Data Mining Assignment 2
Group 65

Agoston Szabo (2738183), Oleg Litvinov (2719624), and Nedim Azar (2728313)

Vrije Universiteit, Amsterdam, Netherlands

Abstract. Recommender systems are used to display search results sorted by their relevance to the user at hand. In this study we used various algorithms to solve the learning to rank problem on the Personalize Expedia Hotel Searches dataset from 2013. After rigorous feature engineering and data preprocessing, we were able to achieve our best results with CatBoostRanker which gave us a NDCG@5 score of 0.41120 on the publicly available competition test data.

Keywords: Data Mining · OTA · Expedia · Machine Learning · Recommender Systems · Learning to Rank · LightGBM · SVD · CatBoost

1 Introduction

Expedia is a hotel booking service, an online travel agency (OTA) where users can input a search query (containing information like desired booking dates, number of people, destination city, etc.) and are presented with a sequence of property recommendations. The users are usually redirected to Expedia from a search engine such as Google or DuckDuckGo, and their activity on the website is saved for later Data Mining purposes. In this report, we explore, engineer, and model the Expedia dataset comprised of 53 columns. Our ultimate goal is to design a recommender system that orders search results by their likelihood of being booked (best case scenario), or being clicked (average case). Various modeling techniques were applied, and the data was transformed to increase the accuracy of the recommendation ordering.

2 Business Understanding

Ranking problem is widely used in industry. Its main applications are search engines and recommender systems. The latter is used in retail (e-shops, merchandise) and media (entertaining content recommendation, advertising). The market for such kind of systems is enormous. Even a slight improvement of a recommender system affects thousands of purchasers and, therefore, company's profits. Companies like Netflix [3] spend millions of dollars for research and development to recommend more suitable content for their users to increase revenue. The goal for conducting a competition was to analyze and improve the customer

experience via improving the hotel sort order. More relevant recommendations increase the likelihood that the user will make a purchase on one of Expedia's website and go to the hotel that was offered to him/her (Expedia also makes a profit by displaying promoted hotels).

From the technical perspective this may be solved via a ranking algorithm optimising (not always directly) some of the ranking metrics like NDCG or MAP. During the original [5] competition the best results were obtained using gradient boosting algorithms for ranking. Namely, [8] used an ensemble of 26 gradient boosted trees with NDCG loss function, [11] used LambdaMART (which is a specific instance of gradient boosted regression trees). Having that and the fact that gradient boosting is the most popular (a very frequently the most powerful) solution for tabular data on Kaggle, this type of algorithms was selected for this study.

3 Data Understanding

When looking at the columns and their descriptions in the task information, we are able to discern 5 different types of variables:

1. **Search Criteria:** These are the values that are inputted by the user into Expedia's GUI. (Eg. date and time of search, destination ID, length of stay, booking window, number of adults, number of children, number of rooms etc.)
2. **Static Property Characteristics:** These variables give us immutable information about a property listed on Expedia. (Eg. property ID, property country etc.)
3. **Dynamic Property Characteristics:** These variables contain information about the properties that is likely to change or fluctuate over time. (Eg. ranking position, price, promotions etc.)
4. **Visitor Information:** These variables describe various user profiles. (Eg.visitor country, distance between visitor and property, mean star rating given by user etc.)
5. **Competitive OTA information:** These are variables that compare facts about properties on Expedia to facts about properties on other OTAs. (Eg. whether Expedia has a lower/same/higher price compared to a competitor, if a competitor has availability, price differences etc.)

Missing Values First we decided to observe the percentage of missing values in each column of the dataset. Furthermore, we wanted to compare these percentages across train and test sets. In **Fig. 1**, we plotted the ratio of missing values for columns where said ratio is not insignificant in the training set ($> 1\%$ of values missing). We can observe that there are no imbalances in missing value ratios in between train and test sets for any of the variables. Furthermore, some columns such as $comp1\_rate$ and $comp6\_rate$ have more than 95% of values missing (rendering them useless), while other columns such as $comp5\_inv$ and

*comp*2_*inv* are missing around 57% of their values, potentially increasing their predictive power compared to the first set.

