Assignment 2 Data Mining Techniques

Christos Petalotis[1 [2739394]], Julia Wesselman[2 [2715996]], and Matteo De Rizzo[3 [2749303]]

Vrije Universiteit Amsterdam, 1081 HV Amsterdam, Netherlands
c.petalotis@student.vu.nl
m.de.rizzo@student.vu.nl
j.a.wesselman@student.vu.nl

Introduction

The assignment is to participate in the following competition on Kaggle: .In this assignment data about hotel bookings is analysed to predict what hotel will be booked by a user of, for example, Expedia. If a site has the best ranking of hotels for a user the most likely the user will book a hotel through that site.

Section 1 the Business Understanding. Section 2 the Data Understanding. Section 3 the Data Preparation. Section 4 the Modelling and Evaluation.

1 Business Understanding

In recent years, numerous data mining competitions have been held to predict the hotel a particular user will book. The goal is, usually, to help companies rank the hotel results they suggest to each particular user on their site. In fact, this is done, as a better ranking leads to more bookings. In this section we are going to focus and analyze the following two competitions from Kaggle: the Expedia Hotel Recommendations competition and the original Expedia competition. More specifically, the methods and models of the winners of those competitions will be explored.

In both competitions different variants of Tree Boosting are used. Tree Boosting is a widely used machine learning method [1]. It combines weak learners, like decisions trees, into a strong classifier [2]. Tree Boosting is effective in many application, like ranking problems and predictions [1], [2]. The variants of tree boosting used, XGBoost, Gradient boosting machine (GBM) and the LambdaMART, will be explained in the analyses of the competitions below.

The competition 'Expedia hotel recommendations' was set up to predict the likelihood a user will stay at a hotel group. The participants must predict a list of the hotel groups that will be booked for every user, based on their search. For this competition only the method of first place, 'idle speculation', is known. Most importantly, 'idle speculation' used factorization machine submodels for the interactions between categorical variables with a large number of distinct values and turned it with 'bursting' into an XGBoost 'rank:pairwise' problem. The XGBoost is a scalable machine learning system for tree boosting giving state-of-the-art results on many problems [1].

The competition 'Expedia personalized sort' is used as base for the competition of this assignment. The goal of the competition was to build a ranking model using the data. The approaches of the winners will be discussed, the 1st place: Owen Zhang, 2nd place: Jun Wang and 4th place: Bingshu.

First of all, all participants pre-processed the data. Pre-processing of the data is needed, because, for example, in the data there are missing values, the data is unbalanced, the data set is too big for some models and some features have non-normality. How the data is pre-processed depends on the model that is used. The winners used different variants of Tree Boosting, in other words different ranking algorithms are used. Owen Zhang used the GBM. According to Friedman (2001) [3], ' Gradient boost-ing of regression trees produces competitive, highly robust, interpretable procedures for both regression and classification, especially appropriate for mining less than clean data'. Jun Wang used the LambdaMART algorithm. 'The LambdaMART is the boosted tree version of LambdaRank' (Burges, 2010) [4]. It is a succesfull algorithm for ranking problem, like in this assignment. Bingshu compared the different models, that could be used for the ranking problem. Bingshu concluded that only using a single model can not achieve the best result.

2 Data Understanding

The provided dataset comprises two parts, one training set and one test set. The training set has 4958347 instances, whereas the test includes 4959183 rows. Each instance represents a different search result, regarding hotel characteristics and search query attributes. The number of different features on both sets is 50. These attributes can be grouped in 5 different groups, based on the object they refer to. The categories are:

- Search query criteria
- Static hotel characteristics
- Dynamic hotel characteristics
- Visitor information
- Competition information

There are also some attributes that are only present in the training set, which are the position of a hotel on Expedia's search results, whether a property was booked, whether a property's search result was clicked and the total value of a transaction for a booked property.

These features include both categorical and numerical values.

Focusing on the training set for this task, after retrieving some basic statistics about it, we realize that there is a very high number of missing values. This is especially true in the case of the features that provide information about the competitors of Expedia in terms of price and availability for hotels on the search results. The average number of missing values for this type of data is 4,089,793. In most of these cases, around 90% of the data is missing.

