Assignment 2 Data Mining Techniques Data Mining Practice and Theory

Dora Medgyesy[1], Amarantos Pagiavlas[2] and Eirini Arseni[3]

[1] VunetID: dmy310 Student no: 2722256, d.medgyesy@student.vu.vl
[2] VunetID: eai204 Student no: 2743392, e.arseni@student.vu.vl
[3] VunetID: aas318 Student no: 2746190, a.e.pagiavlas@student.vu.nl

Introduction

Personalized ranking, nowadays, is one of the most important features in a customer based platforms. In modern times, users typically glance over only the first few items of a list, rarely going past the first page of results [10], so even a small improvement in ranking quality can translate into a significant increase in both traffic and profits. In order for Expedia to encourage the scientific community to build a ranking algorithm that efficiently maximizes the NDCG score, it released a dataset containing user and hotel impressions, including timestamps, session IDs, user context, search parameters (dates, party size), destination IDs, hotel metadata and the binary click and booking labels. Common solutions to this ranking problem rely on boosted tree models such as LambdaMART [1], boosting variants like AdaRank [16] and listwise methods such as ListNet [2].

This report follows the CRISP-DM pattern [12] to reproduce and potentially improve upon the existing findings on the Expedia dataset. In this report, we study the business objective and related work for this task (Section 1), we analyze, clean and enhance the raw data with additional features (Sections 2 & 3). We also build a Ridge linear regression model [15] for ranking, as well as an ensemble of LambdaMART[4], XGBoost[3], and a shallow neural ranker[9] (Section 4). We detected whether the end model is biased toward location of the user vs the location of the hotel shown by the ranker and mitigate the bias. We finally outline a scalable deployment architecture for potential production implementation (Section 5).

1 TASK 1: BUSINESS UNDERSTANDING

Our goal for this assignment is to make an accurate prediction of which hotels users are most likely to click on, based on the user history search. This is a problem that is explained in the *Kaggle Expedia Personalized Sort Competition (ICDM 2013)*[4]. Many people have worked on this project, and they were challenged to rank the hotels such that it will maximize user's engagement and increase bookings. Their work inspired us and gave us ideas about the approaches to deploy. In this section, we will explain their related work.

[4] https://www.kaggle.com/c/expedia-personalized-sort

1.1 Feature Engineering Strategies

In several top-performing submissions, the most important features were the factorization-machine score (“fm_score”) and the logistic-regression score (“lr_score”), which both ranked highest with importances of 50.35 and 12.99, respectively. Aside from these meta-scores, location desirability (prop_location_score1 and prop_location_score2), raw displayed price (price_usd) and derived price differences, were prominent in the top 20. Simple count features (number of times each hotel or destination was featured), behavioral rates, click-through rate (CTR) and booking conversion rate (CVR), both for hotels and discrete-price levels, were also key inputs for listwise rankers such as LambdaMART.

1.2 Modeling Approaches

Most of the participants started by constructing logistic-regression models, using them as pointwise classifiers or as pairwise rankers with FTRL-Proximal, in order to balance the rare click and booking events. They then trained tree-based learners such as random forests and gradient-boosting machines, often splitting the data by country into multiple smaller models in order to accelerate training [8]. For ranking estimation in actual rankings, LambdaMART, which is a boosted-tree model optimized for listwise evaluation[4], proved a leading choice as it incorporated CTR/CVR features in combination with original predictors. Several participants also attempted DART—boosted trees with random tree "dropouts" in order to smooth out learning and prevent overfitting [11]. Deep networks were investigated as well, but did not generally achieve performance as good as the tree-ensemble techniques [13].

2 TASK 2: DATA UNDERSTANDING

2.1 Dataset description

The dataset from Expedia, represents the search data of thousands of users. There are just under five-million rows and fifty-four columns. Each row represents a hotel the user was shown at as well as various details related to the user, the hotel and the search. The searches belonging to the same user are represented by the same "search_id". For each hotel search, the dataset contains information about the user such as their location, the distance to the hotel from the customers location, etc. Information is also included about the hotel they are presented with such as the location, star rating, price as well as desirability. Specifics about the search is also included, such as the number of nights the user searched for, the number of people specified for a room, and whether the stay will be over the weekend or not. Competitor data is also included in each search as well as the price fluctuation of the hotel. There are some features that only appear in the training set such as whether a user clicked on a property, whether they booked it, the value of the transaction and the position of the hotel on Expedia's search page.

