Data Mining Techniques 2 - Group 29

Caterina Buranelli (2718588), Ignas Krikštaponis (2718072), and Charlotte Felius (2648506)

Vrije Universiteit Amsterdam
https://www.vu.nl/nl/index.aspx

1 Introduction

In 2013, Expedia started a challenge aiming to create a robust recommender system for users [8]. More specifically, the aim was to recommend a list of hotels that a user is most likely to book given a specific search query [5]. Moreover, additional information like the time, date, destination and children counts are included in the data set. The challenge of this competition particularly lies in the fact that hotels must be ranked based on queries and additional features, instead of solely ranking hotels on specific hotel features [6].

2 Evaluation

How the model performs is determined by the Normalised Discounted Cumulative Gain (NDCG), an evaluation measure commonly used in information retrieval and learning-to-rank (LTR) methods that has domain in $[0, 1]$, where 1 means that the ranking is perfect [6].

The Discounted Cumulative Gain (DCG) is calculate by the following equation: $DCG_p = \text{rel}_1 + \sum_{i=2}^{p} \frac{\text{rel}_i}{\log_2(i)}$, where rel_i is the relevance score of the hotel at position i (5 - for booked, 1 - for clicked, 0 - for rest) and p means that the measure is accumulated until that rank position, $p = 5$ in our case (in the text the notation is NDCG@p). The measure which is used in practice, suggested by [10] is the DCG normalised for its *ideal* value, calculated as: $IDCG_p = \text{rel}_1 + \sum_{i=2}^{|REL_p|} \frac{\text{rel}_i}{\log_2(i)}$.

3 Related work

Yet there are some researches performed on the Expedia problem set, as this was a public Kaggle competition held in 2013 [1]. The original winning submission was by Owen Zhang, who achieved a score of **0.53984** (evaluated with NDCG@38). Zhang handled missing values by imputing negative values, bounding existing numerical values (such as price) and the down sampling of negative instances for a faster performance. Moreover, he used 5 different groups of features: all original features, numerical features averaged over `srch_id`, `prop_id` and `destination_id`, composite features (e.g. `price_order` which describes the

order of the price within same `srch_id`), EXP features (categorical features converted into numerical features) and an estimated position [12]. The estimated position is an average over an EXP feature based on `prop_id`, `dest_id` and `target_month` and the position of the exact same hotel in the same destination in the previous and next search [12]. Zhang made use of Gradient Boosting Machines (GBM) where he deployed two types of models; models with and without EXP. He states that the most prominent predictors are position, price and location desirability.

The second-best submission [9] has a score of **0.53839** and stands out as they make an explicit distinction between user's rational behaviour and random noise. The principal ranking method used is LambdaMART, which is a nonlinear LTR algorithm . For categorical features they used Linear Regression or a linear LTR model (such as Support Vector Machines). Missing values were imputed by estimations based on hotel descriptions, historical user's data and on competitor descriptions [9]. Feature extraction entails an estimation of the hotel quality and the non-monotonicity of feature utility. The authors also noticed that the hotel price varies in different cities at different times, which made them decide to normalize hotel and competitor descriptions w.r.t. different indicators [9, p. 5]. To evaluate their model, they split the training data in a training set of 80% and a validation set of 20% of the data.

Another contestant [7] achieved a score of **0.53102** with a NDCG@38 evaluation method which marked them as 5th at the Kaggle leaderboard, chose a fairly different approach. Liu et al. used a myriad of different methods including deep neural networks, random forest (RF), GBM's and used LambdaMART as an ensemble method for rankers [7, p.1]. This contestant however too deployed LambdaMART, which therefore seems suitable for this Kaggle competition.

4 Data Background

4.1 Overview

Dataset dimensions The training set contains 4958347 rows and 54 columns. The test set contains 4959183 rows and 50 columns. `booking_bool`, `click_bool`, `gross_bookings_usd`, `position` are excluded in test set.

Missing values 31 columns have missing values (Figure 1). Most of the missing values come from the competitor's data. Logically, some historical user data is also missing as some users are first time Expedia customers. 28 column have more than 50% of missing values with 16 having above 90%.

Summary statistics Table 1 displays general statistics (distributions and click/ booking rates) for selected non-numerical columns in the training set.

