# Data Mining Techniques
## 2nd Assignment

Group 144: Daniel Bemerguy - 2699826 and Corien Westveer - 2714024

Vrije Universiteit Amsterdam

## 1 Introduction

This paper describes an approach to solve the Kaggle competition "2nd Assignment DMT" which was inspired by the Expedia Hotel Recommendations Competition. The goal of this assignment is to predict what hotel a user is most likely to book. The approach will be described on the basis of the DM process model.

## 2 Task 1: Business Understanding

For this task, we will describe the 2 main approaches with the best results in the Expedia Hotel Personalized Searches competition that was held at the International Conference on Data Mining 2013 (ICDM).

The first place was achieved by Owen Zhang [1], with an accuracy of 0.53984. In the pre-processing phase, he imputed the missing values with a negative value, bounded numerical variables, such as *price*, and down sampled negative instances in order to improve learning speed of the model. Furthermore, he used all original features and also some composite features, such as the difference between the current hotel price and the recent price and price order, which is the order of the price within the same *search_id*. Also, he converted some categorical features into numerical ones by calculating the average of the target variable related with the target F.

The second place, achieved by Jun Wang and Alexandros Kalousis [2], showed some interesting pre-processing and feature extraction individualities. They replaced missing values of hotel descriptions with worse case scenario, since people do not like to book hotels with missing values. For the user historical data they highlighted matching and mismatching between historical data and the given hotel data. Furthermore, for the competitors descriptions they set all the values to zero. In the feature engineering phase, they extracted the non-monotonicity utility functions for each feature to check if they are monotonic or not.

## 3 Task 2: Data understanding

The data given for this competition consist out of the hotels resulting from a specific search on an Expedia website. The training set includes 199795 unique searches with for each search on average a list of 25 hotels. The size of the train set is 4958347 rows by 54 columns and the test set contains 4959183 rows and 50 columns (the variables *position*, *click_bool*, *gross_bookings_usd* and *booking_bool* are not given for the test set). The ranking of the hotels is based on whether a hotel is clicked or booked (*booking_bool* and *click_bool*). Only a small fraction of the hotels is booked or clicked: 4,47% of the hotels are clicked and 2.79% are booked, this means that the data set is imbalanced. The variables in the data can be roughly divided in the following categories: search query information, static hotel characteristics, visitor information and competitive information. In the next list the variables that are available in each category are given:

- **Search query information**: search id (*srch_id*), Date and time of search (*date_time*), id for the site (expedia.com, expedia.dk, etc.) (*site_id*), the destination id (*srch_destination_id*), length of the stay (*srch_length_of_stay*), how much in advance the booking was made (*srch_booking_window*), number of adults (*srch_adults_count*, number of children (*srch_children_count*), number of rooms (*srch_room_count*), whether the stay was short and if it included a Saturday night (*srch_saturday_night_bool*).
- **Static hotel characteristics**: hotel id (*prop_id*), hotel country id (*prop_country_id*), starrating (*prop_starrating*), user review score (*prop_review_score*), whether the hotel is independent or part of a chain (*prop_brand_bool*), desirability of hotel location ($prop\_location_score1$ and *prop_location_score*2), historical information about the hotel price (*prop_log_historical_price*).
- **Dynamic hotel characteristics**: rank position (*position*), price (*price_usd*), whether there was a promotion (*promotion_flag*), whether the hotel was clicked (*click_bool*), whether the hotel was booked (*booking_bool*), total value of transaction (*gross_booking_usd*) and probability that the hotel will be clicked on in an internet search (*srch_query_affinity_score*).
- **Visitor information**: country id for visitor (*visitor_location_country_id*), distance between visitor and hotel (*orig_destination_distance*), mean star rating for visitor (*visitor_hist_starrating*) and historical mean price per night for visitor (*visitor_hist_adr_usd*).
- **Competitive information**: if Expedia has lower/same/higher price than competitor (*comp*1*_rate* to *comp*8*_rate*), if the competitor has hotel availability (*comp*1*_inv* to *comp*8*_inv*) and percentage price difference (*comp*1*_rate_percent_diff* to *comp*8*_rate_percent_diff*). These variables where given for 8 different competitors.
- **Other**: whether the search results where returned in random order (*random_bool*).

First we examined the number of missing data for each column. A graph of this is given in figure 1. The variables not included in this graph have no missing data. From the graph it can be concluded that there are a lot of variables that have very high percentages of missing data.

