Assignment 2
Data Mining Techniques

Eui Yeon Jang, Hannah Lim, and Tim van Loenhout

2569512, 2677439, 2525199
Group 106

1 Introduction

Expedia.com is an online travel agency that can be used to search and book vacation related facilities, such as hotel rooms. Users are able to enter a query to the system, which will return a list of relevant entities for the user. For recommender systems such as Expedia.com, the aim is to present a list that is most relevant or preferable to the user, as these systems will be judged based on their recommendations. For instance, a user is less likely to return to Expedia.com when they were not satisfied with the provided recommendations. Hence, for such systems, user satisfaction is important for gaining and maintain a big client base.

This report explores processing raw data obtained from Expedia.com and building models to return the best ranked list of hotels for users based on their search query. As part of a Kaggle competition, the aim is to build the best performing model for a held-out dataset and rank high on the leader board.

2 Related Work

This task is a learning to rank (LTR) task, where purchased and clicked hotels should be ranked on top among a list of hotels retrieved based on a search query. There are several LTR models from the field of information retrieval that can be employed for this task, such as LambdaRank. These models optimise for the number of correct inversions of items with respect to a chosen evaluation metric (such as normalised discounted cumulative gain or average precision) which weighs rankings of top items higher than lower items [2].

Furthermore, ensemble methods, such as bagging and boosting, have shown to be effective in various tasks and have also shown to be competitive with LTR models in a similar Kaggle competition hosted by Expedia [6]. These methods combine weak learners to produce a strong learners, addressing issues with bias and variance in the data.

The winners of the competition offer some interesting insight into the task and dataset. For instance, it was noted that whether certain information about a hotel was missing or not serves as an important feature, as people are less likely to book hotels with missing information [5, 7]. It was also mentioned that users seem to compare prices across different booking systems, and therefore, information about the competition may also be important.

3 Data

The provided raw dataset is divided into a train set and a test set. The train set consists of 4,958,347 entries and the test set of 4,959,183 entries, with 199,795 and 199,549 unique search ids, respectively. Each entry in the dataset represents a hotel property that was retrieved as a result of a user query. It contains various features corresponding to characteristics of the user and the hotel, search specifications, and offers by other competitors, amounting to total 49 features.

The train set contains additional information about whether a user clicked or booked a hotel, the hotel's position on the results page, and if booked, the final value of the transaction (otherwise, `null`). This results in four additional features for the train set, adding up to 53 features.

3.1 Initial analysis

From all the entries from the train set, 4.47 % is clicked and 2.79 % is booked, leading to an imbalanced dataset. On top of the class imbalance, many entries contain missing data. For instance, most of the competitor features were missing for at least 90% of entries (Figure 1).

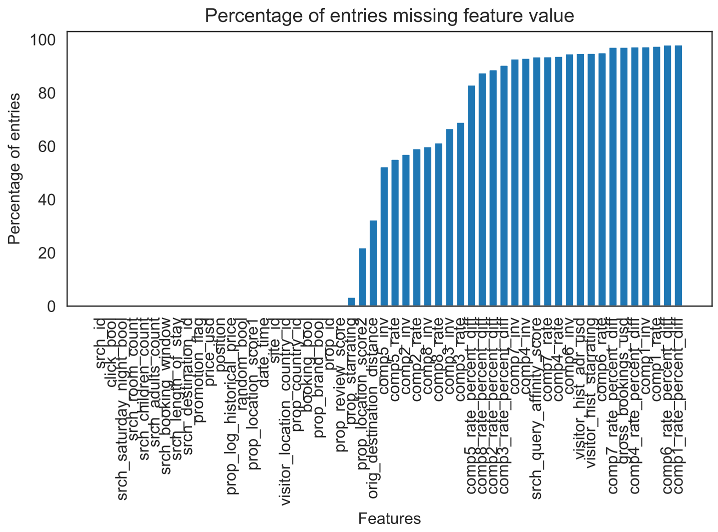
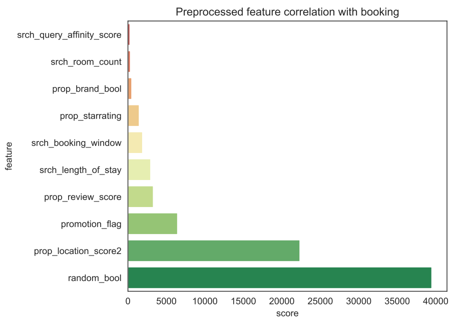


