Data Mining Techniques

Nick Boon 2628257[1,2], Ryan van Koll 10348972[1,2], and Ferry Kruidenberg 2626675[1,2]

[1] Universiteit van Amsterdam, Amsterdam 1012 WX, The Netherlands
[2] Vrije Universiteit, Amsterdam 1081 HV, The Netherlands

Keywords: Data Mining · Machine Learning.

1 Introduction

Data mining is the art of creating value in datasets which may be extremely large and/or unstructured and/or incomplete. Many algorithms have been developed that try to make predictions for new datapoints from these datasets, or try to find new insights from the information hidden in the data.

Since the competition for online services like that from Expedia is high, it becomes important to bind customers to your site. One way to do such is to make sure that the user is able to book a suitable hotel quickly without having the need to switch to another provider. For this reason Expedia want to make personalized hotel searches that have the highest chance of being booked matching given the user search details.

In this report we will consider a dataset provided by the Vrije Universiteit, based on Expedia's dataset on Kaggle[3]. The exact difference between the provided dataset and the original from Kaggle is unbeknownst to us.

The objective posed on the Kaggle site by Expedia is to train a model that can predict which hotel will be booked by the user. Furthermore, an ordering of hotels should be created, with the booked hotel at the top, then the hotels that are clicked on, and finally the hotels that are not interacted with. This information may be used by Expedia's search engine to improve its search result by showing the hotel most likely to be booked as first result at the top of the webpage.

1.1 The dataset

All data is contained in two csv-files, a training and a test set. Both contain around 5 million entries, with each containing information about a search result, such as which property is displayed, the number of stars for the property and more. Each row also contains information about the search itself, such as search id, date of search, and search parameters like the arrival date, duration, and

[3] This dataset is freely available, with some constraints in allowed usage, from https://www.kaggle.com/c/expedia-personalized-sort. We used a modified version of the dataset which is not freely available.

number of adults and children. For a more detailed explanation about all columns in the dataset, we refer the reader to the explanation on the Kaggle site at https://www.kaggle.com/c/expedia-personalized-sort/data. Do note that in the test set, some columns are omitted. These include the `booking_bool` and `click_bool` columns, which will be the targets in this dataset.

1.2 Earlier work in this competition

Earlier entrants to the Kaggle competition have tried to find the best way to rank the personalized search results. One group identified 3 features as most important listwise ranking features, namely the `price_usd`, `prop_starrating` and `prop_location_score2` [1].

The competition winners used an ensemble of 26 Gradient Boosting Machines (GBM), whereas the team on second place used LambdaMART based on Multiple Additive Regression Tree (MART). The team that ended third tried many models, but eventually also settled on LambdaMART. Presentations from the three winning teams can be found at https://kaggle2.blob.core.windows.net/competitions/kaggle/3504/media/ICDM2013_Presentations_2013-12-08.zip.

1.3 Exploration

We will only explore the training set, as the test set does not contain the targets `booking_bool` and `click_bool`. We shall therefore assume that the training set is representative for the data contained in the test set.

The training set contains 4,958,347 search results from 199,795 searches. The maximum number of results per search is 38. Of all searches, 138,390 resulted in a booking but all searches led to at least one clicked result. Many null values exist in the dataset, especially in the visitor history columns (`visitor_hist_starrating` and `visitor_hist_adr_usd`) and some competitor columns where in both cases less than 10% entries may have non-null values. In addition, some columns have impossible values, such as `price_usd` for srch_id 78107. All values are more than 1 million USD with a `srch_length_of_stay` of 4 days. We note that all prices have no cents, but this specific search is undoubtedly unreliable for learning.

The dataset is very unbalanced for the target feature `booking_bool`, as shown in figure 1. Of all 4,958,347 search results only 138,390 (2.79%) results are booked. The large difference in number of negative examples, search results that are not booked, compared to positive examples, search results that are booked, will prove challenging for a model to account for.

