more expensive* over *equally prized* to *cheaper* than the competitor prizes.

Figure 4: click/booking proportion by price.



3 Predictive Modelling

3.1 The Modelling Task

Let $H_i = \{h_{i1}, ...h_{im}\}, i \in \{1, ..., 199795 \text{ search queries}\}, m \in \{1, ..., 38 \text{ hotels}\}$ be a set from Expedia's hotel inventory returned for the search s_i of a certain customer. Each hotel h_{ij} originally has a set of maximally 51 features, some of them hotel specific, some visitor-, some search- and some competitive pricing specific. Furthermore, let $y_{ij} \in [0, 1]$ be the relevance value for hotel h_{ij}. The machine learning task underlying this report now is to find an optimal permutation of the set of hotels in H_i for a given set of features defining search s_i. This is done assigning a score to a each hotel-search pair (h_{ij}, s_i), preferably proportional to the likelihood of h_{ij} actually being purchased by a user.

3.2 Data Preparation

In order to process and adapt the original data provided, we used the python library `pandas` as our basic data infrastructure.

Data Balancing Given the rigorous under-representation of booked and clicked hotels in our data, one has to be aware not to train classifiers that always predicts *not booked*. Even though a ranking task is about the relative and not the absolute probability of a hotel being booked, we can disregard the outsized multitude of negative instances in view of learning structures in the data. Among the various ways to handle imbalanced data, we decided for the direct approach to undersample instances, that are neither being booked nor clicked. In doing so, we kept the entire minority class of booked or clicked instances, while keeping only the k smallest position entries of negative instances per search query, whereas k was chosen to be 5 and 10 in the experiments.

Validation-Train Split Detaching a validation set from the training data enables to assess the performance of single models before actually submitting and applying them to the test set. Therefore, we created a 25% validation set from the instances in the training set, by selecting those search queries whose search ID modulo 4 resulted in 0.

3.3 Feature Engineering

Feature Elimination Good ML models premise a thorough understanding of the provided feature information, which of them are important and which are rather not. In this regard, we considered only one feature, namely `date_time` to be irrelevant for our ranking task.

Feature Imputation For several features, a large fraction of values is missing. We approached the issue of sparse features with imputation. Instead of only imputing by mean or median, we applied linear regression or Random

Forest models. In particular, we used predictors that are most correlated to the feature to be imputed according the threshold 0.15 of spearman correlation. Those features that did not yield any or only one predictor were imputed with a random forest regressor. Features with an enormous amount of missing values, such as `visitor_hist_starrating`, `srch_query_affinity_score` and `visitor_hist_adr_usd` were imputed according to their worst case value. Meaning, for the first and second the min and the mean for the latter. After the entire feature engineering procedure, we still found `na` values which resulted from taking the standard deviation per property id and search id.

Outlier Clipping After assessing the boxplots of the numerical features, we decided to control outliers of the features related to price `price_usd` and `visitor_hist_adr_usd`. The upper outliers deviate stronger, therefore we decide to clip all instances at the 0.975 quantile (439.5).

Feature Transformation We log-transform the price related variables to rectify the originally skewed, cumulated values. All numeric features were normalized over their entire feature vector. To add more features, we additionally normalized some according to group variables, such as `prop_id, prop_country_id` or `srch_id`. Thereby, we yield balanced features with similar ranges to each other. This is deemed beneficial particularly for Logistic Regression, and the Learning to Rank algorithms, we applied in the course of the given ranking task.

Feature Generation Furthermore, we expanded the size of our feature set from 51 to 62 by combining features, that are intuitively correlated and creating new categories.

- difference between: (mean purchase history, price), (property rating, average rating previously booked by customer)
- grouped differences: `relative_country_price`: difference between price and average price of the country it is in; `relative_query_price`: difference between price and average price in the search query
- similarly: `relative_query_stars` (prop_starrating, average starrating in search query); `relative_query_locationscore`: (location score 2 and average location score 2 in the search query)
- average, median and standard deviation of all numerical features separately and additionally grouped by destination id, property id and search id, as well.
- boolean features: `new_customer` (if no purchase and no rating history is given); `children_bool` (children present or not)
- `persons_per_room` (add children and adults divided by the number of rooms)

Target Design There are several possible approaches to model the ranking task. We decided to compare a few variants for the target feature. Firstly, we perceive the ranking problem as a binary classification task, distinguishing `booked = 1` from `not booked = 0`. In accordance therewith, the instances of each search query were ordered according to their probability of being booked. Secondly, we regarded the ranking problem as a regression task with the ordering being based on the magnitude of a score, calculated as follows: Where the *value* 5 was assigned to booked hotels, 1 to clicked hotels and 0 for those tagged with neither action, the final score composes as $[value + 2 - [\frac{2}{1+\exp(0.1*(-position+1))}]]$

3.4 Modelling Approaches

3.4.1 Logistic Regression

When conceiving of the ranking problem as a binary classification task, we primarily used logistic Regression as a standard linear classifier to rank instances. We used the boolean feature `booking_bool` as target, which represents the booked as positive and the rest as negative instances. In our implementation, we adopt the probability of being booked directly as the relevance score for a search query[2]. Initial modelling attempts relied on features selected by correlation with the target but produced low scores, which improved when the top 20 most important features as discriminated by GB were employed to control the weight parameters of the model. In the scope of this project, Logistic Regression chiefly served as a simple baseline model.

