Learning to rank Expedia hotels

Data Mining Techniques applied to a Kaggle challenge
Assignment 2, Group 83

Robert Ioiart 2736834[1], Stergios Ntanavaras 2778274[1], and Chrysoula Pozrikidou 2781703[2]

[1] *VU University, De Boelelaan 1105, Amsterdam, 1081 HV, The Netherlands*
[2] *University of Amsterdam, Spui 21, Amsterdam, 1012 WX, The Netherlands*

1 Introduction

In the past decades, the rise of online travel agencies (OTAs) has significantly transformed the hotel booking landscape. Booking websites such as "Expedia", "TripAdvisor", and "Booking.com" have largely supplanted traditional offline travel agencies. These platforms allow consumers to search for hotel alternatives online and presently account for nearly half of worldwide travel sales [1].

For online travel agencies (OTAs), providing high-quality service is of utmost importance. When a user searches for hotels, the OTAs try to give the most relevant and accurate results that correspond to the user's preferences and requirements. By presenting search results that align closely with the user's needs, the OTA enhances the possibility of the user making a booking through their platform [2]. In response to the significance of service quality and the importance of presenting relevant hotel options to users, the "Personalize Expedia Hotel Searches" competition was conducted on Kaggle in 2013 [3]. The goal of this competition was to develop models for ranking the properties associated to user search queries such that the properties most likely to be booked are displayed first. By sponsoring this competition, Expedia tapped into a vast global pool of professional and amateur data scientists, potentially getting fresh ideas for improving their existing ranking models or at least to attract talent to the company.

2 Business Understanding

Prior to starting the development of the model, the problem needs to be described in more detail and related work investigated. As stated before, the goal is to rank search results meaning that the pool of search results associated to a user query is known, but the ordering is not. This problem can be viewed as a machine learning problem (ML), more specifically a *learning-to-rank* problem. While a classification model could be trained to predict whether a property will be clicked or booked, such models do not work very well in practice as the learning objective does not explicitly enforce an ordering. Classification learning objectives typically minimize average loss over the training set, a loss which can be small even if the relative orderings are incorrect. In order to have a good ordering, a model should be trained with an objective that explicitly includes relative ordering in the objective, since the relative rather than absolute probability of booking is the goal. Ranking problems come with certain peculiarities and challenges some of which will be explored in the modeling and evaluation section.

Despite the fact that the Kaggle competition ended in 2013, a considerable quantity of information about its contributions has been made available in the years thereafter. The winner of the competition was Owen Zhang [3], who accomplished the highest performance by employing an ensemble of gradient boosting machines and optimizing the NDCG (Normalized Discounted Cumulative Gain) metric. He made use of all dataset features, such as mean numerical features per hotel property, search id, and destination. Zhang added composite characteristics for pricing differences and integrated a hotel's position on the result page as a historical feature. Outliers were removed, missing data were imputed, and no-click records were downsampled as part of the preprocessing stages. Finally, EXP characteristics were used to convert categorical information into numerical space.

Team 'Commendo' [4] utilized a combination of models that included the LambdaMART model from RankLib, neural networks, and gradient boosting machines. Their solution received an outstanding NDCG@38 score of 0.54071, placing them unofficially first in the competition. The method entailed utilizing all numerical features and supplementing the description of each hotel property with statistics such as the mean, median, and standard deviation of numerous features. These statistics were generated over both the training and test sets to improve the stability and application of the estimations.

Another noteworthy approach that achieved the runner-up position, was that of Wang [3]. He investigated numerous modeling approaches and discovered that LambdaMART was the most successful. His approach integrated the concept that customers prefer hotels with comprehensive information in first impressions. To fix this, any missing values in search impressions were filled with worst-case values. Additionally, because travel costs change throughout the year and various locations often have varying price ranges, characteristics were standardized depending on criteria such as the property, search IDs, destination, country, and time of year.

The team 'Binghsu & MLRush & Brick-Mover' [5], who secured third place in the competition, produced a solution that involves training different models, including support vector machines (SVM), random forests, factorization machines, and neural networks. A listwise ensemble approach was used to merge these models.

