Predicting hotel preferences from user query search

Marie Corradi (2596976)[1], Elena Garcia (2604289)[1], Alberto Gil (2595259)[1]

MSc Bioinformatics and Systems Biology. Vrije University Amsterdam

Hotel booking and travel websites are one of the most prominent business of the moment. Expedia, for instance, had a net income of 281.8 million in 2016 [10]. The major advantage that companies like Expedia can benefit from is the huge amount of data stored in their data centers. Data driven decisions could have a major impact in the company's revenue if Big Data is handled properly. Nevertheless, this requires highly sophisticated data preprocessing and complex algorithms.

With this motivation, Expedia has launched to date two Kaggle competitions with the "Personalize Expedia Hotel Searches - ICDM 2013" [1] and in 2016 "Expedia Hotel Recommendations" [3].

This project is based on the the 2013 Expedia Kaggle competition. The dataset used was released by Expedia with the aim of improving their ranking on hotel searches. The main objective towards this challenge is facing the ranking problem of Machine Learning [16] in an objective and creative way, as the goal is to create a model able to rank hotels according to the likeliness to be booked.

1 Business understanding

The Kaggle 2013 competitions winners released several (potentially) useful descriptions about their approaches [2].

In Kaggle forum [1] several people used Extreme Gradient Boosting (XGBoost), which is based on Gradient Boosting [7].

We had a look at the 2016's Expedia competition winner's topic [4].

Some tips about how to handle this dataset can be found at Dataquest [18].

Another approach that we found was in [11], in which a separate Random Forest for every country_id was built yielding a NDCG score of 0.51. However, not only the algorithms were discussed, but the importance of feature pre-processing and engineering was also highlighted, with some of the top competitors having engineered over 300 features in addition to the ones initially given by Expedia.

2 Data Exploration

The original dataset was formed by a random selection from Expedia web-page [1], formed by 4.958.348 rows containing information shown on hotel's searches. Around 400.000 different searches comprised this dataset, having information from 129.113 different hotels, and 199.795 different searches (matching in average 25 hotels shown per search).

2.1 Overall exploration

For the data exploration, we used the format pandas data frame [17]. In order to make a better use of the date information, we transformed the column in two: 'year' and 'month'. We then plotted the distributions of the different variables (script data_exploration.py). The graphics (not shown) shows that the distribution of the values in the general features follows a normal distribution in numerical variables and uniform when variables are discrete.

In more detail, visitor_location_country_id has one location (country ID 219) that is the most abundant significantly, although srch_destination_id is similar across all IDs. Other interesting information is that the users search the most for 1 room and below 4 days. The majority of the searches are done for 2 adults and no children. The most common month booked is June (calculated using a new variable month_booked). And mostly there was no promotion flag.

2.2 Variables correlation

According to the winners of the Expedia competition in 2013, prop_starrating and prop_review_score are positively correlated, and there are strong (anti-)correlations between variables of hotel characteristics [2].

As observed in the heatmap[1], several variables are strongly correlated (either positively or negatively). None of the variables has such a strong correlation that we considered removing them because of redundancy.

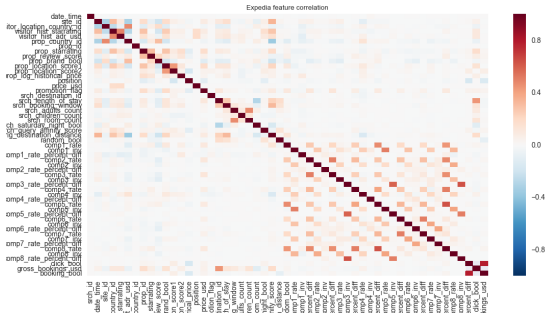

Fig. 1: Heatmap of all-against-all variables.

2.3 Effect of missing information of a query

Several variables lack of information in the dataset. There are variables with ¿90% missing data (specifically the visitors historical data (ratings, shops), the competitors information, the search query affinity score and the gross booking spent). Most of the search_id have 40-60% of the data missing. Around

32% of the rows representing a hotel booked are missing information from orig_destination_distance. We observed how the lack of information of certain variables translated into a lower probability of a hotel being booked [2, 3].

Fig. 2: Percentage of bookings that occur, when certain variables have missing values.

Fig. 3: Percentage of clicks that occur, when certain variables have missing values.