Similarly, the are over 4 million missing values for the attributes related to a customer's booking history and the probability that a hotel will be clicked on Internet searches. The features about a second score for the desirability of a destination and the physical distance between the destination and the customer during the time of the search also contain a high number, of more than 1 million, of missing fields.

The way all these, and those of some other features, missing values are handled is discussed in section 3.

Moreover, we see that the number of search results in the training set that did not lead to a user click - which we call negative results - is disproportionately larger than the number of the results that led to a click. In other words, the dataset is not balanced between positive and negative instances. Particularly, 95% of the search results were not clicked by the user. In order to avoid any model trained on this data being more "favorable" to the negative instances, thus almost always predicting that a hotel was not booked, we will explore downsampling the negative instances. Initially, we decided to increase the percentage of the positive results to occupy 40% of the final dataset. Based on the results of this action in the prediction phase, we will adjust this percentage if needed.

In general, we expect for this balancing action to have a positive impact on the speed with which the preferred model will be trained and on the predictive performance that will result from this training.

Exploring the dataset in more detail, we realized very quickly that from the results that were clicked, 62.4% of them led to a subsequent booking of the property explored, while the remaining 37.6% of the clicked results were not booked by the user. This behavior was expected and it shows that there is a great correlation between the click and book features, indicating that a clicked result has already more probabilities to get selected for a booking.

Important conclusions can also be drawn when looking at the effect of a sale price promotion on a search result on the probability that a result will be booked and/or clicked. More specifically, we found that in general, a promotion is applied to properties with higher prices, and increases the probability that the hotel will be clicked and/or booked. When a search result wasn't booked, only 2.4% of them included a promotion, while the same rate was 3.9% for the booked searches. The difference between these two percentages becomes more prominent when taking into account the imbalance between the negative and positive rows in the dataset. Even though a greater number of non-booked entries would decrease the percentage of entries with a promotion applied, if the amount of promotions offered remained the same, it could also increase the chance a larger part of them had a promotion tag. However, this is not the case. A similar situation is true for the entries that were clicked and not-clicked, with 4% of the not-clicked entries having no price sale promotion, and 6% of the clicked ones being displayed with a promotion on them.

Bringing the length of a booked stay into the picture, analyzing the data shows that the length of stay is also affected by the existence of a price promotion on a property. On average, the length of a stay was almost 0.6 days longer when

a promotion was applied than when it was not, with stays of length of 1.9 days being booked on average for properties with no sale price promotions and 2.5 days for hotels on which a promotion flag was set. Also, as mentioned before, it is realised that promotions are generally applied more often to higher priced properties than lower priced ones. Despite this fact, when a property was booked, the average final price of a property appears to be higher when no promotion was present. The average price of a search result that led to a new booking with no price sale was $263.22, while the new bookings that included a price discount had an average price of $254.28.

Regarding the price that customers saw for the different search results that were returned after their queries, the mean was $254.20, while the max price was $19,726,328 and the lowest price was $0.

The fact that the highest price in the dataset was 19.726.328, with 75% of the data having a maximum price of $184, led us to believe that the data is not calculated always in the same way, and wrong data about the price might be present in the dataset. In fact, regulatory conventions in different countries, in which property destinations can be, could lead to uncscaled prices for some of the instances. These disparities could lead to some of the entries' price to include taxes and fees while in other cases these are not indicated by the listed price. It could also be the case that the price indicates the total booking price, instead of the price per night.

Exploring the listed prices of the search results we realize that our initial assumptions may not be correct. Looking at figure 1, which plots the log price for the different visit lengths in the dataset, we see that most of the outliers are present for short-term stays. On the other hand, longer stays show less outliers, even though we would expect for this to be the case when the price for multiple days is calculated as a whole, thus increasing the chance that the number of outliers is larger. The log of the price attribute was used to normalize the data and make any relations between the different entries for this feature more clear, while reducing the amount of outliers in the data.

One of our assumptions is that some errors might have happened during the data logging process, resulting to multiple abnormal values for the price feature in the dataset. Property prices of $0 could also be explained this way, as we don't expect any of the properties to have been offered to customers for free.

We, then, explored the importance of the date and time that a search was performed on. We assumed that there may be some connection between the time of the year a search is made and the percentage of the booked results.