4.2 Numerical data distributions

Boxplots for selected numerical columns are shown in Figure 2. Many of the variables depicted carry outliers and have wide ranges. We can also observe

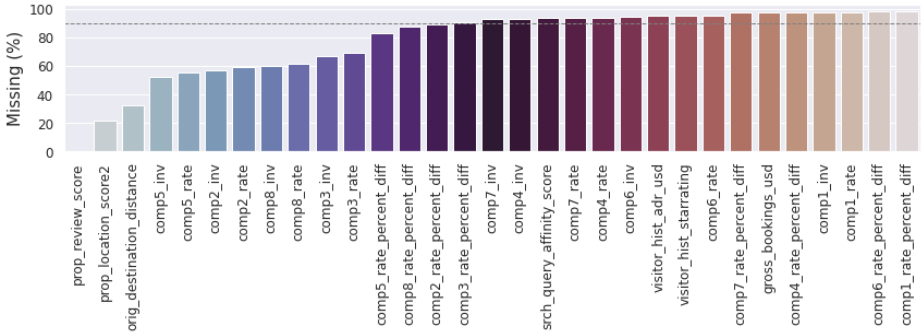

Fig. 1: Missing data. Horizontal line - 90%.

Table 1: Summary statistics for selected non-numerical variables.

Column	Summary statistics
`srch_id`	199795 unique values. Every `srch_id` occurs at least 5 times and a maximum of 38 times.
`prop_id`	129113 unique values.
`date_time`	Data available for the period 2012 11 01 - 2013 06 30
`click_bool`	4.47% of the results were clicked (click rate)
`booking_bool`	2.79% of the results were booked (booking rate)
`promotion_flag`	21.56% of results had a promotion flag. Booking/click rate with a flag - 3.92%/6.03%, without a flag - 2.48%/4.04%
`prop_brand_bool`	63.47% of results come from a major hotel chain. Booking/click rate for major brand hotels - 2.92%/4.49%, for non-major brand hotels - 2.57%/4.45%
`random_bool`	29.60% of the results were returned in a random manner. Random sorting booking/click rate - 0.53/4.66%, algorithmic - 3.74/4.40%.

some of the whiskers extending all the way to the left of the plot (e.g. for `prop_location_score1`) - this is due to the fact that these variables have a significant amount of zeros. `price_usd` has the widest data spread from 0$ to just below 20,000,000$. This makes sense as the price displayed depends on no. of days, no. of adults etc, therefore it might be beneficial to account for this during feature engineering. Furthermore, outliers can have a negative effect on predictive model's performance, therefore columns with many outliers will be treated before training.

4.3 Search position

Expedia displays search results in two ways - in a random order or in an order created by their internal algorithm. The type of result ordering is denoted by `random_bool`. Random result sorting is used for Expedia's algorithm training.

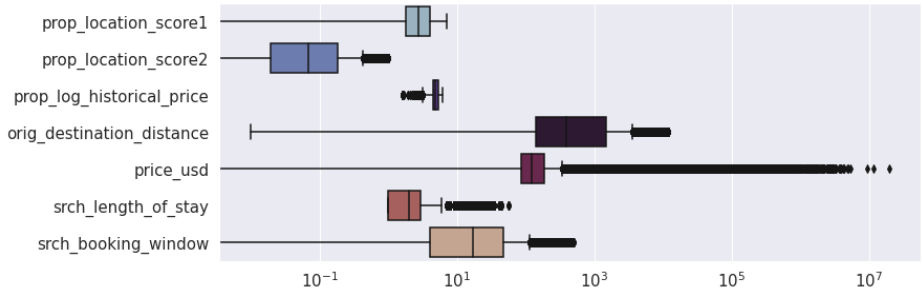

Fig. 2: Boxplots of selected numerical variables.

To explore the position bias we calculated the fraction (rate) of hotels booked/ clicked based on their position in the results query - the results can be observed in Figure 3. It is clear that the higher the hotel was in the search results the more likely it is booked or clicked, both for random and algorithmic ordering. For booking, the success rate gap between random/algorithmic ordering is significantly wider than that of clicking where the trends were similar - this is due to lower booking rate as outlined in Table 1. Interestingly, there are sharp fluctuations in both booking and clicking rates - this could be due to the search result being at the bottom of the page before the customer loads another batch. It is worth exploring if removing hotels at these positions is beneficial for the model's performance.