Fig. 1. Percentages of entries with values missing per feature. For features up to and including `prop_id`, the percentage of missing data is 0.

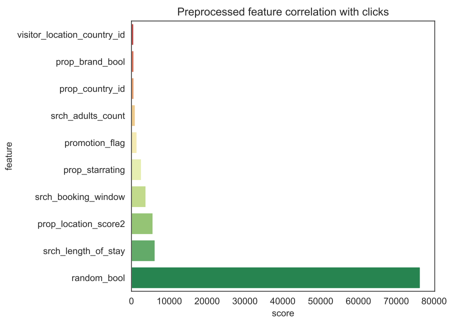
Figure 2 shows the F-regression correlation scores between the top ten most informative features and the `booking_bool` and `click_bool` after having filled in the missing values as described in Section 3.2.4. The top ten correlated features are rather similar for both, as `booking_bool` and `click_bool` are also quite highly correlated (p-value of 0.78).

The highest correlated feature is `random_bool` which determines whether a display was randomly displayed or sorted. The impact from this feature will be discussed more in depth in Section 3.7.

To gain more insight into why for instance the feature `price_usd` is, surprisingly, minimally correlated to both `booking_bool` and `click_bool` we examined its value distribution over the three groups of entries: 1) booked, 2) only clicked 3) neither. Then we compared these distributions to the distributions of the highly correlated feature `prop_location_score2`. The plots (Figure 3) show that compared to `prop_location_score2`, the distributions of `price_usd` are very similar across the groups of entries, indicating that price impacts the probabilities of booking, clicking and not clicking in a similar manner. Hence, from this feature, the model won't be able to extract much distintive information for the three groups.



(a) Correlation with booking.



(b) Correlation with clicked.

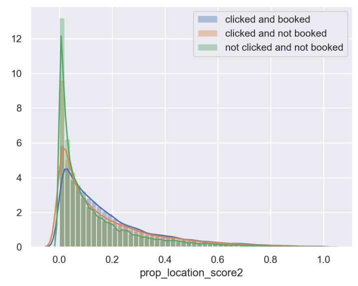
Fig. 2. Top 10 features most correlated with a hotel being booked or clicked using the ANOVA F-values.

3.2 Data preprocessing

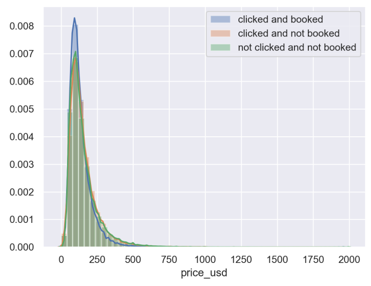
In this section, we detail how we preprocess the raw dataset.

3.2.1 Dropping features. Despite, previous work showing that competitor information is useful, we drop these features as at least 90% of the values are missing. We also ignore `site_id` as from the correlation results, this feature does not appear to contain any useful information. We also drop `orig_destination_distance` for reasons mentioned in Section 3.2.4. This leaves us with 23 (+4 training set) features.

3.2.2 Target values. The raw train set does not contain explicit target values for training a model. We create a few types of possible target values to be used for our predictive models using the provided click and booking information.



(a) `prop_location_score2`



(b) `price_usd`.

Fig. 3. Distribution plots for features `prop_location_score2` and `price_usd` over the groups 1) property is booked (blue), 2) only clicked (orange) and 3) neither (green).

