Data Mining - The 'Personalize Expedia Hotel Searches - ICDM 2013' Kaggle competition

Juan Buhagiar
juan.buhagiar@student.uva.nl
University of Amsterdam
11576014

Daniel Galea
d.e.galea@student.vu.nl
Vrije Univeristeit Amsterdam
vunet_id: dga560
student no.:2621376

Sarantos Tzortzis
sarantos_tzortzis@icloud.com
University of Amsterdam
uva: 11863331
vunet_id: sts950
vu student no: 2624691

https://github.com/sarantinio/Data-Mining.git
May 25th, 2018

1 Introduction

The aim of the project is to predict whether a property will be booked or not, and in order to do so we must go through a data mining process. The data mining process is made up of nine steps, however we did not necessarily follow the steps in the correct order.

The first important thing to do was to read related work and noting any techniques they used and also any faults that they had. This will save us time as we will not repeat the same mistakes and we will also know which techniques work better than others, and why. Following this we analyze the data. Here we look at the data-sets and note how many rows and columns there are, what each column data type is/should be, what percentage of missing values each column has, data balance (such as how many True/False does the field "booking_bool" have etc...), and correlations between columns.

After these two steps we cleaned the data-sets and filled in missing values so that we can run more tests and also visualize the data. We then extracted features and determined which to use, which model(s) to use, and how to split the data. Finally, we ran our models and validated them.

2 Business understanding

The 'Personalize Expedia Hotel Searches - ICDM 2013' Kaggle competition started around 5 years ago and was open for 3 months. During this time there were 337 teams participating and the top submissions had a score ranging between 0.53-0.54. The winning submission by Owen Zhang obtained a score of 0.553984.

We were able to obtain some information about the wining teams as they have posted their presentations online. The problem at hand is split into 3 smaller problems in most literature, in the same way we split it in this paper, Data understanding, Data preparation and Modeling. From the Data understanding point of view, many research we have found describes simple ways of correlation between features, such as a simple Pearson Correlation. In the spectrum of data preparation, the main concepts used here were imputation of missing data in features which have a lower percentage of missing data, dropping of features that have a high percentage of missing data, using 10 % of the data and balancing the data with regards to positive and negative examples.

As for the modeling aspect, there are generally two different approaches, pointwise models and pairwise or listwise models. The former generally includes Random Forests, Gradient Boosting Machine & Logistic Regression. While the pairwise & listwise models generally include Pairwise

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Logistic Regression & Lambda Mart. In general, it seems that almost all the top submissions have utilized with Lambda Mart.

Trying to understand better the intuition behind the choice of the correct model, we observe that the highest rankings were achieved among the ones who used non linear models. In fact using linear models are more difficult to handle categorical (discrete) features. That is probably because features are not linear correlated but we will find out in later sections. Thus non linear models like the LambdaMart seems more prominent.

3 Data understanding

Following the literature review (business understanding) it was time to delve into the data and gather more insight. The first thing we looked at was the size of the data, and here we focus on the training data-set. However, we also analyzed the test data and found that the amount of missing data was basically the same as the training set. Going back to the train set, it is consisted of approximately 5 millions rows and 54 columns. We were interested in learning more about the data. Such as, how many missing values per column there are, how balanced the data is, and also any correlations between columns.

Once we knew what we wanted to look for, we decided to start analyzing the data. First we computed the amount of missing data for each column, and it resulted in the following table:

Column Name	**Missing Data %**
visitor_hist_starrating	94.92
srch_query_affinity_score	93.6
gross_bookings_usd	97.21
visitor_hist_adr_usd	94.9
orig_destination_distance	32.43
prop_location_score2	21.99
prop_review_score	0.15
comp1_rate	97.58
comp1_inv	97.42
comp1_rate_percent_diff	98.1
comp2_rate	59.17
comp2_inv	57.4
comp2_rate_percent_diff	88.78
comp3_rate	69.06
comp3_inv	67.24
comp3_rate_percent_diff	90.46
comp4_rate	93.8
comp4_inv	93.16
comp4_rate_percent_diff	97.36
comp5_rate	55.18
comp5_inv	52.95
comp5_rate_percent_diff	83.04
comp6_rate	95.16
comp6_inv	94.79
comp6_rate_percent_diff	98.06
comp7_rate	93.64
comp7_inv	92.9
comp7_rate_percent_diff	97.21
comp8_rate	61.34
comp8_inv	60.54
comp8_rate_percent_diff	87.6

