Data Mining Techniques - Group 116
Assignment 2

Zakaria Ibn Said - 2558176
z.ibnsaid@student.vu.nl

Eveline Rudolphij - 2660306
e.h.d.rudolphij@student.vu.nl

Xingkai Wang - 2668688
x19.wang@student.vu.nl

Vrije Universiteit Amsterdam

22 May 2020

Abstract

In this report, we specify the data mining process model used to compete in the academic version of the Expedia Hotel Recommendations Contest, which was originally hosted by Kaggle in 2013. The goal was to build a model for Expedia that best ranked hotels so that they could harness their chances of winning sales in the competitive Online Travel Agency landscape. The winning participant of the 2013 contest had a score of 0.53984 and we obtained a score of 0.38275. We explain how we obtained this score by (1) developing business understanding of the contest, (2) conducting exploratory data analysis, (3) discussing the features we have modified and added to prepare the data, and (4) explaining how we built our random forest with the LambdaMART model, LightGBM, and evaluating what we learned from the exercise.

1 Business Understanding

The Expedia Hotel Recommendations Contest was hosted by Kaggle in 2013 and commissioned by Expedia. Expedia aimed to obtain code from the contest that best ranked hotels for specific users with the integration of competitors' rates so that they would gain the best chances of winning sales in the competitive Online Travel Agency (OTA) landscape. For the contest, Expedia provided a large complex dataset that included hotels' characteristics, location attractiveness of hotels, users' aggregate purchase history, and competitive OTA information.

In total, 336 teams participated and 3487 entries were made. Participant Owen received the first place in the contest with a score of 0.53984. Participant Commendo had a score of 0.54074, but was disqualified. The participants who received first, second, and third place, respectively Owen, Jun Wang, and Jeremy, Yongua, and Aaron shared their results in Powerpoint presentations on Kaggle.

The presentations show that the most prominent predictors used were the features (estimated) position, price, and location desirability. Owen used an ensemble of Gradient Boosting Machines (GBM) for his prediction, because he argues that ensembles get better results than single models. He trained the models with and without Expedia features 26 times. In conclusion, Owen remarked that down sampling negative instances improved training time and performance.

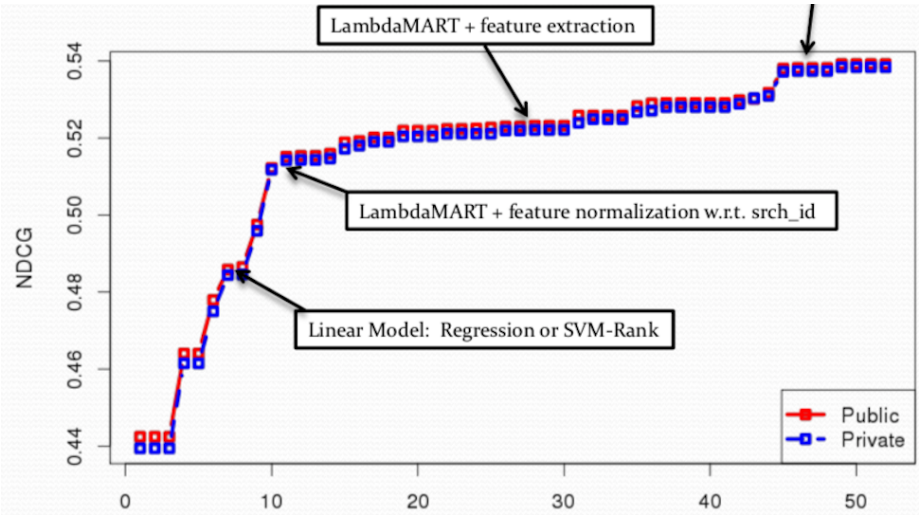

Figure 1: Roadmap Runner-up Jun Wang

The roadmap above shows how Wang received second place with 0.53839, making use of LambdaMART (i.e., a learning to rank algorithm based on Multiple Additive Regression Tree (MART)). Wang argues that LambdaMART leads to better performance in this contest because of its non-linearity and computational efficiency. In regard to feature engineering, Wang stresses the importance of removing redundant features, normalizing features with multiple indicators, and estimating missing values in principle ways. In sum, all winning teams note that there is a strong bias position bias, even for random impressions.