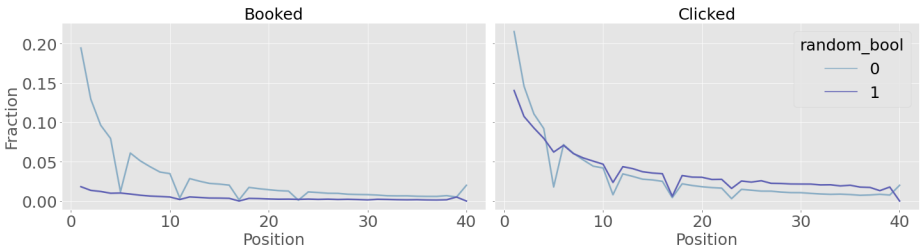

Fig. 3: Fraction of booked or clicked hotels based on their position in the search results.

4.4 Pricing position

Dense rankings of hotels were calculated inside `srch_id` based on `price_usd`. The same booking and click metrics were calculated as outlined in subsection 4.3. The results can be observed in Figure 4. We can see that the price position of the hotel highly affects the probability of the hotel being clicked/booked - both

for random and algorithmic sorting. There is a spike ("clicked") for the most expensive hotels with random ranking - this is due to the small sample size.

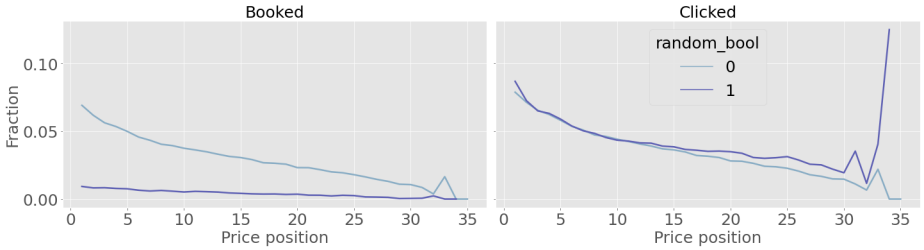

Fig. 4: Fraction of booked or clicked hotels based on their price position in the search results; 1st position - lowest price.

4.5 Seasonality

Booking start date was calculating with `srch_booking_window` and `date_time`. The month was extracted to bin the data into seasons: "winter" (months 12, 1, 2), "spring" (3-5), "summer" (6-8), and "autumn"(9-11). The booking rate was calculated for each `prop_id`, and for each `prop_id` and season combination. `prop_id`s that appeared only in one season were removed from this analysis. Differences between overall `prop_id` booking rate and seasonal `prop_id` booking rates were calculated and a Wilcoxon test was performed. The test produced a p-value < 0.05 concluding that the booking start date significantly affects booking rates. Furthermore, we get the same outcome when performing the analysis on `date_time` column.

4.6 Review score

The probability of hotels being booked/ clicked conditionally to the review score is plotted in Figure 5. The score goes from 1 to 5 in step of 0.5 and 0 stands for absence of reviews.

In both of the figures, we see that the conditional probability increases from score 1 to 4 and then decreases for the last two score points.

This behaviour could be influenced by the fact that highly reviewed hotels are also more expensive, while very low scores discourage the customers; as 4 is the peak (the probability of being booked and clicked within hotels rated with 4 points is respectively around 3% and 5%), it seems to be the best trade-off between what is perceived as a good balance between score and other latent variables such as price.

Moreover, the conditional probability for hotels with no reviews is unexpectedly high, but this could also depend on the price: it often happens that hotels

that just register on a new website allow a discount in the beginning to quickly attract costumers and therefore first reviews.

Nevertheless, analysis of latent variables is beyond the scope of this paper, so it will not be investigated further.

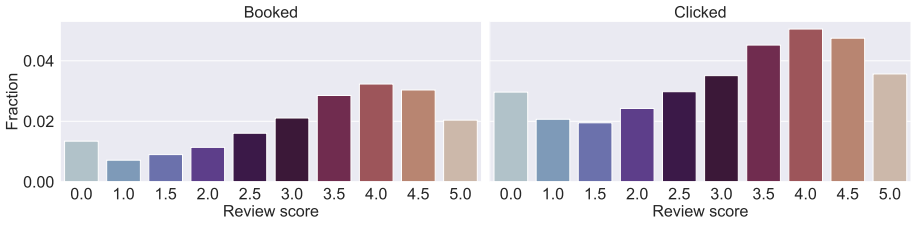

Fig. 5: Fraction of booked or clicked hotels conditioned to their review score.