Looking at figure 2, which plots the percentage of search results that resulted in bookings throughout the time span the dataset covers, we conclude that the time a search is performed greatly affects the booking rate. The behavior of the customers fluctuates throughout the year, as different peaks can be noticed in the graph. The fraction of searches resulting in bookings shows sharp increases for the months of March and June. We further explored the dataset, to understand the effect that the visitor's location country and the destination country might have on the fraction of booked searches.

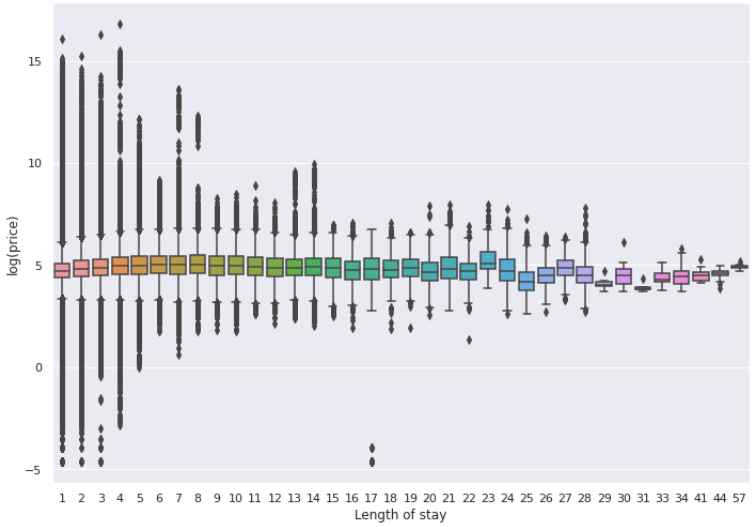

Fig. 1. Price outliers are the highest on shorter stays and decrease as the number of days of stay increases

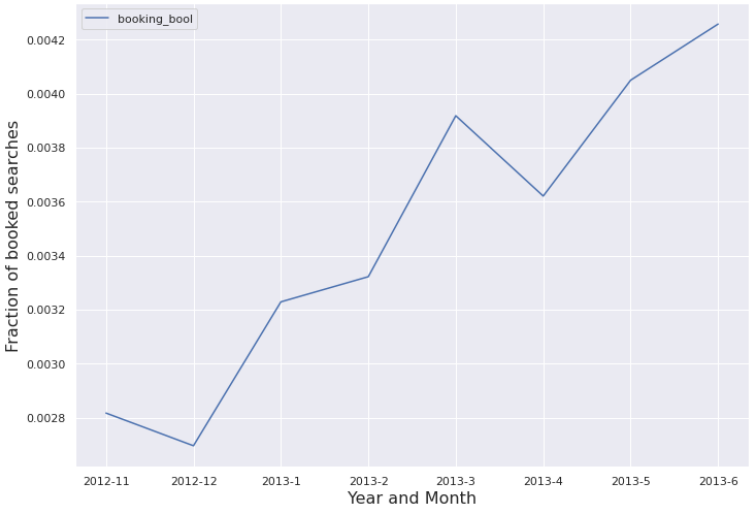

Fig. 2. The fraction of search results that led to bookings has some peaks and dips throughout the year

Table 1 shows that the majority of booked searches originates from customers that are located in country with id 219. As a result, we expect that the trend depicted in figure 2 is highly dependant on the behavior of customers from country 219, and that it approximates those customers' booking trends.

Furthermore, looking at table 2 we realize that the country a hotel is located in might also have a strong effect on the probability of booking too. Particularly, country with id 219 appears to be a great destination for which customers book hotels at as well. Thus, when a hotel is located in this location it is expected to have higher probability of being booked by a user.

Visitor Location Country ID	Booked Fraction
219	0.591
100	0.099
55	0.058
216	0.047
220	0.031

Table 1. Most Frequent Visitor Location Country IDs

Property Country ID	Booked Fraction
219	0.613
100	0.067
55	0.039
31	0.028
99	0.026

Table 2. Most Popular Destination Property Country IDs

We arrived to the conclusion that the date, and more specifically the month of the year, that a search is made provides a strong indication of the traveling behaviors of users of the Expedia site. Therefore, in the next section, we are going to split the date_time attribute of the original dataset to day, month and year, and eventually keep only the monthly component. We will also explore whether having an indication of a visitor's country id and a property's country id can improve the predictions that are made by the selected model.