Table 1: Missing values percentage for each column with missing values from the original training file

As can be clearly seen in Table 1 there are 31 out of 54 columns which have missing values. Three columns have *0% < missing_value_percentage <= 50%* , while the other 28 columns have *50% < missing_value_percentage <= 100%*. Some of these fields can be aggregated and imputed, other fields just need to be imputed, and some columns will simply be dropped but we will go into more detail in the Data Preparation section.

We also noticed that whenever *booking_bool* is true, so is *click_bool*. Therefore, we decided to look at the number of true/false values for *click_bool* and *booking_bool*. The following table shows these figures:

Condition	Amount
click_bool - True	221,879
click_bool - False	4,736,468
booking_bool - True	138,390
booking_bool - False	4,819,957
click_bool and booking_bool - Both True	138,390
click_bool and booking_bool - Both False	4,736,468
click_bool and booking_bool - True and False	83,489
click_bool and booking_bool - False and True	0

Table 2: *click_bool and booking_bool values under different conditions*

As we can see from table 2, there is a huge imbalance in the data when comparing the True vs False amount for each column, and combined. This indicates that we will need to attempt to balance the data in order to achieve better results.

3.1 Data as groups of *search_id*

The *search_id* defines a user search while the *site_id* defines the hotel property. A useful thing to understand is that there is a **unique key** as the pair of (*search_id*,*site_id*) while the *search_id* itself is not unique because for just one search there is a list of hotels that match. That is the first essential understanding as our model should predict a ranking for the *site_id* for each group of *search_id*.

3.2 Finding useful relations among the data.

The column *position*, which only exists in the training set, seems very useful as we assume that Expedia tries to show the more relevant results based on this position value. Then the *booking_bool* and the *click_bool* fields, which again only exist in the training set, inform us on whether the user clicked to see this property and if eventually the user booked it or not. The *prop_review_score* holds the hotel's review and the *price_usd* is the displayed price. These two columns should greatly affect the users on their choice. So we want to find out what is the distribution of the price and the review of the hotel in case that this hotel is clicked and then if it is booked.

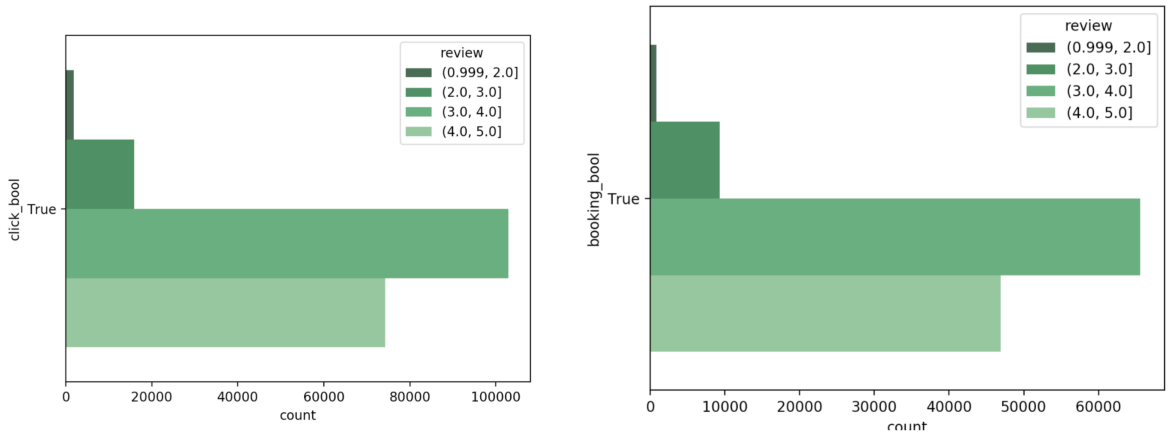

Figure 1: The hotel's review when a user clicks on it or books it.

