Recommendation system to predict hotels ranking

Martin Banchero: 2589681 - mbo440, Davide Bressan: 2652725 - dbn299, and Dennis Dekker: 2656394 - ddr850

VU, Amsterdam, The Netherlands

1 Introduction

During the last decades different data mining tools became more popular especially in commercial applications. As a part of this, recommender systems can be seen as a very useful tool, focus in providing users with different recommendations of items according to their preferences and interests [1]. These recommender systems then aim to predict a rating that a user would give to an item, and thus build a ranking from the most likely to the less likely to be picked. As part of a Kaggle competition the goal of this project is to develop a recommender system to predict a ranking of hotels that a user is most likely to book. In order to achieve this goal a dataset originated from a global travel company called Expedia is used. This dataset contains around five million records, each of these contain the information of queries carried out by users searching for a hotel. In order to build the recommender system a variety of methods are applied, like exploratory data analysis, feature engineering, feature selection and different alternatives to deal with missing values.

Once the data was preprocessed, two models (Elastic net Regression and LambdaMart) are implemented in order to find which of these methods performs best for the ranking task.

1.1 Related work

As part of this project different searches on literature and in previous works were taken into account. Different presentations from the Kaggle competition of 2013 were analysed to find some potentially useful data preprocessed procedures. One of them is to tackle the creation of new significant features, which was found to be a fundamental part for the success in the competition [2].

In particular, an ensemble of models like LambdaMART or Gradient Boosting Machines [3], shown to yield an improved performance in contrast to use a single model.

2 Methods

2.1 Exploratory Data Analyses

The exploratory data analysis (EDA) consists of multiple steps, starting with investigating the structure of the data. This was done in order to observe if there

were dependencies between variables. Also, the expected range of values that can be found in the data. As is shown in the heat map in figure 1 some features show positive correlation (in red) and some others negative correlation (in blue). Following this line figure 2 shows the dependency between the feature *position in the page* vs the occurrences of click bool/booking bool (see 2.3 for hotels, that is an important finding to keep in mind to build a reliable scoring system.

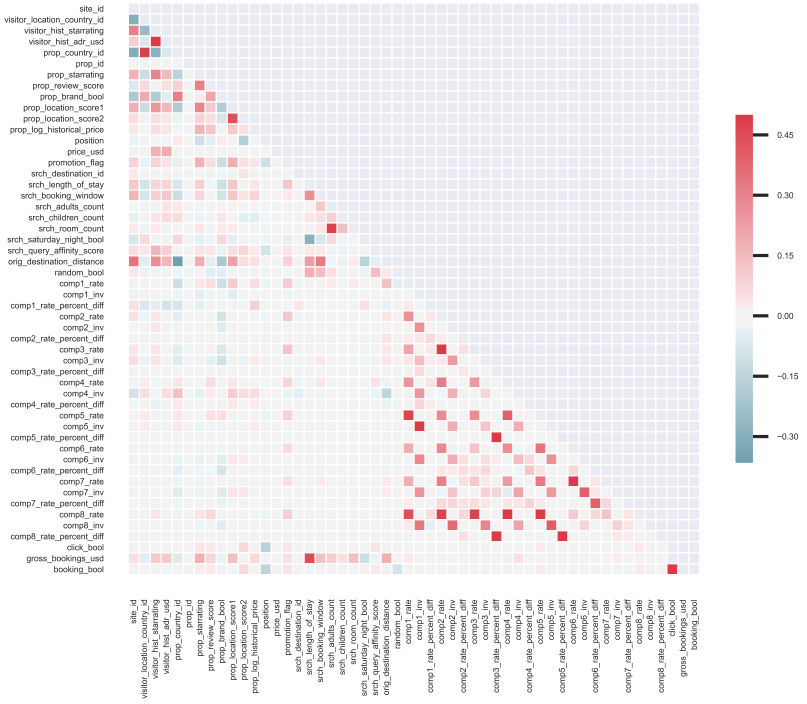

Fig. 1: Heat map of the raw training set, positively correlated features are displayed in red, negatively correlated features in blue.

Another important step is to verify the quality of the data in which, for example, is expected to get an overview of the amount of missing values, outliers and errors. As is shown in figure 5, the range of values for the feature *price_usd* (the price for night in an hotel) has to be corrected due to the presence of outliers.