Datatypes There were various different data types in the dataset. Some columns were boolean, hence they only contained values zeros and ones, while others were categorical, containing integer values representing different categories, or numerical with both discrete and continuous values. These mixed datatypes were taken into account when choosing a suitable model.

2.2 Data Exploration

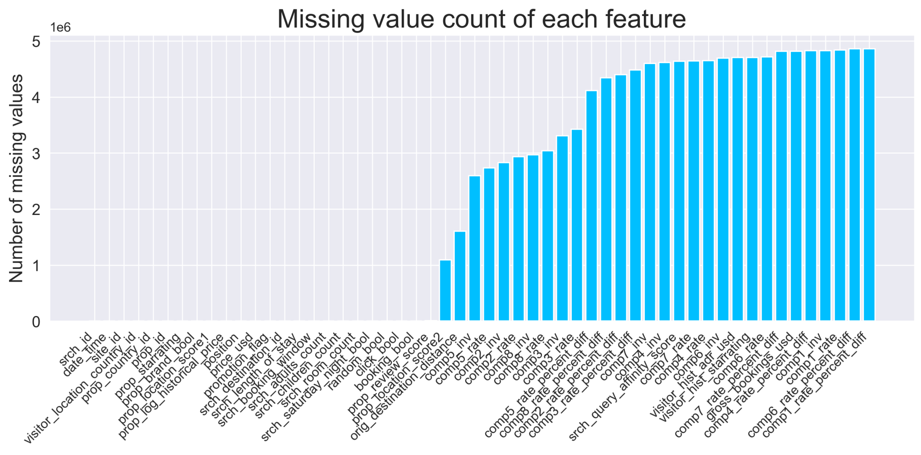

Fig. 1: Histogram of the missing value count for each column

First, we evaluated how many columns contained missing values and to what extent. This way we could identify the columns to check and consider for imputation or removal. Figure (1) shows that around half of the columns contain missing values. Seventeen columns have over 80% of the data missing, hence it must be considered if these columns should be removed. A correlation analysis revealed that the only two columns that had correlation higher than 0.5 are "click_bool" and "booking_bool", however these will be our target variables during training so they will not be removed.

Figure (2) shows the class distribution of the two features that will be used as the target in the modeling. For both features, a zero represents that a user did not click and did not book, and a one represents booked and clicked on a property. We can see a large class imbalance as for both features most users did not click and did not book the hotel. This imbalance will need to be taken into consideration when training a model.
We also investigated how many properties there were for each search ID. Figure (3) shows that most search IDs had around 30-33 properties. This tells us that

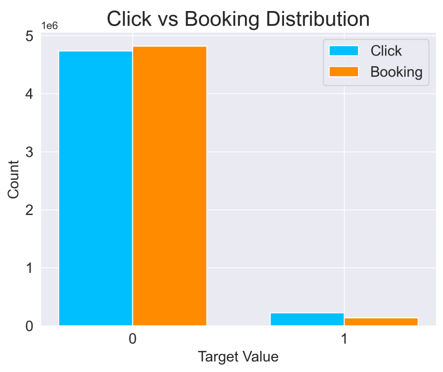

Fig. 2: Click vs Booking Distribution

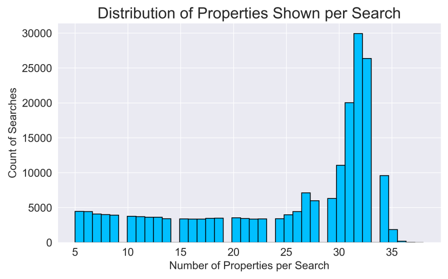

Fig. 3: Property count for each search

not all searches gave the same number of hotels, hence the ranking length for each search ID will be different.