2 Data Understanding

In this section we explore the dataset to identify findings relevant to our task. To efficiently go about the Exploratory Data Analysis (EDA), we first analysed the best practices of the winning teams as were described in the previous section. The Expedia Hotel Recommendation Contest in 2013 taught us that the hotels in the data refer to hotels, apartments, Bed and Breakfasts, hostels, and other properties shown on the Expedia website (Kaggle, 2020). In this regard, room types were not specified and most of the records included searches that resulted in a booking.

Next, we downloaded the test set, training set, and submission set into Kaggle. We started with exploring the training data by looking at the data head and info. As such, we saw that each row represents a search query of a visitor of the website. In this respect, the training and test datasets contain approximately five million records and 54 columns. Upon analysing the categories of the columns, we saw that 24 columns provide competitor information (e.g., price, location, and availability). The other 30 columns provide information about the user, hotel, and booking specifics.

To further explore the data, we created heatmaps that show how the different features correlate. Figure 2 shows a heatmap of columns that provide insights into hotel specifics. We can see that there is a surprising correlation between 'visitorhiststarrating' and 'visitorhistadrusd'. Nonetheless, this can be explained because there are lots of missing values in the two columns that the heatmap mistakes for similar values. Moreover, there is a significant correlation between 'proplocationscore1' and 'proplocationscore2'. This makes sense in light of the fact that both features provide a score outlining the desirability of the hotel's location (figure 2, first).

In addition, we created a heatmap of columns that provide insights into the booking specifics. Figure 2 (last) shows this heatmmap and indicates that 'srchroomcount' and 'srchadultscount' are the only features that significantly correlate within these columns. This makes sense since a higher number of visitors is likely to book a higher number of rooms.

