Data Mining Techniques Assignment 2

Conor Gouraud, Magda Karavangeli, and Jop Meijer

Vrije Universiteit Amsterdam

1 Introduction

In 2013, Expedia embarked on a significant undertaking, initiating a challenge with the goal of constructing a resilient recommender system for its users [8]. The focal point of this challenge revolved around the task of generating personalized hotel recommendations, specifically aiming to provide users with a curated list of hotels that they were most likely to book, based on their search queries [10]. To enhance the accuracy and relevance of the recommendations, the dataset employed encompassed not only the search query itself but also supplementary information such as the time, date and destination. The challenge of this competition lays in the requirement to rank hotels based on a combination of search queries and additional features, as opposed to relying solely on specific hotel attributes [9]. This multifaceted challenge pushed participants to develop innovative approaches in order to create a robust and effective ranking system for hotels.

2 Previous studies

Several research studies have been carried out on the Expedia problem set subsequent to the public Kaggle competition [6]. The top-ranking submission in this competition was developed by Owen Zhang, achieving a commendable score of 0.53984 based on the evaluation metric NDCG@38. Zhang tackled the issue of missing values by employing imputation with negative values, bounding existing numerical values, and implementing downsampling techniques for negative instances, all of which contributed to enhancing the model's performance. In addition, Zhang leveraged five distinct groups of features in his approach. These encompassed the utilization of all original features, averaged numerical features, composite features, EXP features, and an estimated position. The estimated position was derived by calculating the average of an EXP feature, considering the position of the same hotel in the preceding and subsequent searches within the same destination. To construct his models, Zhang employed Gradient Boosting Machines (GBM), creating two variants: one incorporating EXP features and the other without EXP features. Notably, Zhang's investigation revealed that the most influential predictors in his models were the position, price, and desirability of the location.

The second-best submission [4] achieved a score of 0.53839 and distinguished between user behavior and random noise explicitly. They primarily employed

LambdaMART[2], a nonlinear Learning to Rank (LTR) algorithm, for ranking purposes. For categorical features, Linear Regression or a linear LTR model like Support Vector Machines was used. Missing values were estimated based on hotel descriptions, historical user data, and competitor descriptions. Feature extraction involved estimating hotel quality and assessing the non-monotonic nature of feature utility. The authors also observed that hotel prices vary across different cities and times, leading them to normalize hotel and competitor descriptions using various indicators. To evaluate their model, they split the training data into an 80% training set and a 20% validation set.

The preceding studies have served as a source of inspiration for our investigation and approach to analyzing this dataset but first we start by analysing the given data.

3 Data Understanding

The dimensions of the dataset are as follows: In the training set, there are 4,958,347 rows and 54 columns. In the test set, there are 4,959,183 rows and 50 columns. The columns "*booking_bool*," "*click_bool*", "*gross_bookings_usd*", and "*position*" are not included in the test set.

31 columns include missing valueswhich are shown in Figure 1. The columns associated with competitor's data contain a significant portion of the missing values. Furthermore, the absence of historical user data is evident due to the presence of first-time Expedia customers among the user base. 28 column have more than 50% of missing values, with 16 having above 90%.

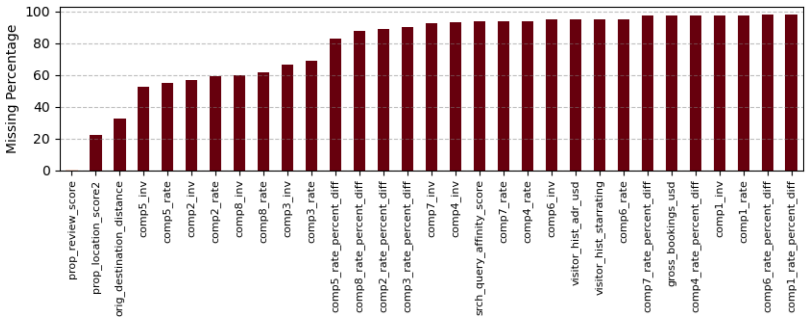

Fig. 1. Data columns and their respective percentage of missing data.

Figure 2 shows us the different clicking and booking rates based on whether properties have certain features missing or not. Initially, it seems quite visually apparent that users have a bias towards clicking and booking properties that don't have missing information. To investigate this further, chi-squared tests were performed to see if the fraction of clicks/books were significantly different. In each of the four cases, the obtained p-value was found to be far below 0.05

(the largest being $3.8\text{x}10^{-5}$) indicating that the null hypothesis is rejected and there is a significance difference between the fractions.