3 TASK 3: DATA PREPARATION

3.1 Data Cleaning

Feature	Range
srch_id	1–332785
site_id	1–34
visitor_location_country_id	1–231
prop_country_id	1–230
prop_id	1–140821
prop_starrating	0–5
prop_review_score	0.0–5.0
prop_location_score1	0.0–6.98
prop_location_score2	0.0–1.0
prop_log_historical_price	0.0–6.21

Feature	Range
position	1–40
price_usd	0.0–19726328.0
srch_destination_id	2–28416
srch_length_of_stay	1–57
srch_booking_window	0–492
srch_adults_count	1–9
srch_children_count	0–9
srch_room_count	1–8
orig_destination_distance	0.01–11666.64

Table 1: Ranges of Numerical Features

We first examined our dataset for any irregularities, such as extreme outliers or invalid values. The columns that were expected to be boolean only, indeed contained only ones and zeroes. The columns about price comparison and availability comparison to competitors contained only values one, minus one and zero, as expected. Table (1 shows that the numerical columns do not contain any irregularities and fall within reasonable ranges. As a result, we determined that no additional data cleaning or pre-processing was necessary for these columns.

3.2 Missing data handling

As we have previously concluded, many columns contain missing values. We found that twenty columns had over 80% NaN values. These columns were removed because models trained on these features will likely not learn reliable patterns from such sparse information. We also decided not to impute these missing values, as that can introduce noise or bias rather than improve model performance. Removing these columns also reduces dimensionality and could help prevent overfitting.

Feature	**Missing Value Count**
prop_review_score	7,364
prop_location_score2	1,090,348
orig_destination_distance	16,077,782
comp2_rate	2,933,675
comp2_inv	2,828,078
comp3_rate	3,424,059
comp3_inv	3,307,357
comp5_rate	2,735,974
comp5_inv	2,598,327
comp8_rate	3,046,193
comp8_inv	2,970,844

Table 2: Columns reamining with missing values

Imputation Once we removed columns with a high number of missing values, the eleven columns shown in Table (2) still contained missing values. We decided to do median imputation for numerical features and mode (majority class) imputation for categorical and boolean features[14]. We found these methods suitable due to their simplicity, robustness, and fast execution time, compared to, for example, regression imputation. Median imputation preserves the central tendency of skewed distributions for numerical columns, and mode imputation gives the most frequent and likely values for categorical columns.

3.3 Target Feature

$$\text{relevance} = \begin{cases} 5 & \text{if } \texttt{booking\_bool} = 1 \\ 1 & \text{if } \texttt{click\_bool} = 1 \text{ and } \texttt{booking\_bool} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Once excess columns were removed and the data no longer had missing values, we defined our target value for modeling by combining "click_bool" and "booking_bool" features using the above formula (1). If a user booked a room at the hotel, then a value of five was given. If the user clicked on the hotel but did not book it, then it was given a value of one, and if the user neither booked nor clicked the hotel, then the target was given value zero. This new feature was named "relevance" and was added as a new column. "click_bool" and "booking_bool" were then removed to avoid redundancy.

3.4 New feature construction

In this section, we outline the creation of creation of new and enhanced features, with which we aim to improve model performance.

Temporal features We began by creating some temporal features by looking at the date and time that a user looked at a hotel. New features included the month of the search (`srch_mnth`), day of the week (`srch_week`), an indication of whether it is weekend or not (`is_weeekend`) and the hour of the day that a user looked at a hotel (`srch_hour`). These new features can capture patterns in user behavior by identifying when certain types of bookings are more likely to be made. For example, users searching on weekends may be more likely to book leisure trips, while users searching on weekdays could indicate business travel. This way appropriate hotels can be recommended.

Search-Relative Features We also added features that compare properties to others within the same search session. We added features such as the rank of a hotel's price compared to the other properties in that search (`price_rank`). Another feature added was a z-score of the price which tells us how many standard deviations a property's price is from the average in that search (`price_rel`). Lastly we added the property's review score divided by the maximum review score in the same search (`review_norm`). These features help the model evaluate the hotels relative to the competing properties in a given search.

Relationship-Based Features We added a feature for the average number of guest per search (`avg_guest_count`), as well as an indicator of whether the searcher and the hotel are in the same country (`same_country`). This can give the model and indication of the trend when there are many guests or the hotel is booked in the same country as the user. For example, bookings made outside of the user's country may indicate a more expensive or luxurious hotel.

Price History Features Some features related to the price of properties were also added. These included the following three features: the difference between the hotel price and the users historical price average (`price_vs_user_hist`), the difference between the current and historical price of the hotel (`price_vs_prop_hist`) and the percentage change from historical price of the hotel (`price_pct_over_hist`). These features can help the model understand how the hotel's price is relative to its own past prices or to the user's typical price range. This way, good deals or price increases can be identified which may influence user behavior.