Lastly, we examined if a user's booking history on the Expedia site could provide any indication about whether a certain search result is going to lead to a booking. The historical data in the dataset, shows that the average price that a hotel is booked for is $176.02, with the average star rating of those hotels being 3.37 starts. Moreover, as we expected, the customer history data shows that higher star ratings of hotels booked in the past were accompanied by higher prices that a user paid in each case. A similar conclusion regarding the price and star rating relation, can be drawn by looking at figure 3, as for the search results that led to a booking, a higher star rating was usually followed by a higher price paid.

3 Data Preparation

In this section we explain the actions we performed to prepare the data for use in the modelling process, which is explained in section 4. These actions will be performed on both the training set and the test set. However, to avoid some

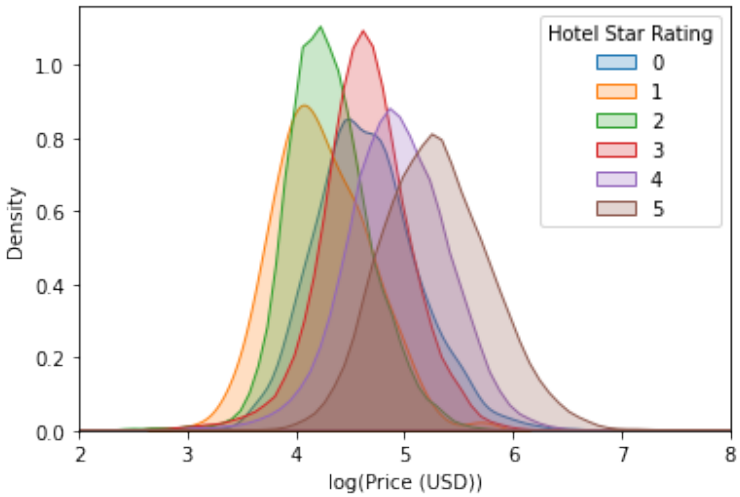

Fig. 3. Higher starred locations in general are booked for higher prices

data "leakage" from the training set to the test set, any missing values on the test set will be adjusted using only information coming from the training set.

The data preparation actions we conducted include Data Cleaning and Feature Engineering.

3.1 Data Cleaning

In the data set there are a lot of missing values. According to Peng and Lei (2005) [6] (from Acuna & Rodriguez, 2004) : 'If the rate of missing is less than 1%, missing data won't make any trouble for the Knowledge Discovery in Databases (KDD) process, 1-5% manageable, 5-15% requires sophisticated methods to handle and more than 15% may severely impact any kind of interpretation'. In the provided dataset there is a lot of missing data. To analyse this data it must be 'completed'.

In this research the missing values are handled as follows. At first, all the missing values in the Expedia competitor related features, such as comp1ratepercentdiff, comp1rate and comp1inv, are set to zero. This is done as we don't want to introduce any bias towards one of the two sides, Expedia or a competitor, and a value of 0 indicates no difference between the two parties. Also, because of the fact that more than 90% of the values of these features are missing, replacing these missing values may lead to results that are do not represent reality. Thus, we decided that we will run our model with and without these features in the dataset to see what works best.

Secondly, the median of the corresponding features is assigned to the missing values of orig_destination_distance and prop_review_score. The missing values in these cases mean that there is no available data, thus we prompted to impute

them with the median. Specifically for the review score attribute, a null value could mean that the corresponding property is relatively new, thus not actual data has been gathered about its reviews yet. The median metric is used instead of the mean because it is more robust to outlier values.

Next, the lowest value is assigned to the missing values of srch_query_affinity_score. The same is done for the prop_location_score2 attribute, setting the missing values for this feature to the minimum of the corresponding srch_destination_id, as we cannot be aware of a property's location desirability, so we decided to assume the worst case.

Regarding the historical data of visitors on the site, the mean of each feature will be use to impute the missing values, as we assume that, in general, a new customer will behave similarly to the majority of the other, known customers, of the website, even if there may be some deviations from this behavior in practice.