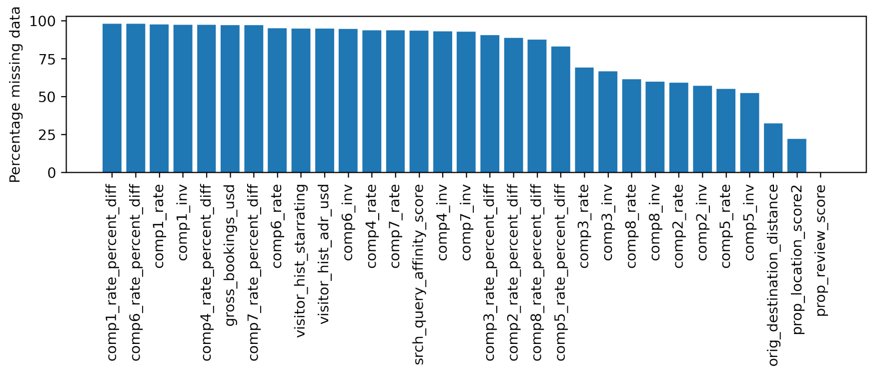

**Fig. 1.** Percentage of missing data in for variables in the training data

Next, the histograms of the variables are examined. An overview of the distributions is given in figure 2.

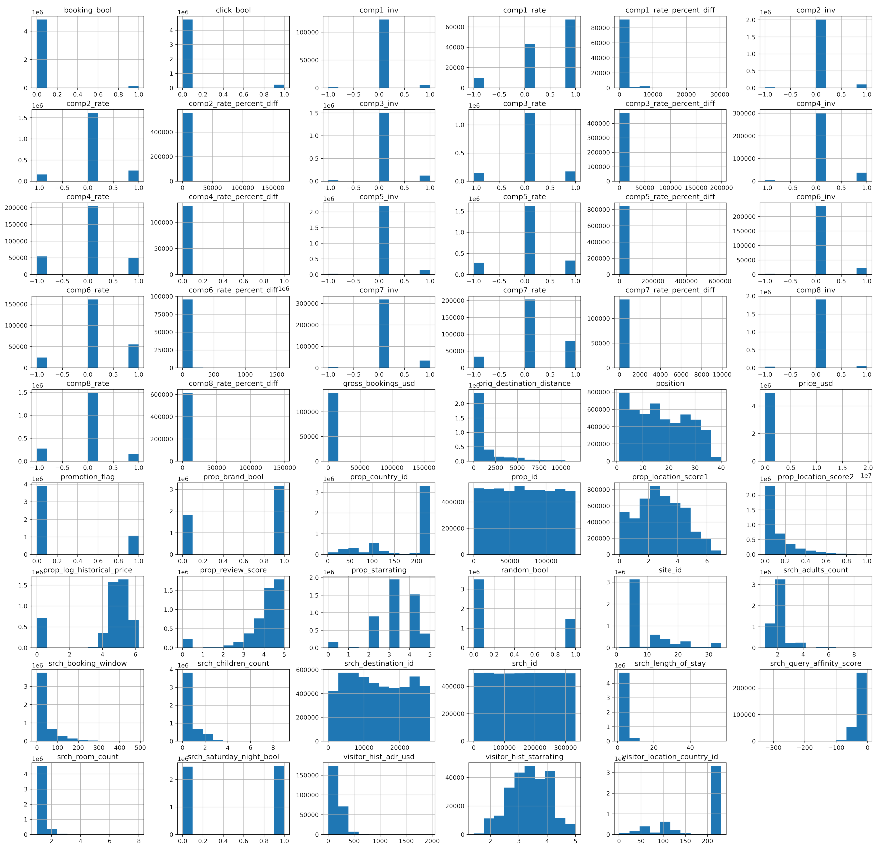

**Fig. 2.** Histograms of variables in training set

Based on the histograms we can conclude that there are some variables with a geometric distribution, such as *destination_distance*, *prop_location_score*2, *prop_review_score*, *srch_booking_window*, *srch_children_count*, *srch_length_of_stay* and *srch_room_count*. Some histograms only show one bar and the x axis is going to very large numbers, which is the case for *comp*1*_rate_percent_diff* tot *comp*8*_rate_percent_diff*, *price_usd* and *gross_booking_usd*. This gives the impression that these variables have large outliers, which results in that the distributions cannot be viewed well in the histograms. To get a better view of these distributions and to examine if these variables have outliers a boxplot is made, which can be viewed in figure 3.

