Capstone Assignment

Abstract. In this report we are going to explain how we are competing in the Kaggle competition '2nd Assignment DMT'. The purpose is to rank the hotels on Expedia according to the likelihood of them being booked. First we will discuss what other people have previously done in the same competition. Secondly we will discuss the data both graphically and numerically. After this we need to prepare the data before we can use it. The steps we made to do this are being discussed. The next step is to build a model that can give us a high accuracy. At last we will have our conclusion and provide some suggestions for further research.

1 Business understanding

The Expedia Hotel Recommendation competition we are participating in was held in 2013[1] and in 2016[2]. The team with the highest score is Commendo with a score of 0.54074. This team was not able to fulfill all the contest requirementsand were subsequenty disqualified. The first place went to the team with the second best score. Team Commendo shared a post on Kaggle with which model and features they used for their second best program with a score of 0.5338. They state that it takes quite a lot of time to train a LambdaMART model. The model that is being used for ranking is LamdaMART. The features that are being used are all the numeric features, the average of numeric features per prop_id, the stddev of numeric features per prop_id, the median of numeric features per prop_id. To improve the score they created an ensemble method with four different models, namely LamdaMART, SGD, NN and GBDT.

Another team named 'Bing Xu MLRush BrickMove' have obtained a score of 0.53101. The only thing they mentioned in a post is that they use deep learning and random forest on 8% of the data. In the same post they made a promise that they will add more information to a blog. Unfortunately the link to the blog does not work.

The winner of the competition is Owen Zhang. Unfortunately the information he share about his solution is quite limit compared to the other winners. In the discussion section he shares some tips with other people. He states that to get started the variable prop_location_score2 is a good feature.

In the competition of 2016 the team 'idle_speciulation' became first with a score of 0.60218. In the discussion section he gave a extensive tutorial of the solution. He used clusters, namely H1=(hotelcountry, hotelmarket, hotelcluster) and H2=(srchdestinationid, hotelcluster). At the end the model sorts out which cluster was more useful. Besides this for optimalization he used gradient descent. It took him 36 hours to run the model and it needed 10^{11} iterations to find the optimal solution.

For our prediction we studied the work of the winning teams carefully and for many features we created we have drawn inspiration from their work.

2 Data understanding

This section provides some information on the data, so that we can have a better understanding of what our data consists. We will explore the dataset and also see how the various attributes are correlated. In the competition of 2013 the data is described as follows:

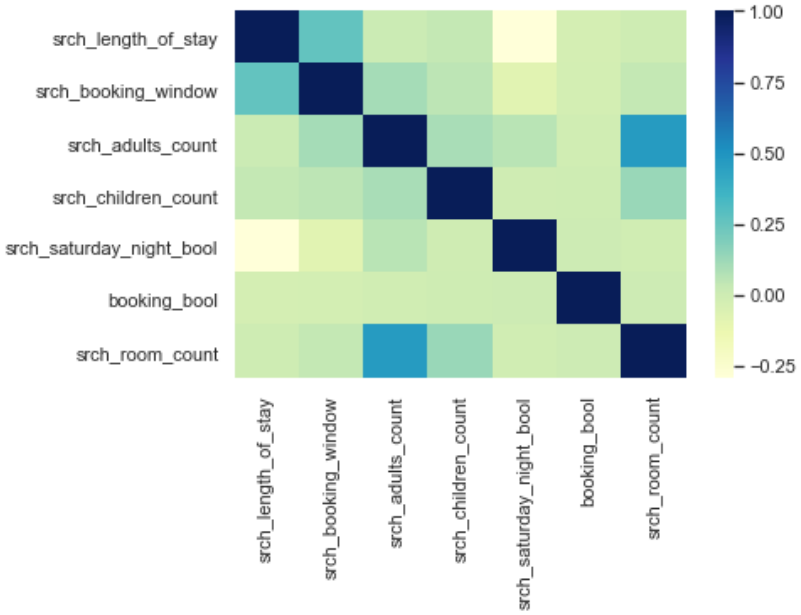
> "Hotel" refers to hotels, apartments, BBs, hostels and other properties appearing on Expedia's websites. Room types are not distinguished and the data can be assumed to apply to the least expensive room type.
> Most of the data are for searches that resulted in a purchase, but a small proportion are for searches not leading to a purchase.

First of all, the dataset is split into a training and test set. Each row represents a search query of a visitor on the website and it contains information concerning that query. Each dataset contains approximately 5 million records. Then there are 54 data variables in the training sample, which cover all the aspects of a single search query. If we divide the data variables, we see that 24 variables are related to Expedia's competitors. This means whether Expedia offers lower prices than some other competitor and the availability of the competition in a certain hotel. For the remaining 30 variables, they describe the properties of the Expedia's users and hotels. We can divide these 30 variables into three groups: User related, Hotel related and Booking related. The User related group includes variables as the ID of the search, date, origin distance, average star rating of previous bookings and the average price the visitor spent. The Hotel related variables are the property location, the average star rating and whether the hotel is linked to a major hotel chain.