It is clear from the previous plot that prop_starrating, prop_review_score and prop_location_score2 have an impact on the user behavior when there is missing information.

2.4 Outliers

We then searched for obvious outliers that would spoil the models performance. Very few searches had price_usd bigger than $5.000 (figure not shown). Expedia is a platform where we would not expect clients to spend that amount of money, so we decided to delete those searches.

2.5 Dominance of negative data in the dataset

In the full training set, we found that 61405 srch_ID have never booked a hotel on their search, whereas 138390 srch_id have booked a hotel. Nevertheless, taking into account the total number of rows, only 2,8% of the original dataset had information about a hotel being booked. Clearly, the dataset is biased towards the presence of negative data.

2.6 Bias in booking and clicking

We found a strong position bias even for random impressions [4,5 1]. The position of the listed hotels has a great impact on the booking counts, but not so on the clicking events.

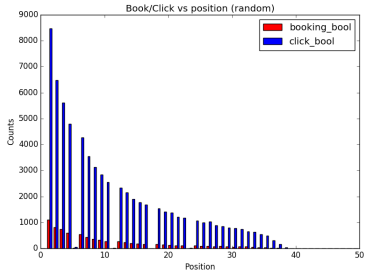

Fig. 4: Position order of the hotels affects the booking number, but not the clicking.

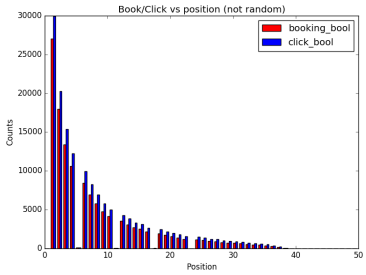

Fig. 5: There is a strong bias in the position and the booking.

Table 1: Bias.

	Random = 1	Random = 0	Total
% book/click	11.4%	85.1%	2.8%
% book/total	0.5%	3.7%	2.8%
% click/total	4.7%	4.4%	4.5%

3 Data Preprocessing

3.1 Missing points

In the course of the data exploration, we realized that there were many missing points. Due to the huge amount of data-points, dropping these rows would have drastically reduced the dataset, leading to a less powerful model. Because of the different nature of the variables, we decided to fill each feature independently [described in 2].

Table 2: Data preprocessing for the missing values, per variable. For reproducibility, the medians and values obtained from the original dataset can be accessed through the google drive directory.

Variable	Approach
gross_bookings_usd	Deleted whole column
visitor_hist_starrating	Substitute by the actual price/rating that the person has spend on the booking, or ...
visitor_hist_adr_usd	... if it has not made any booking, substitute by the median value in the dataset
comp1_inv, comp2_inv, etc	See "Feature engineering"
comp1_rate, comp2_rate, etc	See "Feature engineering"
comp1_rate_percent_diff, etc	See "Feature engineering"
gross_bookings_usd	Deleted
prop_log_historical_price	Substitute by mean value of the corresponding hotel star
srch_query_affinity_score	Substitute by worse possible value in dataset
orig_destination_distance	Substitute by median of the searches
prop_starrating	Substitute by worse possible value
prop_review_score	Substitute by worse possible value
prop_location_score1/2	Substitute by worse possible value

3.2 Feature engineering

Based on the Kaggle competition forum, and the slides [5,6,4] , new features were created using the information of the current variables [3].

Table 3: Feature engineering performed to the dataset.

Variable	Approach
comp_inv	Sum (comp1_inv, comp2_inv,...)
comp_rate_percent_diff	Sum (comp1_rate_percent_diff, comp2_rate_percent_diff,...)
comp_rate	Sum comp1_rate, comp2_rate, etc
season_booked	1 if winter (month=12, 1, 2), 2 if spring (month=3, 4, 5), etc
starrating_diff	$\|Visitor\_hist\_starrating - prop\_starrating\|$
usd_diff	$\|Visitor\_hist\_adr\_usd - price\_usd\|$
prop_starrating_monotonic	$\|prop\_starrating - mean(prop\_starrating[booking\_bool]\|$
Price_norm	$log10(price\_usd)$
month_booked	month + srch_booking_window, fitting into 12 months
scores_merged	$prop\_starrating * prop\_review\_score$
ad_vs_real	$prop\_starrating - prop\_review\_score$
count_hotel	no. of times each prop_id appears
prob_book	Probability of hotel being booked
prob_click	Probability of hotel being clicked
bool_same_country	if visitor_location_country_id = prop_country_id
price_for_one	$price\_usd/[(srch\_adults\_count + srch\_children\_count) * srch\_length\_of\_stay]$
rank_price	Ranked price per query
rank_scores	Ranked prop_starrating per query
roomcount_bookwindow	$F1\_F2 = F1 * max(F2) + F2$
adultcount_childrencount	$F1\_F2 = F1 * max(F2) + F2$

