Data Mining Techniques Assignment 2: A Real Life Competition

Group Number: 166
Lais Wehbi (lwe248)
Tuan Do (2753576)
Darius Barsony (2754003)

Vrije Universitet Amsterdam

Abstract. This report describes our approach and results for the data mining assignment 2, which involves predicting what hotel properties a user is most likely to book based on a large dataset from Expedia. We applied various data mining techniques, such as ... (briefly mention the main techniques you used), and evaluated our performance using the NDCG@5 metric. Our best model achieved a score of ... (report your best score) on the Kaggle competition. We also discuss how our approach can be deployed in a scalable way and what we learned from this assignment.

1 Introduction and Business Understanding

In this section, we introduce the problem and the motivation for this assignment. We also provide some background information on data mining and recommender systems, and review some related work from other people who have tried to solve similar problems.

The problem of predicting what hotel a user is most likely to book is an important and challenging one for the online travel industry. Hotels are complex entities that involve multiple attributes, such as location, price, quality, amenities, etc. Users have different preferences and needs for their trips, which may depend on various factors, such as their destination, budget, travel purpose, etc. Moreover, users may not have enough information or time to compare and evaluate all the available options in a large and dynamic market. Therefore, providing users with personalized and relevant recommendations of hotels can enhance their satisfaction and loyalty, as well as increase the conversion rate and revenue for the online travel agency [13]. One way to approach this problem is to use recommender systems which are a subclass of information filtering systems that provide suggestions for items that are most pertinent to a particular user [8]. They are a subset of machine learning that uses data to help users find products and content. A recommendation engine helps to address the challenge of information overload in the e-commerce space [1]. There are six types of recommender systems which work primarily in the Media and Entertainment industry: Collaborative Recommender system, Content-based recommender system, Demographic based recommender system, Utility based recommender system, Knowledge based recommender system and Hybrid recommender system [1]. Recommender systems usually make use of either or both collaborative filtering and content-based filtering (also known as the personality-based approach) [11], as well as other systems such as knowledge-based systems. Collaborative filtering approaches build a model from a user's past behavior (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users. This model is then used to predict items (or ratings for items) that the user may have an interest in. Content-based filtering approaches utilize a series of discrete, pre-tagged characteristics of an item in order to recommend additional items with similar properties [2].

One prominent approach to solve the hotel prediction problem is to use learning to rank (LTR) methods, which are machine learning techniques that optimize the ranking of items according to some relevance criteria [6]. LTR methods can be divided into three categories: pointwise, pairwise and listwise. Pointwise methods treat each item as an independent instance and assign a score or label to it based on its relevance. Pairwise methods compare pairs of items and try to minimize the number of inversions in the ranking. Listwise methods consider the whole list of items and optimize a global metric such as NDCG (Normalized Discounted Cumulative Gain), which is also the evaluation metric for this competition.

Some examples of LTR methods that have been applied to this problem are:

1. LambdaMART [4]: a pairwise method that combines gradient boosting trees with a novel loss function that directly optimizes NDCG. LambdaMART was the winning algorithm of the original Expedia Personalized Sort competition on Kaggle.
2. RankSVM [9]: a pairwise method that uses support vector machines to learn a linear ranking function that minimizes the number of swapped pairs. RankSVM was used by one of the top teams in the Expedia competition.
3. ListNet [3]: a listwise method that uses neural networks to learn a probability distribution over permutations of items based on their relevance scores. ListNet was also used by one of the top teams in the Expedia competition.

Given these examples of the winning approaches, we observe some of the most prominent predictors to be:

- The user's search query features, such as destination id, length of stay, number of adults and children, booking window, etc. These features represent the user's trip preferences and requirements.
- The hotel property features, such as star rating, review score, location score, price, promotion flag, etc. These features characterize the quality and appeal of each hotel.
- The competitive data features, such as the rate and availability of other online travel agencies (OTAs) for each hotel. These features indicate Expedia's competitive advantage and the user's likelihood of choosing Expedia over its rivals.

- The user's historical behavior features, such as their previous star rating and price per night for booked hotels. These features reflect the user's taste and budget for hotels.
- The user's interaction with the search results features, such as their click and booking behavior for each hotel. These features provide direct feedback on the user's interest and satisfaction with each hotel.