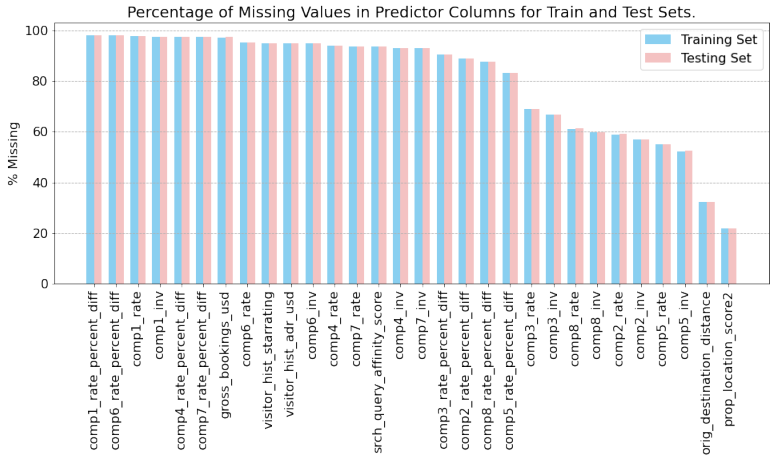

Fig. 1. Missing Value Ratios

Correlations Since we have a considerable amount of predictors ($n = 53$) even before any feature engineering, we decided to visualize correlations and figure out which pairs of variables could be considered to be dropped or merged in order to make training more efficient, and the data less noisy. In **Fig. 2**, we created a heatmap of the 20 most correlated predictors [12]. We define the degree of correlation of a variable with respect to the remaining variables as the sum of absolute pairwise Pearson correlations with every other variable. So the degree of correlation of an arbitrary variable $x \in X$ is $DOC(x) = \sum_{y \in X} |Pearson(x, y)|$. We computed this metric for every variable and observed the most significant 20. Multiple variables that we did not use for modelling show up as the most correlated in this heatmap, examples or this are *comp*1_*rate*, *comp*5_*rate*, and *comp*8_*rate*. The correlation information paired with high missing value ratios, some variables became better candidates for being dropped.

Imbalances Next, we wanted to understand if there are any class imbalances in the data (classes being 0, 1, and 5, respectively "property listing ignored", "property listing clicked", and "property booked"). A simple Pandas query showed us that an overwhelming 95.53% of search results were ignored, while a mere 1.69% were clicked, and a measly 2.79% were booked.

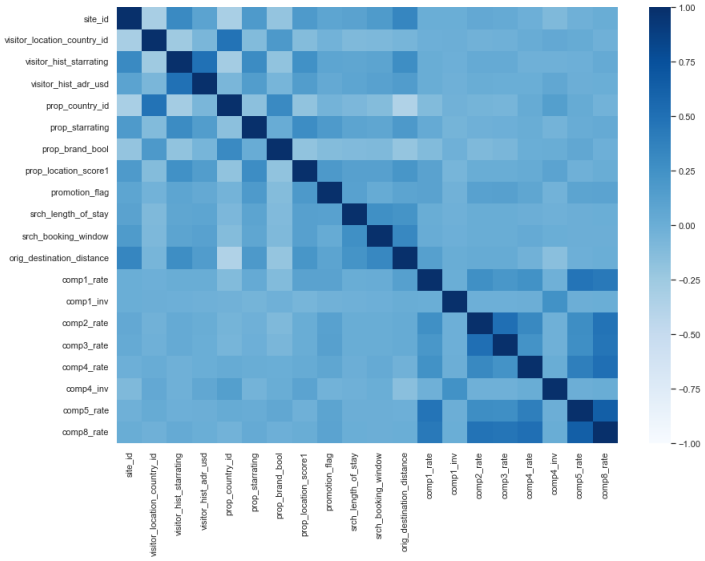

Fig. 2. 20 Most Correlated Predictors

Data Frequency Our goal here was to gain insight about the frequency of the data we are working with. It is important to see if there are any cycles or missing chunks of data, since this leads to different decisions while splitting into train, test, and validation sets.

In **Fig. 3** we plotted the frequency of data instances over the entire dataset. Upon visual inspection, we can see that the average frequency of the data is quite constant over time, but there is a slight positive trend. It is important to note that we do not have any observations from the months of July to October, so this should be taken into account in case we desire to predict rankings for data containing those months.

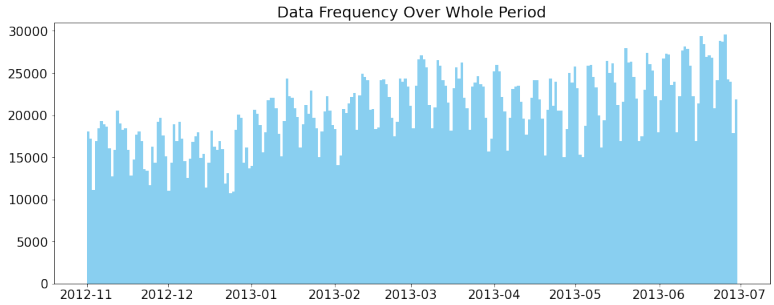

Fig. 3. Data Frequency

4 Data Preparation

4.1 Feature Engineering

Feature engineering may be split into three parts: common sense, orders, and aggregations.

