Data Mining Techniques – Assignment 2

TA: Paulo Fijen

Group 7:

Levi Kamsteeg 2584230
Kilian Wessels 2579173
Steven Logger 2595779

Vrije Universiteit Amsterdam

1 Introduction

In this report we will create a model which is able to predict which hotel a user is most likely to book or click on. We use a dataset which comes from Expedia, where the search results of users and their bookings are stored. The dataset also contains additional data about the competitors' information, locations, and prices. We made use of Python to conduct our analysis.

Predicting the hotel, a user might book based on search results, browser, prices and competitor's data can lead to increased profits for travel agencies like Expedia and Booking.com. As planning trips online becomes more popular, it can be very useful when an algorithm is able to recommend the perfect hotel match for a specific customer.

We find that when we apply a learning to rank model from XGBoost, we are able to make quite reasonable predictions for the preferences of customers. We are able to get a Normalized Discounted Cumulative Gain (NDCG) score of 0.38 on 50% of the test set on Kaggle.

The rest of this report is structured as follows: first we will start with the business understanding, next we will perform some exploratory data analysis and explain our general setup for feature engineering, next we will start to build our first models and evaluate them, finally we will analyse our results and conclude our findings

2 Business Understanding

The dataset and problem are originally from a Kaggle competition hosted by Expedia. The competition is a few years old so other people already worked on creating models for this problem. In this section we will discuss some of their works.

One of the winners of the competition was a team called Bingshu. They addressed a few problems regarding the data and the prediction process: How to deal with the complex data? How to generate good features from them? How to make models train fast in such big data? We will most likely also face these issues. First, there is the issue of non-complete data, some values are missing. Bingshu deals with the missing data by imputing the first quartile to represent the missing data. In addition, the data is too big for some of the models, they therefore randomly samples 10% of the data by the feature srd_id. Moreover, the data is unbalanced, which has been taken care of by balancing the positive and negative data. Several different independent models were created after which an ensemble model has been made, since a single model was not able to get the best result. The models introduced were a linear model, random forest, (pairwise) logistic regression, gradient boosting machine, extremely randomized trees and factorization machine.

Another winner of the competition were Wang & Kalousis. When considering their modelling methodology, they found that a linear model had difficulties handling the categorical (discrete) features, so they decided to look at non-linear models; gradient boosted trees, random forest and LambdaMART. Concerning missing values, they found that the property score for hotels with missing values is much lower compared to hotels without missing values, so they decided to fill the missing values with the worst case scenarios. From both examples we can conclude that linear models do not seem to fit this problem optimally. Both teams described above made use of a gradient boosting machine, which is known for handling structured data very well and that is why we decided to use a gradient boosting machine in this research as well.

3 Exploratory Data Analysis

Before we can start building our models we should look closer into the dataset. In this section we dive deeper into the dataset and assess its statistics and data distributions and show some interesting plots.

First we start by computing the score for each property. As described in the Kaggle competition a score is given as follows: 1 if the user clicked through to see more information on the property, 5 if the user purchased a room for the property and 0 otherwise. In Table 1 we provide summary statistics for a couple of important features. We can see that the average score of a property is around 0.18, which indicates that most properties are not getting clicked on. The highest score of a property equals 6 since it is booked and clicked on by the customer. Another interesting feature is the price of the property in USD.

The maximum of this feature seems unreasonable at almost 20M USD, in addition the standard deviation of this feature is also quite high at 16K USD. When we compare this feature with the gross booking feature it seems that customers do not buy highly priced properties and we might want to handle some of this highly prices properties as outliers. Moreover, from the booking and clicking dummy we can see that on average 2.8% of the properties are being booked and 4.5% is getting clicked on.

Table 1: Summary statistics of a few important features

	Mean	**Std. dev.**	**Min**	**Max**
Score	0.184	0.994	0.0	6.0
Property starrating	3.181	1.051	0.0	5.0
Visitor historical starrating	3.374	0.693	1.410	5.0
Price USD	254.21	16,001.24	0.0	19,726,328.00
Gross booking USD	386.28	821.19	0.0	159,292.38
Booking dummy	0.0279	0.165	0.0	1.0
Clicking dummy	0.0447	0.207	0.0	1.0

Another feature that might be of interest is the property star rating. Intuitively speaking it could be the case that customers are only interested in a minimum of stars when booking an accommodation. We see that the mean score is close to 3 (on a scale of 0 to 5) and by further investigating the feature by plotting it against the average score (our target variable to indicate a click or a booking) we see that four star rated properties have the highest average score. Maybe five star properties are too expensive for the average customer, so customers settle for a four starrating. Also note that the value 0 has quite a high score for a hotel with '0' stars. This might indicate a different meaning then 'worse' hotel and will be examined further in the missing values subsection. Figure 1 shows the average score for each property starrating.