2 Data Understanding

To gain a thorough understanding of the structure and quality of the dataset, an exploratory data analysis was conducted, mostly on the training set. The dataset was provided as a collection of time-stamp search result, in which each line represents a combination of a query and a suggested hotel property, and whether this suggestion led to a click or booking. Both the training set and test set contain a roughly 5 million rows each, with 50 features on both sets and 4 additional features exclusively to the training set. We can group these features into a few categories:

- Query features, including search ID, date and time of search, site ID, destination ID, length of stay, room count, adult and children count;
- Visitor features, including their location country ID, past mean rating and past mean price paid per night;
- Hotel features, including hotel ID, average rating, review score, location desirability score, whether it is part of a hotel chain;
- Query-hotel features, including listed price, whether it is promoted, distance between visitor and hotel, log of probability of the hotel to be clicked;
- Competitor features, including relative comparison of price and room availability during the search period.

The four exclusive attributes on the training set are:

- Position: position of the hotel listed within the search result
- Click and booking: boolean values indicating whether the hotel, shown at a specific position, has been clicked or booked
- Gross booking: the total cost of made transaction in US dollars

Out of these four features, the click and booking value are the most important, and later would be used in the "target" composite feature, which indicates whether the recommendation of a hotel is helpful (in terms of customer's viewing and booking) and is integral in the training models.

In total, 199795 unique searches have been collected in the train set, spread over 4958347 records (rows) for an 8-month period between November 2012 and June 2013. The dataset includes 129113 unique properties at over 18127 destinations in 172 countries. Out of that, 4.47% of record has been clicked, and 2.8% has been booked. For every unique search, a hotel click has been made at least once, but roughly one-third of the search (30.8%) did not end up making a hotel reservation. The number of hotel shown to a search varies between 5 to

38, with an average result of 24.8 properties. We also found that, while visitors are located in 210 different countries and areas, they search for hotels in only 34 Expedia regional sites.

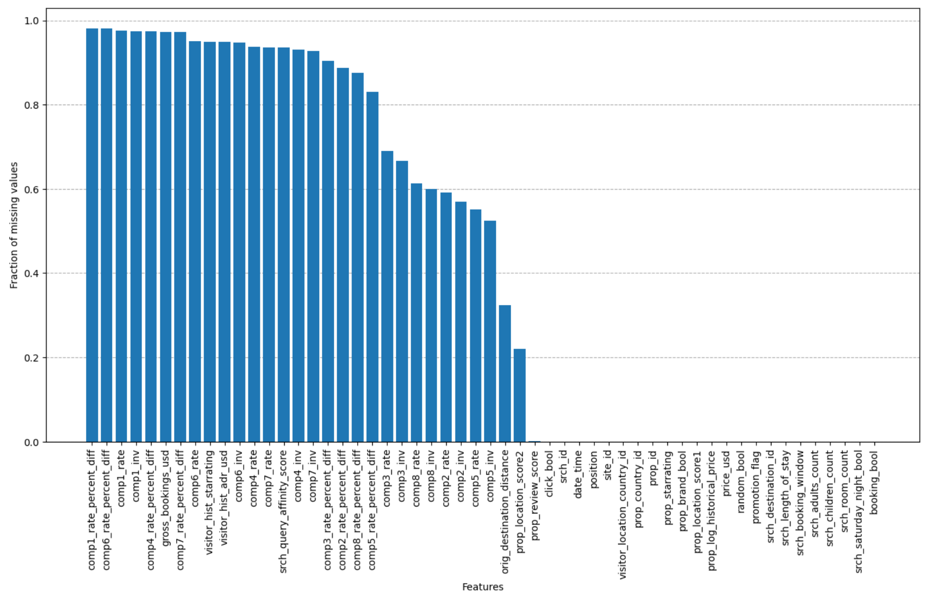

Fig. 1. Missing rates of all attributes in the training data

Fig. 2 shows the ratio of missing values in all original features. Out of 54 attributes, 28 of them suffer severely from missing data, having more than half of data missed. As can be seen from Fig. 1 of distribution of normalized data for selected query-property features, many of them, such as "price_usd" and "visitor_hist_adr_usd", show a long-tailed distribution of extreme values and outliers. This can be interpreted that a hotel can either choose to show their nightly price offer or the price of the entire searched duration, hence the price might not be a reliable feature for prediction, unless being handled carefully to convert to the nightly prices for proper and standardized format, or by outliers elimination.