Fig. 2. Clicking and booking rates for whether properties have missing review scores and property location2 scores or not.

Statistics for columns with numerical data are represented by boxplots in Figure 3. We see the lower whisker of some columns being at 0, showing how frequent the entry 0 occurs. `price_usd` has the widest range, from 0$ to 19,726,328$. The possible reasons for this discrepancy are likely attributed to variations in taxes and fees across different countries, as well as the fact that the value could be specified either per night or for the entire duration of the stay.

A summary of some statistics for the columns that were harder to represent with boxplots is available in Table 1.

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

Column	Summary
`srch_id`	199795 unique values. Each value appearing between 5 and 38 times, with a mean of 24.88.
`prop_id`	129113 unique values.
`date_time`	Values range between 01-11-2012 and 30-06-2013
`click_bool`	4.47% of results were clicked
`booking_bool`	2.79% of results were booked
`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%.

Expedia presents search results using two different methods: randomised order or in an order determined by their internal algorithm. The specific type of result ordering is indicated by a Boolean value `random_bool`. Figure 4 shows the

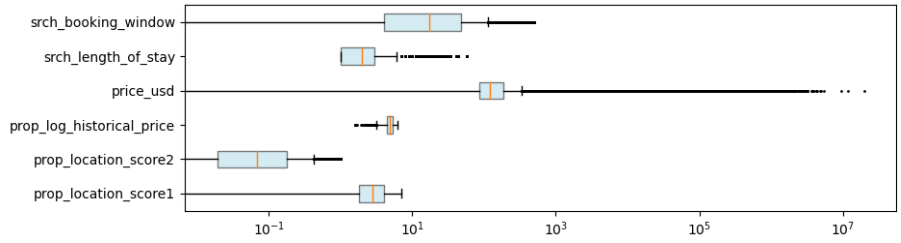

Fig. 3. Boxplots for selected numerical variables.

fraction of clicked and booked properties based on the position they appear in a search. It is evident that a notable bias exists towards properties that are positioned higher in the search results. Additionally, there is a tendency for users to be less inclined to book hotels when the search results are randomized, as opposed to when the internal algorithm is employed. There are sharp drops in booking and clicking rates at certain positions, namely 5, 11, 17, 23. This could be due to the amount of properties appearing on the search page before the customer needs to load more. Weighting properties in these positions in our model so that they are less likely to be clicked or booked could be an interesting aspect to investigate.

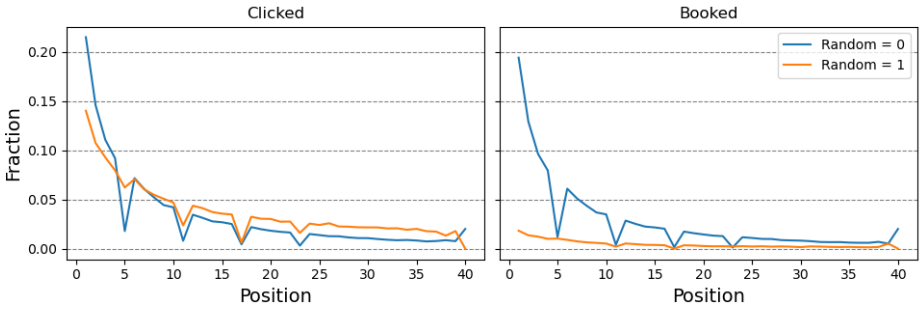

Fig. 4. Fraction of booked or clicked hotels based on their position in the search results.

To investigate clicking and booking bias towards property prices, properties were ordered by price per search ID. In Figure 5, we have the probability density of properties based on their price positioning. We see clearly that, when compared to all properties, properties with relatively lower prices get clicked and booked more often. It is important to note that, when considering all properties, the density will naturally be greater at lower values of price position as there are fewer instances of searches with a greater amount of properties appearing. This is visible when looking at the decline in density as price position increases for 'All Properties'.