In order to get an insight in the distribution of missing values in the data set, the proportion of missing values per feature is represented in a barplot in figure 3. This plot shows that all the *comp_* features (the difference among Expedia and a competitor) present many missing values, this means that there is no competitive data available from the different competitors of Expedia. Also the features *visitor_hist_starrating and visitor_hist_adr_usd* show a high proportion

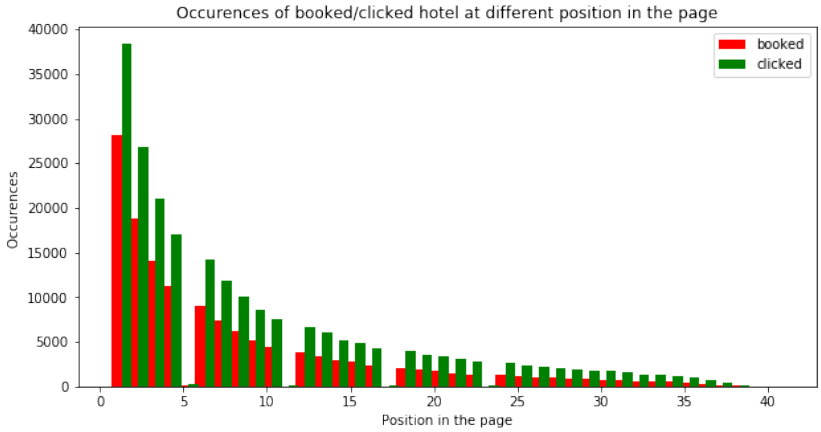

Fig. 2: Plot of position in the page against number of clicked and booked hotels

of missing values. The methodology that was applied to deal with missing values is shown in subsection 2.2.

2.2 Data Pre-processing

The pre-process of the data was performed to deal with missing values and to prepare the dataset to train different models. Below are shown some of the procedures that are performed in this project.

NaNs handling After analysing the training set, it has been pre-processed to tackle the different issues raised in the previous section (2.1). In particular, the presence of Nan is problematic to train a model, and in the dataset are present plenty. They have been handled as follow:

1. **orig_destination_distance**: the NaNs have been replaced by the third quantile of the distribution of the existing values.
2. **prop_review_score**: NaNs replaced with worst case scenario, a score of 0.
3. **prop_location_score2**: NaNs replaced with the median of all data
4. **srch_query_affinity_score**: Replace NaNs for worst case scenario, in this case -326.5675, which is the minimum value for this feature.
5. **visitor_hist_starrating**: NaNs replaced with the median of all data.
6. **visitor_hist_adr_usd**: NaNs replaced with the median of all data.

For the feature *usd_price* outliers have been removed (rows with a value for that feature higher than 50000).

Prepare the dataset In order to train the different models, the following scoring model has been applied: A score of 5 was assigned if the hotel has been booked, 1 if clicked and 0 for neither booked nor clicked. In this way it was

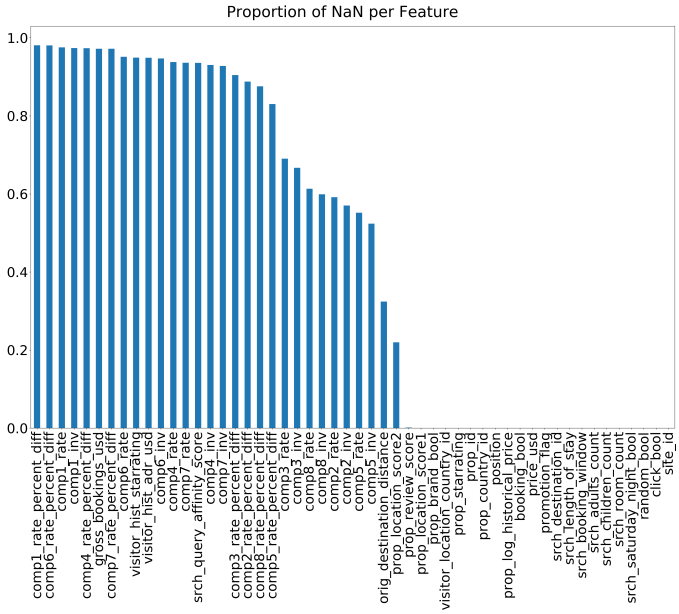

Fig. 3: Plot of the proportion of NaN per feature. All the features from the original dataset are represented.

possible to feed the regression algorithms Elastic Net, that had to predict a value for each hotel in the . Unfortunately, LambdaMART needs an additional step before it could ran. The input of LambdaMART has to be in SVMlight format, that is defined as in table 1.