Finally, the team composed of Xinxing Jiang et al. [6], employed collaborative filtering and multiple conventional classification algorithms to directly predict clicking behavior. Despite conducting diligent feature selection, the team only obtained mediocre classification performance (F1 = 0.38). This outcome emphasizes the limitations of the classifier-based approach for addressing the given problem.

3 Data Understanding

3.1 Dataset

The dataset utilized in the Kaggle competition of 2013 focuses on hotel searches conducted on the Expedia website. The data collection contained 8 months of searches, from November 1st, 2012, to June 30th, 2013. Its objective is to provide insights into the preferences and behaviors of users when they search for and book hotels. The collection comprises various attributes that capture different aspects of each hotel search. These attributes encompass some limited user-related information, some search criteria, and the list of properties associated with each search, along with some features associated with the property. In addition to the predictive information mentioned before, the dataset includes dependent information such as whether a property was clicked or booked, and which position it had in the list displayed to the user.

3.2 Exploratory Data Analysis

Records	
Total Records	4958347
Total Searches	199795
Records Clicked(%)	4.474
Records Booked(%)	2.791

Searches	
Min. Results	5
Avg. Results	24.82
Max. Results	38
Avg. Clicks	1.11
Avg. Bookings	0.69

Attributes	
Properties	129113
Destinations	18127
Countries	172
Sites	34

Table 1: Training Data General Statistics.

Exploratory data analysis was undertaken to gain full knowledge of the dataset's structure and quality, as well as to uncover features that potentially predict booking. Table 1 displays numerous key statistics for the training data.

Specifically, the dataset comprises a total of 4,958,347 entries that document hotel search interactions. Out of these, there are 199,795 unique searches conducted by users. The proportion of records clicked and booked is also supplied, with users clicking on 4.4748% of the records and booking 2.7910% of them. Furthermore, the data reveals a significant variation in the number of search results across queries. On average, users obtained 24.82 search results per query, with a minimum of 5 and a maximum of 38 results. Moreover, the data highlights that users tend to click on more than one property throughout their search process, underscoring the exploratory nature of their hotel decision.

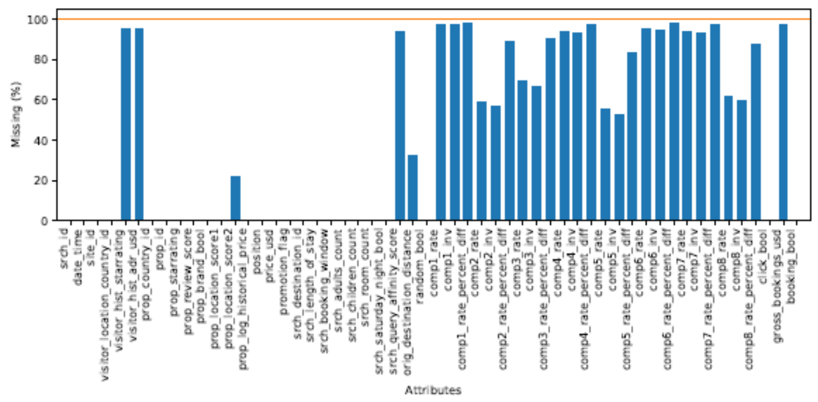


Figure 1: Proportion of Missing Values of each attribute.

Figure 2 displays the data that includes a particular attribute called `'random_bool'`, which provides information about the likelihood of users clicking on hotels and whether they would proceed to book them. Upon additional exploration, it was discovered that the dataset contains a considerable number of missing values across its attributes. The proportion of missing values for each attribute is depicted in Figure 1.

In order to get a clear sense of the structure of the data, we explored how different identifiers relate to each other and what are the one-to-one, one-to-many or many-to-many relationships. We made the following observations about the different present ids.

1. **srch_id**: All records with the same srch_id constitute the search results and search-related information is replicated for each search result. In other words, the data scheme is not normalized. There are approximately 200k searches in each of the train and test datasets and each search will include multiple prop_id but all prop_ids will be associated with the same srch_destination_id.

2. **site_id**: Multiple countries use the same site_id and all records associated with a srch_id share the same site_id. There are 34 sites in total.