Figure 1: The average score, meaning if a customer clicked or booked for each star rating.

Next let us analyze the dependencies of the features. We can do so by exploring the correlation matrix shown in Figure 2. It might be helpful to discard highly correlated features as this will cause a bias in the coefficients of these features. However, considering the correlation matrix there do not seem to be very highly correlated features. The lowest negative correlation is around -0.3 and the highest positive correlations are around 0.5. We see some higher values for the competition values, however due to the fact that a lot of those feature have a huge amount of missing values we will discard some of them, lessening the correlation between them.

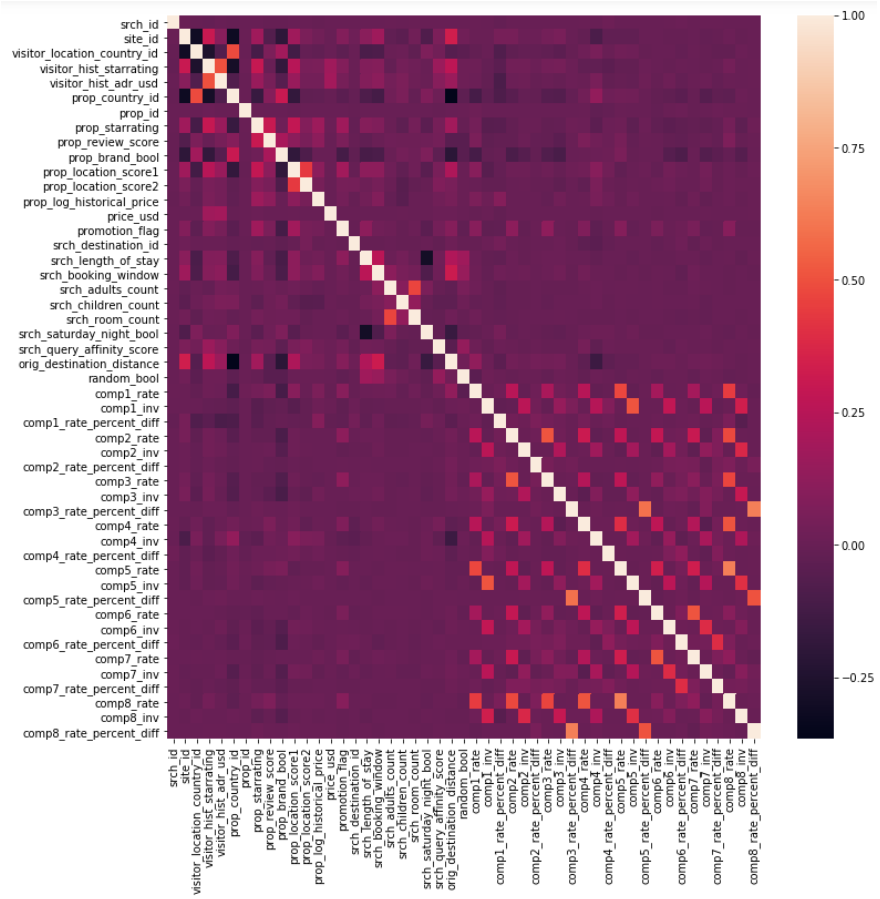

Figure 2: The correlation matrix for all original features in the dataset. Dark colours denote negative correlation whereas light areas denote positive correlation.

4 Data Preparation

We started the data preparation with winsoring a few outliers this was done at a 1 and 99 percentage level for the features price_usd, visitor_hist_adr_usd and all the comp_rate_precent_diff features, as these all seemed to have values that are significant different from the other observations.

Further we will consider two ways of handling the missing values. The first one will be by imputing them using various techniques described in this subsection, whereas the other will be to let the two algorithms make their own decision on how to handle them and is further described in the methodology section.