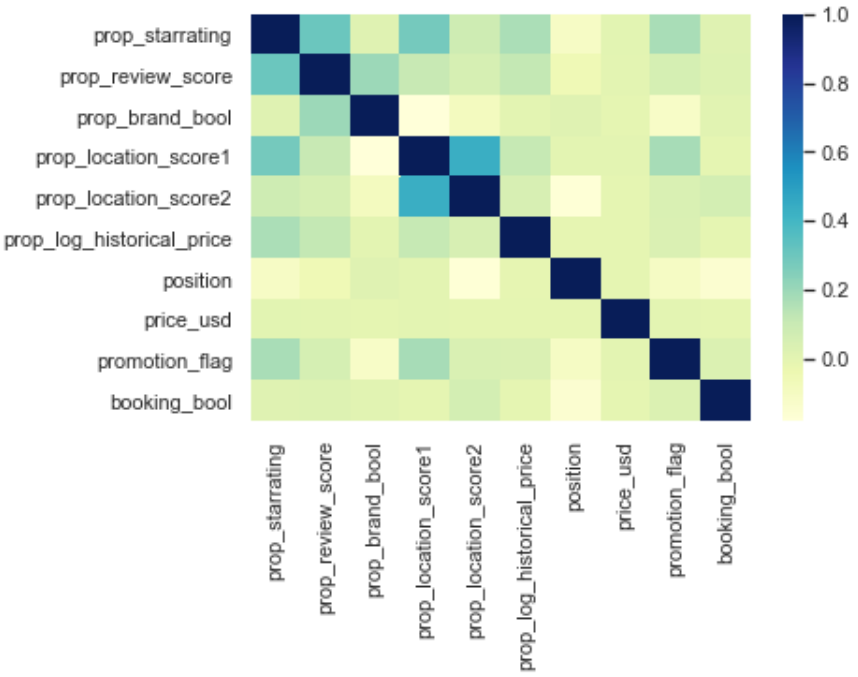
[1] https://www.kaggle.com/c/expedia-personalized-sort/leaderboard
[2] https://www.kaggle.com/c/expedia-hotel-recommendations/leaderboard

Then for the Booking related variables, they include the length of stay, booking window and the number of rooms required. These 30 variables contain several types of data. Tags in integer format such as the ID of the search, time variable, ordinal values and boolean values.

To visualize things, we have made some graphs. If we make a heatmap for the correlation between certain features that are related to the booking, we obtain the following graph:



As we can see, there is not much to see besides a correlation between srch_room_count and srch_adults_count. We also see a slight correlation between srch_length_stay and srch_booking_window, but not any other correlation that we find notable. If we provide the heatmap for the features related to the hotel, then we obtain this graph:



This heatmap of the correlations also shows very few correlations between the features. The only correlation that is mentionable, is that between prop_location_score1 and prop_location_score2.

3 Data preparation

3.1 Missing Values

For the feature engineering we primarily looked at the presentations of Jung Wan and Binghsu, the second and third teams respectively. First we looked at the missing values. We made a distinction between categorical features and continuous features. The categorical features are primarily related to the competition, whereas the continuous features are more related to the company itself. We found that the feature related to the competition had a lot of missing values. Just like the Jung Wang we replace the missing values with a categorical value, but instead of picking 0, we chose -2 to indicate that the missing value is separate a category. However we were unable to apply such an approach for the "comp_rate_percent_diff". Here there were too many missing values, so we decided to skip these features altogether.
For the continuous features with missing values like "prop_review_score" and "orig_destination_distance" we opted to replace the missing values with the median value. This way we hoped to not change the distribution too much. Another possible solution we thought of was performing a regression on the missing values. Unfortunately we were not able to get a model that performed well enough.

3.2 Data cleaning

We noticed that some features had very high outliers. This skews the distribution of the features which can lead to a poorer model performance. We solved this by replacing all values that were more than 5 standard deviations apart from the mean with the value of 5 times the standard deviation. This way we limit the amount of outliers.
We also looked at the feature "orig_destination_distance". Here we found that the distribution of this feature is has a very extreme skew to the left. When we took the log of this feature the distribution turned into a bell shaped distribution. We Think that this is also a great way of removing the influence of outliers.

3.3 Feature Engineering

Here we provide a short description of all the features we created for our model and our motivation behind it. At the end of this section we provide a table with all the extra features we created.

Data cleaning related: The first two features we created are related to the data cleaning process. For the feature "price_usd" we we created a dummy variable "luxury" for every value that was more than 5 standard deviations removed from the mean.
The second feature we created was the log transformation of the "orig_destination_distance" feature.