Furthermore, we decided to remove the feature related to the total value of a transaction (i.e. gross_booking_usd), as this information is not included in the test set, thus we don't expect for it to be useful during the prediction process. Only the price of a property will be used, as there is, also, a high correlation between those two attributes.

Similarly, we removed the site id, property id and search id attributes as their importance was decided to be low for the training of the selected model.

Lastly, we performed some normalization actions on the price and the property star rating attributes based on the search and property ids of the relevant search results. This was done to eliminate any outlier values that could negatively influence the predictive ability of our model, while facilitating the process of comparing and generating connections between them [14]. When this process was done, we decided to drop the original price_usd feature because of the inconsistencies in the way it is logged.

3.2 Feature Engineering

After replacing all the missing values in the dataset, some new features were generated that are believed to lead to a better performance while ranking search results.

More specifically, as mentioned in section 2, we used the datetime feature to derive the day, month and year that each search was performed on, and we added the *month* feature in the dataset.

Then, we made some hypothesis regarding the behavior of the site visitors in order to produce new features in the dataset. We started by assuming that users tend to pick hotels that are either cheaper or have similar prices compared to other hotels at first glance. In the case that some desired results are similarly priced, some differential factors that can make a customer lean towards a certain search result are the reviews that the corresponding property has received, the star rating of it and also the desirability of its location. Following this logic, we generated the following features:

- pricediff: difference in price between a hotel and the average price in results

- reviewscorediff: difference in reviews between a hotel and the average reviews in results
- stardiff: difference in star rating between a hotel and the average star rating in results
- propscore1diff: difference in location score 1 between a hotel and the average location score 1 in results
- propscore2diff: difference in location score 2 between a hotel and the average location score 2 in results

Next, we made the hypothesis that customers usually tend to select hotels that match their history of choices. More specifically, we assume that a customer would be more interested in booking a hotel that is priced similarly to the ones that they booked in the past, than ones that deviate from that price. Similarly, for the star rating of a hotel, users would gravitate towards hotels with star ratings close to the ones they have booked before. For this reason, we produced a new attribute to show the absolute difference between a visitor's purchase history in terms of price and a certain search result's price (histpricediff), and another one to show the absolute difference between the customer's mean star rating of hotels booked in the past and a certain result's such rating (histstardiff).

Furthermore, a hypothesis was made the popularity of a hotel could be derived from the changes in its price. A more popular hotel means that there is a higher probability that it will get eventually booked, thus the corresponding search result should get a higher score. For this reason, a feature called *pricechange* was generated that gives us the difference between the current price of a hotel and the mean price of the same property over the last trading period. As the the mean price is provided as a logarithm, we used the log of the current price to create this feature too.

Lastly, after exploring different options and looking at some of the winning approaches for the original Expedia competition [15] we generated some numerical features that were averaged over the search id, the id of a property and also the destination id. Particularly, we grouped over the srch_id attribute and created new features for the star rating of a property, the desirability scores 1 and 2, separately, for a property, the indication of a promotion flag on a search and a property's review score. These new features were produced using the mean each time. We also grouped separately by prop_id and srch_destination_id to generate a mean feature for the price of each property.

4 Modelling and Evaluation

So far, we have gained a thorough understanding of the importance of the different features present in the provided dataset and the main connections between certain of those features. We have, also, prepared the data for a model to use them as a training means to improve its ability of predicting which search results a user is most likely to click on on the provided test set.

All these steps are necessary for us to be able to make the best selection of a Machine Learning (ML) model that is able to achieve our goal of ranking hotel

search results in the best way possible. In this section, we discuss the two ML models we chose to use for this competition, their configurations, performance and, finally, the score and leadership position they led us to acquire on the competition leaderboard on Kaggle.

4.1 Gradient Boosting Machine

Firstly, the fact that we were more familiar with the notion and usage of classifier models, made our initial approach include the use of the Gradient Boosting Machine classifier. This type of classifier uses Gradient Boosting, a type of ensemble ML model which is constructed from decision tree models, added one at a time to the ensemble, and using the error on the previous models to correct the prediction errors using a gradient descent optimization algorithm [10].

Particularly, we made use of the Light Gradient Boosting Machine (LGBM) Classifier functionality offered by the LightGBM Python module [8]. This is a light-weight, yet powerful implementation of the GBM model, that uses less memory to run and can handle vast amounts of data [9]. As the provided train and test sets include almost 5 million rows each, we consider this implementation of the Gradient Boosting Machine model appropriate to handle the big load of data.