Firstly, we will consider handling the data manually. We find that all *comp_inv*, *comp_rate* and *comp_rate_perc_diff* features (dummies that indicate whether there is availability on the competitors website, whether the price at the competitor is lower and the difference in price among Expedia and the competitor, respectively) all have more than 85% of the values missing. Because of the fact that *comp_inv* and *comp_rate* are categorical values we choose to keep them in the dataset by assigning the null value to a separate category. The same approach has been used for the *property star rating* and *property review score*. We choose to not leave these features as numerical ones because of the assumption that a null value must indicate something else. It could be that hotels decide not to provide the star rating because they find it too low or not representative. Please note that it has been analysed whether null values for a certain property ID were not values in between 1-5 for other instances.

As the rest of the missing values *'competition'* features are numerical and have more than 90% missing values, you might want to say that it does not make sense to include them in our dataset. The features *prop_location_score 2* and *origin_destination_distance* have a relatively low ratio of missing values and we therefore impute those numerical values with a more sophisticated method. This has been done using the *IterativeImputer* class provided by SKlearn, which will model each feature with a missing value as a function of other features. This is done by selecting the feature column with missing values as the target y and the other columns as input X. Then a regressor is fit on (X, y) for known y after which a prediction can be made for the unknown y. As a regressor function we choose Bayesian Ridge. This technique comes in very handy as the regularization parameter is not set beforehand but tuned to the data at hand. For a Bayesian regression the output

y is assumued to be Gaussian distributed around $X\omega$ and is given by (Bishop, 2006):

$$p(y|X, w, \alpha) = \mathcal{N}(y|Xw, \alpha)$$

where α is treated as a random variable that is to be estimated from the data.

Bayesian Ridge then estimates the probabilistic model by setting ω to a spherical Gaussian which is given by:

$$p(w \mid \lambda) = \mathcal{N}(w \mid 0, \lambda\text{^}(-1)\ \mathbf{I}\_p\ )$$

Giving that we had 131 features, it is important to try to reduce dimensionality. This could reduce training time considerably. With the missing values imputed we used standard Principle Component Analysis (PCA). There are alternatives like Nonlinear Iterative Partial Least Squares (NIPALS) and Probabilistic Principal Component Analysis (PPCA) that can construct principal component when there is missing data, however these methods are only tolerant at maximum fifteen percent missing data. As we had features with more than fifteen percent missing we chose to first impute all the missing values and then use standard PCA. To capture 95 percent of the explained variance we needed 64 principal components and for 80 percent explained variance 45 principal components. As this seemed quite a lot we constructed a scree plot. As can be seen in figure 3 there is no clear "elbow" to be detected in the graph. Therefore, we decided to keep all the features and not use PCA (Cattell, 1966).

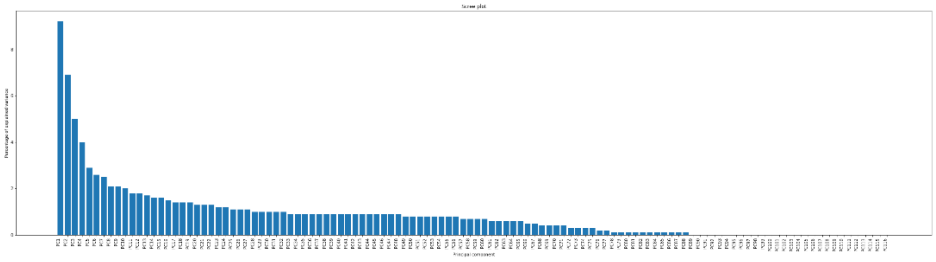

Figure 3: Scree plot with principle components on x-axis and eigenvalues on y-axis.

5 Feature Engineering

Moreover, we created some additional features. We based these additional features on intuition as well as on Bingshu and Wang & Kalousis, who, we mentioned before, scored both high on the Expedia competition.

Considering that we had a date_time feature that recorded the date and time of the search, a feature that recorded the number of days in the future the hotel stay started from the search date and a feature that record the length of the stay we constructed dummies for the day of the week, the month and the year of the search day, checking day and checkout day. In addition, we created a dummy whether the search is done during business hours (08:00 – 18:00), after business hours (18:00 – 23:00) or during the night.