3.3 Test and training set

An independent test set was built from the original dataset by sampling all the rows which represented a search done either in May 2013 or June 2013. This was decided based on the fact that they were the last data-points in time, and we expected that testing our models with a time close to the assignment set would be more accurate. This represented around 1.400.000 rows.

The dataset provided was clearly unbalanced towards searches in which no hotels were clicked (click_bool = 0), and specially not booked (book_bool = 1). We considered that a balanced training data would improve the performance (for instance, decreasing the type-I error) and reduce the time of the process.

In order to prevent an unbalanced training that would bias the results towards non booking the hotels, a down-sampling approach was followed. First, we sampled without replacement from the original dataset all the rows matching the condition booking_bool = 1. Then sampled the same number of rows from the leftover dataset rows (matching booking_bool = 0). As we observed that almost all clicks done in hotels when random_bool=1 didn't end up in a booking, we decided to sample only those rows matching booking_bool=1 and random_bool=0.

4 Experimental Setup

The main objective of the models is to rank hotels so that purchased and clicked hotels are ranked at top among all hotels associated with a search query.

Due to the challenge that a ranking problem presents while building a Machine Learning approach, several ways to handle it were discussed. The evaluation metric used in this competition (normalized DCG; see "Model Validation and Performance") clearly rewarded the correct ranking prediction of hotels being booked towards the hotels being only clicked, and lastly towards the hotels not being booked and clicked, in its relevance score. Thus, we decided to focus the approach as a regression problem.

Independently of the model chosen, a predictor based on a regression approach was built by assigning for every row of the data-frame a target variable. This target variable was named "relevance score" and was assigned a value of 5 if the hotel was booked, 1 if the hotel was clicked but not booked, and 0 if the hotel wasn't clicked.

4.1 Models

Different models were listed to be made.

- Coordinate Ascent (CA) [20] and AdaRank [21], as the most used list-wise models (in [14] it was said that list-wise models have been shown to perform better than the other).
- Gradient Boosted Trees [7], because it was included in a vast majority of the suggested models in the Kaggle discussion forum citeExpediaHttps://www.kaggle.com/c/expedia-personalized-sort.
- LambdaMART (LMART) [13] is a hybrid method that yields better results than others found in the literature, according to.

Furthermore, we tried Random Forest (RF) [12] and Support Vector Machines [15] , as classic machine learning models that usually give a sufficient performance.

The RF and SVM models were implemented on Python 2.3 by using Scikit-Learn library [19] for the machine learning approaches, and Pandas library [17] for the data analysis and preprocessing. The training of the CA and LMART models was performed using RankLib (RankLib-2.1-patched) [9]

Due to time constraints, only four models could be created.

Random Forest Regression Random forests are ensemble models created from several decision trees. In order to achieve their best performance, we performed 10-fold CV for optimum parameter selection. The studied variables were: number of trees ([10,25,50,100,150,300]), maximum depth of the tree ([5,50,150,200]), and the minimum number of samples required to be at a leaf node ([50,100,150]).

A Random Forest Regression was implemented with parameters number of trees: 1000, maximum depth: 40, and minimum number of elements at a leaf: 30. The rest of parameters were set to the default ones from Scikit.

Support Vector Machine Support Vector Machines are algorithms widely used in the field of machine learning. For this project, we used SVM Regression (SVR). One vital step when SVM learning is the tuning of parameters. There are three major parameters to tune: Kernel function [linear, polynomial, RBF and sigmoid], C parameter [0.1, 1, 2, 10] and the γ parameter [1e-1, 1e-2, 1e-5].

The hyper-parameter optimization was performed using the "Scikit-learn" standard function *GridSearchCV* [19] that compares every possible combination of our chosen input values.

Finally, a SVR model was created using the parameters sigmoid kernel function, $C = 2$ and $\gamma = 1e - 5$ (the rest of the parameters were set as default).