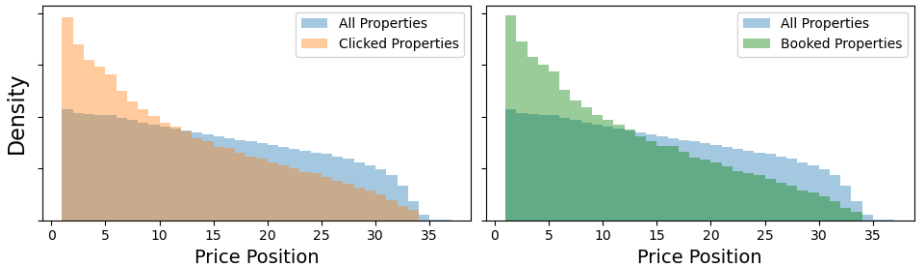

Fig. 5. Probability density function of properties based on their price position in the search results.

The fraction of properties clicked and booked based on review score is represented in Figure 6. We can see a peak at a review score of 4. Higher review scores correlate with larger prices which likely discourages users from clicking/booking such properties, explaining the dip for review scores upwards of 4. Additionally, it is worth noting that hotels without any reviews tend to have a surprisingly high likelihood of interaction, although this might be influenced by their pricing strategy. It is common for newly registered hotels on a website to offer initial discounts as a means to swiftly attract customers and, consequently, accumulate their first reviews.

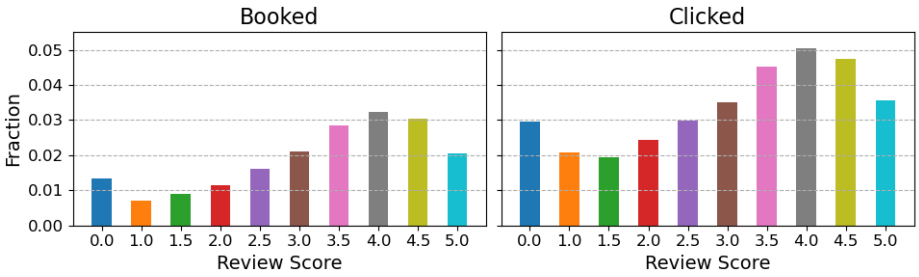

Fig. 6. Fraction of booked or clicked hotels with respect to their review score.

4 Data Preparation

4.1 Imputation

The next step in our research involves addressing the issue of missing values in the dataset through imputation methods. Missing values were imputed in two columns, `prop_review_score` and
`prop_location_score2`. The lower quartile grouped by search ID was used for

imputation in both cases, mitigating bias and preserving data distribution. Missing values in columns such as `orig_destination_distance` and `visitor_hist_adr_usd` were not imputed as there were either too many missing values or no logical reasoning could be made for imputing values.

No removal of outliers was performed in the dataset. Although the possibility of removing large values in columns such as `price_usd` was considered, it was observed that properties with a `price_usd` as high as 3,779,565 were still being booked. Therefore, the decision was made to retain these values in the analysis.

4.2 Feature Engineering

Feature engineering is an important step in the data preprocessing phase of our research, as it involves transforming and creating new features from the existing dataset to enhance the predictive power and performance of our models. Inspired by re-ordering hotels based on their price by search ID, as seen in Figure 5, columns `price_order`, `starrating_order`, `review_score_order`, `prop_location_score1_order` and `prop_location_score2_order` were added. This captures the relative position of each hotel within the dataset.

Feature interaction was also investigated by combining columns. 'Hotel quality' was added by dividing the property price by its star rating and also separately by its user review score, `hotel_quality` = `price_usd`/ `prop_starrating` and `hotel_quality_user` = `price_usd`/ `prop_review_score`. The absolute difference between star rating and review score was also added, `star_diff` = |`prop_starrating` - `prop_review_score`|, aswell as the absolute difference between the hotel price and the the mean price per night of the hotels a customer has previously purchased, `usd_diff` = |`price_usd` - `visitor_hist_adr_usd`|. The reasoning behind the added features so far is that users often exhibit a natural inclination to unconsciously rank or order hotels when conducting searches on websites like Expedia.

When training machine learning models, features with significantly different scales can make it difficult for the model to converge or find an optimal solution. This is very relevant with LambdaMART [2], which is used as the primary ranking algorithm for this assignment. To accommodate this, new normalised columns were created to ensure that features are on a similar scale, which helps the model converge faster and more reliably. Normalisation by both search ID and property ID were investigated. The columns which were normalised were: `price_usd`, `prop_starrating`, `prop_review_score`, `prop_location_score1`, `price_usd`, `price_usd`, `prop_location_score2`, `prop_log_hist_price`, `visitor_hist_adr_usd`, `hotel_quality`, `hotel_quality_user`, `star_diff`, `usd_diff`. Meaning normalisation of both original features and composed features were investigated.