Common sense Having some common understanding of what matters when you search for a hotel, we defined several combinations using the default features.

- Different mathematical combinations of *prop_location_score*1 and *prop_location_score*2
- Number of competitor with prices higher or lower
- Differences between all sorts of prices and their logarithms, powers, and some other mathematical operations
- Min, max, sum, mean, ratio, and difference between review scores and star
- Price per one star and one point of review score
- Time features like day, month, season, etc for the day of booking, trip start and end dates
- Holidays during trip
- The number of people per room, the number of children per adult, total number of people. Boolean feature if there is a separate room for children

Orders It is very important from the user's perspective what is the cheapest hotel from the group of suggested or for the country of search. Moreover, for the same hotel user will be more interested if the price for the hotel is on historical minimums.

- price within the group
- price within the *srch_destination_id*
- price for the *prop_id*

Aggregations We calculated min, max, median, and mean of *prop_starrating*, *prop_location_score*2, *prop_location_score*1, *price_usd*, *promotion_flag* per different columns (and sometimes vice versa):

- *srch_id*
- *prop_country_id*
- *srch_destination_id*
- *prop_id*
- *prop_review_score*
- *prop_location_score*2, where *prop_location_score*2 is rounded to one decimal point
- *trip_start_date_month*
- *trip_end_date_month*
- *trip_start_date_quarter*
- *trip_end_date_quarter*
- different combinations up to 4 grouping columns from the ones presented above

5 Modeling

5.1 Baseline Model

We created a simple baseline model in order to have a basis for comparison. The random baseline simply groups the properties that need to be ranked by their search ID, and shuffles the properties in order to create a random ranking instance. The baseline NDCG@5 scores for train

5.2 LightGBM

Light Gradient Boosting Machine (LGBM) is a machine learning algorithm by Microsoft. It is an implementation of a Gradient Boosting Decision Tree (GBDT) enhanced with Gradient-based One-Side Sampling (GOSS), and Exclusive Feature Bundling (EFB). These additions to the vanilla implementation of GBDT make LGBM significantly more efficient and scaleable [6].

GOSS With this technique, a significant proportion of data instances with small gradients is excluded. This is achievable due to the fact that data instances with larger gradients play a more important role in the computation of information gain [6]. This proves to be extremely useful in our case since we are working with a lot of instances across a large number of features, with relatively low computing power and time.

EFB Exclusive Feature Bundling aims to reduce the number of features. The idea is that high-dimensional data is usually very sparse, and in turn, sparse feature spaces contain many mutually exclusive features, i.e., they ever take non-zero values simultaneously. EFB bundles many exclusive features into much fewer, dense features, which can effectively avoid unnecessary computations [6].

LambdaRank We used the LightGBMRanker algorithm from Microsoft's LGBM framework since it is suited for our task, and it uses NDCG as a built in scoring method. LightGBMRanker makes use of LambdaRank, which turns it into a learning to rank method. LambdaRank uses the gradients of the costs with respect to the model scores (denoted λ's) to adjust the ranks of pairs of items. These gradients are used to apply equal and opposite "forces" to the ranks of each item, one item is pushed up by $|\lambda|$, while an equal and opposite force is applied to the other item.

Hyperparameters We were not able to tune the hyperparameters with GridSearch because of time constraints, however we evaluated multiple models and manually varied the number of estimators and learning rate by intuition to settle on our final parameters: **{n_estimators=5000, learning_rate=0.025}**

Results Paired with our feature engineering, and incremental adjustments to the learning rate and number of estimators, we were able to achieve $NDCG@5 = 0.41120$ on the public task data and $NDCG@5 = 0.40021$ on our local test set with CatBoostRanker.

5.3 CatBoost

CatBoost is an open-source software library developed by Russian company Yandex. It provides a gradient boosting framework with built-in categorical feature encoding algorithm and great visualisations. [2].

Hyperparameter Tuning Hyperparameters were selected via scikit-optimize library. Nothing better than the default set was found.

Results Paired with our feature engineering, and incremental adjustments to the learning rate and number of estimators, we were able to achieve $NDCG@5 = 0.41120$ on the public task data and $NDCG@5 = 0.40021$ on our local test set with CatBoostRanker.