Another challenge that should be addressed during the pre-processing stage is the imbalance of data. Even though the number of unique search, clicks and bookings made increased gradually over time and were higher during summer month, as shown in Fig 3, most search results were not booked or clicked, resulting in an imbalance of negative queries that have no clicks or bookings. A common practice of down-sampling by splitting the dataset into train and test set, therefore, could likely result in data sparsity and model over-fitting. It is also noted that the problem of position bias might exist in the dataset, in which the visitors are more likely to click and book properties shown at the top of

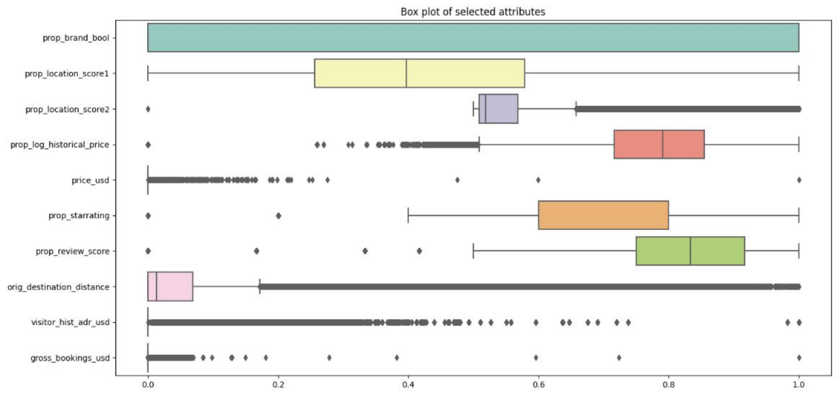

Fig. 2. Distribution of a subset of attributes (normalized)

the result list. As the returned list could be shown randomly or in order, the attribute of "random_bool" might indirectly influence the likelihood of a hotel being clicked, and should also be addressed in the model.

3 Data Preprocessing

During the data preprocessing process, we employed different types of features and came up with two datasets: a base dataset and a derived dataset. The base dataset contains all original features with outliers and missing value handled. Built upon this base dataset, we conducted additional steps of aggregating and transforming existing features, which resulted in the derived set.

3.1 Outliers and Missing Value Handling

As seen in Fig 2, a lot of query-property attributes have wide-spread outliers, in which the data points, represented as black dots, extend all the way to the left or right end. Among them, we paid special attention to the price of the hotel, as it extends from the minimum value of \$0 to a maximum of \$19,726,330 and with a hypothesis that this attribute could be helpful for down-streamed predictive tasks. In the work of [5] on the public Kaggle competition, he decided to remove outliers in hotel price. However, as down-sampling of no-click records was found to cause over-fitting, no removal of samples was performed. Instead, we decided to treat hotels with hotel price over \$10,000 as outliers, which account for less than 1% of the entire dataset, and imputed them with medium value.

In the dataset, a large number of attributes have severe missing ratio with missing rate above 50%. As each of them represents properties of different entities: query, visitor, hotel or property competitor, and is recorded as continuous

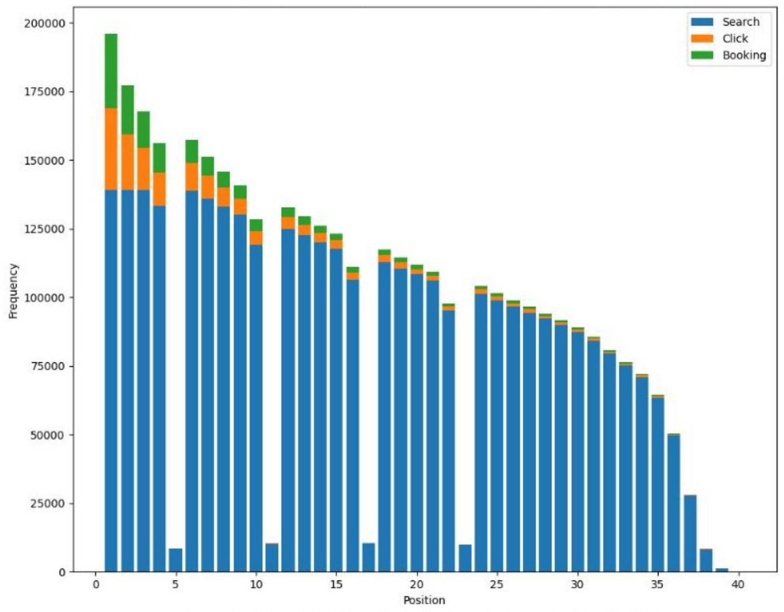

Fig. 3. Count of unique search, clicks, and bookings over position (random_bool=0)