The columns pertaining to Expedia's competitors exhibit the highest occurrence of missing data. According to Woznica [12], customers often compare prices and are more likely to make hotel bookings if the price on Expedia is lower. To capture this information, we condensed it into two columns: `comp_inv_mode` and `comp_rate_mode`. These new columns represent the most frequently observed

value (-1, 0, or 1) across the 8 competitor columns, disregarding missing values. Approximately 30% of the new columns contained missing data, necessitating the imputation of these values as -100 to distinguish them from the remaining data.

The month, day and hour of when a search was made was extracted from the column `date_time` and turned into new columns. The time when a search was made for, `future_month`, was created by adding the number of days in `srch_booking_window` to `date_time`.

5 Modeling and evaluation

5.1 k-nearest neighbors

First model used is the recommender system k-nearest neighbors (k-NN) which is a simple yet powerful algorithm that makes predictions based on the similarity of instances in the feature space. To make predictions using k-NN, the algorithm requires determining the value of k, which represents the number of nearest neighbors to consider. We specify parameters such as the number of neighbors (k=20) and the distance metric to use. The class supports several distance metrics but the one chosen was cosine similarity.

The fit method is used to train the nearest neighbors model on the training dataset. It takes the input data as an argument and constructs an internal representation of it. The k-NN method computes the distances between the query points and all points in the dataset, and then identifies the k nearest neighbors based on the specified distance metric. This algorithm is extremely sensitive to the provided dataset and appropriate normalization described in feature engineering, is important. Based on this similarity metric (distance), the ranking of the features is performed and we end up with predictions.

This algorithm performed poorly compared to the other algorithm employed which in our case is LambdaMart. This was visible from the training set and for that reason we decided to not proceed with it. There are several reasons for that. Some of the most important being that it can be computationally expensive when dealing with large datasets, as it requires calculating distances for each new instance. Moreover, k-NN can suffer from the "curse of dimensionality" in high-dimensional feature spaces, which can degrade its performance.

5.2 LambdaMart

As an alternative to k-NN, we employ the LambdaMART learning-to-rank model [2]. The current objective involves a fundamental task in the realm of Information Retrieval. Over the years, extensive research in this field, coupled with advancements in machine learning, has propelled Learning-to-Rank models to the forefront of ranking competitions [3]. Our particular focus has been on the prominent family of gradient boosting LTR models. Among them, LambdaMART stands out as the most renowned. For this study, we have deliberately opted

for Microsoft's LightGBM [7] implementation of gradient boosting, which offers excellent support for a LambdaRank-inspired objective.

LambdaMart combines the strengths of gradient boosting and the LambdaRank algorithm[1, 5]. It utilizes gradient boosting to create an ensemble of decision trees that collectively learn to make optimal ranking decisions. LambdaRank, specifically designed for ranking tasks, guides the training process by optimizing a pairwise loss function known as the Lambda loss.

The working mechanism of LambdaMart involves several steps. Firstly, the algorithm requires annotated training data consisting of query-document pairs with relevance labels. Feature vectors representing the characteristics of each pair are extracted. Initialization involves creating an initial ranking model. The algorithm then performs iterations of gradient boosting, constructing decision trees that minimize the Lambda loss. Each tree is integrated into the ensemble by adjusting instance weights based on the improvement achieved. The final ranking model combines the predictions of all the decision trees to assign scores or ranks to query-document pairs.

The advantages of LambdaMart include its robustness, flexibility, and state-of-the-art performance. It can handle noisy and incomplete data, accommodate various types of features, and optimize different ranking objectives.

6 Evaluation

To evaluate the performance of ranking models, the Normalized Discounted Cumulative Gain (NDCG) metric was introduced, aiming to assess the consistency of ranking for each search result [11]. This evaluation metric serves two key purposes. Firstly, it measures the extent to which the most relevant results are positioned at the top of the ranking. Secondly, it evaluates the overall relevance of all content within the ranking with respect to the user's characteristics. Equation 5 presents the NDCG formula, which fulfills these evaluation requirements. In this equation, the weights represent the relationship between the position of a result and its relevance, while the summation operation ensures that NDCG values the relevance of all content concerning the user's characteristics. It is important to note that the value of "k" in the equation 1 should be set to 5. rel_i represents the relevance score of the hotel at position i (5 - for booked, 1 - for clicked, 0 - for rest) NDCG ranges between 0 and 1, with a higher value indicating better ranking performance. The IDCG is the ideal score that can be attained, if a perfect ranking is achieved.