5 Data Preprocessing

5.1 Missing Data

Columns with more than 50% missing data were excluded from the analysis, as they were considered not relevant for the too little information they carried; 28 columns were removed after this procedure. We chose to set the remaining missing values as the worst case scenario, therefore the missing values were set as negative (-1), this was inspired by the second best submission in the original competition [9].

5.2 New features

Supplying timestamps into the model is not an option as there are too many unique values, therefore we extracted month, week, day and hour of the query from the column `date_time`; the original column was then deleted. The result in the EDA concerning seasonality (Section 4.5) led to the decision to keep the temporal data in the model, as it highlighted the dependence of the booking rate on the temporal aspect of the query.

The mean price of the hotel over the last trading period was included without scale transformation in the model, while in the original dataframe it was reported in the logarithmic scale (`prop_log_historical_price`). The logarithmic scale is used to reduce the variance, but we saw that in the original shape (not logarithmic) the value of the standard deviation was in the same order of other variables of the dataframe, such as `price_usd`, therefore we decided to inverse the logarithm.

We included mean, median and standard deviation per `prop_id` of the columns related to property: `prop_starrating`, `prop_review_score`, `prop_location_score1` and `prop_location_score2`. This was made to adjust the temporal noise, given by the fact that those property values can change over time, so it can be interesting to have a constant value instead. Also, the winners of the original competition, Owen Zahng's team, included these modified columns.

5.3 Composed features

The columns with the biggest amount of missing data are the ones concerning Expedia's competitors. Woznica [11] emphasised that customers compare prices - they tend to book hotels at a higher rate if the price at Expedia is lower. That is why, in order to retain this information, we had to summarise it in two columns: `comp_rate_mode` and `comp_inv_mode`. These new columns contain the highest frequency value across rows (that can be -1, 0 or 1) among the 8 competitors columns (ignoring the missing values). New columns contained around 30% of missing data, therefore we imputed the missing values with -100 to differentiate the missing values from the rest.

We created a column with the percentage of the booked/clicked for each `prop_id`, overall. This was done to create a column that represents an "overall quality" of the property.

We included some features normalised by `srch_id` in the model, with the purpose of highlighting a specific preference of the user (e. g. that the person prefers on average hotels rated with 4.2). Furthermore, as normalisation is done per `srch_id` this should address the outliers in columns selected for normalisation. We also tried to normalise some features by `prop_id`, following the same reasoning. The features taken into account are the following: `prop_starrating, prop_location_score2, prop_historical_price, prop_review_score, prop_location_score1, price_usd, orig_destination_distance, price_position`.

We created a column `price_per_user_review` which is the `price_usd` column divided by `prop_review_score`. This was done to measure the "value for money" aspect of the property and inspired by the analysis in subsection 4.6.

Finally, we added a column called `price_position` that depicts the price position of the hotel within `srch_id`. This was inspired by price position analysis (Figure 4). Further position columns were created according to `prop_location_score1, prop_location_score2, prop_review_score` and `prop_starrating`.

5.4 Categorical variables

The data set carries a number of categorical variables, namely `site_id`, `visitor_location_country_id`, `prop_country_id` and `srch_destination_id`. Furthermore, date-parts extracted from `date_time` could also be treated as categorical variables. We tried to encode these columns using one-hot encoding, however this substantially increased the size of the dataset, thus crashing our system. Due to these constraint we decided to keep the aforementioned columns in their original format.

5.5 Sampling

To account for the highly unbalanced classes, we undersampled respectively the minority and the majority class. The undersampling algorithm we wrote splits the initial dataset in two; one dataset that contains solely rows that have a relevance score of 0 (referred to as majority class) and another dataset that contains rows with a relevance score > 0 (minority class). Subsequently, from the majority class we sampled around 50% per `srch_id` to account for the imbalanced classes. Note that the percentage depends on the desired final proportion of minority-majority classes. Finally, we merge the majority and minority class datasets together in one dataset, which is therefore always smaller than the original dataset, but with more balanced classes.

5.6 Position bias

To address search results that were unfairly penalised for their position in the search output (Figure 3) we will explore the possibility of removing rows with positions 5, 11, 17, 23.