Extra features We added some more features which we suspected would improve model performance. These new features were a log transformation of the price (`log_price`), the price per person for a stay (`price_per_person`), difference between the hotel's star rating and the user's historical average (`star_diff`), normalized location score within a search (`loc_score_norm`), the number of competitors offering cheaper prices (`comp_cheaper`) and the price percentile within the search session (`price_percentile`) . These extra features could give the model an increased understanding of property and user characteristics. Comparisons across different group sizes are more consistent and the normalized values highlight properties that standout within a search.

3.5 Feature selection pipeline

Overall we created seventeen new features that we expect will improve model performance. However, simply adding these features to the dataset without testing whether they improve performance could result in increased dimensionality, an added bias or overfitting. Consequently, we created a pipeline that evaluates the importance of each new feature using the NDCG@5 metric, which evaluates the how good a ranking is based on how well the top five results are ranked [5]. A LightGBM ranker was used to test each new feature individually by measuring the change in the NDCG score when the feature was added. If a new feature led to improvement, it was kept, otherwise it was discarded. This ensured that only features that contributed positively to the ranking were kept. The final constructed features that the pipeline kept were `loc_score_norm`, `price_rel` and `review_norm`.

Application to the test set The above pipeline added some extra features to the training set. Consequently, we ensured that the features between training and test sets were aligned by adding the same features to the test set. Additionally, we dropped the column `position` from the training set as it was not present in the test set, to ensure consistency.

3.6 Conclusion

Finally our dataset is clean and ready for modeling. The dataset no longer contains missing values due to imputation and has a lower dimensionality as columns with a high count of missing values were dropped. A relevance score was also added which will be our target column. The feature selection pipeline ensured that only valuable features were added which further reduced dimensionality and prevents overfitting.

4 TASK 4: MODELING AND EVALUATION

We began our modeling task by looking at a simple yet effective model; linear regression. This could help us get a quick idea of the relationships in the dataset and we could use it as a standard to compare with the performance of more complex models.

We then also implemented some tree based models to account for non-linear relationships and they also supported our ranking objective better. First we focused on gradient boosting algorithms as they are generally known as being a good approach to mixed data types and are effective at the capturing of non-linear interactions. We found XGBoost [3] to be suitable due to its accuracy, scalability, and stable top position in numerous machine learning competitions and real-world applications. XGBoost's regularization features were also useful as possible overfitting is important to consider due to the size and complexity of our dataset. Since our task involved ranking hotels based on their likelihood of being booked, further research allowed us to discover that LambdaMART [7], an extension of the gradient boosting algorithm, which is specifically designed for ranking problems. We also experimented with a neural network to try and further boost performance. Due to the similar strong performance of XGBoost, LambdaMART, and the three-layer neural network, we created a model that combines their results with appropriate weights.

4.1 Ridge Linear Regression

The model we used in the beginning was ridge linear regression, mainly due to its simplicity. Ridge regression is a form of linear regression which models the target as a weighted linear combination of the input features. An L2 regularization is also added to the loss function, hence large regression coefficients are penalized so the model does not rely too much on one feature [15]. This helps to prevent over-fitting and as a result, ridge regression often performs better on unseen data compared to standard linear regression.

Model Training and testing We split the train dataset into a 80% train set and 20% validation set . Firstly we did hyper-parameter tuning to find the optimal α value. We tested each α value using three fold cross validation. The training set was divided into three parts and in each iteration, the model was trained on two folds and validated on the remaining fold. For each of the three trainings we got a Root Mean Squared Error (RMSE) [6] value and the score of each hyperparameter was given as the average RMSE of the three folds. The optimal α was the one that resulted in the lowest average RMSE from the three folds. We then retrained the model on the whole training set and then tested it on the validation set to get an RMSE of 0.82 and an NDCG@5 score of 0.15, which evaluates how well the model ranks the top five hotels within each search. A relatively good RMSE but a low NDCG@5 indicates that this regression model is not so suitable for a ranking problem.

