Boosting algorithms for session-based, context-aware recommender system in online travel domain

Paweł Jankiewicz
LogicAI
pawel@logicai.io

Liudmyla Kyrashchuk
LogicAI
mila@logicai.io

Paweł Sienkowski
LogicAI
pawel.sienkowski@logicai.io

Magdalena Wójcik
LogicAI
magda@logicai.io

ABSTRACT

With the high competitiveness of the online hotel booking sector, the development of timely and robust recommender systems is necessary. The 2019 RecSys Challenge is focused on the use of session-based and context-aware signals from users to improve the quality of hotel booking recommendations. In this paper, we present our approach to the challenge. We focus extensively on proper problem representation, feature extraction, and model blending. With the final MRR score of 0.685711, our team achieved the 1'st place in challenge out of over than 500 teams.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

recsys challenge, boosting, feature engineering, mean reciprocal rank

ACM Reference Format:
Paweł Jankiewicz, Liudmyla Kyrashchuk, Paweł Sienkowski, and Magdalena Wójcik. 2019. Boosting algorithms for session-based, context-aware recommender system in online travel domain. In *Proceedings of the ACM Recommender Systems Challenge 2019 (RecSys Challenge '19).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As online hotel booking sector becomes more competitive, the need for quick, efficient and robust recommender systems increases. Particularly, previously unused data like click streams, search refinement and filter usage might significantly increase recommendation quality[2]. The 2019 ACM RecSys Challenge[7] requires the usage of explicit and implicit user signals to rank website hotel recommendations from the most desirable to the least. Our approach consists of careful validation, extensive feature engineering and extraction and use of model blending for best results. We found that in this challenge overtraining of the model was difficult, hence we have focused our attention on signal extraction from the data.

RecSys Challenge '19, 2019, Copenhagen, Denmark
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 PROBLEM FORMULATION

2.1 Dataset description

The data available for the challenge was provided by trivago, an online booking company. The dataset consists of user click streams with indicated click-out events, defined as user's transition to a specific hotel detail page. The training set consisted of almost 16MM of click events, that were divided into user sessions. Each event included information about user, session, timestamp, type of action, and details about users location, device and filter preferences. With each click-out item, a list of up to 25 hotels was provided. The challenge was to predict, for a given user, session and timestamp, which presented hotel was clicked by the user.

2.2 Evaluation and optimization metrics

The metric used in the challenge was MRR (Mean Reciprocal Rank):

$$MRR = \sum_{p=1}^{p} \frac{1}{p} \tag{1}$$

where p is the position rank.

MRR is used for optimization tasks where only one item is relevant and we want it to appear as early in the search as possible. However, in LightGBM, our algorithm of choice, there was no MRR implementation, that is why we selected NDCG optimization as our objective function. Here we would like to present that in the case of only one appropriate item it is very similar to MRR objective.

In NDCG it is assumed that there can be few relevant items, and earlier they appear in the search the better. NDCG metric has two conceptual parts: DCG (Discounted Cumulative Gain) and IDCG (Ideal Discounted Cumulative Gain), as can be seen from Equation 2:

$$NDCG = \frac{DCG_{\mathrm{p}}}{IDCG_{\mathrm{p}}} \tag{2}$$

DCG part cares about the relevance of the items and their appearance on the top positions of the search, penalizing when the relevance is reduced logarithmically proportional to the position.

$$DCG = \sum_{p=1}^{p} \frac{1}{log_2(p+1)} \tag{3}$$

The IDCG represents the ideal relevance which serves as a normalization factor across all positions [4]. In Equation 4 $|REL\text{p}|$ represents the list of relevant items up to position p.

$$IDCG = \sum_{p=1}^{|REL_{\text{p}}|} \frac{1}{log_2(p+1)} \tag{4}$$

In the case of only one appropriate recommendation per search, IDCG cancels out as it simplifies to one. The only difference is that NDCG transforms position logarithmically. This simplification in the challenge allowed us to use NDCG as the objective function for our models.

3 OUR APPROACH

3.1 Data Transposition

For each query with a click-out event, there were up to 25 hotels which were shown to the user. We exploded the impressions so that each hotel was a separate observation. Altogether this created 42.7MM observations for the training dataset and 5.7MM for the test dataset.

With this approach, it was possible to treat the problem as a binary classification. We did try to use binary classifiers which were beaten however by ranking models which could leverage list-wise ranking methods.

3.2 Validation process

The data in the competition was composed of a whole week of user interactions. The train data covered Monday till Friday and testing data was Saturday and Sunday. Because of how the problem was defined we tried to mimic the test distribution of events in our validation subset. This means we also selected the last click-out per session for validation purposes.

For each model, we created 2 versions of it. One that was trained on training dataset without the validation dataset and the second with the validation data included. The first model was used to validate the predictions and to create an ensemble of the predictions, while the predictions generated by the second one were used to submit to the competition leaderboard.