Next, we created several dummies. One called “family dummy” equaled one if the number of adults and the number of children is higher than one. Another called “multiple families” which is one if the number of adults is higher than two and they looked for more than one room. We also created a dummy if the hotel is not in the country of the search. And a dummy if the search is for a hotel in the same country as the searcher. In addition, we created three variables. The first is the price a night, this is de price in USD divided by length of stay. The second is the star rating difference between the hotel star rating and the average star rating of previous bookings of the searcher. The third is the difference between the price a night of the hotel and the mean price a night the searcher previously booked. Finally, we created three features to compare competition for the corresponding hotels. One dummy that equaled one if a competitor had a lower price than Expedia. One variable that counted the amount of competitors that had no room available. And lastly, a variable that logged the percentage difference between the price of Expedia and the cheapest competitor.

6 Methodology

As mentioned in the introduction we will be considering two models in this research, namely XGBoost and LightGBM. Both models use gradient boosting techniques which we will introduce shortly before explaining the models themselves.

Let us first consider a gradient boosting machine used for classification. Gradient Boost starts with an initial leaf that contains an initial prediction made for the target feature. This initial prediction is given by the log of the odds. It is comparable in a way to the average probability calculated in a Logistic Regression. To use this prediction, it is convenient to transform it into a probability. Then, for each observation the residuals are calculated by taking the predicted values and subtracting the observed ones (either 0 or 1). Then by applying a learning rate to the previously made tree, the predicted value comes closer to the observed one, by decreasing the error term with each next tree. For a better mathematical understanding please refer to the paper of Freedman (Greedy function approximation: A Gradient Boosting Machine, 1999).

Knowing this we can proceed to the next step in our model, namely the fact that we do not want to classify the property shown on Expedia as correct or incorrect, however we want to know how relevant the property shown is relative to the other properties shown. That is where we focus on Learning to Rank (LTR) models. As said these models are more interested in the relative ordering than the outcome of the single instances, which is why the objective function is different. In this report for both models we use the LambdaMART as an objective function. Using this technique, we rank the instances with a pairwise classification or regression problem. Only a pair of observations is considered at each iteration after which a final ordering of all observations is made. As a more technical explanation of the algorithms is outside the scope of this report these can be found in the paper of Burges (From RankNet to LambdaRank to LambdaMART: An Overview, 2010). To evaluate the ranking that LambdaMART provides we use the Normalized Discounted Cumulative Gain (NDCG), since this is also the score used in the competition. This metric tells us something about the gain of the property regarding its position. Here a relevant property placed in the top of the search will have a larger gain than a relevant property listed at the bottom.

Having defined the objective function and evaluation metric we can analyse the differences between the two models. XGBoost introduces two new techniques over the traditional Gradient Boosting machines. The weighted quantile sketch is one of those techniques describing the approximation of splits that have to be used. The second technique is sparsity-aware split finding, which finds the best possible split in case a value is missing (Chen & Guestrin, 2016).

LightGBM also introduces two main upgrades in order to improve importance, namely Gradient-based One-Side Sampling and Exclusive Feature Bundling. The first technique is designed to improve speed. By

adding a hyperparameter alpha, which stands for the ratio of informative samples that need to be preserved, a linear search over the samples can be conducted as the complexity of the model is lower. The second techniques, bundles feature together as the name already suggests. The outcome is that we can have the same histograms with the bundled features as we would have plotting them individually, substantially decreasing the complexity of the model. (Moreno, 2018)

LightGBM has been build having XGBoost as basis. It mainly stands out in its speed to reach top accuracy over the given objectives. In this report we will test this difference in speed and accuracy of the two best performing LTR models in literature.

7 Model Evaluation

In order to evaluate the two models and also taking into account the large number of missing values for some of the features we will consider four models in total. As both XGBoost and LightGBM have their own way of handling missing values by using Sparsity-aware split finding and Exclusive Feature Bundling techniques respectively, it is interesting to analyse which method performs best taking into account our own manual imputation method described in the data preparation subsection. We have trained all four models on the training set using cross validation. That way we ensure that no information from the test set is leaked into the test set. This procedure results in the following average outcomes of the folds, as can be seen in Table 2:

Table 2: The average folds scores for all four models including the training time in seconds.

	LightGBM imputed missing values	**XGBoost imputed missing values**	**LightGBM no imputation**	**XGBoost no imputation**
Time	39.85 seconds	854.37 seconds	43.51 seconds	549.93 seconds
NDCG	0.367	0.371	0.369	0.377

We see that the results are relatively close to each other, which is in line with the results in literature (Ke, et al., 2017). What also has been tested in this report is the speed of the models, the results are that the LightGBM trains a lot faster than the XGBoost and is quite close in performance. However due to the fact that training time does not really play a role (since we would like to achieve the highest score possible), we decided to stick with the XGBoost for further model optimization.