6 Modeling

6.1 LambdaMART

Inspired by the winners of the Expedia Competition in 2013, the general model we used was LambdaMART via `xgboost` package in Python (under `XGBRanker` function). The model stems from LambdaRank, which on its turn is based on RankNet. LambdaMART differs since it makes use of boosted trees and has proven to be a successful ranking algorithm. [3]

LambdaMART is a gradient boosting forest (*MART* stands for Multiple additive regression tree) where every new tree is trained to predict the error of the previous tree, computed by the loss function (*Lambda*). The loss function has a huge influence on the quality of the results; in this project the pairwise and listwise loss functions will be investigated and tuned as parameters of the model. In the pairwise comparison the probability that the hotel x_i with score i is preferred over the hotel x_j with score j is estimated. The listwise comparison is simply the pairwise function, adjusted with NDCG weighting (see Section 2). The loss function aims at minimizing the difference between the predicted and the true preference probabilities for all the pairs.

6.2 Feature engineering evaluation

To select the most effective columns for LambdaMART training we started with all the original columns (after removing columns with $> 50\%$ missing data), inverted the logarithm in `prop_log_historical_price` and added the price rank and the average numerical values per `prop_id`. The base parameters supplied to the XGBRanker were as follows: `n_estimators` $= 110$, `learning_rate` $= 0.1$, `max_depth` $= 6$, `objective` $=$ "rank:pairwise". The initially trained model provided us with a score of 0.37187 on Kaggle.

Removing average values per ***prop_id*** To improve the model's score, further we tried removing the average, mean and standard deviation columns per `prop_id`. This resulted in an improvement of Kaggle score to 0.38165.

Adding normalised columns We noticed that some features normalised for `srch_id` increased the performance of the model. In order to select the features responsible for the improvement we first looked at the importance feature plot of the model, where we chose the 8 most important normalised features; then, we evaluated the model with and without a combination of these 8 features (e. g. with only the first 4 vs with all the first 8), on 10% of data sampled by `srch_id` and with 70-30% train-validate split. By comparing the values of NDCG@5 we selected four features: `prop_starrating`,`prop_location_score2` `prop_review_score` and `prop_location_score1`. We inserted them in the model both in their original shape and normalised; they improved the Kaggle score to 0.38292; including only the first two of the normalised columns listed above would decrease the Kaggle score to 0.38231, so we kept the initial four.

We followed the same method when normalising for `prop_id`, but since the initial exploratory experiments did not bring any further improvement, we discarded the idea. This is coherent with what was found in the previous paragraph.

Removing "unfair" positions Next, we wanted to test the hypothesis that removing "unfair" search position (as seen in Figure 3) would improve the performance of the model. Unfortunately, after training the model 5 times with 10% of sampled data according to `srch_id`, we concluded that the performance of the model was not significantly different. This could be due to the fact that validation data also carries position bias, therefore addressing it is not beneficial.

Adding book and click rates Next, we added columns that represented the fraction of times a specific `prop_id` was clicked or booked. On the training set this translated into much improved predictions, however this did not translate into an improvement of Kaggle score. This can be explained by the phenomena of data leaking as we trained the model on columns that were directly derived from the columns that we wanted to predict.

Adding competitors' columns Next, we added aggregated `comp_rate_mode` and `comp_inv_mode` columns as explained in data prepossessing section. This raised the Kaggle score to 0.38442.

Adding additional rank columns Next, we added ranking columns analogous to price rank column, but this time based on star rating, user review rating, and two location scores. After testing the model we saw a decrease in its' performance, therefore these column were not used. This could be due to the addition of too many columns. It would be beneficial to explore addition of these column separately, however due to time constraints this was not performed.

Adding price per user review Lastly, we added a column depicting price per user review. After training the model on sampled data 5 times we concluded that performance of the model was not improved.

Undersampling For our results, undersampling the majority class did not significantly contribute to the NDCG@5 score. This could be due to the fact that undersampling greatly reduces the size of the dataset, thus the model has less data to train on.