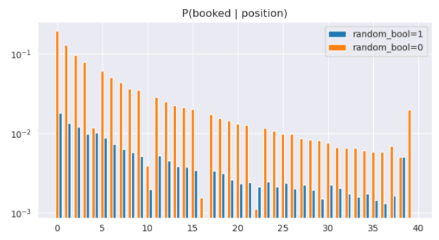
3. **visitor_location_country_id**: There are 218 unique customer country location IDs in the dataset, while there is a strong but not perfect correlation between visitor_location_country_id and site_id.

4. **prop_country_id**: There are again 218 unique hotel country location IDs. Some srch_destination_ids are associated with more than one country (196 out of 23715).

5. **prop_id**: The hotel prop_id is unique per srch_id. A prop_id has a one-to-many relationship with the srch_destination_id and a one-to-one relationship with the prop_country_id. There are approximately 130k prop_ids in each of the data sets while there are about 7500 prop_ids exclusively in one of the two data sets (e.g. 7500 values occur in the test but not in the training set and vice-versa).

6. **srch_destination_id**: Each srch_id refers to a single srch_destination_id. This means that all prop_ids associated with a search correspond to the same srch_destination_id. This attribute can be viewed as the id of a set of properties, the same property can be in multiple such sets. There are about 18k srch_destination_ids in each of the sets, while 5600, or almost 30% of them belong exclusively to one set but not to the other.

During the exploration phase, several initial observations and patterns occurred. For instance, a noteworthy pattern that emerged was a class imbalance in the target variables `‘click_bool’` and `‘booking_bool’`. The majority of the search outcomes did not result in clicks or bookings, indicating a large class imbalance. To provide solid predictive performance, this mismatch may necessitate additional attention throughout the modeling process. Given that we are interested in ranking rather than classification, and since only 4.474% of the results were clicked, downsampling the data prior to training is likely to result in data sparsity and, as a result, overfitting.



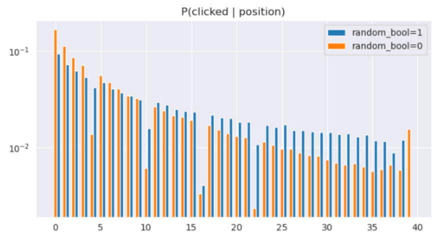


Figure 2: Booked and Clicked distribution over Position.

The Figure 3 also displays another notable aspect, namely the importance of features in a certain model or algorithm, as evaluated by their influence on both decision splits and information gains inside the model. The feature importance values represent how much each feature contributes to the model's overall predictive ability and performance. Precisely, each bar's length corresponds to the magnitude of the feature's relevance. Longer bars represent attributes that have a greater influence on decision-making and contribute more to the model's overall accuracy and performance.

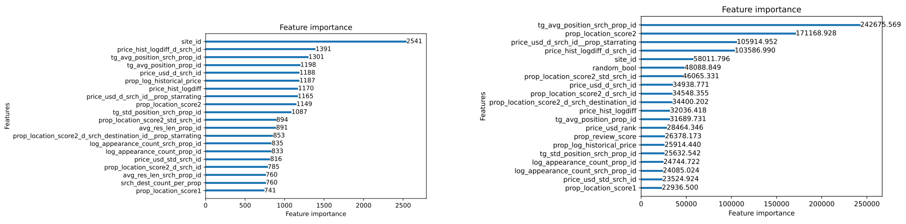


Figure 3: Feature Importance in Splits and Gains.

Finally, Figure 4 represents the Shap values for each attribute in a visually intuitive manner. We can find which features have the most important effect on the model's output and obtain insights into their relationships with the expected outcome by evaluating them. Positive and negative numbers indicate whether an attribute contributes beneficially or adversely to the prediction.

