Assignment 2 - Data Mining Techniques

L. Prast 2628427, H. Choudhary 2754741 and N. Levi 2696717

Group 109: Vrije Universiteit Amsterdam

1 Introduction

Search engines can be found on different types of websites, for example - Google, movie recommendation websites and hotel searching websites. The goal for these websites is to give recommendations that are preferred by the user, so that they will come back later because this website is the best in predicting what they want. In this report we tried to predict what hotel a user is most likely to click on. We made our predictions with the data from the online travel agency Expedia and our task was to rank every search of a person by most likely to click on to least likely to click on. We did this by using different type of features of the properties such as how much the hotel costs, reviews of the properties and for how many persons the searched for property was.

1.1 Business Understanding

2 Exploratory Data Analysis

The training data-set has around 5M instances with 54 feature columns. The features are related to search,hotel, user and competitor descriptions.

- Search Features : Date and time of the search, Destination Id, Length of the stay, Booking window(in days), number of adults and children, number of rooms and if the stay includes a Saturday night.
- Hotel features : Hotel id, Hotel country id, star-rating, price, promotion flag, probability a hotel will be clicked, review score, if hotel is a part of major hotel chain, desirability of hotel's location, historical price of the hotel, total value of the transaction, if clicked, if booked.
- User features : visitor country id, purchase history(average star rating and average price per night) and distance between hotel and user.
- Competitor Description : If Expedia has a lower, similar or higher price than a competitor, price difference, hotel availability.

We analyze the data-set for empty cells in the training data and find that percentage of missing values for "prop_review_score" $\approx$ 0.15%, "prop_location_score2" $\approx$ 22%, "orig_destination_distance" $\approx$ 32% ,"visitor_hist_starrating" $\approx$ 95%, "visitor_hist_adr_usd" $\approx$ 95% and competitor information have more than 50% of their values missing.
We observe that out of 5M instances,there are 199795($\approx$ 0.2 million) unique

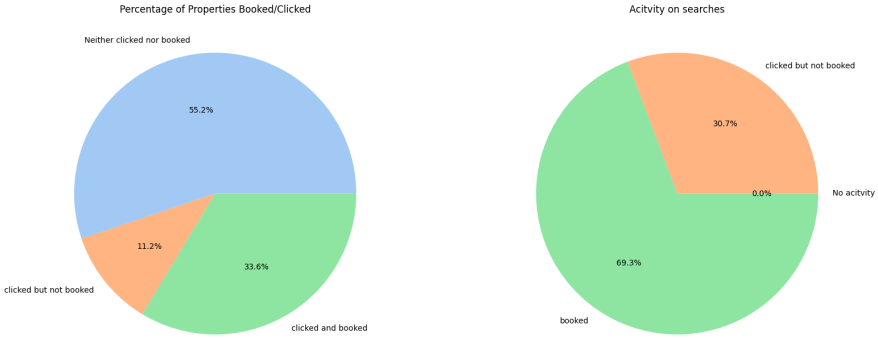

Fig. 1. Left Figure - Distribution of Unique property counts based on if they were clicked or booked and Right Figure - Conversion of searches into Bookings.

searches("srch_id"), 129113($\approx$ 0.13 million) unique properties, 210 different countries the user belongs to and 172 property country ids. Out of the 5M instances, 221879($\approx$ 0.22 million) were clicked and 138390 were booked($\approx$ 0.14 million).

In Fig. 1, we see that around 55% of the properties were neither clicked or booked by any of the users. Out of the clicked properties, 75% of the properties were booked by at least one user. And with respect to queries, all the searches showed some activity and clicked on at least one of the properties recommended to them. The conversion rate on search ids is around 70% , this implies out of all the unique searches, 70% of them ended up booking a hotel on Expedia.