3.4.2 Random Forest

Next up, we trained a Random Forest model as a nonlinear regressor, that is specifically approved for its favourable handling of heterogeneous and noisy data sets and for taking account for non-linear dependencies in the data [3]. Regarding model parameters, we used the maximum amount of regression trees allowed for by our computational resources (100-500) with $\sqrt{n\_features}$ variables to be tried at each split. We adapted the estimator instance to the target design, employing RF regression for `score_rank` and RF classification otherwise. To predict the booking probabilities for each hotel in each search query we ranked the properties according to the predicted values in the case of regression and according to the prediction probabilities in the case of classification. For the target designs discriminating between clicked and booked properties a weighted sum of the probabilities for both was used to sort (4:1 book/click weighing produced the highest scores). Based on this ranking prediction, we calculated the NDCG.

3.4.3 Gradient Boosting Trees

Apart from Random Forest ensembles, predominantly Gradient Boosting approaches win data competitions hosted on Kaggle. As these tend to be computationally costly, we drew on the efficient `python` implementation `catboost`

[4]. Gradient boosting sequentially optimizes the creation of decision trees in the ensemble by fitting the residuals of the previous tree and is thus using the errors to improve the result. The number of instances of a feature used in Gradient boosting decision tree's nodes is proportional to its effect on the overall performance of the model and different to RF it does not bootstrap but uses the entire data set. Apart from the total number of trees to be created, it also inherently performs parameter tuning. Depending on the target and specified loss function, we either classify or regress. [5]

Pairwise

Additional to rank by supervised learning, specific approaches to ranking exist. In particular, the family of pairwise learning methods that optimize the objective function defined on pairs of documents retrieved for a given query. Given a pair of documents, they try and come up with the optimal ordering for that pair and compare it to the ground truth. The goal for the ranker is to minimize the number of inversions in ranking i.e. cases where the pair of results are in the wrong order relative to the ground truth. Pairwise approaches tend to work better in practice than pointwise approaches because predicting relative order is closer to the nature of ranking than predicting class labels or relevance scores. [6] With `Pairlogit` and `PairlogitPairwise` we refer to two pairwise settings enabled by the `catboost` package.

3.5 Model Evaluation

The preceptive evaluation metric for this competition is the average Normalized Discounted Cumulative Gain over the first 5 search queries. Denoted as NDCG@5, this metric can account not only for class membership of the test set, but also relative ranking of the result. Normalizing the resulting DCG by the best possible DCG per search query to guarantees that a perfect ranking at position n is 1.

Our predefined relevance function $f(x)$ highly rewards hotel purchases, and marginally prefers clicks over no user action as follows:

$$f(x) = \begin{cases} 5 & \text{if the user purchased a room} \\ 1 & \text{user clicked for hotel information} \\ 0 & \text{user neither booked not clicked the hotel} \end{cases}$$

For each model, we used simple hold-out cross validation and calculated the NDCG@5 as the average of both results.

4 Experiments

4.1 Feature importance

In addition to experimenting with the models and their parameter settings, we extracted features based on feature importance obtained by the gradient boosting trees. We end up with the features in Table 1. In this regard, feature importance was assessed with catboosts default option, so called Loss-Function-Change, which considers a difference of the loss function and the SHAP value of a feature.

Considering that many highly important features are generated ones, we regard the feature engineering as succesful. We also remark that most of the important features are either price or location score related. Removing those features increased speed drastically, but in our final ensemble tree learners performance also dropped. Therefore, it rather serves as an observation to identify important features and helped to decrease model selection time.

Feature names
srch_diff_locscore2,
prop_loc_score1, prop_rating,
prop_loc_score2,
norm_srch_diff_locscore2,
prop_log_hist_price,
srch_diff_price, prom_flag,
srch_ave_loc2, srch_ave_star,
srch_diff_prop_review_score,
norm_srch_diff_price,
srch_diff_star,
ave_num_prop,
norm_price_usd_prop_id,
prop_review_score,
star_diff, srch_len_stay,
norm_srch_diff_locscore2,
srch_dif_locscore1,
srch_diff_locscore2,
norm_price_usd_srch_id,
srch_ave_loc1,price_usd,
norm_country_diff_price

Table 1: Features sorted according to decreasing feature importance.