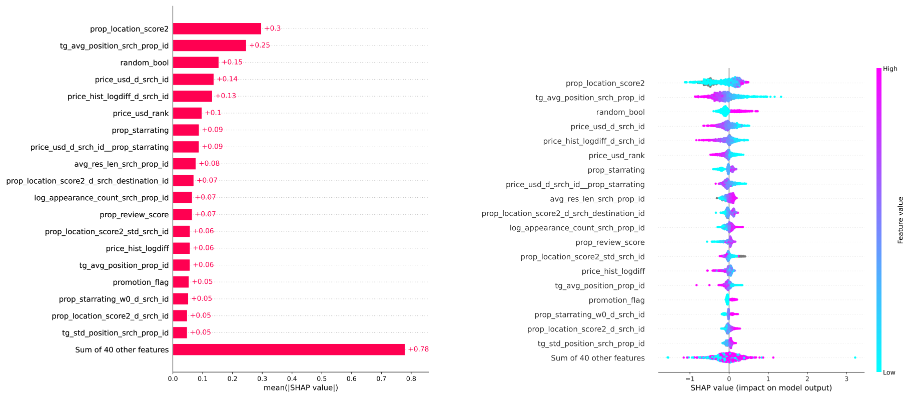


Figure 4: Shap Values.

4 Data Preparation

4.1 Missing Values Imputation

During the preprocessing step of our data, we took advantage of LightGBM's LambdaRank ability to handle null values effectively. For example, we replaced every entry with the feature '`price_usd`' set to zero with a null value. The rationale for this step is grounded in the nature of the data: a price of zero is implausible and more likely indicates that the price is unknown. With the LambdaRank algorithm, we can allow for such unknowns directly, putting these instances into a separate bin during the model's binning process. This approach not only preserves the integrity of the data but also avoids interference when calculating aggregate statistics, as a value of zero would.

Furthermore, the ability to accommodate missing data without resorting to imputation is a significant advantage of using the LambdaRank model. Imputation, the practice of filling in missing data with estimated values, inevitably introduces a degree of artificiality into the dataset, and may not always accurately reflect reality. In a scenario like ours, where several features have a significant percentage of data missing, any imputation can heavily skew the feature's effectiveness. Thus, choosing a model that naturally accommodates null values, such as LambdaRank, is both a safer and more accurate approach to handling missing data.

Following this method, even with features having more than 90 percent missing values, we recognize the potential importance these variables may still hold. Despite the high proportion of missing data, these features can provide valuable insights contributing to the robustness of the model and the nuanced understanding of the patterns within the data.

4.2 Feature Creation

The feature concerning the date and time of search initiation was first transformed into an appropriate DateTime datatype. Subsequently, a novel attribute was constructed, which encodes the mid-stay month, week, and day of the desired stay. Specifically, the new feature was created based on the following formula.

$$mid\_stay = date\_time + booking\_window + \frac{srh\_length\_of\_stay}{2} \tag{1}$$

This feature aims to aid understand seasonality patterns by providing insight into demand patterns during the actual stay period, which could be different from the search date or the start of the stay. We observed the patterns exposed by the booking date weeks and noticed that both in the train and test set, the mid-stay weeks show that the summer months are missing, indicating lower demand during these months. The original DateTime column was discarded since it may not provide as much interpretative value.

We additionally processed the property star rating and review score features with the goal of maximizing their relevance. We created a new feature that encapsulates the average property star rating per search destination and specific property star rating. This relative measure positions each hotel within its local context, providing a benchmark that informs us about a hotel's competitive standing within a specific destination. In a similar fashion, we have created a novel feature encoding the average property review score per search destination which informs us about a property's relative standing, in terms of customer satisfaction, within a particular location.

A secondary feature was developed, showcasing the disparity between a customer's usual hotel star rating and that of the current property, giving insight into any deviation in their hotel quality preference. We've also developed two features: the average property star rating and the average review score per search ID, offering a comprehensive view of the property quality and customer satisfaction in each search. Finally, a feature considering both the average review score per destination ID and property star rating has been included, enhancing our understanding of a hotel's appeal in a specific location and potentially boosting model prediction accuracy.

Focusing on both property location scores which encapsulate the attractiveness of a hotel's locale, we created the same aggregated features using each of the two available scores respectively. Firstly, we introduced an attribute representing the average property location score per search destination. By aggregating location scores for all properties within a specific search destination, we gain insight into the overall desirability of locations within that destination.