The most basic is the set {0,1,2}, where 0 stands for no click, 1 for a click, and 2 for a booking. An alternative to this is setting the value for booking to 5 instead of 2. This creates a larger gap between the booked and non-booked entries to force the model to put more emphasis on the booked, which contains most weight in the ranking scoring metric. We refer to these values as '012' and '015' values, respectively.

When using historical user data in ranking problems, the occurrence or absence of a click is not only dependent on the relevance of an entry, but is also influenced by noise and position bias [3]. Noise refers to the added uncertainty due to the unpredictable nature of human users, which can be reduced by simply using a larger sample size. This approach does not suffice for position bias. This problem refers to the concept that higher ranked entries are more likely to be clicked upon, regardless their relevance. In order to address this issue, we construct two regression targets, where each score is weighted by its examination probability. This approach assumes that the examination probability of a function is correlated to its position in the ranking, which is stored as the training feature `position`. For weighting, the score is either divided by the absolute value of the corresponding rank, referred to as average relevant position (ARP) weighting, or by the $\log 2$ of the rank, resulting in discounted cumulative gain (DCG) weighting.

3.2.3 One-hot encoding

While decision trees can learn categorical features, they can introduce bias in other models, hence we apply one-hot encoding of the features `date_time`, `visitor_location_country_id`, `prop_country_id`, `prop_id`, and `srch_destination_id`.

As the period of the year and the moment of the day that is searched for a hotel could influence the chances of booking a particular hotel, we decide to keep the temporal information. As it would be difficult to process dates and times using one-hot encoding, we decide to interpret the `date_time` features into two new types of one-hot features: *part of day* and *month*. The former is divided into the categories ‘night’ (00:00-06:00), ‘morning’ (06:00-12:00), ‘afternoon’ (12:00-18:00) and ‘evening’ (18:00-00:00), and the latter into eight categories since no entries where from the months July, August, September, and October for both the training and the test dataset.

3.2.4 Missing value interpolation. In this section we discuss how we interpolated the missing values for each feature.

The feature `orig_destination_distance` contains information about the distance between the hotel and the customer at the time of search. This feature has been proven useful in predicting hotel clusters when interpolated with matrix completion model [1]. However, this was too computationally expensive to train. Also, given the large number of missing values, we concluded that interpolating by averaging over the dataset or search query would introduce unnecessary bias, hence we decided to drop the feature entirely.

The missing values of `srch_query_affinity_score` are interpolated by first looking if the property has a value somewhere else in the dataset (the same property is indicated by sharing the same value for `prop_id`). If no such value could be found, we interpolate the missing values by averaging the values of the other properties of within the search. When values are still missing, we simply interpolate by taking the average over the entire dataset.

Unfortunately, this approach not applicable for the `prop_starrating` and `prop_review_score` as properties which contained missing values for these features missing in the entire dataset. Therefore, we first interpolate by taking the average with respect to the search id, and interpolate the rest by taking the average of the entire dataset.

As for `visitor_hist_starrating` and `visitor_hist_adr_usd`, no user id information is present, due to which we are not able to interpolate the missing values for these feature by using the values from the same user. Therefore, they are interpolated by taking the average of the feature over the entire dataset.

Finally, for `prop_location_score2`, we build a simple linear regression model to predict the missing values given `prop_location_score1` as input, as they were shown to be highly correlated and `prop_location_score1` does not contain any missing values. This resulted in a model with mean squared error of 0.041. Figure 2 shows that the interpolated `prop_location_score2` is in fact the second most correlated feature with `booking_bool`.

3.3 Creating new features

3.3.1 Missing-flag features. As seen in related work, hotels with missing values are less likely to be booked, and therefore it is desired to retain the information about whether a feature originally present or interpolated. Therefore, for the features `visitor_hist_starrating`, `visitor_hist_adr_usd`, `prop_starrating`, `prop_review_score`, `prop_location_score2` and `srch_query_affinity_score` we generate binary features indicating whether their value was initially missing.