6.3 Model ensemble: Random Forest

We attempted to use a RF model that was trained on all preprocessed features [4]. The reasoning behind choosing RF is that it is known to be able to handle both categorical variables with a high cardinality and numerical variables quite well. We are aware of the fact that also for RF the best way would be to one-hot encode categorical features, but due to the cardinality of features we decided to leave these features as they are. For deploying a RF model, we used the pre-built model from `sklearn` and assessed the accuracy with cross validation to have an indication of the quality of the RF itself

To incorporate the prediction of the RF model, we composed a final score by adding (a percentage of) the prediction probability on the final score that is obtained from the LambdaMART model. To determine whether the RF model contributed, we experimented with different combinations of weights (`weight = [0, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.5]`) for the predicted probabilities of RF and LambdaMART. However, none of the combinations that included the predicted probabilities by the RF model positively influenced the final NDCG@5 score. We expected this since the cross validation scores were already low in the RF model. Therefore, we decided to exclude the RF model from our final model.

6.4 Parameter Tuning

`XGBRanker` takes 27 parameters in total [2]. Due to time constraints we limited parameter tuning to 4 parameters that directly control over/under-fitting and ranking mechanisms: `n_estimators` (number of trees that the algorithm makes the decision with - higher number of trees incorporate more information into the model, however there is an upper threshold where the accuracy gain is negligible; integer range $(0, \infty)$), `max_depth` (maximum path length between root and end leafs - smaller depth makes the model faster, however risks under-fitting; integer range $(0, \infty)$), `learning_rate` (shrinks the feature weights to make the boosting process more conservative, range: [0,1]), `objective` (determines the ranking of the elements - can take "rank:pairwise", "rank:ndcg", "rank:map").

To gain a better understanding of the parameters before running full-scale parameter tuning, we chose to evaluate the model once for every combination (630 in total) of the following parameter settings: `learning_rate` = [0.3, 0.1, 0.05, 0.025, 0.01], `max_depth`=[6, 8, 10, 12, 15, 20], `n_estimators` =[110, 150, 200, 300, 500], `objective` = ["rank:pairwise", "rank:ndcg", "rank:map"]. The

model was evaluated with 10% of data sampled on `srch_id` with 70-30% train-validate split. From the preliminary tuning results we could observe that 8 out of top 10 NDCG@5 scores had pairwise ranking. Furthermore, 9 out of 10 `max_depth` were 6. 5 out 10 `n_estimators` were 500, signifying that we might not have reached the optimal maximum value with the chosen parameters. The results of `learning_rate` in the top 10 were mixed: 4/10 were 0.1 and 4/10 were 0.05, therefore the range of `learning_rate` will not be changed for later tuning.

Based on this information we decided to restrict full-scale parameter tuning to the following parameter settings: `learning_rate` = [0.3, 0.1, 0.05, 0.025, 0.01], `max_depth` = 6, `n_estimators` =[300, 500, 700, 850, 1000], `objective` = "rank:pairwise". We chose to perform parameter tuning by sampling 10% of data (based on `srch_id`) to get 5 distinct samples, train the model on each of these samples for every combination of parameters and calculate the mean NDCG@5 (based on 70-30% train-validation split). The best performing parameters were found to be: `n_estimators` = 300, `learning_rate` = 0.05, `max_depth` = 6, `objective` = "rank:pairwise" with NDCG@5 of 0.48195.

6.5 Final model

Given the columns selected as above and the parameters found with the parameter tuning, we trained the final model with 70-30% train-validation split. The NDCG@5 score on the training set evaluation was 0.48256, while the score given by Kaggle was 0.37598. Unfortunately, our best Kaggle submission (score 0.38499) was found with the same columns, but with the following parameter setting: `n_estimators` = 300, `learning_rate` = 0.35, `max_depth` = 6, `objective` = "rank:pairwise". This can be due to random dynamics, therefore the model found via parameter tunning - with the mean NDCG@5 - was selected as the final model.

The feature importance plot is reported in Figure 6; beside the very first one (`prop_location_score`), the top five positions are occupied by features that were modified by ourselves and that can be considered a satisfying result. Normalised features have the highest scores as well as price-related ones. Competitor columns do not weight as much as others, even if in the previous analysis it was shown that they positively contribute to the performance of the model. Despite the results in EDA, also temporal features have quite low scores.

7 Discussion

The main challenge of this dataset was its size: handling 1.3 gigabytes of data is not easy from a computational point of view, because every run of the code takes some time (for instance, getting the predictions took from 2 to 5 minutes) and this changes the approach to the whole project. To speed up some experiments we used only 10% of the whole dataset, but we had to take into consideration that we were looking at queries to be ranked, so we needed to build a specific function that sampled for `srch_id`.