To further optimize our final model, we perform an exhaustive grid search on three parameters, namely the number of boosts, the learning rate and regularizations parameter gamma. We do not focus on the tree-based parameters as we lack the computational power to do this thoroughly. The results of the optimization are as follows: for the number of boost we get 500 boosts as our optimal score. The scores for 400 and 300 boosts are relatively very close to the score of 500 boosts, however as we do not really care about the training time we can set the number of boosts to 500. The parameter Gamma which, if set higher than 0 it helps to reduce the complexity of the model with regularization gives the best results when kept at 0. This means that the complexity of the model will not decrease, however as Gamma is often used to increase speed of the training this will not be a problem for this case. At last we look at the learning rate which is used to update the last made tree with as described in the methodology subsection. The learning rate provides the optimal score over the folds when set to 0.05.

After creating our final model, we want to take a closer look into the feature importance of our model. Figure 4 shows the feature importance of the most ten most important features of our model. We can see that the features 13 (price_usd), 7 (prop_starrating), 11 and 10 (prop_location_score1 and 2) are the most important features for our model.

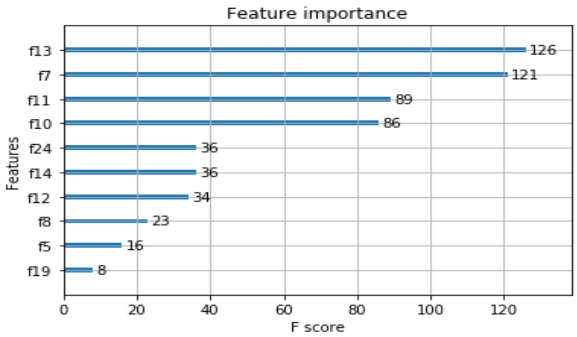

Figure 4: Feature importance plot of the ten most important features for our final model (XGBoost, boosts = 500, gamma = 0, learning rate = 0.05)

Uploading our final output of the optimized model trained over the whole training dataset into Kaggle we get a decent NDCG score of: 0.377

8 Conclusion

Learning to rank models are different from regression or classification models, where the relative position of the instances matters a lot. This kind of problem results in finding models that are adequate in handling this relative position objective as well as learning the coefficients of the features in establishing these relevant positions. We find that Gradient Boosting Models: XGBoost and LightGBM are capable in understanding these relationships. The final NDCG score of 0.377 can be interpreted as 38% of our listed properties on the website are relevant to the user and listed correctly. Naturally there is room for improvement, which could be done by a more extensive feature engineering part and a more sophisticated grid search for optimal hyperparameters. Having that said, we still observed good performance of the XGBoost and LightGBM for learning to rank problems. LightGBM comes very close in performance, however is way faster in converging to the optimal scores. The final XGBoost model has shown its strength in this research.

Doing this research, we learned quite a lot. At the start we were all quite unfamiliar or did not even know learning to rank models. In addition, at the start we have not used boosting algorithms before, so getting experience in this field by doing this research we all learned a lot. Since the data we were working with was quite large we also ran into computational problems. We solved this issue by looking into more efficient code so that the code wouldn't have to run for several minutes. Moreover, we also learned how to work with gradient boosting libraries and their syntaxes. Finally, doing this all on Kaggle has interested us in the field of data mining and some of us might want to participate in another competition in the near future.

9 References

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.* New York: Springer.

Burges, C. J. (2010). From RankNet to LambdaRank to LambdaMART: An Overview. *Microsoft Research Technical Report*.

Cattell, R. B. (1966). The Scree Test For The Number Of Factors. *Multivariate Behavioral Research*.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Association for Computing Machinery*.

Friedman, J. H. (1999). Greedy function approximation: A Gradient Boosting Machine. *Stanford University*.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., . . . Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *31st Conference on Neural Information Processing Systems*.

Moreno, H. (2018, October 15). *Medium.com.* Retrieved from Gradient Boosting Decision trees: XGBoost vs LightGBM (and catboost): https://medium.com/kaggle-nyc/gradient-boosting-decision-trees-xgboost-vs-lightgbm-and-catboost-72df6979e0bb