$$\text{NDCG}(k) = \frac{\text{DCG}_k}{\text{IDCG}_k} \in [0,1], \text{where DCG}(k) = \sum_{i=1}^{k} \frac{\text{rel}_i}{\log_2(i+1)} \tag{1}$$

6.1 Feature Evaluation

To select the most effective columns for LambdaMART training we started with all the original columns aswell as including all the engineered columns in the

previous section. The base parameters supplied to the LGBMRanker were as follows: `n_estimators` = 200, `learning_rate` = 0.1, `max_depth` = 6, `objective` = "lambdarank".

Next, we investigated the impact of adding normalised columns with respect to the search ID feature, in total there were 11. Initially, we examined the feature importance plot generated by the model and selected the eight least significant normalized features. To evaluate the influence of these selected features, we conducted experiments using a 80-20% train-validate split. We compared the model's performance with and without combinations of these eight features, including scenarios where only four or all eight features were considered. We utilized NDCG@5 values as the evaluation metric. It was found that the only normalised-by-property-ID column which negatively affected the NDCG@5 value was `visitor_hist_adr_usd`, and hence was excluded. A similar method was used to investigate the impact of the columns normalised by property ID. Out of the six that were created, only three were kept, `prop_location_score2_norm_byprop`, `price_usd_norm_byprop` and `prop_log_historical_price_byprop`. As any combination including the others negatively affected the NDCG@5 score.

To investigate the potential benefits of mitigating position bias, we conducted an experiment aimed at testing the hypothesis of improved model performance through the removal of "unfair" search positions, as depicted in Figure 4. Our approach involved training the model with and without the properties in the positions 5, 11, 17 and 23. However, our findings indicate that the performance of the model did not exhibit a statistically significant difference. Consequently, our efforts to address position bias did not yield discernible benefits in enhancing the model's performance.

We had also incorporated columns representing click and booking rates for specific property ids. While this led to improved predictions on the training set, it did not result in an enhanced Kaggle score. This can be attributed to data leaking, as the model was trained on columns derived directly from the ones we intended to predict. These observations emphasize the importance of vigilance in feature engineering and the need to carefully consider potential data leakage.

Adding the two columns `comp_inv_mode` and `comp_rate_mode` only very slightly increased the NDCG@5 score for our model, by approximately 0.002. This does not match up with conclusions made with some of the very successful models in the original Expedia Kaggle competition which claimed that utilising the competitive columns significantly increased their scores [6]. This could be an indication that further efforts towards refining the integration of competitive columns should be made in our case.

Similar conclusions could be made about the addition of the time extracted columns from the feature `date_time`. The hour, day, month and month that a search was made for had little to no effect on the NDCG@5 score and hence were excluded from our model.

In the end, after the feature evaluation process, a total of 48 features were kept in our model.

6.2 Parameter Tuning

Considering the size of the data set and consequently the computational effort required to train the model, we consider the following hyperparameters for the optimization of our model: the number of boosted trees (`n_estimators`), the learning rate (`learning_rate`), and the number of leaves for each tree (`num_leaves`). We do not perform every possible combination of parameters, as our computational resources do not allow for that, but instead optimize the number of leaves at default `n_estimators` = 100 and default `learning_rate` = 0.1. The evaluated range is [20, 30, 40, 50, 60]. The underlying intuitive assumption is that the influence of the number of leaves is independent of the parameters that are kept constant. Similarly, we vary `n_estimators` over [100, 120, 140, 160, 180, 200] and `learning_rate` over [0.1, 0.09, 0.08, 0.07, 0.06, 0.05] at default `num_leaves` = 31, with the same assumption of independence of `num_leaves`. We test the parameters by splitting the training data 80/20 into training and test data, and implementing our own NDCG@5 function.