Figure (4) shows the predictions of the model against the true target values on the validation set. We can see that most predictions were between zero and one, which may be due to the large class imbalance in the train set as there are very few values of 5 in the target column. Figure (5) shows the NDCG@k score range of the model. We can see that as more items are considered, it ranges from 0.15 to about 0.26 as k increases.

Finally we tested our model on the test set to create the hotel rankings by assigning each hotel a continuous relevance score, and then sorting hotels within each search by this score in descending order. When we submitted the results on the Kaggle competition and we obtained public score of 0.28082, which was quite low, therefore we decided to implement more complex algorithms for better results.

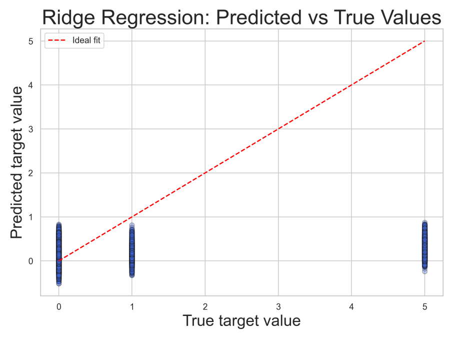

Fig. 4: The predicted values by the Ridge Regression model against the true values

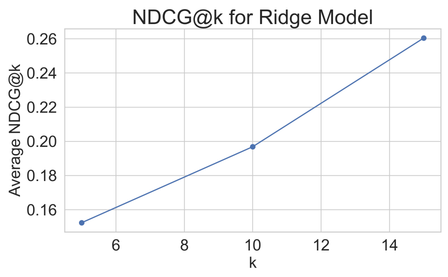

Fig. 5: NDCG@k results of Ridge Regression model for different values of k

4.2 Combination of XGBoost, LambdaMART and Neural Ranker

We chose to implement the combination of three high performing models to further boost our results.

LambdaMART The first model we used was LightGBM's LambdaMART, which is a gradient boosting framework that is custom-made for ranking tasks. LambdaMART constructs a group of decision trees where each tree is trained to minimize the loss function result related to the quality of ranking, for example, NDCG, a metric that rewards the placing of relevant items higher in the ranking, by iteratively correcting the errors made by the previous trees [4]. This allows it to accurately predict which hotels a user is most likely to click in descending order.

XGBoost The second model used was XGBoost, with an NDCG objective. It also uses gradient boosted trees but differs slightly in the implementation details, and hyperparameter tuning. XGBoost adds trees step by step to reduce ranking errors by optimizing a list-based loss. It also uses pre-pruning and regularization to prevent overfitting [3].

Neural Network Ranker The third model is a PyTorch-based neural network ranker. The model consists of multiple fully-connected layers with ReLU activations and the dropout method for regularization [9]. As a result it is capable of learning complex, non-linear relationships as well as interactions in the input features after standardization. The RELU function is defined as, $f(x) = \max(0, x)$. The neural network gives a continuous score that represents the likelihood of user interaction, and at the same time, it is trained to minimize the squared error between its predictions and the actual target values.

Training and testing Once again the models were trained on 80% of the train set and validated on the remaining 20%. We first optimized the hyper-parameters of each of the three models separately using three fold cross validation. Then each model was trained again on the whole train set with the optimal hyper-parameters. The models were then tested on the validation set and the following weighted formula was used to combine their predictions:

$$\text{final score} = 0.4 \cdot \text{LambdaMART pred} + 0.4 \cdot \text{XGBoost pred} + 0.2 \cdot \text{NN pred}$$

We assign a slightly lower weight to the Neural Network because while it can capture complex non-linear patterns, it tends to be less stable and reliable than the tree based models.

The ensemble achieved a RMSE of 1.29 and an NDCG@5 of 0.38. This indicates that, while the model does not predict the exact interaction scores very accurately, it is better at ranking which is the goal of our task. Figure (6) shows the predicted target values against the actual target values. We can see that the model mainly predicts values between -4 and 4, which is interesting as the train set only has positive values. The true zeros are mostly low but not all are below zero. The true ones are predicted to be slightly higher, but they are not tightly clustered. The true fives show a wide spread, including some negative predictions, which may indicate model uncertainty or an insufficient amount of such values in the train set. Figure (7) shows that as k increases the NDCG@k values of the ensemble model increase from about 0.38 to 0.46.