Several features emphasizing property pricing were added to the dataset. We calculated the log-scaled price difference between a hotel's current and historical price, where a negative result indicates a price increase and a positive one signals a more favorable deal. This feature is further aggregated with the search ID to depict deal quality relative to other properties in the same search. We also introduced a feature utilizing the median price over the search ID, serving as a robust measure of the central tendency to contextualize relative affordability during a search. Finally, we introduced a feature that considers the median price and property star rating over the search ID, providing a nuanced perspective on hotel cost and quality.

The dataset has been additionally supplemented by a feature that encodes the average search query affinity score over the search ID. As the logarithmic probabilities are negative, this feature offers an inverse perspective on the hotel's popularity in terms of online search interactions. This inclusion not only captures the hotel's online visibility and attractiveness but also provides a form of implicit user feedback, reflecting user interest during the browsing stage, prior to the booking process. When aggregated over the search ID, it reveals the average click-through propensity for hotels within a particular search, thus offering an additional measure of the hotel's appeal.

We additionally included a feature encoding if the country ID of the hotel matches the country ID of the customer's location, in order to provide information about the geographical preference of customers, indicating whether they are more inclined towards domestic or international travel, potentially shedding light on the travel patterns of different users

We created a novel feature related to competitors' pricing and supply advantages over Expedia for specific hotels, denoting the proportion of competitors offering better pricing and availability. A positive value implies a majority of competitors have superior offerings, while a negative value suggests the opposite. This single, succinct feature eliminates the need to evaluate 18 separate features. We also removed the feature indicating the absolute percentage price difference between Expedia and competitors, deeming its essential information as already covered in the new features.

Ultimately, we engineered a feature incorporated that includes the average hotel position over the search destination ID. This feature encapsulates the average rank of a hotel within the search results for a specific destination. It reflects how prominently a hotel is displayed in the search results. As higher-ranked hotels generally attract more attention and are more likely to be clicked on and booked, this feature provides insight into the visibility and perceived attractiveness of each hotel within a particular destination.

On a final note, the new features were normalized to scale them to a standard range. While our LightGBM model, specifically the LambdaRank algorithm, is generally not sensitive to the scale of input features because it uses decision trees to rank items. Decision trees split on individual features and the scale of the feature doesn't impact the quality of the split[7]. Nevertheless, this process can still be beneficial. Scaling may help in handling potential outliers, improve computational efficiency, and make feature importance comparisons more interpretable. However, it's important to note that the main goal of scaling in this context is not to prevent any single attribute from dominating the model due to its scale.

5 Modeling and Evaluation

5.1 Splitting the dataset

In order to develop and validate the hotel recommendation model, we divided the training dataset into data used for training and data used for validation based on the search ID feature. This approach ensured that each search instance, with its unique combination of properties and user preferences, is wholly contained within either the training or the validation set, avoiding potential leakage of information between the two. To facilitate a fair distribution and to maintain the random nature of the division, the search ID values were randomly shuffled prior to the split. Subsequently, 90% of the data was allocated to the training set, which serves as the primary resource for building the model. The remaining 10% was reserved for the validation set, which is utilized to evaluate the performance of the model on unseen data and to fine-tune its parameters.

5.2 Model selection

5.2.1 LambdaRank

As our main model, we chose LambdaRank [8]. It is a specialized model that's particularly effective for handling information retrieval and ranking problems. It builds upon traditional ranking models by incorporating the use of gradients in the cost function, specifically focusing on the changes that reordering pairs of items would have on the overall system performance. Essentially, LambdaRank provides an optimized solution for learning to rank by using a pairwise loss function that exploits the unique characteristics of ranking metrics and directly optimizes the performance measure that matters for ranking tasks, such as Normalized Discounted Cumulative Gain at k (NDCG@k) or Precision at K (P@K).

In our case, we optimize the ranking for the NDCG metric.

$$NDCG_k = \frac{DCG_k}{IDCG_k} \in [0,1], whereDCG_k = \sum_{i=1}^{k} \frac{\mathrm{rel}_i}{\log_2(i+1)} \qquad (2)$$

The relevance grades of hotels for each user query were assigned as [5, 1, 0] for the case of a booking occurring, a click observed and none of the two present, respectfully.