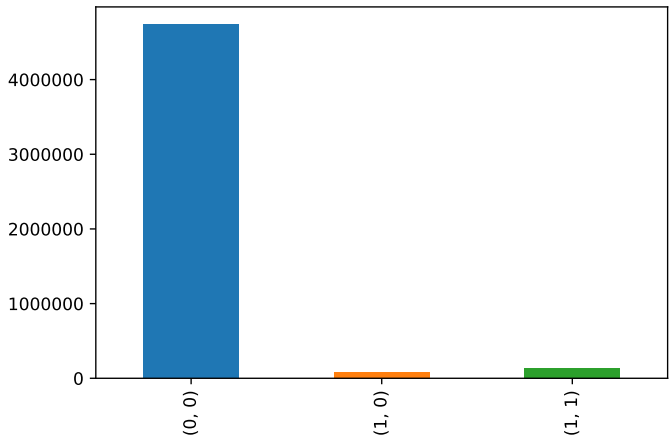

Fig. 1: Count data of occurrences of clicking and booking. (0,0) means not booked and not clicked, (1,0) clicked and not booked and (1,1) clicked and booked.

The linear correlations between all features are shown in figure 2. The largest correlation between two distinct features is 0.782889, found between `click_bool` and `booking_bool`. This is caused by the fact that, before a hotel is booked one must first click on the hotel. All `booking_bool` rows must, and do have, click_bool. This fact allows for only training on `booking_bool`, as the model is then also indirectly trained for `click_bool`.

Another large correlation is found between `position` and `booking_bool` with correlation -0.147918. This is explained by Expedia's search algorithm, which already puts hotels which have higher chance of being booked, at top of the search result page. Do note that Expedia's search algorithm is only used if random_bool equals 0. The correlation increases to -0.175761 if only non-random searches are included. An higher `booking_bool` (true is signified by the value 1) is often found with lower `position`, which explains the negative correlation found. Unfortunately, this column is not available in the test set. As the `random_bool` feature is only useful in combination with the `position` feature, we will not use the `random_bool` feature for training the model.