Table 1: Example of SVM light format. The first value in the rows is the score, followed by the search id and finally the list with all the features

0 qid:1	1:0.53	2:0.1		...
5 qid:1	1:0.13	2:0.1		...
1 qid:2	1:0.87	2:0.4		...
⋮	⋮	⋮	⋮	⋱

Additionally, a new scoring system based on the position on the page of the hotels was tried out (due to the discoveries in EDA 2.1, see figure 2). Higher position on the website page means higher score, and thus the range of values

will spread. As a result of this implementation, a booked hotel not in the first positions has a lower score than a booked hotel in the first positions.

2.3 Feature engineering

Before the models were trained with the data, feature engineering processes were applied to the train and test data. The new features that were created are shown in table 2.

Table 2: Features that have been added to the original ones

Feature Name	**Feature Handling**
starrating_diff	$\|\|visitor\_hist\_starrating - prop\_starrating\|\|$
usd_diff	$\|\|visitor\_hist\_adr\_usd - price\_usd\|\|$
avg_comp_rate	mean(comp1_rate..comp8_rate)
booked_percentage	(booking_bool / counts) * 100
clicked_percentage	(click_bool / counts)*100
month	From the timestamp

In order to use the *booked_percentage* and *clicked_percentage* feature for prediction, these features have to be added to the test data. While the other feature can be calculated based on the test data, these two feature can not be extracted from the data as the features required to do this were not present (*booking_bool* and *click_bool*). This information is copied from the train data, what could be done for 121665 out of 129438 hotels, as those were present in both train and test data. For the remaining hotels both features were set to the first quantile of the other data.

Afterwards, other features were built, in order to improve the performance and to train another LambdaMART model (see table 3. Additionally, some of the original features that were seen to be useless were removed: *gross_bookings_usd*, *site_id* ,*srch_saturday_night_bool*, *srch_query_affinity_score*,*month*, *srch_room_count*, *srch_children_count*,*visitor_hist_adr_usd*.

Table 3: Additional features that have been created subsequently

Feature Name	**Feature Handling**
bool_same_country	1 if user and hotel are in the same country
roomcount_bookwindow	$F1\_F2 = F1 * max(Max(F2)) + F2$
adultcount_childrencount	$F1\_F2 = F1 * max(Max(F2)) + F2$
ad_vs_real	prop_starrating - prop_review_score

2.4 Models

Two models have been tested on the dataset, in order to find the most suitable for the ranking task. In particular:
- Elastic net (Regression)
- LambdaMART (Learning to Rank)

Regression The first two algorithms are regression methods, while the third one is a learning to rank algorithm. Thus, the dataset has been pre-processed in order to be used in either way (see 2.2). Random Forest and Elastic Net are well known methods for a regression task, and have been implemented using the package scikit-learn [4] and pyltr [5]. The former builds an ensemble of decision tree and then select those that gives the lowest error [6], while the latter is a linear regression method that incorporate LASSO and Ridge as regularizers.

Learning to rank A learning to rank algorithm is a supervised task that aims to order a list of items in the optimal ranking. Hence, for each query (searches in the website) is associated a list of documents (hotels) with their relevance (booked or clicked) [7]. LambdaMART is a learning to rank algorithm based on boosted regression trees (MART, Friedman, 2000) but using the cost function defined in LambdaRank (Burges et al., 2006) [6]. Among the others learning to rank methods (AdaRank, ListNet), LambdaMART gives better performance (see [3]). The python package employed to apply LambdaMART is a learning to rank toolkit (see [5]).

2.5 Evaluation

The evaluation metric used is the normalised discounted cumulative gain (NDCG), defined in equaton 2.5 [7].

$$NDCG(k) = G^{-1}_{\max,i}(k) \sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}} - 1}{\log_2 (1 + \pi_i(j))} \tag{1}$$

The parameter of the position in the ranking k is set to 5, and G (see 2)is the gain function, where y is the label (correct rank) of the hotel $d_{i,j}$ in the ranking list π_i.

$$G(j) = 2^{y_{i,j}} - 1 \tag{2}$$

3 Results

The results that have been obtained are shown in table 4. The first LambdaMART model (1) has been trained for 12 hours and 45 minutes and it fitted 714 estimators (0.1 of learning rate, 100 stop), whereas the second (2) for 34 hours and 27 minutes

with 2000 estimator and 0.02 as learning rate (100 stop). On the other hand, Elastic Net took only one hour and 12 minutes, and the parameters used were an alpha of 1e-05 and l1_ratio of 0.2.