numerical or boolean value, different imputation strategies were necessary. First of all, we decided to remove all hotel's competitors features; all of them have missing fraction above 50%. Other attributes with more than half data missed are historical price the customer previously paid, past rating of the customer, total booking price (in the train set), and the log probability a hotel being clicked generally. The first three attributes were imputed with the value of -1, while the last attribute (log of probability of a hotel being clicked on general Internet search) was sanitized and then imputed with a default value of 0, as this value was the most intuitive.

This left us with three attributes with missing data: the property location score (prop_location_score2), the distance between destination and visitor's location (orig_destination_distance), and the historical review score of the property (prop_review_score). For other attributes without missing values and outliers, we left them intact in the base dataset.

In specific, for the historical review score with only 7364 out of almost 5 million samples being missed, we applied the worst-case imputation method by filling up the missing with a value of -1. As a part of the feature engineering step, we later introduced a new attribute "prop_reviewed" that indicated whether a property has been reviewed (+1) or not reviewed or NaN (0). Inspired by the past work of Liu [7] on which the author imputed the missing value with first-quartile data of all properties, missing in "prop_location_score2" were imputed with the first-quartile of the score of all properties that located in the same country. For the missing in visitor and destination distance, we employed a heuristic-based imputation method, in which the missing data are approximated by the mean of distance between other visitor who came from the same country and other property at the similar property country. If the value were still missed, it would then be imputed with mean of all distance between visitor's location and destination.

3.2 Feature Engineering

For the task of feature engineering, we employed different approaches, some of them inspired by literature review and exploratory data analysis on the dataset, to come up with newly added numerical features. Such features can be categorized into 3 main groups: composite and temporal features, statistical features, and probability-based features

Composite and Temporal Features
In the work of Liu [7], who finished at the fifth place in the public Kaggle competition by combining a linear model, random forest and gradient boosting machine into LambdaMART ensemble, the group proposed 3 new composite features: rating difference between visitor's past rating and property rating, expected final payment, and the maximum number of units (as room per day) occupied. While it has been little to know technical or heuristic foundation on the explanability of such composite features, they were shown to be powerful attributes in the predictive model. Here, we re-implemented their combined number of units occupied "count_window", calculated as:

```
count_window = srch_room_count * max(srch_booking_window) + srch_booking_window
```

As the dataset contains time-stamp data on the search and it has been shown that the number of click and booking made varied by month and day of the week, we expected that the search time, to a certain level, might affect consumer's behavior and divided them into three other temporal attributes: month, week, and date of the week. The feature "prop_log_historical_price" was saved as a logarithm with natural base, so we also converted them into a regular historical price, as we did in the "srch_querry_affinity_score".

Statistical Features
As shown in section 1, statistical features play essential roles in achieving top performance for the winner of the public Kaggle contest in 2013. In this study, for each property, we computed the feature statistics on the mean, median, and standard deviation of numeric features and by each property. The computation was conduced on the whole dataset (as the concatenation of training and test set) and over the following attributes shown in Table 1:

Table 1. List of attributes with statistical methods

visitor_hist_starrating	prop_location_score1	srch_adults_count
visitor_hist_adr_usd	prop_location_score2	srch_booking_window
prop_starrating	orig_destination_distance	srch_room_count
prop_review_score	srch_children_count	srch_length_of_stay

Probabilistic Features During our data exploration stage, we observed that the number of times a property is shown in a search varied significantly, raging from 1 time to 2357 times. Our hypothesis is that the more a hotel is returned in the result list, the more accessible and generally popular the property is to the set of visitors in the dataset. Taking this into consideration, the prior probabilities of clicking and booking were calculated for each property. These features functions as indirect estimations of other customers' impression on the hotel, and were respectively computed as:

$$p_{clicked_{ih}} = P(clicked|i,h) = \frac{(\sum_{j \in C_h} click\_bool_{jh}) - click\_bool_{ih}}{|C_h| - 1}$$

$$p_{booked_{ih}} = P(booked|i,h) = \frac{(\sum_{j \in B_h} booking\_bool_{jh}) - booking\_bool_{ih}}{|B_h| - 1}$$

Here, $p_{clicked_{ih}}(p_{booked_{ih}})$ is the probability that, given a hotel with property ID h and search with search ID i, the hotel will be clicked (booked) when being shown in the search result of search i. $C(h)$ and $B(h)$ are two identical sets of all query to which h has been returned . $click\_bool_{jh}$ and $booking\_bool_{jh}$ are boolean values representing whether the hotel h, when returned for search j, was clicked and booked respectively.