One of the key strengths of LambdaRank lies in its ability to effectively capture and quantify the relative importance between items. By considering the pairwise interactions and deriving the lambda, or change, it ensures that the model is sensitive to the ordering of items and emphasizes getting the top items correct. This makes it especially beneficial for recommendation systems where the correct ranking of items is crucial. Additionally, it is consistent with the objectives of most information retrieval systems, making it a suitable choice for tasks where the goal is to return the most relevant items at the top of the search results. By optimizing the ranking directly, LambdaRank tends to deliver better and more precise results in comparison to other ranking algorithms. The chosen set of hyperparameters for the LambdaRank model implemented using the LightGBM framework was determined based on rigorous experimentation, which ultimately yielded the best performance for the hotel recommendation task at hand.

5.2.2 Point-wise model

In the exploration of different models for the hotel recommendation task, we also experimented with the "HistGradientBoostingClassifier" from the sklearn's ensemble module. It is a method that uses a gradient-boosting framework, a point-wise approach to ranking known for handling categorical features effectively and being robust to different types of data. However, despite its proven capabilities, the model did not outperform the LambdaRank model in this particular task.

It focuses primarily on classification problems, seeking to minimize misclassifications in the training data. In contrast, LambdaRank is a learning-to-rank model, which explicitly aims to optimize the ordering of items as previously mentioned. In the context of a hotel recommendation system, where the goal is to rank hotels based on their relevance or appeal to the user, LambdaRanks's ranking approach is more aligned with our objective than the classification approach of 'HistGradientBoostingClassifier'.

Furthermore, the LambdaRank model leverages group information based on the search ID, enabling it to consider the relative relevance of hotels within the same search query. This group-wise ranking capability is something that the "HistGradientBoostingClassifier" lacks, as it treats each instance independently.

5.2.3 K-Nearest Neighbors

Finally, we experimented with K-Nearest Neighbors (KNN), a popular collaborative filtering approach that generates recommendations by leveraging the preferences and actions of similar users [9]. It calculates the distance or similarity between users in a multidimensional space, where each dimension represents a different feature or preference essential to the recommendation task. In the context of hotel searches, for example, these features might include criteria such as hotel location, facilities, ratings, or previous booking history.

To measure user similarity, we utilized the Pearson correlation coefficient, a frequently used statistic in collaborative filtering. In our example, the Pearson correlation coefficient measures the linear relationship between two variables: user preferences or behaviors. It ranges between -1 and +1, where a value of -1 signifies a strong negative correlation, +1 indicates a strong positive correlation, and 0 implies no correlation.

To compute the Pearson correlation coefficient, we first normalized the relevant variables. This step guarantees that the similarity computation is fair. Following this, we applied the formula for Pearson correlation.

After obtaining the correlation values between users, we selected the K users with the highest correlation scores as the nearest neighbors. The parameter "K" is critical to KNN's efficacy since it sets the number of nearest neighbors to consider. It is critical to select an acceptable value for "K" in order to strike a balance between accuracy and individuality.

5.3 Model configuration

In order to obtain the optimal model configuration to maximize performance we experimented with different hyperparameter settings which we aim to discuss in this section.

We set the hyperparameter for histogram bin construction to 400,000 samples, which can offer more precise data distribution estimates to potentially boost model performance. The computation and memory usage increases were manageable. The model was designed to fit 10,000 boosted trees, surpassing the default value of 100, to enhance performance without overfitting. The learning rate was left at the default of 0.1 to balance training speed and performance. We observed that lower rates slowed training without notable improvements, while higher rates bypassed optimal solutions, oscillating between suboptimal values.

The label gain was set to [0, 1, 5] in order to assign more weight to instances that led to a booking (5) over instances that just led to a click (1). These gains are used to calculate the NDCG value so that the model is optimized with the same objective as the competition metric. This a distinct advantage over cross-entropy loss based classification models. This also addresses the problem that often clicks are meaningless since they can be the result of a missclick or be biased from the position of the proposed property. On the other hand, booking instances are more reliable because they consist of a clear indication of successful recommendations.