4.2 Model selection

In order to find the best performing model and to draw conclusions about the challenge, we conducted several experiments. First of all, we test our previously mentioned target score variants for either the full data or an undersampled version by rank, as described above, which is favorable due to significantly lower computational costs. Also, different targets imply different model specifications, as `score` and `score_rank` are numeric and thus predicted by regression, whereas `book` and `book_click` are classified and ranked according to probability.

Algorithm	**Target**			
	score	score_rank	book	book_click
RF full	.3341	.3300	.3299	.3340
RF rank10	.3465	.3341	.3470	.3472
RF rank5	.3454	.3347	.3460	.3457
GB full	.3621	.3572	**.3780**	**.3804**
GB rank10	.3609	.3576	.3767	.3776
GB rank5	.3595	.3586	.3743	.3761
GB full pair	**.3721**	-	.3724	.3726
GB rank10 pair	.3676	-	.3672	.3678
GB rank5 pair	.3645	-	.3647	.3654
GB rank10 pairlogit	.3674	-	.3668	.3673
GB rank5 pairlogit	.3648	-	.3651	.3650

Table 2: Experiments regarding the type of target, algorithm and degree of undersampling according to `NDCG@5`. RF: Random Forest, GB: Gradient Boosting trees, full: no undersampling, rank 10/5: no action instances filtering down to top 10/5 positions per query. All GB models fitted 500 trees.

In Table 2 we observe lower scores for RF models across all specifications. GB models score higher for the entire data set compared to undersampled ones. Surprisingly, the pairwise models did not outperform the simple supervised one. As we only fitted 500 trees, this speaks for overfitting of the basic version and underfitting of the pairwise ones. As a final option, we decided to use up to 3000 iterations, much more than the previously deployed 500 trees. This lead to a highest score with the full dataset on the submission and due to time limitations only ran 2000 iterations and could not be investigated across all settings and models. As `score_rank` yielded low submission scores, we spared the costly pairwise computations (marked as -). During the model phase, we observed slightly different scores between our validation and the submission set. Table 3 shows that we tend to overfit, as a gap between test and validation scores is observable.

Algorithm	`validation`	`submission`
Random Ordering	.157	-
Logistic Regression	.330	.3336
Random Forest	.349	.3626
GB full book_click	.3804	.329
GB rank5 book pairlogit	.3651	.3624
GB full score pairlogit	.3556	.3303
GB rank5 score pairlogit	.3648	.3616
GB rank 10 book_click pairlogit	.3721	.3702
GB rank full book_click pairlogit	-	**.3759**

Table 3: Validation and final submission performance. Best submission on 2000 iterations.

5 Conclusions and future considerations

We have showed that logistic regression, which can intuitively be extended to tackle learning to rank problems produce results significantly higher in quality than a random ordering.
We noted that the major challenges of this task consisted in dealing with a massive size, a great number of missing values, undersampling to cure the data imbalance, feature engineering and carefully designing the target variable. However, compared to our competitors we apparently did not or too late rely on computational power to boost performance. Earlier relying on more iterations might have helped to identify the best setting.
To our surprise, considering the previously ranked position did not influence but actually worsen our performance. To us, this is counterintuitive, as we expected the information of the previous ranking to additionally highlight important factors (Figure 3). However, none of our models regarded the position bias introduced by Expedia's yet applied ranking algorithm, which could have limited the performance of target `score_rank`.
Also, condensing the competitor information into one variable was not important to our final model according to feature importance. This is surprising, as we thought this would be favorable over not considering the mostly missing competitor features.
As we undersampled with regards to the feature `position`, we might have introduced unwanted noise to our dataset, as about 30% of the data was ranked randomly and thus `position` is not representative. One way to deal with this, might have been to predict, for instance by using random forests, the position values of instances that were ranked randomly.
Finally, an even higher score should have been possible by deploying proper computational power over more iterations.

References

[1] Xudong Liu, Bing Xu, and Zhang. Combination of diverse ranking models for personalized expedia hotel searches. 2013.

[2] Saurabh Agarwal, Luke Styles, and Saurabh Verma. Learn to rank icdm 2013 challenge ranking hotel search queries. *space*, 2:3.

[3] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[4] Catboost: fast, scalable, high performance open-source gradient boosting on decision trees library. https://catboost.ai/docs/, 2018.

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, 2016.

[6] Andrey Gulin, Igor Kuralenok, and Dimitry Pavlov. Winning the transfer learning track of yahoo!'s learning to rank challenge with yetirank. In *Proceedings of the Learning to Rank Challenge*, pages 63–76. PMLR, 2011.