As information on "click_bool" and "booking_bool" of the hotel was not provided in the test set, the prior probability was inferred from the training data. With hotel that are only shown in search of the test set, because there has been no information of such properties in the training set, we imputed the probability value as 0.

4 Modeling and Evaluation of Results

In this section, we present our choice of prediction models and explain how we apply them to our feature engineered dataset developed from Section 2. Our model selection is mainly influenced by our findings from the short literature review conducted in Section 1. Namely we use two algorithms to predict what hotels properties listed as a result of a hotel search a user is most likely to click on: **Collaborative filtering with latent features and LambdaMart**.

4.1 Two-staged Collaborative Filtering

Collaborative filtering (CF) is a widely used technique in recommendation system that aims to use similarities between users and items to provide recommendation. It is leveraged on the principle that similar users would have similar tastes and opinions on similar items. Overall, CF techniques can be categorized into two main categories: memory-based CF (also called as neighborhood-based) and model-based. Memory-based approaches, such as user-based method and item-based method, compute similarity measures between users or items, through the use of similarity metrics such as cosine similarity, and use this information to make predictions. Model-based approaches, meanwhile, employ supervised or unsupervised machine learning methods to capture latent factors and patterns from the user-item matrix.

In this study, we conduced a two-staged collaborative filtering with K-Nearest Neighbor (KNN), a neighborhood-based approach, and Singular Value Decomposition Plus Plus (SVD++), a model-based matrix factorization approach built upon SVD. First, query-property interaction matrices were built for all of the query and property shown in the training and test set, in which each row represents the array of a query's interaction scores on all item, and each column represents the array of all query's interaction scores on that property. The interaction score s_{ih} in the matrix indicates whether a property h, when being shown to query i, would be clicked, booked, or ignored (non-click), and is given as:

$$s_{ih} = \begin{cases} 5, & \text{if } h \text{ is booked by } i \\ 1, & \text{if } h \text{ is clicked by } i \\ 0, & \text{if } h \text{ is not clicked by } i \\ null, & \text{if } h \text{ is not shown to } i \end{cases}$$

A downside with matrix factorization method is that it suffers from cold-start problem, in which there are no historical interaction data of certain users and properties, resulting in empty rows or columns in the rating matrix, which is the case in the current test set.

To overcome this, the KNN-based approach was introduced to find out K-nearest neighbors of users and properties and their ratings were then used to impute the missing values, before applying the SVD method. An overview of the procedure can be seen below.

Algorithm 1 Steps for finding similar query and property with KNN approach

```
item Load the datasetSplit the dataset into training and test sets based on unique
search IDs Define the list of unique query and unique property only shown in test
set For each query (property) in the list, calculate top-K similar queries (properties)
and impute interaction scores from these similar queries (properties)
```

To calculate similar queries to a given search, profiles of all queries were constructed using static visitor and query-related features, including search ID, site ID where the search was conducted, consumer's country ID, consumer's past rating and price paid, destination ID, length of stay, booking window, number of children and adults, and room count. Similarly, profiles of all property were constructed with static property attributes, which are property ID, country ID where the property is located, previous rating and review score, location desirability scores, historical price, and whether the property is part of a hotel chain. As the dataset contains search data over a long period of time, some property features of a same property, including previous rating and review score, historical price, and location scores, varied between different queries with large time gaps. Hence, we took the mean of such attributes when building the property profile. As we need to impute each empty row and column in the search-property rating matrix with some interaction scores, but not too much so that the data might be distorted considerably, and the evaluation metric is on NDCG@5, we decided to conduct the method with K = 1. The most nearest neighbor was then calculated using cosine similarity, and its array of non-null interaction scores was imputed for the given query or property.