3.3 Iterative process

Because the training dataset was quite large in the first phase of the competition we developed our solution using a much smaller validation dataset. First, we used 1 million observations for training and validation (split by time). This proved to be a sufficient method to validate the models for the first half of the competition ie. the validation was correlated with the leaderboard. For the second half of the competition, we had to prepare a validation strategy that was more similar to the size and characteristic of the test set (described in more details in 3.2).

4 FEATURE ENGINEERING

4.1 Numerical features

During the competition, we created 220 numerical features which mainly encoded the context of the query and 30 categorical features which were one-hot-encoded. The most important features were based on the frequency and the recency of interactions with items.

Most of the features were centered around **item** and **user-item** interactions. Limited informations about the **user** were not very helpful. Similarly to [8] we calculated various **item-to-item** similarity measures.

4.1.1 Item-item similarity. One of the easiest and surprisingly good were features that compared the item from query to items that users interacted with.

Let $U = \{u_1, u_2, ..., u_n\}$ be a finite sequence of users of size N and $H = \{h_1, h_2, ..., h_m\}$ be a finite sequence of hotels of size M. Each hotel had a list of properties $R_h = \{r_1, r_2, ..., r_v\}$ of size V. For each hotel h in the impression we can calculate the Jaccard distance to the list of selected hotels before. Let $T = \{h_{c1}, h_{c2}, ..., h_{ct}\}$ be a set containing a finite sequence of length t of hotels h_c user clicked-out within a time window. The Jaccard distance was calculated from Equation 5.

$$d_{uh}^{\text{JAC}} = \frac{1}{t} \sum_{h_c=1}^{t} \frac{|R_h \cap R_{\text{h}_\text{c}}|}{|R_h \cup R_{\text{h}_\text{c}}|} \tag{5}$$

Using this formula, we also created Jaccard distances for hotels based on their prices and the number of points of interest.

4.1.2 Diversity in user-item interactions. To understand how users interact with the new hotels, we calculated the probability of the click-out of the hotel based on the user's interactions with new hotels before.

If $T = \{h_{c1}, h_{c2}, ..., h_{ct}\}$ is a set of length t containing a finite sequence of hotels h_c user clicked-out on within a time window, then number of distinct hotels is $|H|$, where H is the set of $h_c \in T$.

For each user u, we calculated the probability of the click-out $P_u(y)$, where y is the click-out event for the hotel (Equation 6):

$$P_u(y) = \begin{cases} 1 - \frac{|H|+1}{|T|+2}, & \text{if user interacted with hotel before} \\ \frac{|H|+1}{|T|+2}, & \text{otherwise} \end{cases} \tag{6}$$

This feature was one of the Top 20 most important and helped us differentiate users with a tendency to click-out on the new hotels from users with stable preferences.

4.2 Ranking transformations within the click-out group

Apart from the raw numerical features, we have applied some contextual transformations within the query. The transformations included:

(1) Ranking of the feature within the query i.e. is the price of the hotel the maximum on the impressions list.
(2) Value of the feature above and below the impression i.e. what is the price of the hotel that is above the hotel.
(3) Difference of the same feature with the features above and below i.e. what is the difference of the price vs the hotel above and below.

The example of such transformations can be found in Table 1.

Table 1: Numerical feature augmentation

Price	Rank	p1_diff	m1_diff	p1	m1
100	2	NULL	20	NULL	80
80	1	-20	-120	100	200
200	4	120	50	80	150
150	3	-50	100	200	50
50	0	-100	NULL	150	NULL

p1 - the item below, m1 - the item above, _diff is a difference between the item above/below with the current item

4.3 Accumulators

Most of the features were created incrementally so that the metrics could only use the information before the query.

```
initialize accumulator;
while not last event do
    read event;
    if event.actiontype == "click-out item" then
        feature = accumulator.getStats(event);
    end
    accumulator.update(event);
end
```

Algorithm 1: Accumulator algorithm

Accumulators following the same methodology were created to track all the events that happened within the session or the user context.

The drawback of such an approach meant that at the beginning of the process the values of the features need some time to warm up. For example, when we calculated CTR (Click Through Ratio) for the hotel the first observations were using a very small subset of the data.

Given more time we would like to investigate if removing early observations could improve the performance of the model.

4.4 Additional features

4.4.1 Hotel's properties. Firstly and foremostly, we used the hotel's properties as a L matrix, where l_{hp} represented if the hotel had a specific property. Additionally, we created standalone features for each h such as: rating, number of stars, type of the hotel (hotel, resort, etc.), number of properties for each hotel and binary representation of the few more relevant features when it comes to choosing the hotel: free Wi-Fi, car park, all-inclusive, etc.