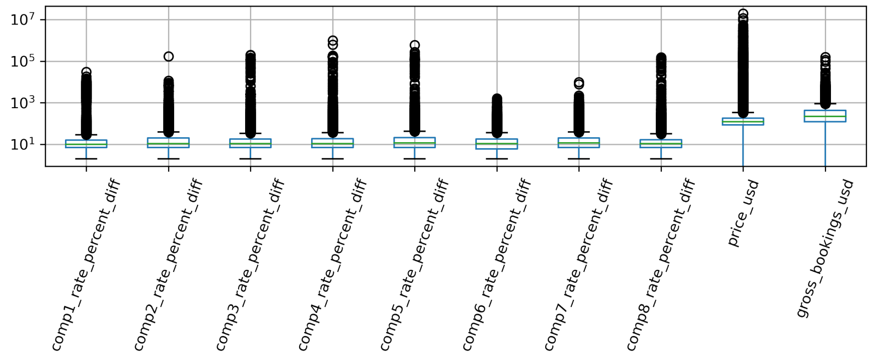

**Fig. 3.** Boxplots

The boxplots indicate that there are indeed outliers present in the shown variables and that these distributions are skewed (the y-scale of the plot is logaritmic).

At last, the correlations between the variables are viewed. This was done by using Pearson's correlation coefficient (r). The highest correlation was found for *click_bool* and *booking_bool* (r = 0.78), which can be declared by the fact that if a person books a hotel, a person has also clicked that hotel. The other highest correlations are found for the variables *comp*8*_rate* and *comp*5*_rate* (r = 0.64), *comp*8*_rate_percent_diff* and *comp*3*_rate_percent_diff* (r = 0.63), *comp*3*_rate_percent_diff* and *comp*5*_rate_percent_diff* (r = 0.59) and *comp*1*_inv* and *comp*5*_inv* (r = 0.51), meaning that there is correlation between the competitor related variables. There where no very big linear correlations found for variables that could be used as features and *click_bool* and the feature variables and *bookings_bool*, the highest correlation was found for the variable *position* (-0.16 for *click_bool* and -0.15 for *bookings_bool*), which indicates that there is a positional bias in the data (meaning that hotels with a higher position have a greater change to be booked or clicked).

## 4 Task 3

After we did the exploratory analysis, we verified a large number of missing values in some features. For instance, all the competitors features have more than 50% of missing values, and, some of them achieve the mark of more than 90% of missing values. Moreover, the *srch_query_affinity_score*, *visitor_hist_starrating* and *visitor_hist_adr_usd* have more than 90% of missing values as well. There are other features with missing values, however, the proportion oh them is not so high as the ones above. The list below describes the treatment each feature received:

- **Competitors features**: replaced NAs by zeros (Approach done by Jun Wang);

- **srch_query_affinity_score** : replace NAs with the worst case score (approach done by Jun Wang)
- **visitor_starrating**: removed the feature, since it had 95% of missing values, in addition we replaced it by a new feature called *starrating_diff*, which is calculated by the absolute value of the difference between *visitor_starrating* and *prop_starrating*;
- **prop_review_score**: replaced missing values with the lowest star rating (approach done by Jun Wang)
- **prop_location_score2**: replaced missing values by worst case location score
- **orig_destination_distance**: tried to replace by avarege distance of people with the same srch_destination_id and same visitor_location_country, but this approach was computationally inefficient, hence, we replaced missing values by the sample average
- **gross_booking_usd**: removed feature due to its irrelevance for the problem
- **price_difference_history**: new feature created by (*prop_log_historical_price - log(price_usd)*)
- **price_difference_user**: new feature created by (*visitor_hist_adr_usd - price_usd*)

After applying this transformations to existing data and also creating new features, we did some normalization on numerical data. The features *price_usd* and *price_difference_user* were normalized using the MinMaxSacaler funtion in python. The MinMaxSacaler works according to the following equation:

$$m = \frac{x - min(x)}{max(x) - min(x)} \tag{1}$$

## 5 Task 4: Modelling and Evaluation

We build different types of models to predict the optimal ordering of the hotels. Here we will describe the different models.