6.3 Final Model

Our final model is the LambdaMART algorithm with the following optimized parameters: `num_leaves` = 60, `n_estimators` = 200, and `learning_rate` = 0.05. All the other parameters are the `LightGBM` default parameters. Our final model takes into account the following features :

```
site_id
visitor_location_country_id
visitor_hist_starrating
visitor_hist_adr_usd
prop_country_id
prop_id
prop_starrating
prop_review_score
prop_brand_bool
prop_location_score1
prop_location_score2
prop_log_historical_price
price_usd
promotion_flag
srch_length_of_stay
srch_booking_window
srch_children_count
srch_query_affinity_score
orig_destination_distance
random_bool
comp2_rate_percent_diff
comp3_rate_percent_diff
comp5_rate
```

comp5_rate_percent_diff
comp8_rate
comp8_rate_percent_diff
relevance
price_order
starrating_order
prop_location_score2_order
prop_location_score1_order
hotel_quality
hotel_quality_user
star_diff
usd_diff
prop_starrating_norm_bysearch
prop_review_score_norm_bysearch
prop_location_score1_norm_bysearch
prop_location_score2_norm_bysearch
prop_log_historical_price_norm_bysearch
price_usd_norm_bysearch
hotel_quality_norm
hotel_quality_user_norm
star_diff_norm
usd_diff_norm
prop_location_score2_norm_byprop
prop_log_historical_price_norm_byprop
price_usd_norm_byprop
comp_inv_most_frequent
comp_rate_most_frequent

The importance of each of these features is displayed in 7.

7 Deployment

To deploy the LambdaMART approach in a scalable way within Expedia's systems, several methods from big data engineering and infrastructure can be utilized. Expedia, being the world's largest online travel agency, handles a vast amount of data and needs to account for potential changes in data characteristics over time.

Distributed file systems can be leveraged such as Hadoop Distributed File System (HDFS) or cloud-based storage solutions to efficiently store and manage the large volume of data. These distributed file systems offer scalability and fault tolerance enabling effective storage and access to data across a cluster of machines. In order to process the large-scale data, Expedia can also employ the MapReduce programming model and framework. By distributing tasks across the cluster, Expedia can benefit from improved scalability and performance.

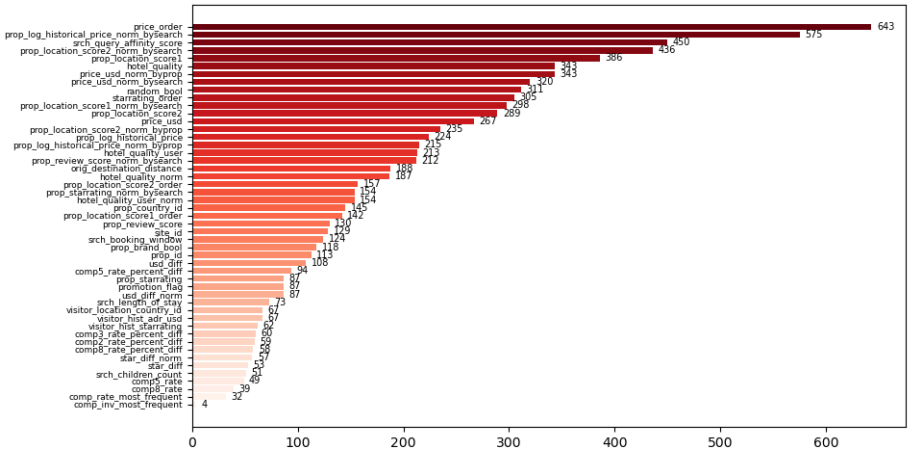

Fig. 7. Important feature plot of the final model, measured by 'gain'. The gain represents the total improvement in the model's objective function achieved by splitting on a particular feature.

Hash functions can also be used to distribute and partition data across a cluster. That way even distribution of the data required for training and inference tasks is ensured. This enables efficient parallel processing and optimal resource utilization within the cluster. While not directly applicable to LambdaMART deployment, generative AI techniques can complement the process. The use of generative AI techniques can enhance the training data and further boost the model's performance. Furthermore continuous monitoring and adaptation of the deployment strategy is recommended as the data characteristics evolve over time, ensuring scalability and the ability to capture changing user preferences and market dynamics.

Expedia can also utilize the findings from deploying the LambdaMART approach for marketing purposes in several ways. By implementing LambdaMART, Expedia can offer highly personalized hotel recommendations to users, emphasizing its ability to match them with the most relevant options based on their preferences and price competitiveness. Last but not least, the improved user experience resulting from LambdaMART can be a focal point in Expedia's marketing efforts.

8 Conclusion

What We Learned

During this competition, our primary focus was on constructing a model, which utilized LambdaMART for predicting hotels that users were most likely to book and engage with. Regarding our choice among libraries, LightGBM turned out to be beneficial. The maturity and extensive development since its release 2016

contribute to this Through this process, we gained valuable insights and lessons in handling various challenges, including missing values, feature engineering, and predicting the most relevant outcomes. The large 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.