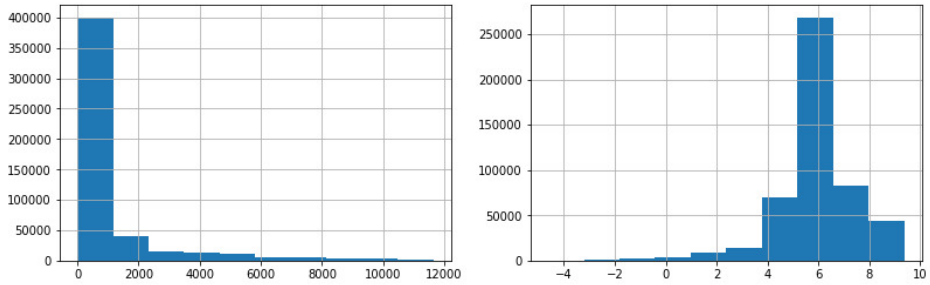


Fig. 1: Left: Distribution of the orig_destination_distance feature, Right: Distribution of the log˙distance feature

Conversion related: From the presentations of Jun Wang and Binghsu we know that that the perceived quality of a hotel by the customer can be determined by relating the probability of clicking and/or booking with the "prop_score" feature per "prop_id". We tried the replicate all the features as mentioned in the presentation of Jun Wang regarding this for both booking and clicking.

Business related: We also created date related features for the year, month, day and season. We did this because we know that the destination of customers often varies depending on the time of year.
Finally we constructed some simple features ourselves for such as the the "number of visitors" by adding the number of adults and children. Another example is taking the difference between the historical and the current price of a hotel room. All these features were created by doing simple operations like adding and subtracting.

Feature	Description
luxury	binary, 1 for very expensive hotel, else 0
log_distance	log transformation of orig_destination_distance
year	year
month	month in numbers (e.g. 8, 9, 10)
day	day of the month
summer	season (June, July, August)
winter	season (December, January, February)
dayofweek	the day of the week (1 - 7)
booking_score	Probability score of booking a particular property
click_score	Probability score of clicking on a particular property
price˙usdXprop˙location˙score1	price multiplied by prop˙location˙score1
price˙usdXprop˙location˙score2	price multiplied by prop˙location˙score2
price˙usdXsummer	price multiplied by summer dummy
price˙usdXwinter	price multiplied by winter dummy
diff˙log˙price˙usd˙prop˙log˙historical˙price	difference between current het historical log price
diff˙log˙price˙usd˙prop˙log˙historical˙priceXpromotion˙flag	difference between current het historical log price multiplied by promotion
price˙usdXpromotion˙flag	price multiplied by promotion flag
prop˙brand˙boolXpromotion˙flag	brand boolean multiplied by promotion flag
srch_id_booking_score	mean booking score for given search id
prop_id_booking_score	mean booking score for given prop_id
prop_country_id_booking_score	mean booking score for given prop_country_id
srch_destination_id_booking_score	mean booking score for given srch_destination_id
month_booking_score	mean booking score for given month
srch_booking_window_booking_score	mean booking score for given search booking window
srch_id_click_score	mean click score for given search id
prop_id_click_score	mean click score for given prop_id
prop_country_id_click_score	mean click score for given prop_country_id
srch_destination_id_click_score	mean click score for given srch_destination_id
month_click_score	mean click score for given month
srch_booking_window_click_score	mean click score for given search booking window
prop_starrating_price_usd	(prop starrating / price) ratio
prop_review_score_price_usd	(prop review / price) ratio
visitor_hist_adr_usd_price_usd	(historical price / price) ratio
srch_adults_count_srch_children_count	total number of people
srch_adults_count_srch_children_count_price_usd	(total number of people / price) ratio
diff_visitor_hist_starrating_prop_starrating	difference between historical and current starrating
price_usdXsrch_room_count	total price of booking

Table 1: List of newly generated features with a short description

4 Modeling and Evaluation

For this assignment we started out creating our first model in Python. Unfortunately we kept running into problems. Our model performance was very poor resulting in a score of 0.000 even after repeated uploads. We thus decided to pick R for the modelling part, while still using the python code for the preprocessing part. Since we did spend considerable time working on both the Python code as well as the R code we will describe them both.

4.1 Python

After preprocessing we wanted to create our first simple model to function as a base line, which we would then later expand upon. From experimenting with the data we found that a model the "click_bool" feature is very important for predicting the the bookings. unfortunately this feature is not available in the test set. To compensate for this we opted to first predict the clicks. With these predictions we wanted to predict the bookings.