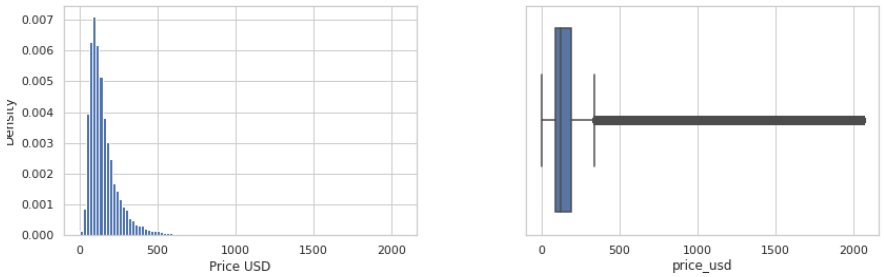

Fig. 2. Left Figure - Distribution graph of displayed price of the hotel for the given search and Right Figure - Box plot of displayed hotel price shown on a search.

In Fig.2, distribution of price of the hotel feature("price_usd") is right skewed. Mean, median and 75th percentile values for this feature are 155.18, 122.0 and 184.96 dollars respectively. The above graphs are plotted after masking values above 99.9th percentile with 99.9th percentile which is 2060.03 dollars.

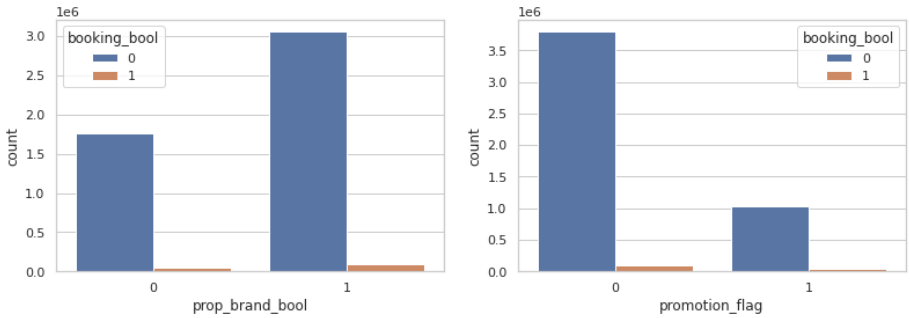

Fig. 3. Left Figure - Graph of number of instances with Hotel being part of a major chain(label - 1) and independent hotels(label - 0). Right Figure - Instances with hotels having sales price promotion and hotels without promotion flag.

In Fig. 3, we see that there are more number of recommendations with hotels belonging to a chain as compared to independent properties. With respect to displayed promotional price on a hotel, less number of such offers are shown on the searches/queries. We further mathematically evaluate the conversion rate of searches and find that $\sim$ 2.5% of the recommendations were booked when promotion_flag was 0 and $\sim$ 4.0% of the recommendations were booked when promotion_flag was 1. This implies a slight preference of properties with displayed sale price advertisement. However, for branded properties $\sim$ 2.9% instances were booked and $\sim$ 2.6% were booked for independent hotels. The difference in conversion for branded vs independent hotels is very small.

In the below Fig. 4, the distribution of feature "prop_log_historical_price" is close to a Normal Distribution but with an abrupt right tail. The graph is plotted with training instances where "prop_log_historical_price" is not zero. And, the feature "srch_booking_window" is similar to a Geometric Distribution.

Graph of Historical data of the user in Fig. 5 indicates that users with purchase history tend to book their stay on Expedia.

Table. 1 reflects the distribution of "Property Starrating", "Property Review Score", "Location Score", "Price(USD)" and "Gross Booking Price(USD)" of hotels clicked or booked by the users.

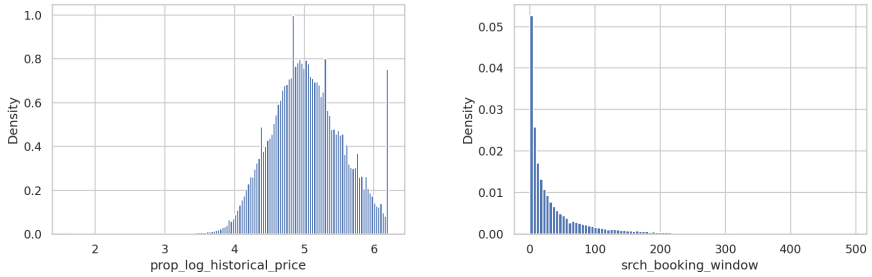

Fig. 4. Left : Distribution of the logarithm of the mean price of the hotel over the last trading period with mean ≈ 4.3 dollars of the properties that were sold in that period. Right : Distribution of number of days in the future the hotel stay started from the search date