3.3.2 Price per person features. We create two additional features with regards to the price of the hotels. Price per adult ($\frac{\texttt{price\_usd}}{\texttt{srch\_adults\_count}}$) and per person, including children ($\frac{\texttt{price\_usd}}{\texttt{srch\_adults\_count}+\texttt{srch\_children\_count}}$), could give a more detailed indication of the costs of the hotel.

3.3.3 Delta features. Similar Kaggle competitions have showed that the difference between star ratings and prices of hotels can be an important indicator on the likelihood of hotels being clicked or booked [4]. Therefore, as extra features, we consider the difference between the star rating of the hotel and the average star ratings of the previous hotels the user has booked, and the difference between the price of the hotel and the average price of the hotels the user has previously booked.

3.3.4 Cross-products. Seeing that few features are highly correlated with booking, we aim to promote more emphasis on these relevant features. To do so, we take the top 10 features that were most correlated with a hotel being booked and we create 45 different combination of features and multiply their values. ANOVA F-values showed that the cross-product features are also correlated with being booked. For the final features, we only consider the top five of these created cross products, which are (`random_bool`, `delta_starrating`), (`random_bool`, `missing_prop_starrating`). (`random_bool`, `prop_starrating`). (`random_bool`, `prop_review_score`), (`prop_location_score2`, `prop_review_score`), and (`prop_location_score2`, `missing_prop_starrating`).

3.4 Detecting and removing outliers.

In order to keep the model from potential bias from outlier noise, we count how many of the features for an entry have values that are outliers and remove entries with more than n such feature values from the training set. A feature value is considered an outlier when its value lies outside of the z standard deviations of the feature's mean, where z is a hyperparameter. The maximum number of outliers for an entry, n, is also a hyperparameter which determines the trade-off between the size and reliability of the dataset.

3.5 Normalisation.

Some learning models put more emphasis on features containing larger values. However, the models we consider (Section 4) are invariant to such scale invariance and do not require normalisation over all data entries. Nevertheless, we experiment with normalising per query. We believe that the internal query distribution is most important to be represented in the feature for learning a ranking. Furthermore, this could also diminish the potential bias from discrepancies between different countries and seasons in features such as the property price.

3.6 Reducing class imbalance.

While the ranking is mainly based on the correct prediction of bookings and clicks, these entries are underrepresented in the dataset. Hence, in order to force the model to put more emphasis on learning the characteristics from these relevant entries, we experiment with balancing the train set. This is done by upsampling relevant entries until a certain ratio between the classes is reached. With ratio of 1-1-1 between bookings, clicks, and no clicks, the latter two are sampled from the dataset with a higher probability, resulting in a new training dataset containing an equal number of entries per class.

3.7 Using only random entries.

In order to reduce the previously discussed position bias, we also experiment with training only on entries that were displayed either randomly or sorted, determined by the feature `random_bool`. From the train set, the entries retrieved using random display (29.59 %), 0.53 % is booked and 4.13 % is only clicked. Among the entries retrieved using sorted display (70.41%), 8.90% is booked and 1.56 % is clicked. This confirms that users are more likely to click on properties which are ranked higher in the list. Training on either type of rankings does not completely eliminate this bias, however, it does make the bias more consistent.

3.8 Summary of settings for experimentation.

In the previous sections, we have outlined the different ways of preprocessing and engineering features. We test each of these different methods, which we refer to as feature settings, to evaluate which produce the best results. The final list of feature settings and their options are as follows.