Table 4: NDCG(5) scores for the three models, obtained on the test set.

Model	NDCG(k=5)
Elastic Net	0.30312
LambdaMART(1)	0.36941
LambdaMART(2)	0.36120

Additionally, in figure 4 is possible to see the features importance that came out from the trained model LambdaMART. The most important features are *booked percentage*, *clicked percentage*, *promotion flag*, *historical price* and *price usd*.

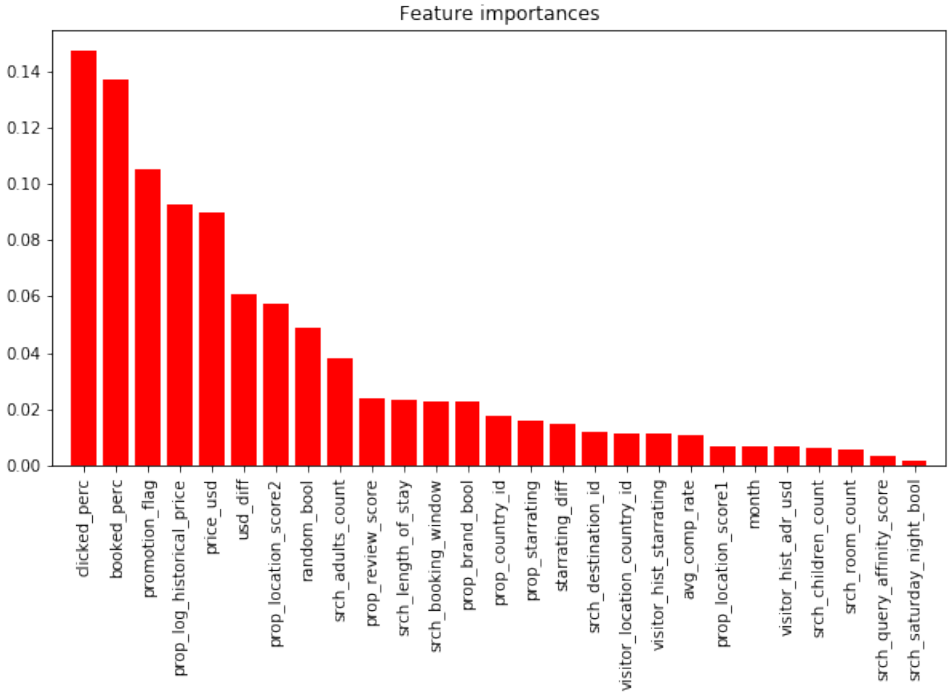

Fig. 4: Feature importance for the trained LambdaMART model (1).

4 Discussion

The performance (NDCG-5) of the second LambdaMART model (2) is worse compared to the first one. This was unexpected, as the second model has been trained on the dataset after removing some less useful features (based on figure 4) and after adding some new features (see 2.3). One possible explanation could be that the additional features did not improve the performance, and could therefore result in useless splits in the decision trees.

Another reason that led the second model to performed worse could be that the learning rate parameter (0.02 instead of 0.1) was not proportional to the amount of estimators (from 714 to 2000). This is more likely, as the first model terminated early (meaning that is used less estimators than the given due to the stop parameter) while the second model terminated when the amount of estimators (2000) was reached. If the algorithm was given more estimators, the second model could have outperformed the first model.

An alternative could be to increase the learning rate, for instance to 0.05, this to lower the amount of estimators needed. However, as we were limited by time and computational power, we did not used these parameters. The most important features that allowed for a big improvement on the performance were clicked_perc and *booked_perc*, also thanks to their transferability to the test set. Finally, it has been observed that a LTR model as LambdaMART outperforms a popular regression algorithm as Elastic Net for the given ranking problem.