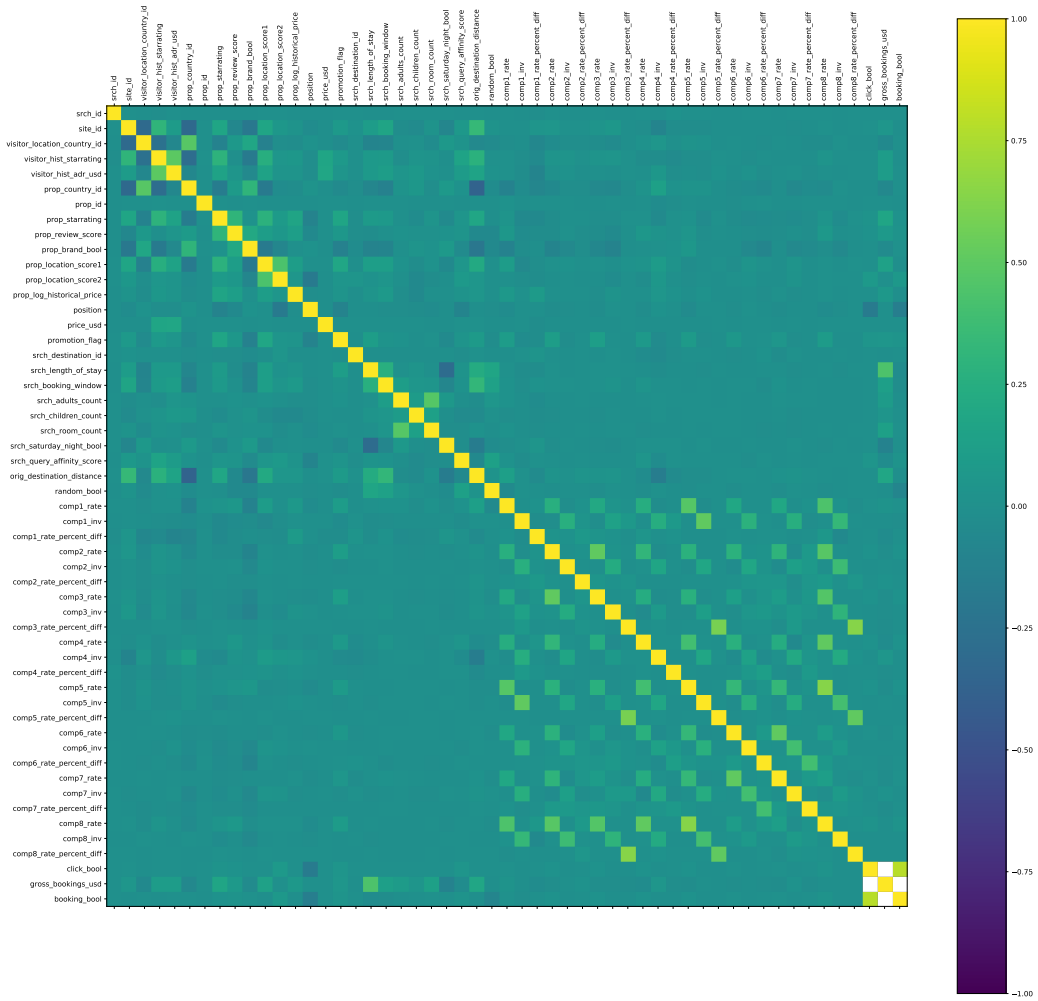

Fig. 2: Correlations between all features initially available in the dataset.

2 Methods

2.1 Pre-processing

First, we load the complete training set from the provided csv-file. Then we convert the object-typed `date_time` column to a date_time-object and add the `srch_booking_window`, the number of days between the search and arrival date at the hotel, to obtain `book_start`. To this, we add `srch_length_of_stay`, the number of days at the hotel, to obtain `book_end`. From these date_times, we extract the weekday, month, quarter and year as new features. If a hotel is only booked visited during the summer, we now have some information on which the model can be trained.

Next, we take the minimum of all `comp#_rate` columns, such that the value will be 1 if Expedia is cheaper than all, 0 if one or more competitors charge the same price, and -1 if one or more competitors are cheaper than Expedia. We assume all competitors are fairly large and known, such that all customers at Expedia will compare prices at these competitors. If one competitor is cheaper, the chance for the hotel to be booked at Expedia will decrease. This `comp` feature has a correlation of 0.018681, which is higher than the separate comp#_rate features.

We also multiply the `comp#_rate` and `comp#_rate_percent_diff` columns, such that we combine information about cheaper / more expensive and difference. For negative values, the competitor's price is -value percent cheaper and for positive values the competitor is +value percent cheaper. This method also partially solves have non-zero `comp#_rate_percent_diff`, but `comp#_rate` being zero. The first column states the price is different by some percentage, while the second states the price is equal. We also drop the `comp#_rate` and `comp#_rate_percent_diff` columns after creating the new columns.

Furthermore, we add averages per `srch_id` for the columns `prop_starrating`, `prop_location_score1`, `prop_location_score2`, and `price_usd`. These features are prefixed with `avg_`. We also calculate a difference between the original columns and the averages, these are prefixed with `avg_` and suffixed with _diff. A hotel which is higher in stars than the average search result may have higher chances of being booked. This information cannot be inferred from a single row, so the addition may be beneficial.

Finally, all null values in the dataset are set to zero.