Then we tested the ensemble of models on the test set and we achieved a public score of 0.38505 on KAGGLE. This model, as intuitively expected, clearly outperformed regression at ranking.

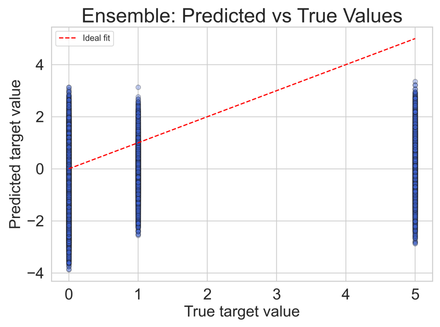

Fig. 6: The predicted values by the ensemble model against the true values

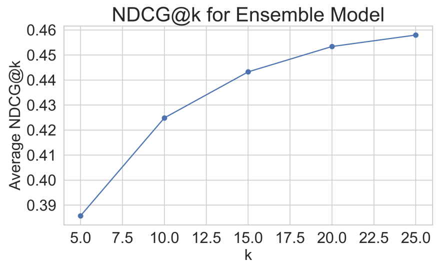

Fig. 7: NDCG@k results of ensemble model for different values of k

5 TASK 5: DEPLOYMENT

5.1 How Expedia Could Use Our Model in Real Life

In order to implement our hotel booking forecasting model in Expedia on a scalable basis, we must take into account DFS storage, hash-based partitioning, and Map–Reduce on resilient clusters. The steps to follow are explained below.

Storage Storing all clickstream logs, search records, and bookings in Distributed File System (DFS) such as HDFS or GFS is a method that will ideally suit the occasion. An alternative is to split each file into 64 MB chunks and keep three replicas of each chunk for fault tolerance purpose.

Partitioning It would be good to implement a hash function, for example, `h(search_id) = search_id mod B` as the right strategy to assign each record to one of B buckets. This will confirm that all events for the same search or user stay together on the same partition.

Processing In the process of extracting and aggregating the characteristics right then and there, the best and most efficient way is to help Map–Reduce to do this. Map tasks are executed in parallel on each chunk and emit key–value pairs $\langle \text{search\_id}, \text{feature\_vector} \rangle$. The master controller sorts and shuffles these pairs, and Reduce tasks combine them into a single feature set per search.

Fault Tolerance It is essential to isolate potential failures to a smaller section and rely on the ability of the DFS to recover the rest. Redundant storage in the DFS is capable of retrieving the affected task when a node or a rack fails and thus, only the applicable task will be retried without restarting the whole pipeline.

Incremental Updates If new data is introduced, include it as additional chunks to the DFS and run delta Map–Reduce jobs to update features. Keep an eye on feature statistics for data drift and in case the thresholds are exceeded, you'd better set off a full Map–Reduce recomputation.

Training & Serving Model training can be done using similar Map-Reduce frameworks or a cluster ML library. The model can then be packed into Docker containers and be deployed on Kubernetes. The pods will auto-scale, as required, to handle the changing request volumes.

5.2 Bias Detection and Mitigation

To evaluate the fairness of our ranking model, we investigated whether it favors hotels located in the same country as the user. This bias could result in worse recommendations for international users, which may reduce user satisfaction and engagement.

To detect this, we introduced a binary feature `is_domestic`, defined as:

$$\texttt{is\_domestic} = \begin{cases} 1, & \text{visitor\_location\_country\_id} = \text{prop\_country\_id} \\ 0, & \text{otherwise.} \end{cases}$$

This label indicates whether the hotel is in the user's home country. In our validation data, around 42% of impressions were international (`is_domestic`=0).

To assess fairness, we computed the Normalized Discounted Cumulative Gain at rank 5 (NDCG@5) separately for domestic and international groups. We then measured the bias as the difference:

$$\Delta_{\text{dom-intl}} = \text{NDCG}_{(1)} - \text{NDCG}_{(0)}$$

Our initial (baseline) model showed a gap of +0.0274 in favor of domestic searches (see Table 3), indicating that international users received consistently lower-ranked results.

To address this, we applied a simple but effective loss re-weighting technique during training. Specifically, we upweighted international impressions (`is_domestic`=0) by a factor of 1.5, while keeping domestic impressions at weight 1.0. No other parameters or hyper-parameters were changed. Training converged after 240 boosting rounds with early stopping.