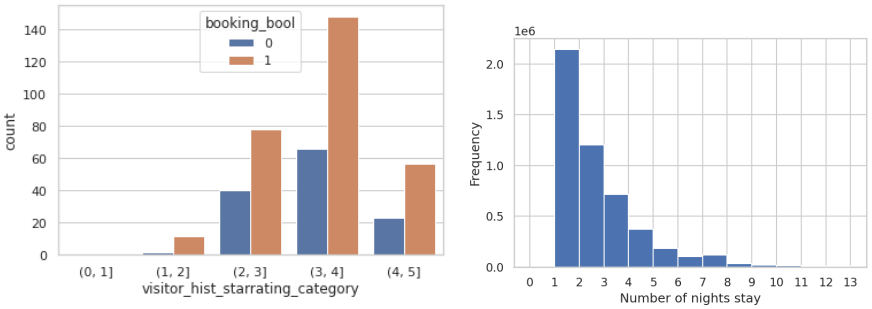

Fig. 5. Left : Historical star rating information of the user(Counts are based on unique search ids). Right : Frequency distribution of Number of night stays in the search queries.

Table 1. Data Statistics for Clicked/Booked Hotels

	Starrating	Review Score	Location Score(2)	Price-USD	Gross-USD
25th percentile	3.0	3.5	0.042	84.71	124.0
Median	3.0	4.0	0.119	118.0	218.4
75th	4.0	4.5	0.257	169.0	429.79
Max	5.0	5.0	1.0	3779565.0	159292.38
Mean	3.33	3.89	0.181	291.91	386.28

We decided to split the column date time into the columns hour, day of the week, month and year and analyse what kind of effect they had on the clicking and booking of the searches which resulted in the following graphs. In Fig. 6 it is shown that for every day of the week whether a given property was clicked on or booked.

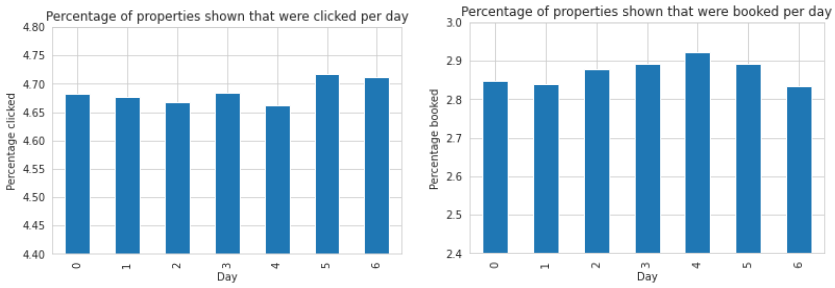

Fig. 6. In the left figure the percentage of the properties that were shown that were clicked on for every day of the week is seen, where 0 is Monday and 6 is Sunday. In the right figure a similar figure can be seen, but instead of clicked on properties it is about booked properties

Although the difference is not that significant we can see in Fig. 6 that in the weekend a larger percentage of the shown properties are clicked on and on Friday a larger percentage of the properties are booked. In fig 7 the percentage of booked and clicked on properties are shown per hour.