2.2 Model evaluation

The model will be evaluated using the NDCG@38 method, as prescribed on Kaggle at https://www.kaggle.com/c/expedia-personalized-sort#evaluation. Each row is given a relevance score, which is defined by

$$\text{Relevance score} = \begin{cases} 5 \text{ if } \texttt{booking\_bool} \text{ equals } 1 \\ 1 \text{ if } \texttt{click\_bool} \text{ equals } 1 \\ 0 \text{ otherwise} \end{cases}$$

Table 1: Parameter values for the `RandomForestClassifier` from the `sklearn.ensemble`-package in Python. The values, with exception of the n_jobs parameter are set according to the benchmark code.

Parameter	Value
bootstrap	True
class_weight	None
criterion	'gini'
max_depth	None
max_features	'auto'
max_leaf_nodes	None
min_impuritity_decrease	0.0
min_impuritity_split	None
min_samples_leaf	1
min_samples_split	10
min_weigh_fraction	0.0
n_estimators	50
n_jobs	4
min_samples_split	10
oob_score	False
random_state	1
verbose	2
warm_start	False

The NDCG@k-score is then defined by calculating

$$\mathrm{NDCG_k} = \frac{\mathrm{DCG_k}}{\mathrm{DCG_{k,optimal}}} = \frac{\sum_{i=0}^{k} \frac{2_{\mathrm{k}}^{\text{Relevance score}} - 1}{\log_2(i+1)}}{\mathrm{DCG_{k,optimal}}}$$

where $\mathrm{DCG_{k,optimal}}$ is the discounted cumulative gain using the optimal ordering, such that the best possible NDCG-score is 1. Note that k limits the number of results taken into account. For the current dataset, the maximum number of results is 38, which explains Kaggle's decision to use NDCG@38. For each unique `srch_id`, the NDCG@38-score is determined, after which the score is averaged over all `srch_id`. This results in the final score for the model.

2.3 Model definition

We use the random forest classifier from the `sklearn.ensemble`-package using Python. This is the same model as used in the benchmark code which is freely available on https://github.com/benhamner/ExpediaPersonalizedSortCompetition. The `sklean`-package is a widely-used package for machine learning and therefore well documented. As we do not have much experience using machine learning, using this package was a safe choice.

The model parameters are shown in table 1. These parameters are chosen in accordance to the benchmark code. We only changed the `n_jobs`-parameter to

the number of physical CPU cores available, to allow for more efficient use of the available computation power and therefore faster training of the model.

In this competition, we need to order search results based on `booking_bool` and `click_bool`. Due to the large correlation between these variables, as shown in section 1.3, and the higher relevance score for correctly sorting search results with `booking_bool` we choose to only train a model for `booking_bool`.

The model can use all information available in the training set, except features not available in the test set and, as discussed in section 1.3, the `random_bool` feature. Furthermore, the date_time features cannot be used by the model and are therefore also not available. All other features, including calculated features, are available to the model, as the same calculations can also be performed on the test set. The total number of features used to train the model is 61.

Random decision forests Random decision forests is an ensemble learning method. An ensemble uses multiple learning algorithms and then combining those to get a better result. In the case of a random decision forests the underlying algorithms are decision trees. If one would use a decision tree on its own it has a high probability of over-fitting and this reduces the generalizability of the model. The random decision forests tries reduce the over-fitting by training multiple trees on various subset of the data and average them. This not only reduces the over-fitting but also improve the predictive accuracy of the model. When training a deep decision tree the model tends to fit highly irregularly patterns. This fitting behaviour leads to a high variance. So when using multiple of these model and averaging them tries to reduce the variance.

How are decision trees made? The nodes are made based on which feature has the highest amount of information gain. The information gain = entropy parent - weighted sum entropy of the children. The children are the split in data that is made by choosing a feature. Splits are made until the information gain is zero.

3 Results

10-fold cross-validation In a 10-fold cross-validation on the training set, we obtained a final NDCG@38-score of 0.477 ± 0.002 with 95% confidence intervals. As we do not have the target values for the test set, we unfortunately cannot test our model on the test set.