We implemented L2 regularization, or Ridge Regression, to prevent overfitting by adding a penalty term to the loss function proportional to the squared model weights, effectively simplifying the model. This technique, using a lambda value of 0.01, balances bias and variance for improved performance on unseen data while minimizing overfitting. We additionally employed Gradient-based One-Side Sampling and Exclusive Feature Bundling, limiting the maximum tree depth to six to control model complexity. We utilized 24 threads for enhanced computation speed through parallel execution and introduced early stopping after 300 consecutive non-improvement rounds to prevent overfitting. Finally, to ensure reproducible results, we set seeds and eliminated non-deterministic behavior sources.

5.4 Results

As a data mining project is iterative, ad-hoc, but increasingly structured, experiments have been performed during the entire duration of this challenge. Most of the experiments revolved around refining and selecting features to include in the model. Much of the feature selection has been done manually, by trial and error. While using a systematic feature selection technique would have been potentially better, the computational costs of such automated techniques are significant if not prohibitive for such a short time frame. Table 2 show contains the NDCG@5 scores for the different train, validation, test, and prediction (leaderboard) sets. Two models are included namely the LambdaRank pair-wise model and the point-wise model trained on the final feature set. The **VT** suffix stands for the model being evaluated on a validation and local test set. The **V** suffix means that the model had no local test set, only a validation set (the test set was included in the training set), and finally the **F** suffix means that the model was trained on the entire training set, with no validation set for early stopping and no test set, instead relying on an estimated number of steps from the previous two models, which incorporated early stopping on a validation set. The main goal with these variants is to fully utilize all training data, as overfitting is a problem and the model's performance would increase with more training data. While the initial point-wise model performed poorly, successive feature engineering steps allowed for a decent performance, exceeding a score of 0.4.

Model	Training	Validation	Test	Leaderboard (public test set)
Ranker VT: Base features	0.4325	0.3804	-	-
Ranker VT: Base + Statistical features	0.44964	0.39746	-	-
Ranker VT: Base + Statistical + Position feature	0.48043	0.41409	0.40948	-
Ranker V: Base + Statistical + Position feature	0.48890	0.41387	-	-
Ranker F: Base + Statistical + Position feature	0.49617	-	-	0.41409
Pointwise VT: Base + Statistical + Position feature	0.46546	0.40744	0.40317	-

Table 2: Model performances

5.5 Discussion

Feature engineering was the most challenging part of this assignment but also the one that led to model performance improvements. There were many more avenues that could not be investigated due to time constraints. Several features were not included in the model as they would require significant effort to engineer into something informative. The orig_desination_distance and the visitor_location_country_id were completely discarded. One could argue that users from different countries have different preferences, but given the relatively limited amount of data available, other features proved to be more informative.

Another important aspect of the chosen model is the fact that it handles NAs naturally as mentioned in Section 4.1. Imputing missing values has a large time-wasting potential while it alters the data's true distribution. Having a model treat missing values as a distinct category is a big advantage when it comes to using informative features with many missing values as was the case for this challenge, where the historical price and historical star rating had a noticeable effect on model performance.

Additionally, NAs had another surprising benefit. One of the main factors contributing to exceeding 0.41 NDGC@5 score on the test set was the fact that estimated position values from the training set have been removed and set to NA on purpose, to match the absence of this feature on the test set. There is little meaning in having a complete feature in the training set, but partial in the test set. Removing some estimated positions from the training set forces the model to use other attributes more effectively when this informative feature is not present.

5.6 What we have learned

The evaluation metric NDCG@5 presented considerable challenges for our team. We initially used the *sklearn.metrics* implementation, but after thorough testing we concluded that it was completely out-of-line with the provided definition [1]. Thus, we implemented our own function and managed to get perfect alignment on the validation set with LightGBM's reported values. It is important to note that LightGBMs implementation uses the natural logarithm instead of base 2.