– **target value**: 012, 015, ARP, DCG
– **normalisation**: none, normalise with respect to entire training set, normalise per search query
– **outlier reduction** ($z = 3$)[1]: none, $n = 1$, $n = 1$, $n = 3$
– **random entries**: all entries, only random entries, only non-random entries
– **upsampling**: none, upsampling to a ratio of 1-1-1.
– **feature inclusion**: basic features, add missing flags, add month features, add part-of-day features, add country-id features, add `prop_id` feature, add price per person features, add delta features, add cross product features

The default features are features from the original dataset: `visitor_hist_starrating`, `visitor_hist_adr_usd`, `prop_starratng`, `prop_review_score`, `prop_brand_bool`, `prop_location_score1`, `prop_location_score2`, `price_usd`, `promotion_flag`, `srch_length_of_stay`, `srch_adults_count`, `srch_children_count`, `srch_room_count`, `srch_saturday_night_bool`, `srch_query_affinity_score`, `srch_booking_window`, and `random_bool`. Each addition of features is considered in isolation.

[1] We found that 3 standard deviations from the mean is the most optimal z value.

4 Models

Based on related work, we focus our attention on Random Forest, LambdaRank, and Gradient Boosting models.

4.1 Random Forest models

Random forest models are ensemble models built with multiple decision trees to classify data, and average over the trees to obtain a final prediction. This is one of the simplest algorithms for classification and regression tasks. Due to spreading the predictive power over multiple classifiers, these models generally reduce the probability of overfitting.

For this task, we experiment both with classification and regression. For classification we used the classes 'booked', 'clicked' and 'neither'. For regression we experiment with the different target values as discussed in Section 3.2.2. We use `sklearn`'s RandomForest model. As the regression model perform better than classification, the latter is not be further discussed in this report.

We consider the following hyperparameters of the RandomForest model. `min_samples_split` determines the number of data points required for a split and `min_samples_leaf` being the minimal number of data points required per leaf node. `n_estimators` controls the number of decision trees, which is negatively correlated to the model variance. On the other hand, increasing the value for texttt max_depth creates a more fine-grained model as more features are consulted, which promotes predictive power while inducing a larger risk of overfitting. Finally, `n_jobs` relates to the computational efficiency of learning the model.

Although this model is rather simplistic for a ranking task, it serves as a good baseline.

4.2 LambdaRank models

The problem with general machine learning models is that they do not directly optimise a ranking. That is, learning a ranking is not a classification or regression model. In classification and regression entries are considered independently from each other, and therefore the loss function does not always represent ranking quality.

Ranking metrics cannot be used directly as a loss function as they are not differentiable. An alternative is to optimise a function that considers pairs of entries and maximises the the number of correct inversions. However, this approach treats every pair as equally important. To emphasise that certain items are more important to be ranked correctly, weighted inversions are used, where the importance of an inversion depends on its impact on the ranking score. This approach is called LambdaRank.

In this assignment we use the LambdaRankNN library from PyPI, which builds the LambdaRank loss objective on a neural network, where the hyperparameters cover the network architecture, consisting of the number and size of the hidden layers and their corresponding activation functions. Adam is used as optimiser, which combines momentum with adaptive learning rates for steady convergence while reducing the risk of getting stuck in a local minimum. We alter the source code to extend it to also learn on the DCG and ARP target values.

4.3 Gradient Boosting models

Gradient Boosting models are ensemble models that combines weak models in a iterative fashion to create one strong model. Each additional model creates experts in different regions of data where the previous models had difficulty. This gives a stronger learner that reduces the bias of its component models.

`XGBoost` offers a gradient boosted tree model for ranking task, XGBRanker, which optimises the LambdaMart ranking algorithm, which combines aspects of the two previous algorithms by applying the LambdaRank learning objective of minimising the pair/listwise loss on an ensemble of decision trees. The hyperparameters of interest for XGBRanker correspond to the learning objectives (pairwise or listwise), type of booster (tree, linear or DART), number of boosted trees, maximum depth for base trees, and learning rate.

4.4 Stacking models

Lastly, we also consider stacking the three different models. As opposed to Random Forest and Gradient Boosting models, in stacking the combined predictions of *different* models serve as new data for a meta-model. Instead of creating this meta-model using static weights, we apply linear regression, for which we consider the combination of components: XGBRanker and RandomForest, and XGBRanker and LambdaRank.