The Fig. 1 actually indicates that the review of the hotel strongly affects the users' choice. In fact most users will neither click or book the hotel if its score is not above average. That means that the column *prop_review_score* should be essential for our model.

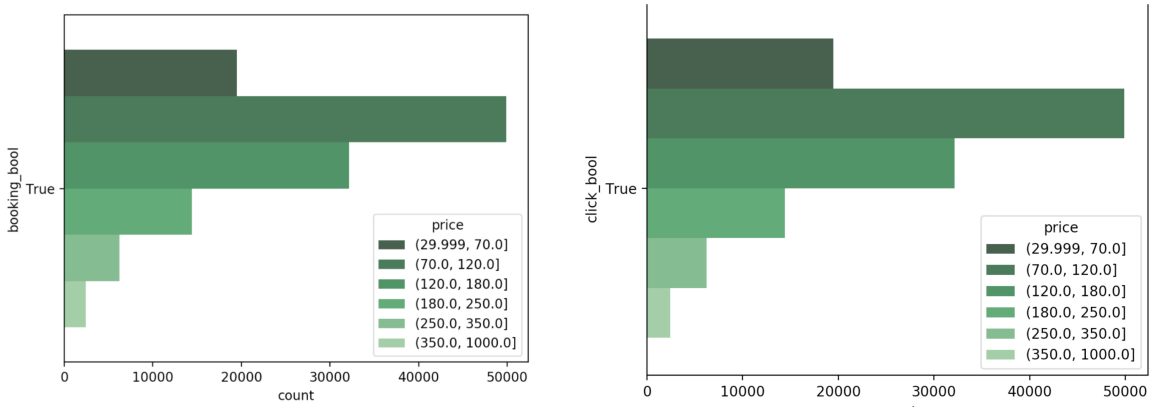

Figure 2: The hotels' price when a user clicks on it or books it.

The Fig. 2 shows that users are mostly inclined to click or book hotels which are cheaper.

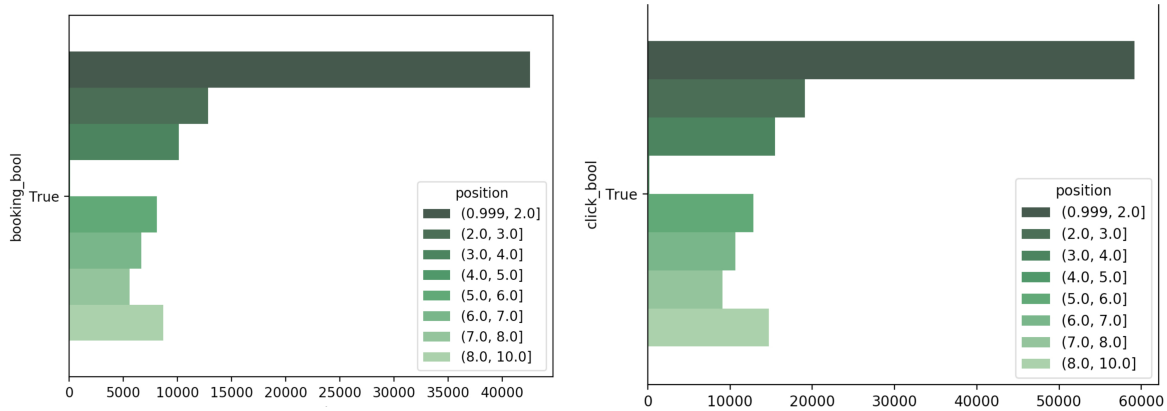

Figure 3: The hotels' position when a user clicks on it or books it.

Fig. 3 confirms our belief that users are more likely to book hotels which have a better position. Meaning the ones that appear first in the list.