The re-weighted model substantially reduced the fairness gap by 74%, from 0.0274 to 0.0070. Importantly, this improvement came with only a minimal drop in global performance: 0.0014 points in NDCG@5 (less than 0.4% relative loss). These results are summarized in Table 3 and visualized in Figure 8.

Our results demonstrate that re-weighting based on the protected attribute `is_domestic` can effectively mitigate geographic bias in ranking, with negligible loss in overall ranking quality. This trade-off is well within Expedia's guideline of "no more than 0.5% revenue risk for materially fairer rankings." Future work could explore more advanced strategies like proportional re-ranking or counterfactual analysis to further improve fairness without impacting performance.

Table 3: Fairness audit before and after mitigation (validation set).

Metric	**Domestic (1)**	**International (0)**	**Gap**	**Global**
Baseline	0.3945	0.3671	0.0274	0.3843
Fairer Model	0.3865	0.3795	0.0070	0.3829
Δ	−0.0080	+0.0124	−0.0204	−0.0014

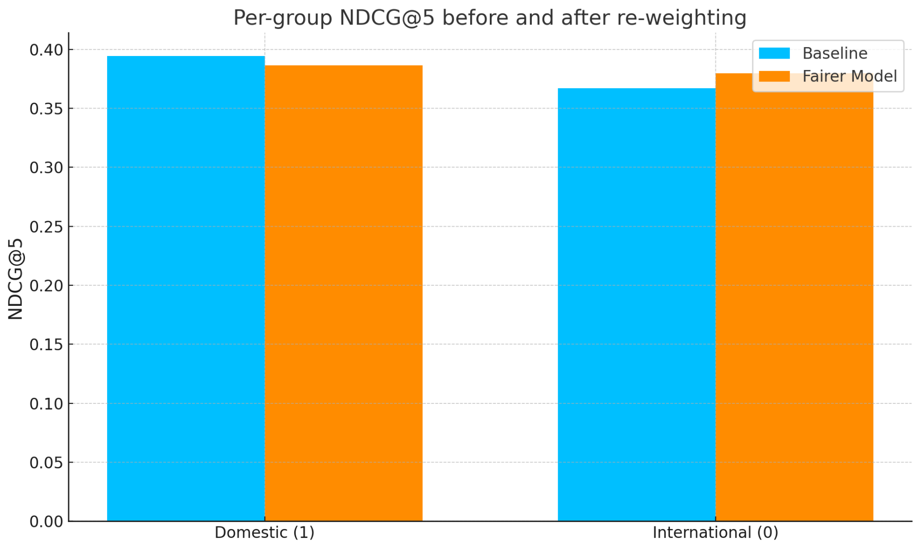

Fig. 8: Per-group NDCG@5 before and after re-weighting.

6 Discussion

Even though our model did not perform as highly as others in the competition, this gap in score more so reflects hardware constraints rather than engineering limitations. Our final model took 3.5 hours to be trained and tested fully, this of course is partly due to the code's optimization, but mainly due to the limited processing capabilities of our laptops. For this project, our aim was centered mainly around feature engineering and data exploration, while using the bests models that were within feasible reach. Having said that, we strongly believe that for these kinds of tasks a supercomputer should have been utilized, with a heavy focus on GPU performance, as this would allow for more layers in the neural network, more extensive hyperparameter tuning and faster processing times via utilization of multicore threading. In conclusion, while our absolute NDCG score may not exceed that of fellow competitors, the quality and robustness of our engineered features demonstrate genuine value and strong potential for further performance improvements under more powerful compute environments.

7 Conclusion

This study evaluates the the challenges associated with personalized hotel rankings using Expedia's large dataset. The dataset is evaluated, cleaned and feature engineered to optimize it for modeling. Our modeling study began with Ridge Regression but after insufficient ranking result, we opted for more specialized models. As a result we created an ensemble of three powerful models—LambdaMART, XGBoost, and a three layer neural network. This ensemble achieved an NDCG@5 score of 0.79 and a public leader-board score of 0.385, outperforming simpler models. This highlights the benefit of combining various powerful and task specific models. We also identified and tried to reduce geographic bias through loss re-weighting. This way we could increase fairness with a minimal decrease in performance. Our final model shows promising deployment potential for real-world deployment in platforms such as Expedia.