Another classifier that we considered using for the modelling process was the Random Forest algorithm, which creates multiple trees from the provided data, and can also provide strong performance. However, we prompted for a Boosting solution instead, because we consider the ability of the model to learn from previous errors and improve as something that can positively influence the quality of the final predictions.

Using the LGBM Classifier, we started the training phase by looking for the best configuration possible, in terms of hyper parameters of the model. For this reason, we split the provided train set into two subsets - a (new) train set and a validation set, with each one being 70% and 30% of the initial training set, respectively. After that, we used stratified folds on the new training set to create 5 splits that are then used in a grid search operation with the goal of realizing the optimal combination of model hyper parameters for our problem. Once this process was complete, and the suggested parameters were extracted, we created a new instance of the LGBM model using these parameters. Some of the hyper parameters that we used are a number of 2000 estimators, with the number of leaves for the base learners being 12 and the learning rate at 0.11.

Next, this model was trained on the training split of the initial train dataset, and then evaluated on the evaluation set we created earlier. The feature that was used as the target for the classifier to predict for each search result is the booking_bool attribute. More specifically, the model was configured to predict the probability of a result having a booking_bool attribute set to 1. Then based on this probability, we sorted the results in descending order.

As the Kaggle competition uses the (NDCG)@5 metric for evaluation of the submissions, we prompted to measure the performance of the trained LGBM

model using this metric as well. The best score we managed to achieve using this algorithm locally was 0.51.

We made several attempts to train and improve the score that this model gave us, using different combinations of features. The run of this model that gave us the best score used the entirety of the dataset for the training, while having the missing values imputed and the newly created features (discussed in section 3) included as well.

4.2 LambdaMART

The fact that the goal of the competition is to provide a ranking for the search results displayed to a certain user, having to recommend the ones with higher chance of being booked to be placed higher in the list of results, quickly led us to realize that a classification model may not be the best solution for this case. For this reason, we prompted to use a ranker model, that is able to better perform predictions that rank entries based on their features.

More specifically, we utilized the Light Gradient Boosting Machine Ranker functionality provided, again, by the LightGMB Python library. This deploys a Learning to Rank (LTR) supervised Machine Learning model, the LambdaMART model, a boosted version of LambdaRank, which is based on RankNet [11]. Such model is appropriate for solving ranking problems on a list of items with the goal of ranking those items. The most common application of a Learning to Rank model is the ranking of search results, which makes it appropriate for our case [13].

Before using the LambdaMART model for training on our training set, we split the set into a (new) training set and an evaluation set. In this case again, similarly to what we did for the LGBM Classifier, we prompted for a 70-30% split accordingly. However, this time we used a group shuffle split [12] method to perform the splitting, as we deemed it necessary that instances with the same search id are not spread throughout different splits .

Once this was done, we trained the LambdaMART model and performed evaluations using different configurations. The version of model that gave us the best score locally, and also on the competition leaderboard on Kaggle included the hyper parameters of 3000 estimators (or number of boosted trees to fit), a random state of 65, and a learning rate of 0.1.

Moreover, multiple training datasets were used for the training of this model too. The dataset that worked the best for us, meaning that resulted in the best performance on the competition, had a total of 69 features, including the data about Expedia's competitors and all the features we engineered, and discussed in section 3.

Using this type of model, the goal is to assign a score to each search result in the test set, with a higher score meaning placing the entry higher in the list of the final search results to the user. For this purpose, and to align with the evaluation logic of the competition, we added a relevance score feature to each instance in the training set, and set this feature as the target for the model during the prediction phase, performed on the test set. More specifically, when

a search result in the training set is marked as booked, it receives a relevance score of 5, if it is only clicked it receives a relevance score of 1, and in the case a certain entry is neither clicked nor booked, then it gets assigned a relevance score of 0.

As the features of whether a search result is clicked, booked or non of them is missing from the test set, the model has to learn how the rest of the features available for each instance are related to this relevance score, and assign the appropriate relevance to each search result present in the test set. The final result is a file containing the search id and the property id combination of each search result, sorted based on the relevance score in descending order. Thus the results that are more probable to lead to a booking for a certain are placed higher in the list, while the ones with less booking probability are displayed further down the search result list.