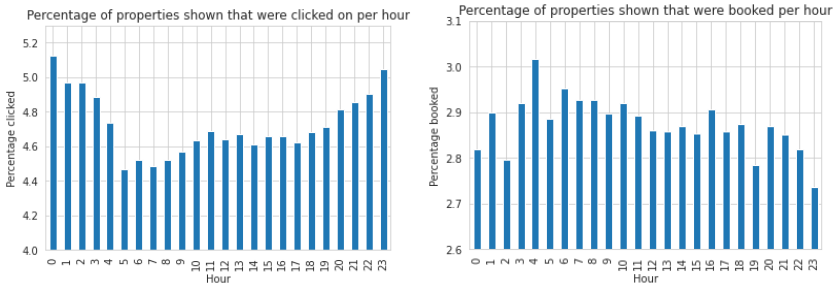

Fig. 7. In the left figure the percentage of the properties that were shown that were clicked on for every hour is seen. In the right figure a similar figure can be seen, but instead of clicked on properties it is about booked properties

In Fig. 7 it can be seen that in the evening and midnight a higher percentage of properties are clicked on then during the afternoon, while the highest percentages of bookings are in the morning. In Fig. 8 the percentage of booked and clicked on properties per month can be seen.

In Fig. 8 it can be seen that in the spring and summer months a higher percentage of properties are clicked on. The months of May and June are the months that users are most likely to book a property. The percentage of booked and clicked on properties between 2012 and 2013 was not significant and therefore we removed that feature. It should be noted that the difference between clicking

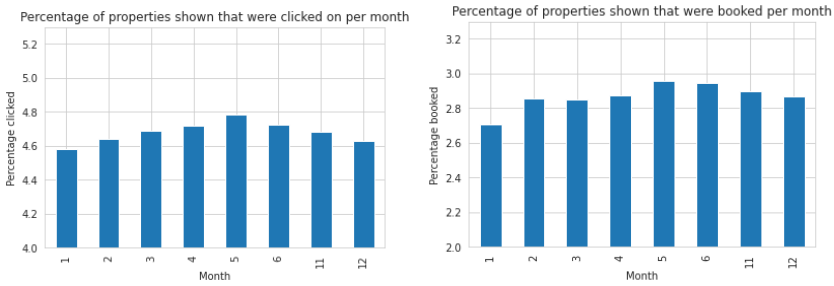

Fig. 8. In the left figure the percentage of the properties that were shown that were clicked on for every month is seen. In the right figure a similar figure can be seen, but instead of clicked on properties it is about booked properties

and booking between the day of the week, the months and the hours are not that large and it is questionable if it will affect the model.

3 Data Preparation

3.1 Handling NA values

For data pre-processing we start with analyzing empty values in the feature columns. The percentage of empty rows per column can be seen in Fig. 9.

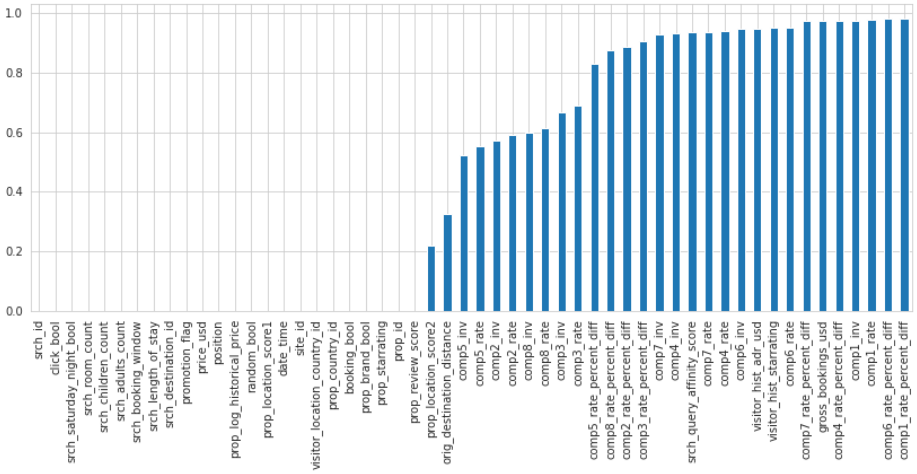

Fig. 9. Percentage of missing values per column

Inspired by an approach used by one of the wining teams in the original Kaggle Competition 2013, we plot the features - "prop_review_score" and

"prop_location_score2". We observe that in Fig.10, a huge fraction of the hotels clicked/booked have some value in the "prop_review_score" and "prop_location_score2" feature columns.Therefore, user is more likely to book or click on a hotel with non-missing values for property review score and desirability of the hotel's location. Hence, fill the missing values in these columns with worst case scenario and that is 0.