3.3 The issue of linearity

A traditional approach to the problem would be to use machine learning linear algorithms like Linear Regression, Decision Trees etc. The reason that this task is interesting is that we have various columns which we are not sure if or how they are linearly correlated. In order to check this we will run a correlation test, using the Pearson method, which will result in a correlation matrix. In the following table we show columns which where of interest to us in terms of correlation and linearity. This table is not the correlation matrix for all of the columns.

	prop_review_score	**prop_starrating**	**price_usd**	**click_bool**	**booking_bool**	**position**
prop_review_score	1	0.30645	0.00140	0.02342	0.02580	-0.05398
prop_starrating	0.30645	1	0.00777	0.03079	0.02121	-0.10772
price_usd	0.00140	0.00777	1	0.00051	0.00007	-0.00188
click_bool	0.02342	0.03079	0.00051	1	0.78289	-0.16499
booking_bool	0.02580	0.02121	0.00007	0.78289	1	-0.14792
position	-0.05398	-0.10772	-0.00188	-0.16499	-0.14792	1

Table 3: Correlation between specific columns

The table shows that there is very low linearity between columns that we know are highly correlated. Previously we saw in Fig. 3 that *click_bool* and *position* are correlated, because the better the position was the higher the amount of *click_bool* **True** were. However, the correlation test showed that there is a negative correlation between the *click_bool* and the *position* and this makes sense. The reason being that *position* and *click_bool* do not hold the same value type and so they cannot be linearly correlated. We can even see that there seems to be very low correlation between *price_usd* and *click_bool*, or *price_usd* and *booking_bool*. Logically this doesn't make sense to us, and in fact this low linear correlation doesn't mean that they're not correlated ,and in Fig. 2 we can clearly that the price does have an affect on whether a property is clicked on and booked.

Also, we noticed that the *date_time* held the day, month, year, hour, minutes, and seconds. We figured that the year and time is not so important, while the day and month are more important due to high seasons etc. Therefore this can be split into day and month, as separate columns.

Summing the above, we understand that a linear model will not work as intended for our task. Therefore, based on previous work, we have decided to go with the LambdaMART model. More on this will be discussed later.

4 Data preparation

In this section, we investigate all the different features present and modified some to improve their discriminatory ability. We look also at different approaches to solve the problem of missing data.

The initial features we investigated were the competitor data, which have a large percentage of missing data. The competitor data contains 3 features for each of the 8 competitors which include the comparison of the rate, availability and the price difference between the competitor which on average have 78.12%, 77,05% & 92.58% of missing data respectively. Our idea was to combine the competitors features into one set. We have done this by iterating over every data row and performing the following:

1. For each competitor rate data row, we find the most common value amongst the 8 competitors and set this as the new value.
2. For each competitor availability data row, we perform the same operation as the rate data.
3. For each competitor difference between rates, we do an arithmetic average over the available data.

With this procedure we merge the 27 features from the 8 different competitors into just 3 features named 'comp_rate', 'comp_inv' and 'comp_rate_percent_diff'. This leads to the new comp data having 34.62%, 33.35% & 68.12% of missing data respectively. This results in 43.5%, 43.7% & 24.46% less missing data then the previous features on average.

Another feature which we decided to change was the 'date_time' feature which was a string of the date & time when the search was performed. We parsed the date time string and converted the information into two features, 'month', the month when the search was performed as an integer and 'day', the day of the month when the search was performed.

Table 4 shows the final list of features that have missing data. We have included the percentage of missing data and if we will impute the missing data or remove the feature all together. We have based these decisions on the % of missing data. To impute the missing data we use a KNN classifier similar to [1].

Feature	**% of missing data**	**Impute (I) or Remove (R)**
comp_rate	34.62	I
comp_inv	33.35	I
comp_rate_percent_diff	68.12	I
gross_bookings_usd	97.21	R
orig_destination_distance	32.43	I
prop_location_score2	21.99	I
prop_review_score	0.15	I
srch_query_affinity_score	93.6	R
visitor_hist_adr_usd	94.9	R
visitor_hist_starrating	94.92	R

Table 4: The table shows features which have some missing values and their respective percentage of missing values. For each feature we include our decision if we are going to impute the data or remove the feature

To prepare the data for the classifier, once we have no more missing data, we decided to "balance" the data. We select all the data rows that have been booked. This leads to 138,390 rows of data. We then look at the search id of each booked property and get a maximum of 10 other rows for the same ID that were shown but not booked. However, there were cases where there would be less rows, such as five rows for a particular ID, four of them marked as not booked and one marked as booked. This will lead to 1,463,147 data rows which will be the final rows of data used.