5 Experiment Setup and Results

For each model described in Section 4, we evaluate which settings described in Section 3.8 performs best. Then given the best settings, we tune the model hyperparameters. Due to time constraints, we do not perform a full grid-search on the settings nor the hyperparameters. Instead, we test each setting in isolation and choose the combination of best performing settings as the final setting for the model. Each setting performance is compared to a baseline for each model, which is trained on a default set setting: '012' target value, no normalisation, no outlier reduction, no upsampling, using all entries (random and non-random) and the basic features.

Unfortunately, due to the large size of the dataset and the various hyperparameter searches to perform, we utilise only the first 500,000 entries of the training set. We then split this subset into 8:2 training and validation set. We decide that this subset is large enough to not implement cross-validation. Each model has their respective learning objective for training, and we evaluate on the validation set using NDCG@5.

Before predicting the test dataset and submitting the results on Kaggle, we train a model with the optimal settings and hyperparameter values on the entire training set.

5.1 Random Forest models

The Random Forest model has a baseline NDCG@5 of 0.2270 with default settings and features. Experimenting showed that it performed best with `012` target values, without normalisation, with outlier reduction at 2, using only non-random entries, without upsampling, and using all additional features except the cross product, price per person and price per adult features. Hyperparameter tuning resulted in the following optimal settings; `n_jobs`=-1, `n_estimators`=150, `max_depth`=10, `min_samples_split`=500, `min_samples_leaf`=100. This resulted in an NDCG@5 score of 0.2945 for the validation set and 0.33898 on the Kaggle test set.

5.2 LambdaRank models

The LambdaRank model has a baseline NDCG of 0.3230 with default settings and features. Experimenting showed that it performed best with `015` target values, with normalisation per query, with outlier reduction at 3, using only non-random entries, without upsampling, and using the following additional features; missing flag, month, part of day and cross product. We found that the optimal hyperparameter settings are `hidden_layers`=3, of `size`=32,16 and 8 with a ReLU activation after each layer. This resulted in an NDCG@5 score of 0,3415 for the validation set and 0,3502 for the Kaggle test set.

5.3 Gradient Boosting models

The XGBRanker model has a baseline NDCG@5 of 0.3302 with default settings and features. Experimenting showed that it performed best with `012` target values, without normalisation, with outlier reduction with $n = 2$, using only non-random entries, without upsampling, and using the additional missing flag and delta features. Surprisingly, the hyperparameter search showed that no other hyperparameter values outperformed the `XGBoost` default values – pairwise loss, tree boosters, maximum depth of 3, 100 estimators, and learning rate of 0.1. This resulted in an NDCG@5 score of 0.3563 for the validation set and 0.36613 for the Kaggle test set.

5.4 Stacking models

For each of stacking combinations, we take the tuned models of each model type. Combining XGBRanker and RandomForest retrieved NDCG@5 of 0.3463 on the validation set, while XGBRanker and LambdaRank performed 0.3676. Using the latter on the Kaggle test set resulted in an NDCG@5 of 0.36612.

6 Evaluation

The results show that RandomForest was the worst performing model, followed by LambdaRank, while the best models were XGBRanker and the stacked model.

This was expected as XGBRanker utilizes the best of both worlds; the predictive power from RandomForest and the for rankings optimized learning objective from LambdaRank. Also, previous Kaggle competitions have shown similar LambdaMART models to achieve high performances.

As for the best feature settings, XGBRanker performed best on both {0,1,2} or {0,1,5} labels, while the regression labels ARP and DCG resulted in lower scores. Between ARP and DCG, the latter performed slightly better, which could be explained by its relatively softer weighting due to dividing the score by the logistic instead of absolute rank.

XGBRanker is not susceptible to normalisation over the entire dataset given the scaling-invariant nature of decision trees, however, normalising per query seems to degrade performance. This could be explained by the fact that when normalising per query, the ratio within the query is not affected, hence the inter-query re-scaling does not provide any extra information to the model. However, this approach does smooth out the average feature value distributions over the different queries, setting the minimum for each query to 0 and the maximum to 1, consequently discarding their relative information.