4.1 Improvement for Team Members

During previous courses of our Bioinformatics Master, we explored the basics of Machine Learning and Data mining. The basic concepts of Machine learning were explained during the course "Machine Learning", in which methods like k-NN and decision trees were explained. Later we also learned more complex methods like neural networks and feature selection algorithms. In this course we also tried to implement different methods to get a better understanding and hand-on experience. This course in particular thought us ways to handle missing values, normalisation and methods to extract the most information from data and make reproducible pipelines.

Another important point is that the datasets used in the previous courses were content-wise, which is different from the data in this course. However, the methods to handle data remain the same.

In this course we expected to learn even more complex methods to make predictions based on data, and also to integrate all different concepts mentioned above. However, starting with the advanced assignment, we underestimated the work on data wrangling and ended up rather unsatisfied with the final result. This thought us that most of the work was not in creating a classifier or model, but in the data exploration and data handling. Keeping this in mind and with a lot of enthusiasm, we really wanted to excel in the second assignment. Even with this extra drive, the second assignment was still a lot of work.

4.2 Modelling limits

The first struggles we encountered the magnitude of the data, as well as the amount of time to run the models. Even with our computers on the highest performance settings, the shortest LambdaMart run-time was around 3 hours. Later we even performed a run of 35 hours, which unfortunately resulted in a lower prediction and a full day lost.

The amount of time was not the only problem, also the amount of data was sometimes too much to be handled and different workarounds had to be used to circumvent these issues. In the end, the most important lessons were that sometimes a limitation on the performance of a model is due to the limited computational power. Next to this, we now have a better understanding on how different concepts can influence each other, like NaN handling on the learning process or the choice of method on the amount of work to get the formatting correct. As a final remark, we really feel we now are more experienced and more confident in approaching a complex problem like this assignment.

5 Supplementary

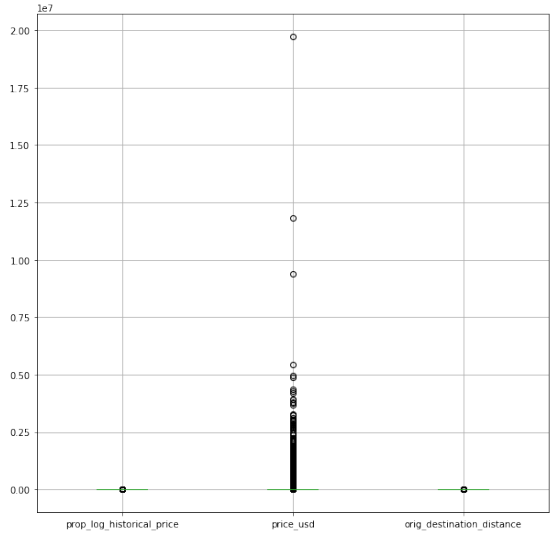

Fig. 5: Box plot for different features. Notice the outliers for the price usd.

References

[1] Maryam Khanian Najafabadi, Azlinah Hj. Mohamed, and Mohd Naz'ri Mahrin. "A survey on data mining techniques in recommender systems". In: *Soft Computing* 23.2 (Jan. 2019), pp. 627–654. ISSN: 1432-7643. DOI: 10.1007/s00500-017-2918-7. URL: http://link.springer.com/10.1007/s00500-017-2918-7.

[2] Owen Zhang. *ICDM 2013 — IEEE International Conference on Data MiningDallas, Texas / December 7-10, 2013*. URL: https://icdm2013.rutgers.edu/ (visited on 05/24/2019).

[3] Christopher J C Burges. *From RankNet to LambdaRank to LambdaMART: An Overview*. Tech. rep. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.180.634%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

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

[5] Jerry Ma. *Python learning to rank (LTR) toolkit*. https://github.com/jma127/pyltr. 2016.

[6] Kristofer Tapper. "Learning to rank, a supervised approach for ranking of documents". In: June (2015). URL: http://publications.lib.chalmers.se/records/fulltext/219663/219663.pdf.

[7] Hang Li. "A Short Introduction to Learning to Rank". In: *IEICE Transactions on Information and Systems* E94-D.1 (2011), pp. 1–2. ISSN: 0916-8532. DOI: 10.1587/transinf.E94.D.1. URL: http://times.cs.uiuc.edu/course/598f14/l2r.pdf%20http://joi.jlc.jst.go.jp/JST.JSTAGE/transinf/E94.D.1?from=CrossRef.