Before we start we divide our training set in a train and test set. For this we use the train_test_split function in the sklearn package. The same package is used for all the models used for prediction. For all models we apply a 5 time cross-validation.
For the first models we used a simple logistic regression. The reason for this is that the model is specially designed for binary prediction. It also trains very fast, making it an ideal option for quick feature evaluation. Unfortunately, this model proved to bee too simple, leading to very bad results. Even when we added weights and even when we downsampled the data. We therefore chose a random forest. This model had a slightly better performance, but the difference was not of any significance.

Finally we tested the gradient boosting classifier, which performed best. However, when we evaluated the accuracy and F1 score of both models. The accuracy was either very high, around 95% with an F1 score of 0.000 or the accuracy floated around 12% with an F1 score of 0.62. This hold for both the click as the booking prediction separately, regardless of whether the click predictions were incorporated in the booking prediction or not.

After consulting with our TA we thought that the poor result could due to our relatively limited set of features at the time. We revisited our preprocessing code and we took the presentations of the winning teams as inspiration for further feature engineering. This resulted in an accuracy of 68% and a F1 score of 0.69 for the gradient boosting classifier for the click data and an accuracy of 13% and a F1 score of 0.63. Downsampling the data did not improve these results very much.

With these results we generated the score for our prediction. We then create a vector to use for ranking the different hotels. If our model predicted that a particular hotel got clicked on, we award it with 1 point. In case we predict that the hotel is booked we award it 5 points. in all other cases we award 0 points. We repeat this process for the test set as well, but now using the actual clicks and bookings. We then sort the whole dataset on site id (ascending) and the score(descending). Here we create an additional index vector to keep track of the order of the actual predictions. After this we reorder the dataset using the score for the predicted clicks and booking in the same way as described before. This causes the index vector to be shuffled around a little. We then put the index vector in the nDCG5 function.

Despite our painstaking efforts to program this whole process, we got a nDCG score of almost zero. When we then made our predictions for the real test set and uploaded this to Kaggle we also got a score of 0.000. After two weeks of trying various things and rewriting our preprocessing and modelling code times we decided to continue the modelling part in R, while still keeping the preprocessing code in Python since that works fine.

4.2 Things we tried

Some of the newly generated features rely on the clicks and bookings being available. Unfortunately for the testset this is not the case. We wanted to work around this problem by first predicting the clicks and bookings to then add these features to the testset as well. However, due to the fact that we were not able to get our python code to work and the time constraint as a result of that, we were unable to succesfully incorporate these features in our R code. Instead we chose to create some new features that were easy to implement in our python preprocessing code and that do not rely on clicks or bookings.

Another thing we tried was predicting the missing values in the data by using a regression model. The idea was that some missing data in the features could be estimated by a applying a simple model. For this model to work it would only have to outperform replacing the missing values with the mean or median. However, actually doing this proved more difficult. There are a lot of features that had missing values, which meant that these rows could not be used in our model. To actually make this work we should then very carefully pick features that could be used and then make sure that they did not have any missing observations, because this would crash the model. Furthermore we knew that this approach would only work on the continuous features. Any categorical feature would have to be estimated by a multinomial model or be turned into dummy variables to then individually estimate with a classifier. Given the fact that this could take up a lot of time without knowing whether this approach would yield significant results, we decided to spend our time on other features.

Finally we tried to use the Principle Component Analysis to reduce the amount of features that went into the model in the hopes of reducing the computation time. Unfortunately we thought of this only a day before the final deadline, which is why we could not apply it to the data to generate new predictions. We wanted to reduce the number features from 69 to 30. We expected a small decrease in the model performance all other thing being equal. However, because of the lower memory usage we would be able to increase the number of observations in our trainingset, the number of trees and the interaction dept.

4.3 R

Regardless of the fact that we lost two weeks trying to get a model to work in python, we did learn a few things about the data. One thing we learned is that the gradient boosting classifier performs better than the other models. We also see this in the presentations of the winning teams, where they use both the GBM and the LambdaMART model, which also uses GBM. Knowing that we were short on time we directly tried to implement a LambdaMART model as the winning teams all concluded that this model performed best in their case.
We splitted our preprocessed trainingset into a trainingset and a validationset. This way we were able to evaluate our model performance and calculate the nDCG score for our data. In total we used 2 million observations for testing and the remaining 2.9 million observations for validating our model. We would have liked to use a larger trainingset. Unfortunately this was not possible due to a lack of memory.

For our GBM model we used the pairwise distribution with the nDCG score as a metric. We used a total of 1000 trees as we saw from running our code multiple times that our model converges after around 970 trees. Furthermore we used an interaction dept of 12. Finally we used a five times cross-validation on our data.
After running our model for approximately 9 hours we able to extract the best iteration and use this to make predictions on the testset. Finally we saved the results and uploaded it to Kaggle. In the end we were able to get a score of 0.36583.

In the end we found that the creation of simple features and some interaction features worked very well in improving our nDCG score on Kaggle. Furthermore we found that increasing the size of the trainingset also increases the prediction score.