LambdaMART LambdaMART is a learning-to-rank method that uses Gradient Boosted Regression Trees in order to solve the ranking task. At first, we used RankLib with parameters ntree set: 500, number of leaves: 10, number of threshold candidates: 256, learning rate: 0.1, stop early: 100, and NDCG@10 as a metric in order to generate this model.

A second tentative was done by using a model that gave the best NCDG overall to predict the ranking in our independent test set.

Coordinate Ascent Coordinate Ascent is an optimization algorithm, alternative to gradient-based. The model was created using RankLib, taking NDCG@10 as a metric, 2 random restarts and 20 iterations (the rest of the parameters were set as default).

4.2 Model Validation and Performance

Cross-Validation A 10-fold cross-validation approach was followed to tune the parameters of the models. The Root Mean Square Error function was used as the one to be minimized during the learning process of the algorithm.

True error estimation. The Normalized Discounted Cumulative Gain (NDCG) score was used for evaluating the quality of the rankings obtained. The function is well explained in the Kaggle web-page [8]. The relevance function was calculated as follows: 5 if the hotel was booked, 1 if the hotel was clicked but not booked, and 0 if the hotel wasn't clicked.

A script was developed to compute the NDCG score as follows:

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{1}$$

$$nDCG_k = \frac{DCG_k}{IDCG_k} \tag{2}$$

, where DCG_k represents the Discounted Cumulative Gain score for all queries $IDCG_k$ is the Ideal DCG, and k stands for the number of hotels recommended.

Benchmarking In order to benchmark our models, a random sorting to the hotels shown was performed, as proposed in [11]. The NCDG from this model was calculated. To do so, an independent test set (see 'Training and Test Set') was used.

4.3 Final submission

The final model was chosen to be Random Forest Regression due to reporting the highest NDCG score. The model was re-trained with the same parameters but using the full dataset downsampled (training set + test set) in order to enable better generalization.

Once estimated the predictions for the 4959183 rows from the delivered test set (129438 different hotels and 199549 different searches), for every search ID the hotels were ranked according to the predicted estimations to be booked (ranging from 0 to 5) and the results were submitted in a comma separated file (EstimationsExpedia_Group18.csv).

5 Results

Various models were built using their "optimum" parameters, chosen by cross-validation. They were sorted using their NDCG score.

In 4 there is a list of the scores of our working models, as well as of the Kaggle competition winners[1] and a random model.

Table 4: Performance of our models, random model and models from Kaggle competition winners[1]

Algorithms	**NDCG Score**
Winner #1 algorithm (Owen) [2]	0.53984
Winner #2 algorithm (Jun Wang) [2]	0.53839
Winner #3 algorithm [2]	0.53366
Random Forest Regression	0.37
Support Vector Machine	0.36
Random method [11]	0.35

Among the models that we tried, Random Forest Regression shows a better performance. Its performance is better than a random model. However, the NDCG score is distant from the best scores.

Neither of the LambdaMART approaches worked. With the first implementation, half of the folds had the same NDCG, and the other half a different one. With the second one, all scores predicted were 0.

[1] Note: the winner's algorithm performance was assessed in a different test set than the one used in this report

Similarly to the LambdaMART training, the Coordinate Ascent model failed to work. In this case it looked as a technical issue, probably due to the program version and the huge size of the data.

6 Discussion

Among the models that we created, the Random Forest Regression is the one that performs better. However, we could not compare many of the other models due to several errors when building the rest.

We think that an extensive exploratory data analysis helped to understand the dataset better, and probably it helped to improved the performance over a random model. Furthermore, the amount of missing points present in the datasets was very large. Thus, many of the data-points were inferred. For a better ranking model, we suggest collecting more information about these variables.

Throughout this project, the lack of time was a key-point in the results of the ranking problem. We severely underestimated the time consumed for the creation of each of the models. This ended in us not being able to analyze the models in depth. In addition, it prevented us from creating other models.

We consider that the implementation of an ensemble method of all the built models could have been implemented in order to increase the performance. Moreover, a more complex algorithm could have been used (such as the ones mentioned in 'Methods') However, the training times of the individual models took us several time (several hours) which mainly limited the possible implementations of it.