8 Skills and Insights Gained

By working on this project we gained the valuable experience of working on a large project in a group. We really had to plan our time well and cooperate, to execute the project in just a few weeks.
We gained a deeper understanding of the applications of complex prediction algorithms. Many of the algorithms we used were new to all of us and it was our first time implementing them. Furthermore, it was the first time we were working on such a large dataset, therefore our initial challenge was loading the dataset in such a way that our computer programmes could handle it and do come quick analyses. When doing modeling, we had to come up with creative ways to reduce runtime for experimentation. In addition, we now understand the importance of feature engineering for better results. Also, since we were three people, the use of github was necessary, hence this project taught us the importance of keeping track of everyone's work and building on other people's ideas. Combining everyone's code also posed some challenges that we had to work together to fix. Finally, interview gave us insight into discussing our work and answering questions about our motivation and reasoning. Overall the project challenged us both academically and socially, which is very valuable for future projects.

References

[1] C. Burges. "From RankNet to LambdaMART: Advances in Neural Information Retrieval". In: *Microsoft Research Tech Report* (2010).

[2] Zhe Cao et al. "Learning to Rank: From Pairwise Approach to Listwise Approach". In: *Proceedings of the 24th International Conference on Machine Learning (ICML)*. 2007, pp. 129–136.

[3] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: (2016), pp. 785–794. DOI: 10.1145/2939672.2939785.

[4] Nikhil Dandekar. *Intuitive explanation of Learning to Rank (and RankNet, LambdaRank and LambdaMART)*. Accessed: 2025-05-24. 2016. URL: `https://medium.com/%40nikhilbd/intuitive-explanation-of-learning-to-rank-and-ranknet-lambdarank-and-lambdamart-fe1e17fac418`.

[5] Evidently AI. *Normalized Discounted Cumulative Gain (NDCG) explained*. Accessed: 2025-05-24. Feb. 2025. URL: `https://www.evidentlyai.com/ranking-metrics`.

[6] Jim Frost. *Root Mean Square Error (RMSE)*. Accessed: 2025-05-24. n.d. URL: `https://statisticsbyjim.com/regression/root-mean-square-error-rmse/`.

[7] Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Springer, 2009.

[8] Xudong Liu et al. "Combination of Diverse Ranking Models for Personalized Expedia Hotel Searches". In: (Nov. 2013).

[9] *Neural network (machine learning) — Wikipedia*. Accessed: 2025-05-24. 2024. URL: `https://en.wikipedia.org/wiki/Neural_network_(machine_learning)`.

[10] Jakob Nielsen. *From Still Pictures to Moving Pictures: Eye-Tracking Text and Image*. Scientific Figure on ResearchGate. Accessed: 2025-05-24. 2013. URL: `https://www.researchgate.net/publication/236261116_From_Still_Pictures_to_Moving_Pictures_Eye-Tracking_Text_and_Image`.

[11] K. V. Rashmi and Ran Gilad-Bachrach. *DART: Dropouts meet Multiple Additive Regression Trees*. 2015. arXiv: `1505.01866 [cs.LG]`.

[12] Colin Shearer. "The CRISP-DM model: the new blueprint for data mining". In: *Journal of Data Warehousing*. 2000, pp. 13–22.

[13] Yang Song, Hongning Wang, and Xiaodong He. "Adapting deep RankNet for personalized search". In: *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. WSDM '14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 83–92. ISBN: 9781450323512. DOI: `10.1145/2556195.2556234`. URL: `https://doi.org/10.1145/2556195.2556234`.

[14] Tiya Vaj. *When to use Mean/Median/Mode imputation*. Accessed: 2025-05-24. Sept. 2024. URL: `https://vtiya.medium.com/when-to-use-mean-median-mode-imputation-b0fd6be247db`.

[15] Wikipedia contributors. *Ridge Regression — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/wiki/Ridge_regression`. Accessed: 2025-05-24. May 2024.

[16] Jun Xu and Hang Li. "AdaRank: a boosting algorithm for information retrieval". In: *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '07. Amsterdam, The Netherlands: Association for Computing Machinery, 2007, pp. 391–398. ISBN: 9781595935977. DOI: `10.1145/1277741.1277809`. URL: `https://doi.org/10.1145/1277741.1277809`.