Fig. 10. Left : Plot of features with empty(labeled "na") and non-empty(labeled "not na") values in the training dataset for Clicked Hotels. Right : Plot of features with empty(labeled "na") and non-empty(labeled "not na") values in the training dataset for Booked Hotels.

We replace NA for visitor_hist_starrating and visitor_hist_adr_usd with median of their non-NA values. The rest of the the missing values were just kept and the final lightGBM model is able to handle those values by entering 0 for the missing values. Furthermore we removed the columns: position, the booking_bool and click_bool for the training data. Since the first was not present in both the training and test data and the last two were used as a metric to train the model, which is further described in the next section.

3.2 Derived Features

As earlier mentioned in the Exploratory Data Analysis section we separated the Date Time column into three columns: hour, month and day of the week.
Feature "visitor hist starrating" has around 95% empty instances in the training dataset so we create a new column "hist starrating bool" that is given a value of 1 if the search contains a numerical value in "visitor hist starrating" else we assign 0. We use "price rank" feature idea from one of the winners from kaggle competition [1], "price usd" is ranked in ascending order within each unique "srch id". This features gives a personalization for the searches.
"price diff" captures the difference between "visitor hist adr usd" and "price usd". This reflects the variation in price of recommended hotel from historical purchase of the user. Similarly "price diff rank" is a rank feature derived from "price diff". When we fit out model with these additional features we find that

"price diff","price diff rank","price rank" have high feature importance for our model(XGBoost and lightGBM). Some of the other features that we tried but didn't improve the accuracy of the model are -

- "fee per person" : we calculate the "price usd" per individual and number of individuals is summation of adults and children in the search query. We think this could affect the choice of hotel for the user and if "fee per person" is small then it could be preferred by some customers.
- "prop location score 2" / "prop location score 1" : Both these features are related to desirability with respect to hotel's location.
- "prop location score2" * "srch query affinity score" : From table 1 we see that median and 75th percentile of "prop location score2" are 0.1191 and 0.2573 for purchased or clicked hotels. Therefore, this implies that hotels with low "prop location score2" were preferred by most of the users. We filled NA values in "srch query affinity score" with -1 i.e. probability a hotel will be clicked = 0.1, which is low. Therefore, "prop location score2" * "srch query affinity score" with values close to 0 could be preferred by the customers.
- Evaluate "clicked percentage" and "booked percentage" for unique property ids from training data. Match common property ids between train and test dataset. For hotels that are new in the test data, fill them with mean of "clicked percentage" and "booked percentage". This feature drastically reduced the accuracy of the model to 0.28. It might have overfit the data.

The last used derived feature is the relevance grade, which is built up as following:

- 5 The user purchased a room at this hotel. This information could be obtained from the feature 'booking_bool'.
- 1 The user clicked through to see more information on this hotel. This information could be obtained from the feature 'click_bool'.
- 0 The user neither clicked on this hotel nor purchased a room at this hotel. This information could be obtained by the fact that the property was neither booked or clicked on.

This feature was used to train our model with.

4 Modeling and Evaluation

4.1 Random Forest Regression Model

A random forest regression model [2] is used to predict the probability of clicking and booking a property (hotel) by the customer (search_id). The model is trained twice, once for click probability and then for booking probability. The training data has an unbalanced class distribution - 4.736.468 instances were neither clicked nor booked and only 221.879 instances were clicked or booked. Therefore, we balance the training data. With basic feature engineering of filing missing NA values for "prop review score" and "prop location score2" with 0, we test out

the model and get an accuracy of around 0.31. But, this limits the number of training instances for our model. Hence, we explore XGBoost and LightGBM for further analysis.