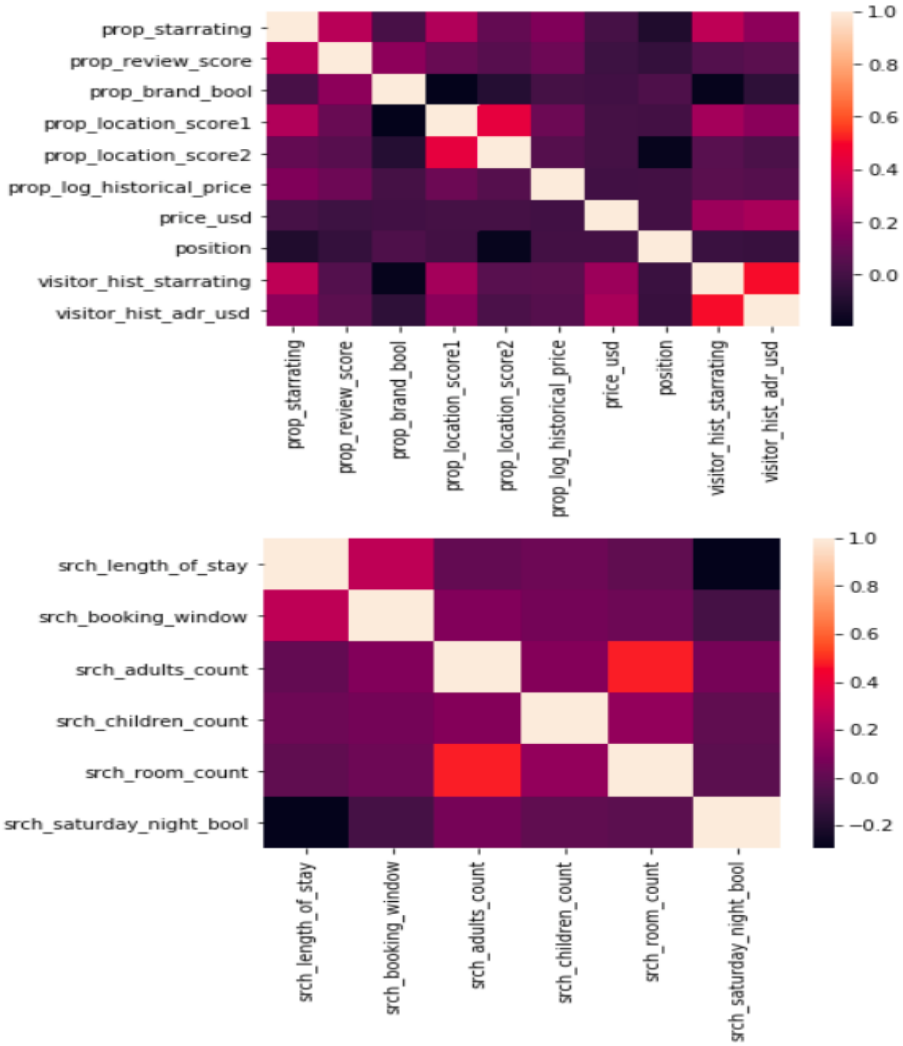

Figure 2: Heatmap correlation hotel(first) and booking(last) specifics

3 Data Preparation

In this section we address the features we have modified and added to prepare the data for modelling.

3.1 Missing Values and outliers

We looked at the null values of the training and test data and identified that there were several columns with many missing values. We are aware that missing values need to be taken of because they reduce the quality of performance metrics. In this regard, missing values may lead to wrong predictions and may cause a high bias for any model used (Jimoh, 2018).

Subsequently, we chose to drop the columns ranging from 'comp1rate' up until 'comp8ratepercentdiff' and both 'srchqueryaffinityscore' and 'grossbookingsusd', since around 80 percent of the data on these columns was missing. We are aware that dropping columns may reduce the quality of the model because the sample size is reduced. However, we chose to drop these columns, as the best practices' presentations did not highlight their importance and they had a significant percentage of missing values.

Furthermore, 'visitorhiststarrating' and 'visitorhistadrusd' also showed missing values, but Wang's presentation showed us that people do not like to book a hotel without historical data. Hence, we considered it useful to keep the columns for analysis. As such, we replaced all missing values with -1, indicating that the hotel does not have any historical information. Moreover, the columns 'propreviewscore', 'proplocationscore2', and 'origdestinationdistance' also showed null values, but we chose to keep them for analysis as the best practices showed us that these were prominent predictors. By importing seaborn and distplotting the remaining columns with null values (e.g., see figure 2), we were able to come up with useful approaches to clean the data. In this respect, we chose to fill all null values with the median values to avoid bias.

In addition, we found out that some features had significant outliers, which could lead to a poorly performing model as they skew the distribution of the features. As such, we decided to remove all outliers that were 3 standard deviations set apart from mean. The figures 4 5 below illustrate this and show the differences among the 'priceusd' and the 'proplocationscore2' with and without outliers.

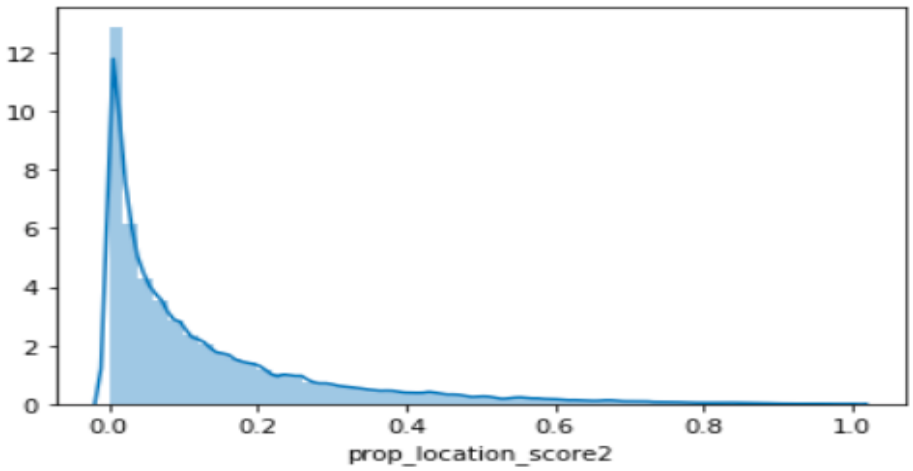

Figure 3: Distplot 'proplocationscore2'