Performance on Kaggle dataset Using the original Kaggle dataset,[4] we can upload our prediction file to Kaggle for evaluation. We then obtain a private score of 0.48391 and a public score of 0.48574. This would put us at position 102 on the public leaderboard and position 101 on the private leaderboard.

[4] This dataset is freely available after registration at https://www.kaggle.com/c/expedia-personalized-sort/data

4 Discussion

During the exploration of the huge dataset provided, we learned a lot about using Python for these purposes. Especially the Pandas-package was really convenient, as it still worked really fast. Furthermore, we got some practical experience in analyzing and modifying (adding features) these datasets which is important in this field. This all resulted in a ranking just outside the top 100 on the Kaggle leaderboards, which is not too bad for a first attempt.

We do note that the Kaggle-score is far beyond our confidence interval of our cross-validation on the training set. This is likely caused by differences in datasets, but also due to having almost double training examples in Kaggle's training set. The size of the Kaggle version was 2.19GB compared to our version's 1.18GB of which we used 90% in the 10-fold cross-validation to train on. The other 10% was used to test the trained model on.

As we have not implemented any form of outlier detection, our data preparation is not perfect. For improving on our result, we must consider outlier detection to only train the model on correct data. Furthermore, we should better account for missing data. In our current data preparation procedure, we just set all null values to zero, but there may be information hidden in the fact that data is missing. For example, a competitor may not be available

References

1. Liu, X., Xu, B., Zhang, Y., Yan, Q., Pang, L., Li, Q., Sun, H., Wang, B.: Combination of diverse ranking models for personalized expedia hotel searches. CoRR **abs/1311.7679** (2013), http://arxiv.org/abs/1311.7679

5 Appendix

5.1 Process report

Schedule Our project schedule for each week is shown in table 2.

Table 2: Project schedule

When	What
First week (April 23 - April 29)	Loading dataset / make histograms / research earlier work
Second week (April 30 - May 6)	Feature engineering / implement k-fold cross-validation
Third week (May 7 - May 13)	More feature engineering / cross-validation of model on training set
Fourth week (May 14 - May 20)	More feature engineering and testing / started work on report
Last week (May 21 - May 27)	Finalize code and report / implement creating a prediction file on test set

Who did what For the most part every team member contributed to every stage of the project. In the early stages we did most of the tasks together like loading the dataset and make some exploratory histograms. As for the research into earlier work we all looked into the algorithms, methods and feature engineering that was done in the past. Ferry looked into the LambdaMART method that was used by the top 3 contestants in the kaggle competition. In the end we didn't use this method because there were no good implementations in python and this method is much slower to train than our current method. Nick looked into the benchmark code which was provided in the Kaggle competition. As this code was based on the well-supported `sklearn`-package in Python, it eventually formed the base of our final code. Ryan studied what the 3rd place of the competition did and how it could have improved our own search for the best prediction.

Nick also worked on the k-fold cross-validation implementation in python to test our models and get a indication how well our models would preform on the actual test-set. Feature engineering was done by the whole group. Each of the members added some features that made it into the final model. The same team effort was made to the report where anyone did its part.

Reflection Working on a longer-term project is always more difficult in a team; there are many smaller tasks which have to be divided between team members. For some tasks, it is beneficial to meet in person to discuss major choices in the project, for example the model to use. This was also the main difficulty within our group: not all the members follow the same courses. Therefore the only time

we spend together in class was during this course. Furthermore we lived pretty far apart from each other. For these reasons meeting in person was often not feasible. To overcome this hurdle we made a Whatsapp group and a github to communicate our progress.

Many tools do however exist to reduce the problem of not being able to meet in person. GitHub is a wonderful tool to merge and synchronize code. This allowed us to each find new features on our own, so we could later merge the best features into our final code. Furthermore, we can easily work on a single report using ShareLatex.

Still, all these tools do not remove the necessity of working together on this project in person. In hindsight, we may not have worked together often enough to really make the most of this project.