In the RankLib models, we were not able to create a working model. Although the NDCG metric score of the program did not have the same relevance function (5 if booked, 1 if clicked, 0 if none), we thought that it could work as fine ranking the hotels. As it did not, we tried retrieving the information about the predicted label in order to calculate manually the NDCG score, but again, we could not. The lack of experience working with such tools have prevent us from obtaining a working model. More learning can be done in the future.

Despite we compare our models with a random method, the statistical significance of the model could have been better assessed e.g. with a permutation test. All in all, this model seems to be better than a random sorting of the results, and would very probably lead to more hotels booked, the final goal of Expedia.

Scripts The scripts created for this project can be accessed through the following link: $https://drive.google.com/drive/folders/0B7x1X9kzuXl6MmZfWG0xcy1JNGc?usp=sharing$

References

1. Expedia 2013 Kaggle competition. https://www.kaggle.com/c/expedia-personalized-sort, `https://www.kaggle.com/c/expedia-personalized-sort`

2. Expedia 2013 Kaggle competition winners slides - https://www.kaggle.com/c/expedia-personalized-sort/discussion/6203, https://www.kaggle.com/c/expedia-personalized-sort/discussion/6203
3. Expedia 2016 Kaggle competition. https://www.kaggle.com/c/expedia-hotel-recommendations, https://www.kaggle.com/c/expedia-hotel-recommendations
4. Expedia Hotel Recommendations — Kaggle, https://www.kaggle.com/c/expedia-hotel-recommendations/discussion/21607
5. Expedia Hotel Recommendations — Kaggle, https://www.kaggle.com/c/expedia-hotel-recommendations/discussion/21646
6. Expedia Hotel Recommendations — Kaggle, https://www.kaggle.com/c/expedia-hotel-recommendations/discussion/21588
7. Introduction to Boosted Trees xgboost 0.6 documentation, http://xgboost.readthedocs.io/en/latest/model.html
8. Normalized Discounted Cumulative Gain — Kaggle, https://www.kaggle.com/wiki/NormalizedDiscountedCumulativeGain
9. The Lemur Project / Wiki / RankLib, https://sourceforge.net/p/lemur/wiki/RankLib/
10. Expedia, Inc. Reports Fourth Quarter and Full Year 2016 Results (2017), http://files.shareholder.com/downloads/EXPE/3880984722x0x927398/D6FADF0D-8265-4833-9C34-8BF23AFA21E6/Q4{_}2016{_}Earnings{_}Release.pdf
11. Agarwal, S., Styles, L., Verma, S.: Learn to Rank ICDM 2013 Challenge Ranking Hotel Search Queries http://www-users.cs.umn.edu/{~}verma/letor.pdf
12. Breiman, L.: Random Forests (2001), https://www.stat.berkeley.edu/{~}breiman/randomforest2001.pdf
13. Burges, C.J.C.: From RankNet to LambdaRank to LambdaMART: An Overview https://pdfs.semanticscholar.org/0df9/c70875783a73ce1e933079f328e8cf5e9ea2.pdf
14. Cao, Z., Qin, T., Liu, T.Y., Tsai, M.F., Li, H.: Learning to Rank: From Pairwise Approach to Listwise Approach (2007), http://www.machinelearning.org/proceedings/icml2007/papers/139.pdf
15. Cristianini, N., Shawe-Taylor, J.: An introduction to support vector machines : and other kernel-based learning methods. Cambridge University Press (2000), https://www.aaai.org/ojs/index.php/aimagazine/article/view/1710
16. Liu, T.Y.: Learning to Rank for Information Retrieval. Foundations and Trends{®} in Information Retrieval 3(3), 225–331 (2007), http://www.nowpublishers.com/article/Details/INR-016
17. McKinney, W., Team, P.D.: Pandas - Powerful Python Data Analysis Toolkit. Pandas - Powerful Python Data Analysis Toolkit (2015)
18. Parachuri, V.: How to get into the top 15 of a Kaggle competition using Python, https://www.dataquest.io/blog/kaggle-tutorial/
19. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, É.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12(Oct), 2825–2830 (2011), http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html
20. Wright, S.J.: Coordinate Descent Algorithms https://arxiv.org/pdf/1502.04759.pdf
21. Xu, J., Li, H.: AdaRank: A Boosting Algorithm for Information Retrieval http://www.bigdatalab.ac.cn/{~}junxu/publications/SIGIR2007{_}AdaRank.pdf