4.3 Conclusions

As described in this section, we have used two different models to make predictions about which search results a customer is more likely to click on. In order to configure and train these models, we used various hyper parameter combinations, appropriate for each type of model, and training sets in an attempt to get better predictions and ranking of those results.

We concluded that some of our initial assumptions were not true, while other were verified and let to better model performance. More specifically, as mentioned in section 2, we made the assumption that increasing the percentage of positive instances in the dataset, by removing some of the negative ones, would lead to better performance and less bias in the prediction ability of the model. However, we found that this is not true in practice. Even though we got faster training times when we used the reduced dataset, the scores that each model received were actually slightly worse in this case, compared to using all the instances of the dataset. Thus, we decided to use the full dataset, in terms of instances, and this set led to the best score we received in the competition.

Furthermore, in section 2, we mention that we want to explore whether a country id of 219, either for a visitor or a property, would influence the ranking of the search results. This hypothesis was not true though. Each time we run a model on the training set, the importance of this feature was perceived to be one of the lowest in the list of features. Therefore, we came to the conclusion that our assumption was wrong, and we didn't use this attribute in the dataset for the later attempts.

We, also, contemplated the presence of competitor data in the dataset. We run all of our model training processes and predictions once with this data included and once without it. The final verdict was that the competitor data had a positive effect to the performance of the model used.

Regarding the Kaggle competition itself, we made all of our submissions using the predictions that came out of the LambdaMART model. The best score we managed to achieve for this competition was 0.38531 and our final position on the leaderboard was place 67 at the time of writing this.

Bibliography

[1] Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).

[2] Liu, S., Xiao, J., Liu, J., Wang, X., Wu, J., & Zhu, J. (2017). Visual diagnosis of tree boosting methods. IEEE transactions on visualization and computer graphics, 24(1), 163-173.

[3] J. Friedman. Greedy function approximation: a gradient boosting machine. Annals of Statistics, pages 1189–1232, 2001.

[4] Burges, C. J. (2010). From ranknet to lambdarank to lambdamart: An overview. Learning, 11(23-581), 81.

[5] Pratama, I., Permanasari, A. E., Ardiyanto, I., & Indrayani, R. (2016, October). A review of missing values handling methods on time-series data. In 2016 International Conference on Information Technology Systems and Innovation (ICITSI) (pp. 1-6). IEEE.

[6] Peng, L., & Lei, L. (2005). A review of missing data treatment methods. Intell. Inf. Manag. Syst. Technol, 1, 412-419.

[7] LGBMClassifier `https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html` Last accessed on 27 Apr 2022

[8] LGBMClassifier `https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html` Last accessed on 27 Apr 2022

[9] What is LightGBM Algorithm `https://www.analyticssteps.com/blogs/what-light-gbm-algorithm-how-use-it` Last accessed on 27 Apr 2022

[10] How to Develop a Light Gradient Boosted Machine (LightGBM) Ensemble `https://machinelearningmastery.com/light-gradient-boosted-machine-lightgbm-ensemble/` Last accessed on 27 Apr 2022

[11] RankNet: A ranking retrospective `https://www.microsoft.com/en-us/research/blog/ranknet-a-ranking-retrospective/#:~:text=RankNet%20is%20a%20feedforward%20neural,data%2C%20called%20the%20training%20set` Last accessed on 27 Apr 2022

[12] GroupShuffleSplit `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GroupShuffleSplit.html` Last accessed on 27 Apr 2022

[13] Learning to Rank `https://medium.com/@nikhilbd/intuitive-explanation-of-learning\-to-rank-and-ranknet-lambdarank-and-lambdamart-fe1e17fac418` Last accessed on 27 Apr 2022

[14] Why Data Normalization is necessary for Machine Learning models `https://medium.com/@urvashilluniya/`

why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029 Last accessed on 27 Apr 2022

[15] Owen Zhang, Personalized Expedia Hotel Searches – 1st place https://www.dropbox.com/sh/5kedakjizgrog0y/_LE_DFCA7J/ICDM_2013?preview=3_owen.pdf Last accessed on 27 Apr 2022