Algorithm 2 SVD Plus Plus for Hotel Recommendation

```
1. Load the dataset
2. Split the dataset into training and test sets based on unique search IDs
3. Build the query-property interaction matrix, with empty row and column imputed
   from similar query and property using KNN approach
4. Perform grid search for hyperparameter tuning with cross-validation on training
   set
5. Get the best model from grid search
6. Train the best model on the entire training set
7. Predict interaction score on the test set
8. Convert the interaction score matrix into a DataFrame
9. Sort the DataFrame by search ID and predicted score in descending order
```

In our task of predicting recommended property, the idea behind conducting SVD and its variant is that it is highly likely that there are some generalities to be found in the interaction of visitors and property results, for example, the purpose of the booking (e.g. family or business trip), or the size of the room, which can be captured in terms of a much smaller number of latent factors. SVD++ is an extension of SVD, which takes into account not only the ratings or explicit feedback provided by the visitors as in SVD, but also implicit feedback like clicks or viewing of the property, which are essential in the process of booking a hotel. Given the query-property interaction matrix, SVD++ factorizes it into three main matrics: the U matrix represents the queries and their corresponding latent factors, the diagonal matrix indicating the importance of each latent factor, and the V matrix represents the latent factors and properties. To estimate unknown scores between a query and property, the algorithm minimized over the regularized root-mean-squared error (RMSE) between true interaction score and predicted score using latent matrix, and updated parameters with gradient descent. A grid search was also conducted on the number of latent factors, learning rate, and number of epochs to find optimal hyper-parameter of the algorithm, which resulted in the best model of 50 latent factors, learning rate of 0.001 and 10 epochs. The regularization term was set to default value of 0.02. The Algorithm 2 outlines our implementation with SVD++ and hyper-parameter tuning, implemented in Python and with surprise package.

4.2 LambdaMART

LambdaMART is a ranking algorithm that combines the strengths of MART (Multiple Additive Regression Trees) and LambdaRank. MART is a form of gradient boosting that uses decision trees as the base learners and minimizes a differentiable loss function. LambdaRank, on the other hand, is an extension of RankNet, a pairwise ranking model. LambdaRank improves upon RankNet by directly optimizing the Normalized Discounted Cumulative Gain (NDCG), a popular metric for evaluating the quality of a ranking, with the idea that to train a model, we do not need the costs themselves, but only the gradients (of the costs with respect to model scores). The gradient should be bigger for pairs of properties that produces a bigger impact in NDCG by swapping positions.

By combining these two approaches, LambdaMART is able to handle the ranking aspect of the problem in a pairwise fashion. In specific, for each pair of properties (i, j) from the result list, it computes the gradient λ_{ij}, which then act as a "force" to push pairs of properties apart or together (up or down the list of results), through the change in NDCG score when swapping property positions. As the result, the method can levarage the power of gradient boosting to minimize the non-smooth loss function, making it a highly effective tool for predicting user behavior on the hotel dataset with NDCG evaluation metric.

The pseudocode for the LambdaMART algorithm, implemented in Python and with XGBoost package, can be seen below.

Algorithm 3 LambdaMART for Hotel Prediction

```
 1. Load the dataset
 2. Split the dataset into training and test sets based on unique search IDs
 3. Define the features and target variable
 4. Convert the data into DMatrix format for XGBoost
 5. Train the model using XGBoost with the 'rank:pairwise' objective
 6. Predict the target variable for the test set
 7. Convert the predictions into a DataFrame
 8. Create a DataFrame to hold the search ID, property ID, and predicted score
 9. Sort the DataFrame by search ID and predicted score in descending order
10. Compute the NDCG score for the ranked list of properties
```

First, the hotel dataset is loaded, and unique search IDs are extracted. These IDs are then split into training and test sets, ensuring that all data associated with a particular search ID is either in the training set or in the test set, but not both. This step is crucial to prevent data leakage between the training and test sets.

Next, the features (all columns except 'target') and target variable ('target') are defined for both the training and test sets. The data is then converted into DMatrix format, a data structure required by XGBoost, the gradient boosting library used to implement LambdaMART.

The model is trained using XGBoost with the "rank:pairwise" objective. This is LambdaMART's objective and it is used because the task is to rank the hotel properties based on the likelihood of being clicked on. The model then makes predictions on the test set, and these predictions are converted into a DataFrame for easier manipulation.

A DataFrame is created to hold the search ID, property ID, and predicted score. This DataFrame is sorted by search ID and predicted score in descending order, resulting in a ranked list of properties for each search ID. Finally, the NDCG score is computed for the ranked list of properties. This score is a measure of the quality of the ranking produced by the model which we show in the next section.