5.4 Singular Value Decomposition

Singular Value Decomposition (SVD), is the factorization of a matrix $\mathbf{A}$ into the product $\mathbf{U\Sigma V^H}$. Where $\mathbf{U}$ is a unitary matrix (which represents searches in our case), $\mathbf{\Sigma}$ is a diagonal matrix (which encodes the "importance" of a property in our case), and $\mathbf{V^H}$ is another unitary matrix (which represents properties in our case) [10]. There exist different algorithms to compute Singular Value Decomposition.

We opted to use the SVD implementation from Surprise's [4] **matrix_factorization** module. The implementation is based on Simon Funk's technique for the 2006 Netflix Prize competition [3]. Matrix factorization methods prove to be superior to traditional nearest neighbour methods for ranking problems, and allow the incorporation of additional information such as implicit feedback, temporal effects, and confidence levels [7, 1].

Hyperparameter Tuning We used **GridSearchCV** from the Scikit-Learn [9] library to search the hyperparameter space. The following parameter grid was used:

n_epochs	5, 10, 20
n_factors	50, 100, 150
lr_all	0.005, 0.01
reg_all	0.4, 0.6

The best hyperparameter instances for our purposes, and therefore the ones we used for the Kaggle competition were: **{n_epochs=5, n_factors=50, lr_all=0.005, reg_all=0.6}**

Results Paired with our feature engineering, and hyperparameter tuning through GridSearchCV, we were able to achieve $NDCG@5 = 0.27837$ on the public task data and $NDCG@5 = 0.28473$ on our validation set with SVD.

5.5 Overall Results (NDCG@5)

	Training	**Testing**	**Validation**	**Public**
Random Baseline	0.15920	0.15297	0.15364	N/A
LGBMRanker	0.39448	0.39731	0.39082	0.39566
SVD	0.29143	0.27376	0.28473	0.27837
CatBoostRanker	0.49272	0.40021	0.40407	0.41120

Takeaways From Project

Agoston This project taught me how to look at a seemingly large and complex dataset and study it until it becomes familiar. Feature engineering was a major part of the challenge after we found an algorithm that seemed to do a good job, which was a first experience for me. The importance of clean code and documentation within the group became more visible as the project progressed and this experience definitely changed the way I will work on my next project.

Ned Working with such a large dataset taught me to really think about the efficiency of the code I write. Processing five million rows can take a while, or even overflow your machine's memory if not done with thought. This was also my first Machine Learning public task, and the competitive nature of the process showed me how you really get diminishing returns when you push yourself towards the end, but nevertheless there is always more to gain. And finally, things take longer than they seem at first.

Oleg I learned how to validate properly. Found interesting techniques for feature selection. Found a lot of ways to leak.

References

[1] Linas Baltrunas, Tadas Makcinskas, and Francesco Ricci. "Group recommendations with rank aggregation and collaborative filtering". In: *Proceedings of the fourth ACM conference on Recommender systems*. 2010, pp. 119–126.

[2] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. "CatBoost: gradient boosting with categorical features support". In: *arXiv preprint arXiv:1810.11363* (2018).

[3] Simon Funk. *Netflix Update: Try This at Home*. Dec. 2006. URL: http://sifter.org/~simon/journal/20061211.html.

[4] Nicolas Hug. "Surprise: A Python library for recommender systems". In: *Journal of Open Source Software* 5.52 (2020), p. 2174. DOI: 10.21105/joss.02174. URL: https://doi.org/10.21105/joss.02174.

[5] Expedia Inc. *Personalize Expedia Hotel Searches - ICDM 2013*. 2013. URL: https://www.kaggle.com/c/expedia-personalized-sort.

[6] Guolin Ke et al. "Lightgbm: A highly efficient gradient boosting decision tree". In: *Advances in neural information processing systems* 30 (2017).

[7] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems". In: *Computer* 42.8 (2009), pp. 30–37.

[8] Zhang Owen. *Personalized Expedia Hotel Searches – 1^{st} place*. 2013. URL: https://www.dropbox.com/sh/5kedakjizgrog0y/_LE_DFCA7J/ICDM_2013?preview=3_owen.pdf.

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

[10] Gilbert W Stewart. "On the early history of the singular value decomposition". In: *SIAM review* 35.4 (1993), pp. 551–566.

[11] Jun Wang and Alexandros Kalousis. *Personalized Expedia Hotel Searches – 2^{nd} place*. 2013. URL: https://www.dropbox.com/sh/5kedakjizgrog0y/_LE_DFCA7J/ICDM_2013?preview=4_jun_wang.pdf.

[12] Michael L. Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.