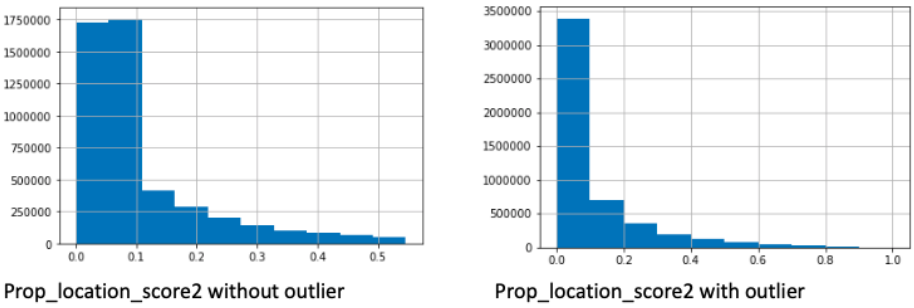

Figure 4: 'proplocationscore2' without and with outlier

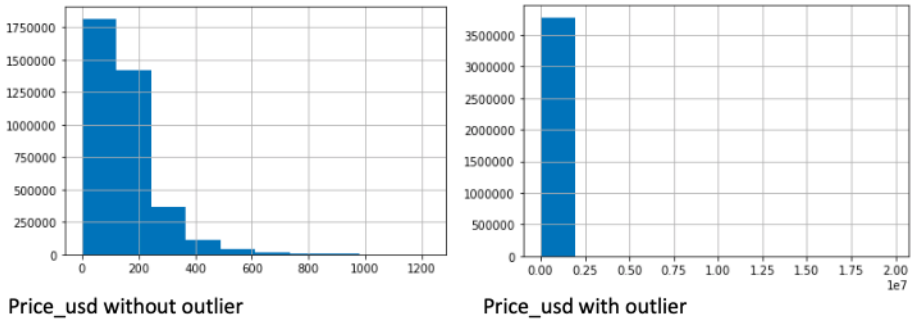

Figure 5: 'priceusd' without and with outlier

3.2 Feature Engineering

As aforementioned, Wang's presentation showed us that people do not like to book a hotel without historical data. As such, we created two features, 'histrateexist' and 'histusdexist', based on the 'visitorhiststarrating' and 'visitorhistadrusd' columns. In this regard, 0 represents a hotel without a historical rate and price and 1 represents a hotel with a historical rate and price.

Next, we identified that the type of the feature 'datetime' was not datatime. Thus, we transformed it to datatime type and created some additional features. That is, we made a 'daytimenight' feature with 1 representing daytime between 07:00h and 20:00h and 0 representing nighttime between 20:00h and 07:00h. In addition, we made a 'season' feature with 0 representing spring (i.e., months 3-5), 1 representing summer (i.e., months 6-8), 2 representing autumn (i.e., months 9-11), and 3 representing winter (i.e. months 12-2). Finally, we created a 'year' feature.

According to Wang and Binshu's presentations it was of significant importance for the predictive performance to include features describing the booking rate and clicking rate. As such, we created the following features based on the columns 'bookbool' and 'clickbool':

- 'booknum': the number of times a hotel has been booked
- 'clicknum': the number of times a hotel has been clicked on
- 'allnum' : the number of times a 'propid' occurs in the dataset
- 'bookingrate' : 'booknum'/'allnum' : rate being booked
- 'clickingrate': 'clicknum'/'allnum' : rate being clicked
- 'bookingclickingrate': 'booknum'/'clicknum' : rate being booked when clicked

Furthermore, we considered it relevant to create the feature 'logdiffhistoricalcurrentprice', describing the difference between 'loghistoricalprice' and 'logcurrentprice'. We also created the feature to describe the number of 'familymembers', based on 'srchadultscount' and 'srchchildrencount'. With this, we also calculated the feature 'totalprice' multiplying the 'srchroomcount' with the 'priceusd'. Subsequently, we calculated the feature 'averagepriceoneperson' by dividing the 'totalprice' by the 'familymembers'.