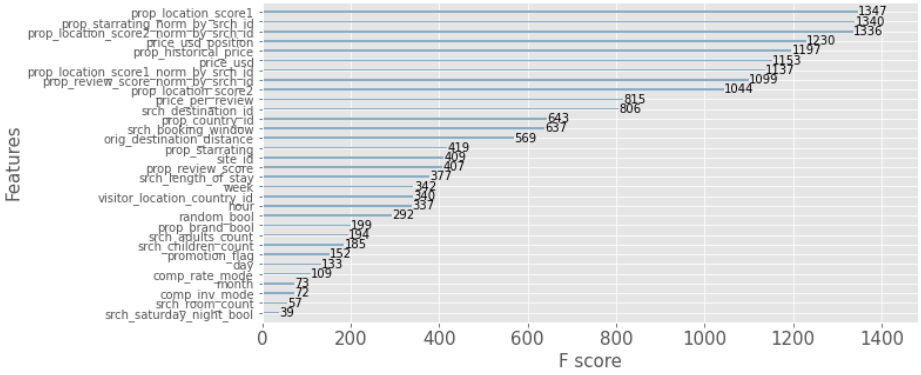

Fig. 6: Important feature plot of the final model.

The big amount of variables at hand was conceptually challenging, as we had to make a selection of what was worthy to be analysed and with which methods, especially when it came to EDA; indeed, in order to start this task, it was very useful to look at the previous work available online from the original competition. The related work acted as springboard also for the feature selection part, but it did not solve the problem of managing how to test the way and to which extent some new features were improving the predictions. In more clear words, how features were evaluated and selected was reported in Section 6.2 in chronological order, but we could not say with certainty that performing the same process in a different order would have brought same results. Experimenting with this was too demanding in terms of time and computational effort, but it could raise interesting conclusions.

Another critical point was the fact that the NDCG@5 score given by our evaluation was very different from the result given by the Kaggle submission, as we already stated when talking about the best and final model. Eventually, we decided to consider our NDCG@5 score, as it would allow us to get more robust results, because we could compute it over several simulations and get its mean and standard deviation, while the submissions on Kaggle were limited to two per day. Nevertheless, we finalized this approach only toward the end of the project, some analysis had to be repeated.

As future work, it would be interesting to experiment with the encoding of categorical variables, a topic that we touched only on the surface. Furthermore, generating artificial data points of the minority class instead of only using undersampling is an unexplored path that is left to the posterity.

References

1. Personalize expedia hotel searches - icdm 2013. https://www.kaggle.com/c/expedia-personalized-sort/overview (2013)
2. Xgboost documentation. https://xgboost.readthedocs.io (2020)
3. Burges, C.J.: From ranknet to lambdarank to lambdamart: An overview. Learning **11**(23-581), 81 (2010)
4. Cutler, A., Cutler, D.R., Stevens, J.R.: Random forests. In: Ensemble machine learning, pp. 157–175. Springer (2012)
5. Dai, B., Shen, X., Wang, J., Qu, A.: Scalable collaborative ranking for personalized prediction. Journal of the American Statistical Association pp. 1–9 (2019)
6. Liu, T.Y., et al.: Foundations and trends® in information retrieval. Foundations and Trends® in Information Retrieval **3**(3), 225–331 (2009)
7. Liu, X., Xu, B., Zhang, Y., Yan, Q., Pang, L., Li, Q., Sun, H., Wang, B.: Combination of diverse ranking models for personalized expedia hotel searches. arXiv preprint arXiv:1311.7679 (2013)
8. Shenoy, G.G., Wagle, M.A., Shaikh, A.: Kaggle competition: Expedia hotel recommendations. arXiv preprint arXiv:1703.02915 (2017)
9. Wang, J., Kalousis, A.: Personalized expedia hotel searches, - 2nd place (2013)
10. Wang, Y., Wang, L., Li, Y., He, D., Liu, T., Chen, W.: A theoretical analysis of NDCG type ranking measures. CoRR **abs/1304.6480** (2013), http://arxiv.org/abs/1304.6480
11. Woznica, A.: Personalized expedia hotel searches - data description (2013)
12. Zhang, O., Woznica, A.: Personalized expedia hotel searches, - 1st place (2013)