One key takeaway from this competition was the importance of thoroughly conducting exploratory data analysis (EDA) and extensively researching relevant materials before diving into the experimentation phase. By investing time in EDA and studying applicable models, we were able to identify implementable approaches that had already been validated by others. This strategy allowed us to avoid exploring numerous models independently, ultimately helping us determine the best-fit model for the task at hand. Furthermore, the presence of a leaderboard, which ranked participants' solutions, played a pivotal role in continuously driving and motivating us to enhance and optimize our model.

Dealing with the dataset in question posed a significant challenge due to its large size. From a computational standpoint, handling such a volume of data proved to be demanding as each code run incurred considerable time, obtaining predictions took between 4 to 10 minutes. This necessitated a reconsideration of the overall project approach. To expedite certain experiments, we opted to utilize only a subset comprising 20% of the complete dataset. However, it was important to remember that our objective was to rank queries, which called for the development of a specialized function that performed sampling based on the search ID.

One major shortcoming of our work was the lack of time spent parameter tuning and feature selecting. A greater amount of time spent on this would likely have increased our Kaggle score.

As a potential direction for future research, it would be interesting to further investigate the encoding of categorical variables. Although we briefly touched upon this topic in our study, there is room for more in-depth exploration. Additionally, instead of solely using undersampling techniques, an unexplored approach worth considering is the generation of artificial data points for the minority class. This alternative method holds promise and could be a valuable avenue to explore in subsequent studies.

References

[1] C. J. C. Burges, R. Ragno, and Q. V. Le. "Learning to Rank with Non-Smooth Cost Functions." In: *Advances in Neural Information Processing Systems*. 2006.

[2] Christopher Burges. "From ranknet to lambdarank to lambdamart: An overview". In: *Learning* 11 (Jan. 2010).

[3] Olivier Chapelle and Yi Chang. "Yahoo! Learning to Rank Challenge Overview". In: *Proceedings of the Learning to Rank Challenge*. Ed. by Olivier Chapelle, Yi Chang, and Tie-Yan Liu. Vol. 14. Proceedings of Ma-

chine Learning Research. Haifa, Israel: PMLR, 25 Jun 2011, pp. 1–24. URL: https://proceedings.mlr.press/v14/chapelle11a.html.

[4] D.R. Cutler and J.R. Stevens. "Random Forests". In: *Ensemble Machine Learning*. Ed. by C. Zhang and Y.Q. Ma. New York: Springer, 2012, pp. 157–175. DOI: 10.1007/978-1-4419-9326-7_5.

[5] Jerome H. Friedman. *Greedy Function Approximation: A Gradient Boosting Machine*. Tech. rep. Also published in *Annals of Statistics*, 2001. IMS Reitz Lecture, Stanford, 1999.

[6] Kaggle. *Expedia Hotel Recommendations*. https://www.kaggle.com/c/expedia-hotel-recommendations. Online; accessed on 27th May 2023. Accessed 2023.

[7] Guolin Ke et al. "Lightgbm: A highly efficient gradient boosting decision tree". In: *Advances in neural information processing systems* 30 (2017), pp. 3146–3154.

[8] H. "Andy" Lee, Basak Denizci Guillet, and Rob Law. "An Examination of the Relationship between Online Travel Agents and Hotels: A Case Study of Choice Hotels International and Expedia.com". In: *Cornell Hospitality Quarterly* 54.1 (2013), pp. 95–107. DOI: 10.1177/1938965512454218. URL: https://doi.org/10.1177/1938965512454218.

[9] Tie-Yan Liu et al. "Learning to rank for information retrieval". In: *Foundations and Trends® in Information Retrieval* 3.3 (2009), pp. 225–331.

[10] Gourav G. Shenoy, Mangirish A. Wagle, and Anwar Shaikh. *Kaggle Competition: Expedia Hotel Recommendations*. 2017. arXiv: 1703.02915 [cs.IR].

[11] Yining Wang et al. "A theoretical analysis of NDCG type ranking measures." In: *Conference on learning theory*. PMLR. 2013, pp. 25–54.

[12] Woznica, A. *Personalized expedia hotel searches - data description (2013)*. https://www.kaggle.com/competitions/expedia-personalized-sort/overview. Online; accessed on 27th May 2023.