Also, we modified and created features for the review, star, and location scores. We created the feature 'diffrate' by subtracting the 'propstarrating' from the 'propreviewscore' to illustrate the differences between the review score and hotel score. Then, we created the 'totalproplocationscore' by adding the 'proplocationscore1' to 'proplocationscore2'. Lastly, we created the feature 'totalrate' by adding the features 'propstarrating' and 'propreviewscore', while grouping the feature into three classes with 0 representing low quality with a 'totalrate' of less than 3, 1 representing middle quality with a 'totalrate' between 3 and 7, and 2 representing high quality with a 'totalrate' between 7 and 10.

Finally, we did some additional feature engineering to improve the performance of the model while modeling. First, we created composite features. These are simple features that capture the relative value of a property. Then, we created ranked features. For each search ID we created a ranking from 1 till n for the existing features, since we realized that as we were working with a ranking model, these type of features work quite well. We first created ranked features for the basic features and then we created ranked features for the features we created ourselves. All above features are listed below.

- 'priceproprating' : 'priceusd' / 'propstarrating'
- 'pricepropreviewscore' : 'priceusd' / 'propreviewscore'
- 'pricepropreviewscore' : 'priceusd' / 'propreviewscore'
- 'priceprop1score'] : 'priceusd' / 'proplocationscore1'
- 'priceprop2score' : 'priceusd' / 'proplocationscore2'
- 'propreviewXproprating' : 'propreviewscore' * 'propstarrating'
- 'stardiff' : 'visitorhiststarrating' - 'propstarrating'
- 'pricerank' : groupby('srchid')['priceusd'].rank
- 'priceratingrank' : groupby('srchid')['propstarrating'].rank
- 'propreviewscorerank' : groupby('srchid')['propreviewscore'].rank
- 'prop1scorerank' : groupby('srchid')['proplocationscore1'].rank
- 'prop2scorerank' : groupby('srchid')['proplocationscore2'].rank
- 'logdiffpricerank' : groupby('srchid')['logdiffhistoricalcurrentprice'].rank
- 'stardiffrank' : groupby('srchid')['stardiff'].rank
- 'totalpricerank' : groupby('srchid')['totalprice'].rank
- 'totalproplocationscorerank': groupby('srchid')['totalproplocationscore'].rank
- 'pricepropratingrank : groupby('srchid')['priceproprating'].rank
- 'pricepropreviewscorerank' : groupby('srchid')['pricepropreviewscore'].rank
- 'priceprop1scorerank' : groupby('srchid')['priceprop1score'].rank
- 'priceprop2scorerank' : groupby('srchid')['priceprop2score'].rank

4 Modeling and Evaluation

In this section we explain how we built our model and evaluate what we have learned from the exercise.

4.1 Modeling

The idea was to create a kind of random forest with the LambdaMART model. The LambdaMART is the boosted tree version of LambdaRank and has proven to be a very successful algorithm for solving real world ranking problems (e.g., an ensemble of the LambdaMART rankers won Track 1 of the 2010 Yahoo! Learning To Rank Challenge) (Burges, 2010). Moreover, research shows that ensembles of decision trees to make them to vote for the most popular class improved the classification accuracy significantly (Friedl et al., 1999; Pal and Mather, 2003). Specifically random forests can be useful as they add an additional layer of randomness to begging, changing how regressions trees are constructed (Breiman, 2001).

Having created a random forest with the LambdaMART model implies that we ran a number of slightly different models, which each had a slightly different prediction. We then averaged these predictions in order to get a ensemble model that would be better than any single model, as the winner of the 2013 competion, Owen, suggested.