Furthermore, another learning point was about the power and flexibility of LambdaRank in general and LightGBM specifically. This library supports multiple objectives and the implementation is multi-threaded allowing for some local scalability. We learned to appreciate the advantage of a model being able to deal with NAs. Not only does it lead to less work as it avoids the problem of data imputation, but as mentioned in Section 5.5 it can even improve performance.

Finally, each of us initially looked at the problem independently and then discussed our findings, leading to more questions. Thus, a concept that was reinforced, was the iterative nature of data mining and machine learning.

[1] `https://en.wikipedia.org/wiki/Discounted_cumulative_gain`

6 Deployment

In deploying the LambdaRank model on Expedia's systems in a scalable way, one of the main considerations is dealing with the vast volume of data available and the continuously changing nature of this data. The solution lies in leveraging big data engineering methodologies and infrastructure that allow for efficient data processing and model updating.

The large volume of data can be handled using a distributed file system like the Hadoop Distributed File System (HDFS) [10], which can store and retrieve large volumes of data quickly and reliably. Data can be partitioned across multiple machines in a computer cluster using hash functions, ensuring efficient data access. Expedia could also utilize Big Data clustering, to allow for handling of large-scale data. This way the computational load will be reduced by grouping similar data, which can then be processed in parallel. In our case the clusters could be initiated by grouping similar hotel searches, making it easier to predict their rankings. The initial clusters will be iteratively refined until no further improvement can be made.

In order to enhance the clustering process's speed we can reduce the computations required at each iteration. Approximate nearest-neighbor techniques can be leveraged for this purpose [11]. These methods involve approximations to find the nearest data points or clusters instead of calculating exact distances, resulting in significantly less computational load. Despite the approximations, these techniques can still provide sufficiently accurate results for many use cases, including ours.

The process of clustering could also significantly benefit from parallelization, which involves dividing the workload across multiple computing resources to simultaneously process different portions of data. This involves dividing the data into smaller partitions, for example, based on attributes of hotel searches, distributed across computing resources for simultaneous processing to avoid bottlenecks. After partitioning, local clustering is performed independently on each partition. Post-local clustering, a global clustering step combines and re-clusters all local clusters to achieve a unified result. Lastly, the local clusters are refined based on the global clustering results for improved quality and consistency.

By integrating these big data engineering principles, LambdaRank can be deployed effectively within Expedia's systems, ensuring it can handle large, dynamic data in a scalable, efficient manner.

References

[1] Shalini Talwar et al. "Why do people purchase from online travel agencies (OTAs)? A consumption values perspective". In: *International Journal of Hospitality Management* 88 (2020), p. 102534. ISSN: 0278-4319. DOI: https://doi.org/10.1016/j.ijhm.2020.102534. URL: https://www.sciencedirect.com/science/article/pii/S0278431920300864.

[2] Anindya Ghose, Panagiotis G Ipeirotis, and Beibei Li. "Examining the impact of ranking on consumer behavior and search engine revenue". In: *Management Science* 60.7 (2014), pp. 1632–1654.

[3] Kaggle. URL: https://www.kaggle.com/c/expedia-personalized-sort/data (visited on 05/10/2023).

[4] David Kofoed Wind. "Concepts in predictive machine learning". In: *Maters Thesis* (2014).

[5] Xudong Liu et al. "Combination of diverse ranking models for personalized expedia hotel searches". In: *arXiv preprint arXiv:1311.7679* (2013).

[6] Xinxing Jiang, Yao Xiao, and Shunji Li. *Personalized Expedia Hotel Searches*. 2013.

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Second Edition. Springer Science & Business Media, 2009.

[8] Christopher Burges et al. "Learning to Rank using Gradient Descent". In: Jan. 2005, pp. 89–96. DOI: 10.1145/1102351.1102363.

[9] Xiaoyuan Su and Taghi M Khoshgoftaar. "A survey of collaborative filtering techniques". In: *Advances in artificial intelligence* 2009 (2009).

[10] Konstantin Shvachko et al. "The Hadoop Distributed File System". In: *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE. 2010, pp. 1–10.

[11] Sunil Arya et al. "An optimal algorithm for approximate nearest neighbor searching fixed dimensions". In: *Journal of the ACM (JACM)* 45.6 (1998), pp. 891–923.