To be able to evaluate the data using the Normalized Discounted Cumulative Gain (NDCG), we introduce a target feature called 'relevance_score' which assigns the following scores:

5 - The user purchased a room at this hotel (clicked == True & booked == True)

1 - The user clicked through to see more information on this hotel (clicked == True & booked == False)

0 - The user neither clicked on this hotel nor purchased a room at this hotel (clicked == False & booked == False)

5 Modeling and Evaluation

5.1 Model Choice

The initial step is to select a model to train. Based on the research in the previous work and the analysis of our data, we understand that our data is not linearly spreadable because of the lack of high linear correlation. Therefore, we look at non-linear models such as Gradient Boosted Trees, Random Forest, LambdaMART and GBM. Among these LambdaMART seemed the most popular model that many competitors used on Kaggle and so we try to use this model for our training and predictions.

5.2 Learning to Rank and LambdaMART

Learning to Rank algorithms are techniques of supervised Machine Learning in order to rank. Mainly, the difference between the traditional ML approach and LTR is that the former aims to predict a single instance (classification label, a score, spam detection for a specific mail), while the latter pays attentions to their relative ordering of the instances without necessarily comes up with a score.

LambdaMART model translates the ranking into a pairwise classification problem. Intuitively we understand that the model will try to predict which is best between a pair at every time, and finally it will come up with an descending sorted list(highest to lowest relevance)

The chose of LambdaMART specifically were made because on average this model scores better as it combines properties from both LambdaRank and Multiple Additive Regression Trees.

The *pyltr* library was imported which is learning-to-rank toolkit with ranking models and evaluation metrics using the LambdaMART.

5.3 Training and Evaluation

Once this library is installed, the next step is to feed the model with our dataset. As described at the previous section our training data is now consisted of roughly 1.5 millions rows which is a balanced dataset including all the rows with a true booking values and 10 respective false booking values whose missing data was imputed and a relevance label column was added.

To evaluate the model we use our balanced training dataset explained above with 10-fold cross validation. The 10-fold cross validation is performed on the unique search ids and then the respective features and labels are obtained. This is done so that we do not have any searchids in different folds. This procedure will help us understand how well/accurate our model works. Also the *pyltr* library provide us with a built in NDCG metric. That's why we created the column *label relevance*. The NDCG metric is set with a length of 10 (ie. NDCG@10).

In order to achieve a higher score a parameter tuning is also necessary. Empirically and after some experiments best score was obtained with *learning rate* at 0.01. The *n_estimators* which is the number of boosting stages was set to 500, while by default was just 100. The intuition behind this is that gradient boosting is robust enough to overfitting so, to a certain extent, the larger the better. The rest of the features contributing to the tree that *LambdaMART* is constructing were set empirically after trying different values. The Table 5 indicates the final tuning.

Our final Average NDCG@10 score over the 10-fold cross validation is **0.21**.

Table 5: LambdaMART parameters.

Parameters	**Value**
n_estimators	500
learning_rate	0.01
max_features	0.5
query_subsample	0.5
max_leaf_nodes	10
min_samples_leaf	64

5.4 Prediction

Now the model is ready to predict ranking on the test set. The model is re-trained using the full balanced train data using around 1.2 million rows as training data and 500K rows as evaluation. Finally, we used the trained model to predict the full test set, which had the same feature engineering and missing data imputation as the train set. From these predictions, for each search id, the rows are sorted by their predicted score and outputted to the prediction file.

6 Conclusion

6.1 Knowledge Gained

In this project we faced quite challenging tasks. Categorical values with low linear correlation make the ranking task harder. For this case we do not just feed our data to a classifier but we did lot of preprocessing. The relevance label score is a sound example of combining columns (click_bool and booking_bool) to gain more knowledge. We also tackle issues as how to fill missing value and based on what tuition. We also understood differencies between traditional ML approaches that classify and LTR that ranks.

References

[1] Acuna, Edgar, and Caroline Rodriguez. "The treatment of missing values and its effect on classifier accuracy." Classification, clustering, and data mining applications. Springer, Berlin, Heidelberg, 2004. 639-647.