We created the different versions of the model as follows:

1. Randomly removing some features (just like an actual random forest)
2. Differing random seeds for the train test split
3. Differing random seeds for the lightgbm model itself

Unfortunately we were not able to apply any bootstrapping to the model. Instead we opted to use different random seeds to replicate the same effect.

The LightGBM model requires categorical faetures to be listed. In order to keep track of the categorical features when we are randomly removing some features we create a dictionary. We made a function which randomly removes features from the data. However, some features should not be removed or else the model would crash.

We could not use the standard train_test_split function. In stead we split the data on 'srchid' and we also create a target variable where 'bookingbool' weighs more that 'clickbool'.

We made use of the following hyper-parameters:

- We set the objective to 'lambdarank', which ensured that the lightgbm would be a ranking model
- We also used the 'nDCG' score as a metric for model performance

The nDCG was only used to determine the top 5 ranking. Eventually our model outputs values between 1 and -1. These were sorted to create the final prediction that has been uploaded to Kaggle.

Please find an overview of what happens in our overview below.

1. Faulty features were removed
2. Some extra features were generated
3. A dictionary was initialized to track the individual model performances
4. An empty array was initialized in which we shall store the individual model predictions
5. A 'try' function was made to save our results in case of an error
6. We looped n times. The index of the loop was then used a the random seed
7. Random features were dropped
8. Data was split into training set en validation set
9. The hyper-parameters were initialized with a random seed
10. The lightGBM started training
11. Individual model results were stored in 'modeldict' and the model is saved
12. The model was then used to create predictions and these predictions are stored in the predictions array
13. The stored predictions were averaged by row and then sorted to get the final prediction. This is then also saved
14. The 'modeldict' was returned

4.2 Evaluation model and key take-aways

The main learning from this assignment and course as a whole revolves around having gained more tacit and explicit knowledge on structuring data and algorithms for data mining to come up with efficient solutions when processing large volumes of data. In this respect, we have become more skilled at treating missing values, feature engineering, and preparing training and test data for modeling. Specifically, we learned to put tree based learning algorithms to use in ranking data.

The main difficulties we encountered throughout this assignment was coming up with useful features. For example, in the data preparation we created features based on the bookings and the clicks. Nonetheless, we found out that these were not useful because the test set did not have these features. As such, we removed these features since both the training and test set should have the same features.

Furthermore, it was quite challenging to create the final ensemble model and merge all predictions together. Firstly, we did not expect it to be so complicated to keep track of all the categorical features while randomly dropping some of them. Finally, it took over four hours for our model to run once, leaving very little room for error. In sum, we are proud of the predictive performance we obtained in the end.

References

Breiman, L. (2001). Random forests. Machine Learning, 45(1): 5–32, 2001.

Burges, C. (2010). From RankNet to LambdaRank to LambdaMART: An Overview. Microsoft Research Technical Report. MSR-TR-2010-82

- https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/MSR-TR-2010-82.pdf

Friedl, M. A., Brodley, C. E., and Strahler, A. H., 1999, Maximizing land cover classification accuracies produced by decision tree at continental to global scales. IEEE Transactions on Geoscience and Remote Sensing, 37, 969-977.

Jimoh, H. (2018). The tale of missing values in Python. Retrieved 17 May 2020, from

- https://towardsdatascience.com/the-tale-of-missing-values-in-python-c96beb0e8a9d

Pal, M., and Mather, P. M., 2003, An assessment of the effectiveness of decision tree methods for land cover classification. Remote Sensing of Environment, 86, 554-565.

Kaggle (2020). Personalize Expedia Hotel Searches - ICDM 2013. Retrieved 19 May 2020, from

- https://www.kaggle.com/c/expedia-personalized-sort/data

Presentations winning teams (2013). Link to the dropbox account with the presentations of the winning teams. Retrieved 08 May 2020, from

- https://www.dropbox.com/sh/5kedakjizgrog0y/_LE_DFCA7J/ICDM_2013

Wikipedia page of Random Forest

- https://en.wikipedia.org/wiki/Random_forest