4.2 XGBoost

The general idea of the gradient boosting is that it is an ensemble method, which works as following: it first starts fitting an initial model and then a second model that focuses on accurately predicting the cases where the first model did perform poorly. In the case of XGBoost and lightGBM these models are decision trees. The combination of these models should perform better than the first model alone. This process of boosting is repeated multiple times. The trick in Gradient boosting is that the nodes of every decision tree have different subset of features when they select the best split and therefore the individual models are different and can capture different signals in the data. Gradient boosting relies on the fact that the combined boosted ensemble plus the new next model minimizes the prediction error. How XGboost exactly works mathematically is explained in more detail in the following paper Xgboost: A scalable tree boosting system. [3] With the gradient boosting algorithm XGBRegressor with objective rank:pairwise we achieved a highest score of 0.35692.[4]

4.3 LightGBM

The final model we used was LGMRanker of the lightGBM library.[5] Since this is an applied data mining course we will not delve into the mathematics of this algorithm, but how it exactly works is described in the following article LightGBM: A Highly Efficient Gradient Boosting Decision Tree.[6] Both XGboost and lightGBM use gradient boosting and therefore are similar. The most important difference between the gradient boosting of XGBoost and lightGBM is that the trees grow leaf-wise instead of level-wise. The split is done on the leaf node that has a higher loss. The difference between both algorithms can be seen in Fig. 11.

Fig. 11. In the left figure level wise tree growth is shown, which is used in XGboost and in the right figure leaf wise tree growth is shown which is used in lightGBM.

The decision trees of lightGBM can create more complex trees by following the leaf wise growth compared to the level wise growth, which results in a higher accuracy. However, it can also lead to over fitting, which can be avoided by

setting the max depth parameter. With this model we got our best score on the test data, which was: 0.38427. This score was achieved with the standard parameters of the model except for the n_estimators, which we changed from 100 to 300. One of the nice features of lightGBM is that you can plot the importance of the features that were used in the model which can be seen in Fig. 12. We also tried to remove the least important features (< 50) to see if this would improve our score, but it did not.

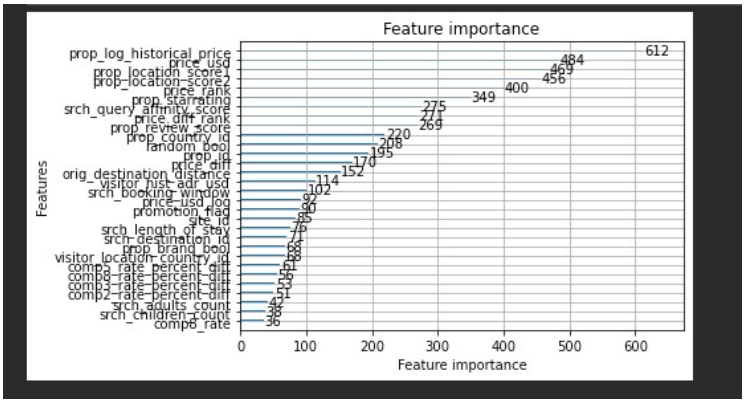

Fig. 12. The importance of the features of the lightGBM model

4.4 Hyperparameters Used In The Model

As per the lightGBM documentation[5], some of the hyper-parameters of the model can be fine-tuned in order to achieve accuracy or reduce the chance of over-fitting. In particular, to achieve better accuracy the following parameter changes can be made:

- Use large max_bin (may be slower). With lightGBM, each continuous numeric feature is split into discrete bins, thus creating a histogram. It then finds the optimal split point while creating a weak learner. Hence,large max_bins improves accuracy whereas small max_bin causes faster speed.

- Use small learning rate with large num_iterations. The number of iterations in the model controls the the number of boosting iterations - i.e how many trees to build. The more trees we have, the more accurate the model is. The learning rate generally determines the step size at each iteration while moving toward a minimum of a loss function.

- Use large num_leaves (may cause over-fitting). According to the documentation, this is one of the most important hyperparameters. It sets the maximum number of leaves each weak learner has.