### 5.1 Benchmark model

We started with a benchmark model, so we could compare our other models to this benchmark (and at least score better than this benchmark). For this we used the benchmark code provided to create the basicPythonBenchmark for the Personalize Expedia Hotel Searches from 2013.[1] This benchmark uses a random forest classifier. A random forest is a ensemble machine learning method which uses many decision trees for the prediction. The algorithm makes use of bagging when building trees and uses random subsets of features when splitting the nodes of the trees. For the implementation of this algorithm the RandomForestClassifier of scikit-learn was used. [3] The hyperparameters used for the algorithm where 50 trees and a minimum of 10 samples for splitting a node. The criteria used was the gini index, there was no maximum depth, only 1 sample required at

[1] https://github.com/benhamner/ExpediaPersonalizedSortCompetition

a leaf, the number of features considered when looking for the best split was set to the quare root of the number of features, unlimited number of lead nodes, no early stopping of tree growth based on impurity, bootstrap was used for building trees and for the remaining hyperparameters the default values of the sklearn implementation where used. The predictions where done for the target variable *booking_bool* (*click_bool* was not used). The data used for training this model was the original training data (not our transformed training data) and all features where used for training except *click_bool*, *booking_bool*, *date_time*, *position* and *gross_bookings_usd*. Using this model to predict the likelyness of being booked on the test data resulted in an NDCG@5 score of 0.25694.

### 5.2 Random Forest Classifier

The next model used was again a random forest classifier. For this classifier the same implementation and hyperparameters where used as for the benchmark model. But in this case the transformed dataset as described in task 3 was used. For training the features of the complete dataset was used, except for the variables *click_bool*, *booking_bool*, *date_time* and *position*. The model was trained two times, ones with the target variable *booking_bool* and ones with the target variable *click_bool*. The two trained models were used the make predictions, the first was used to predict the likelyness of being booked and the second was used to predict the likelyness of being clicked. These were combined by simple summing the likelyness of being booked with the likelyness of being clicked (with equal weights of 1). Based on this combined likelyness the list was ordered from highest likelyness to lowest likelyness for every *srch_id*. This resulted in a NDCG@5 score of 0.32884 on the test data. This is a big improvement compared to the score for the benchmark model, indicating that our feature transformation did help in creating better predictions and that it is wise to predict for both the *booking_bool* and *click_bool* (which is not very surprising).

### 5.3 Extreme Gradient Boosting Classifier

In order to make prediction we used the Extreme Gradient Boost Classifier. This is an implementation of the Gradient Boosting Decision Tree algorithm with improved speed performance. The main advancement in this method is the faster execution speed and its scalability when compared to other implementations of gradient boosting [4]

Gradient Boosting is a technique where the final prediction is made by adding a new model that can predict the residual errors of a prior model. In other words, the Extreme Gradient Bossting model is an ensemble of random forests models. We were inspired by Owen‘s approach, which achieved the first place on the original Kaggle competition.

When training the model we used the holdout approach to split the dataset into training and testing data. The data was devided into 90% training and 10% testing. For our model parameters we chose: *number of estimators* = 5000, *learning rate* $= 0.01$, *gamma* $= 0.2$ and *max_depth* $= 4$. The model took approximately 11h to train, hence we decided not to do any hyperparameter tuning. The reason we choose these parameters was to avoid overfitting and also based on Owen's approach.

The implemented model aim to predict two variables, *click_bool* and *booking_bool*. In order to do so we wrapped the XGBoost model in the *MultiOutputClassifier* class, contained in the *sklearn* library. This wrapper allows our model to predict two outputs at the same time. However this approach assumes that these two output variables are independent, which might not be a correct assumption, however, this approach still provide effective results [5].

After predicting the two target variables, we used our model on the test data to predict the probability of each property of being clicked or booked. After predicting these probabilities we calculated what we called the expected relevance grade. This relevance grade was calculated by adding the probability of clicking with the probability of booking. Before trying this approach we tried to sum the probabilities giving more weight to the probability of booking, according to the following equation:

$$expected\_relevance\_grade = Prob(clicking) + 5 * Prob(booking) \tag{2}$$

However, we have got better results when we didn‘t give weights to the probabilities. That was not an expected result. A possible explanation for this could be that weighting the probability of booking 5 times is too high and resulting in a ordered list solely based on the booking bool. After calculating the expected relevance grade we sorted the results to rank the properties for each user.

This approach achieved score of 0.34930. The following plot shows the feature importance for the XGBoost approach. As we can see, the 5 most important features were *prop_location_score2*,*promotion_flag*, *prop_location_score1*, *prop_starrrating* and *prop_review_score*.