4.3 Evaluation

Evaluation Setup We splitted the training set into ... for local comparision of two approaches.

Evaluation

5 Deployment

Our approach can be deployed in a real-world scenario by Expedia or similar companies to provide personalized hotel recommendations for their users. We use LambdaMART and recommender system as our solution methods, which are both scalable and effective for ranking problems [12]. However, One of the main challenges of deploying our solution is to handle the large volume and high velocity of data generated by Expedia's users and hotels. This requires efficient data processing and storage techniques, such as distributed computing and cloud services, to ensure the timely and accurate update of the features and models. Another challenge is to deal with the cold start problem, which occurs when new users or hotels enter the system and have no or few historical data. This requires adaptive methods that can leverage other sources of information, such as user profiles or hotel descriptions, to provide reasonable recommendations for them [10].

One of the main opportunities of deploying our solution is to improve the user experience and satisfaction by offering personalized and relevant hotel recommendations. This can increase the user loyalty and retention, as well as the conversion rate and revenue for Expedia. Another opportunity is to incorporate feedback mechanisms that can collect and utilize the user's preferences and ratings for the recommended hotels. This can enable online learning and model updating, as well as provide more insights into the user's behavior and needs.

Some possible improvements or extensions that can enhance our solution are:

- Incorporating more features that capture the user's context, such as their location, time, device, etc. These features can help to account for the user's situational factors that may affect their hotel choice [12].
- Applying more advanced methods that can model the user's sequential behavior, such as recurrent neural networks or attention mechanisms. These methods can capture the user's long-term and short-term interests, as well as their attention patterns for different hotels [12].
- Exploring more diverse and novel recommendations that can balance the trade-off between exploitation and exploration. These recommendations can expose the user to new hotels that they may not have considered before, but may also like [10].

6 Conclusion

In this section, we summarize our main findings and contributions from this assignment. We highlight the strengths and limitations of our approach and reflect on what we learned from this experience. We also provide some directions for future work or research on this topic.

References

1. Pegah Malekpour Alamdari, Nima Jafari Navimipour, Mehdi Hosseinzadeh, Ali Asghar Safaei, and Aso Darwesh. A systematic study on the recommender systems in the e-commerce. *IEEE Access*, 8:115694–115716, 2020.
2. Poonam B.Thorat, R. M. Goudar, and Sunita Barve. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, 110(4):31–36, 2015.
3. Zhe Cao and Tao Qin. *Learning to rank: From pairwise approach to Listwise approach*, 2007.
4. Eric He. Why we chose lambdamart for our hotel ranking model, Mar 2020.
5. David Kofoed Wind. *Concepts in Predictive Machine Learning: A conceptual framework for approaching predictive modelling problems and case studies of competitions on Kaggle.* PhD thesis, Technical University of Denmark, 2011.
6. Yinxiao Li. Handling position bias for unbiased learning to rank in hotels search. *arXiv preprint arXiv:2002.12528*, 2020.
7. Xudong Liu, Bing Xu, Yuyu Zhang, Qiang Yan, Liang Pang, Qiang Li, Hanxiao Sun, and Bin Wang. Combination of diverse ranking models for personalized expedia hotel searches, 2013.
8. Linyuan Lü, Matúš Medo, Chi Ho Yeung, Yi-Cheng Zhang, Zi-Ke Zhang, and Tao Zhou. Recommender systems. *Physics Reports*, 519(1):1–49, 2012. Recommender Systems.

9. Aditi Mavalankar, Ajitesh Gupta, Chetan Gandotra, and Rishabh Misra. Hotel recommendation system. 08 2019.
10. Phong Nguyen, Jun Wang, and Alexandros Kalousis. Factorizing lambdamart for cold start recommendations. *Machine Learning*, 104(2-3):223–242, 2016.
11. Maria Augusta Nunes and Rong Hu. Personality-based recommender systems. *Proceedings of the sixth ACM conference on Recommender systems*, 2012.
12. Doug Turnbull. Lambdamart in depth. *Doug Turnbull's Blog*, 2022.
13. Yabing Zhao, Xun Xu, and Mingshu Wang. Predicting overall customer satisfaction: Big data evidence from hotel online textual reviews. *International Journal of Hospitality Management*, 76:111–121, 2019.