Furthermore, the reduce over-fitting the documentation proposes the following:

- Use small max_bin
- Use small num_leaves
- Try lambda_l1, lambda_l2 and min_gain_to_split for regularization. This parameters help to combat over-fitting.
- max_depth to avoid growing a deep tree. This parameter control the maximum depth of each trained tree and will have impact on training time and the best value for the num_leaves parameter.

the default parameters of the model in terms of number of estimators is 100, and a learning rate of 0.1. Furthermore, the default num_leaves is 31 while max_bin is 255. We wanted to increase the accuracy of the model, so we began by quick trial where we choose a large number of estimators$(10,000)$ with a small learning rate of 0.001. Running the model took significantly longer, as expected, and the results on the test data-set where not improved, and even slightly worsened compared to the default.

We then tried a different approach: Using an external library to optimize the hyper-parameters. After conducting research on the topic, we found a library called FLAML - A Fast Library for Automated Machine Learning Tuning. As input, we choose the exact same model and metric to evaluate. The result that came were: Best hyperparmeter config: 'n_estimators': 154, 'num_leaves': 4772, 'min_child_samples': 16, 'learning_rate': 0.03175026, 'log_max_bin': 9, 'colsample_bytree': 1.0, 'reg_alpha': 0.010431886, 'reg_lambda': 0.18887 Best r2 on validation data: 0.7779. This achieved a similar result as before.

While this did not improve our score on Kaggle, it may have made to model more general and less prone to over-fitting.

4.5 Evaluation of model created

As opposed to the more common tasks of regression or classification used in Machine Learning, for this particular problem we were interested in the prediction of ranking of hotel entries. In order to evaluate the model we used the *Normalized Discounted Cumulative Gain* which would produce a rank for each hotel entry. In this metric, we used the following function to evaluate *relevance* for each hotel entry:

$rel_i \leftarrow 0$
if $booking\_bool = 1$ **then**
    $rel_i \leftarrow 5$
else
    if $click\_bool = 1$ **then**
        $rel_i \leftarrow 1$

end if
end if

From [7], we know the traditional formula of DCG accumulated at a particular rank position p is defined as:

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{1}$$

However, since the size of results lists may vary in length, the standard DCG cannot be used on its own to compare search results. For address this issue, we use the normalised discounted cumulative gain, or nDCG, is computed as:

$$nDCG_p = \frac{DCG_p}{IDCG_p} \tag{2}$$

Where

$$IDCG_p = \sum_{i=1}^{|REL_p|} \frac{rel_i}{\log_2(i+1)} \tag{3}$$

is known as the *Ideal discounted cumulative gain*

5 Discussion

With around 5 million occurrences, the training data set is quite large. We learned how to work with a large dataset. For example - we had to optimize our algorithms that build a new feature based on the computation of other feature columns. Our system crashed when we employed brute force.

We learned how to explore data, create meaningful graphs and deduce information mathematically and graphically. Data exploration can sometimes reveal aspects that may have an impact on the target variable. We realized that this is a crucial phase in the problem-solving process. When researching the Expedia recommendation project, we looked into the winning teams' solutions as well as other related discussions. That aided us in determining how to approach feature engineering in terms of managing NA values, and extracting distinct features in a useful manner. We'd like to investigate the use of aggregate functions in feature engineering in the future.

As we discovered, features that are critical for one machine learning model may not be as important for another. Furthermore, we learned about different models to use for ranking problems, namely two of the most important ensemble machine learning algorithms: random forest and gradient boosting. The gradient boosting algorithms proved to be the best for the way we prepared our data and especially the lightGBM model, which had two advantages: the accuracy and the speed.

References

1. "kaggle-winners." https://www.kaggle.com/competitions/expedia-personalized-sort/discussion/6203.
2. G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, no. 2, pp. 197–227, 2016.
3. T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.
4. "Xgboost model documentation." https://xgboost.readthedocs.io/en/stable/python/python_api.html.
5. "Lightgbm model documentation." https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMRanker.html.
6. G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.
7. K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Transactions on Information Systems (TOIS)*, vol. 20, no. 4, pp. 422–446, 2002.