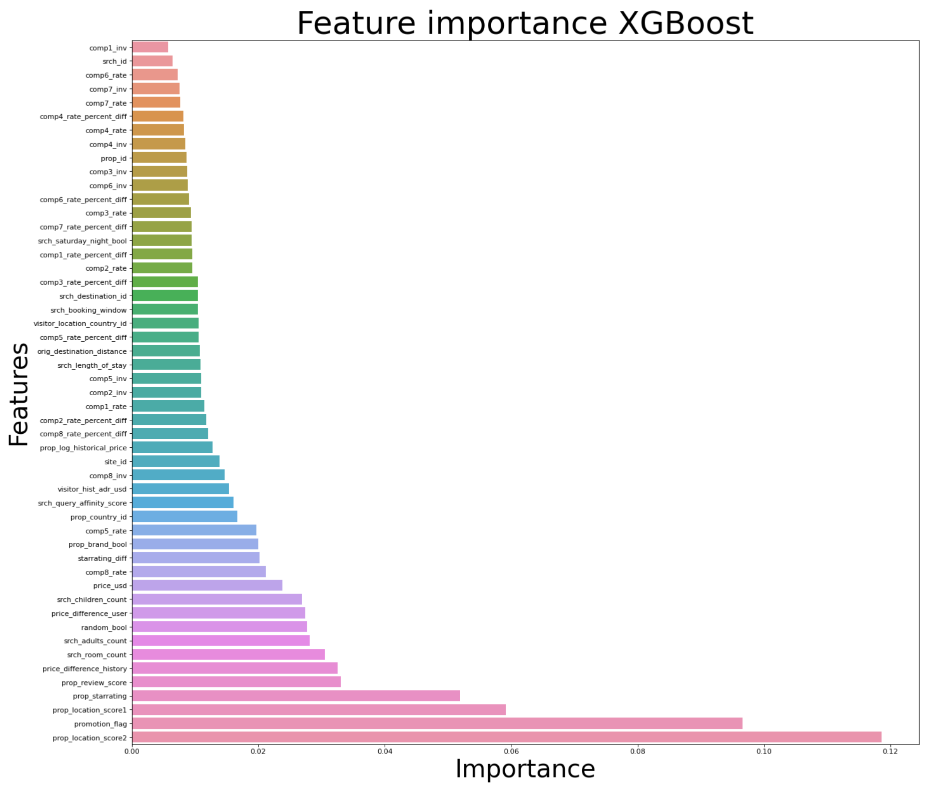

**Fig. 4.** Feature Importance XGBoost

There were some ways to improve our model. The first way would have been to combine different models, however since the model training was taking too long and we did not know how to ensemble new models together we decided not to try this approach. Another way to improve our results would have been to do some hyperparameter tunning. This was not done due to the same reason as before.

## 6 What we learned

From this assignment we learned a lot about recommender systems. We both had no previous knowledge on this topic. That you should order the instances based makes a recommender system quit different from the more classic classification or regression machine learning tasks. We learned that you can approach such a problem in all sorts of different ways, like binary class, multiclass, regression or ranking problem. In first instance we approached the problem as a simple classification problem, where we tried to predict if it was a booking/click or not. But soon we realized that only predicting a boolean made it hard to sort a list in a good order, so we decided very soon that we should predict a likelyness or probability of being booked or clicked to create an ordered list. Besides that we gained knowledge on feature transformation. In previous machine learning models we made, the datasets where already quit clean, but from this we learned how to handle missing data and that creating addition features can make a big difference in the performance of your model.

The main difficulty in this assignment was how we should rank the hotels. One of the approaches was making two separate models and than weighting the score for click and the score for booking equally and another way was to weight the booking bool more, because it counted more in the NDCG. Besides that, the long training times due to the size of the training set where inconvenient.

One of the unexpected outcomes was that in first instance the extreme gradient boosting classifier performed worse than the random forest classifier. We were surprised by this as you would expect the extreme gradient boosting classifier to perform better. However, we found out when we looked in more detail that this difference in performance could be declared by the different target variables than where used. For the random forest classifier we predicted the probability of being a click or a book and simply summed these changes. For the extreme gradient boosting classifier we used two features we generated our self, which was the number of times a property was clicked divided by the total number of times the property appeared in the dataset an a comparable feature for the bookings. This turned out to be a bad idea, and this can be declared by losing a lot of information as you will have with these features always have the same probability for a property for all different visitors.

## References

[1] Owen Zhang. *Personalized Expedia Hotel Searches –1st place.* ICDM - International Data Mining Conference. 2013.

[2] Alexandros Kalousis Jun Wang. *Personalized Expedia Hotel Searches –2nd place.* ICDM - International Data Mining Conference. 2013.

[3] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[4] Nunung Nurul Qomariyah, Dimitar Kazakov, and Ahmad Fajar. “Predicting User Preferences with XGBoost Learning to Rank Method”. In: Dec. 2020, pp. 123–128. DOI: 10.1109/ISRITI51436.2020.9315494.

[5] *How to Develop Multi-Output Regression Models with Python.* URL: https://machinelearningmastery.com/multi-output-regression-models-with-python/. (accessed: 10.05.2021).