Furthermore, outlier reduction seems to have a positive impact on the XGBRanker performance. However, this only holds up to a maximum of 2 outlier features, as by removing more entries, the increased quality of the features does not outweigh the negative impact from reducing the number of entries.

Moreover, the results indicate that using either non-random or only random queries for training supports the model in coping with positional bias. This could be explained by the assumption that effect of the positional bias is different between the two, with the former containing a more self-reinforcing bias, while in the latter, bias is more uniformly distributed. Hence, using only either one could cause a more consistent nature of the bias, which might be easier for the model to cope with. Furthermore, in the case of using only non-random queries, implicit information from the Expedia.com ranking algorithm can be inferred by the model.

Finally, upsampling did not result in an improved NDCG score, while adding the missing flag, delta and country id features did. Other features such as price per person/adult and using cross products did not result in any improvement, which could be explained by that fact that these features are simply combinations of other features, which is information that can also be inferred by the model itself. Moreover, manually adding these features could even introduce bias to the model as they put extra emphasis on assumed correlations which might not be beneficial to the predictive power.

7 Discussion

Despite the positive impact from the engineered feature settings, and the confirmed strong relative predictive power of XGBRanker, our final NDCG@5 score was mediocre compared to other submissions on Kaggle. We have several ideas why this may be the case. Firstly, the low score on the default feature set could indicate a misstep during the initial steps in prepossessing. For instance, it could be that some of the left out features contained valuable information nonetheless, or perhaps something had gone wrong during the missing value interpolation. While using the mean for filling in missing data feels most intuitive, we could have overlooked a more optimal solution.

Furthermore, more intensive training of the models could have boosted performance, however due to time constraints, limited computational power and the educational objective of the assignment, we instead decided to use our time for more conceptual thinking about the inner workings of the models and by experimenting with different prepossessing approaches.

Finally, there is a possibility that we might not have picked the perfect set of features for the default models. Discarding potentially informative features has a vital impact on performance. This is, for instance, shown by that when removing `srch_destination_id` from the the default feature set, XGBRanker performance drops to below 0.34. This shows that having the right features is crucial.

It was interesting to see that the stacked model performed significantly better during validation but was outperformed by the plain XGBRanker on the test set. It would therefore be interesting to do further experimentation on these models. For example, by stacking more models or separating their focus onto either clicks or bookings.

The most important lesson that we take away from this assignment is the necessity of scalable processing and training methods. We spent quite some time waiting for certain parts of the data to be processed, while other preprocessing steps were infeasible due to the amount of time they required. Moreover, the large size of the data and limited computing power meant we could only afford to train on the entire training set when ready to make predictions on the final test set. All together, more scalable and more efficient methods could be an important factor for faster processing and training in further experimentation.

References

1. Expedia hotel recommendations 1st place solution summary, https://www.kaggle.com/c/expedia-hotel-recommendations/discussion/21607
2. Burges, C.J.: From ranknet to lambdarank to lambdamart: An overview. Learning **11**(23-581), 81 (2010)
3. Croft, W.B., Metzler, D., Strohman, T.: Search engines - information retrieval in practice (2009)
4. Khantal, N., Kroshilina, V., Maini, D.: Rank hotels on expedia.com to maximize purchases (dec 2013)
5. Wang, J., Kalousis, A.: Icdm 2013, https://www.dropbox.com/sh/5kedakjizgrog0y/_LE_DFCA7J/ICDM_2013?preview=4_jun_wang.pdf
6. Woznica, A.: Icdm 2013, https://www.kaggle.com/c/expedia-personalized-sort/discussion/6203
7. Zhang, O.: Icdm 2013, https://www.dropbox.com/sh/5kedakjizgrog0y/_LE_DFCA7J/ICDM_2013?preview=3_owen.pdf