4.4.2 Hotel's prices. We also created features of the hotels concerning the price. The most standard ones were the average, minimum and maximum price of the hotel gathered from the price impressions from the main dataset. Additionally, we used the number of different prices of the hotel, price percentiles grouped by the city and the platform, ranking for each hotel based on the prices sorted ascending and descending as values and in percentiles, relation of each price to the minimum price within the impression, and the price range for the hotel.

Table 2: Performance change with dart

Boosting type	Learning rate	Number of trees	Max leaves	Min data in leaf	MRR
gbdt	0.2	1600	128	5	68.724
dart	0.2	1600	128	5	**68.825**

4.4.3 Rating features. We created features that estimated rating, distance and popularity of the hotel with respect to other hotels, e.g. how good is the hotel within the impression list.

4.4.4 User's previous interactions. We also calculated statistics on how many users interacted with the hotel and how did they interacted with it: viewed images, info, deals, and ratings.

4.4.5 Time difference. One of the significant features was the time difference between the click-outs. When the difference was small, it usually meant that the next click-out item was close to the previous one. When creating time difference features we also used the position of the item and platform. Although the importance was huge for small time differences, as the time difference increased it was harder to derive the information from the feature.

4.5 Data leakage

We noticed that there exists a relation between the position of the last hotel that the user interacted with, the time between the interaction and the click-out and the final ranking. Based on only 3 features it was possible to create a model with 63% MRR. We think that for future research the time of the final click-out event should be removed for validation purposes.

4.6 Feature importance

The table 3 presents the most important numerical features according to LightGBM feature importance measure (number of usages of the feature in tree splitting).

5 MODELS

For the challenge we used LGBMRanker which is an algorithm for ranking problems within the LightGBM framework. LightGBM is a gradient boosting framework that uses a tree-based learning algorithm. The choice of LightGBM was due to the fact, that this framework builds the trees leaf-wise instead of the depth-wise. It adds performance boost but needs special treatment to prevent overfitting [3].

5.1 Parameter tuning

5.1.1 Early stages. In the beginning, we were building classification models tuning mainly the number of estimators and learning rate. We noticed, that it was relatively hard to overfit to the training dataset, as can be seen in Figure 1. Due to that fact, increasing the number of estimators was our prime strategy for some time.

5.1.2 Boosting type. We started from the 1600 estimators and *gbdt* boosting type, but the main increase from the hyper parameter optimization was due to the switch to the *dart* boosting type, as can be seen in the Table 2.

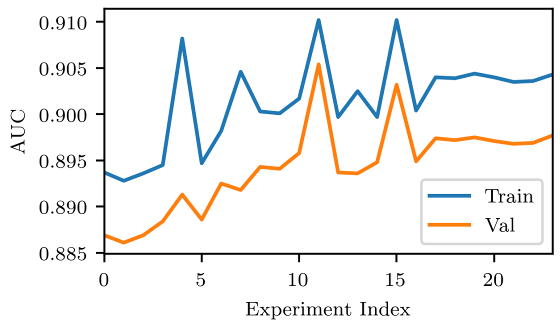

Figure 1: AUC score for train and validation datasets

Dart (Multiple Additive Regression Trees) is an ensemble model that uses boosted regression trees. It successfully handles the over-specialization: the later trees less affect the predictions. Dart implements the idea of *dropout*, extensively used in deep neural networks. Dart algorithm drops some amount of existing trees from previous interactions (a hyperparameter that can be controlled by *drop rate* value). For more details please refer to [6].

5.1.3 Top 10 models. Since we created 37 models, we would like to concentrate on this section on the Top 10. The full description of the used models can be found in the appendix (Table - 4). Most of the competition was spent on features. In the last 2 weeks we tried to do a parameter sweep of the most important parameters: learning rate, number of boosting iterations. This procedure is used often in the competitions [1]. In our case, we also observed how a single model affects the ensemble. It turned out that even models with relatively high learning rate and worse score were improving the ensemble. The hyper parameters that were stable throughout the Top 10 models were the learning rate 0.2, boosting type *dart* and objective function NDCG.

Number of trees. The number of trees for the Top 10 models varies between 3200 and 5000.

Number of leaves. It is known, that number of leaves control the complexity of the model [5]. Most of the models had value 64 as the number of leaves, we also used the value of 32 with the default drop rate.

Drop rate. We tested the default value in LightGBM, which is 0.1. The rest of the values were used based on the [6]. As shown in the Table 4 the best model has a drop out rate of 0.015.

Feature and Bagging fractions. For this hyper parameter we tested two approaches: the whole dataset and feature fraction of 0.7 with bagging fraction of 0.8. Due to the drawback of the accumulators (described in 4.3), the lower values of fractions could cause problems with the inference, although it wasn't investigated further in the practice.

Max position. Apart from that, in some models we changed *max position*. This hyperparameter limits the maximum position to optimize the objective, which can be important if a click-out item was on the 20+ position. The default value in LGBMRanker is 20, although in some models we used 25 as the number of hotels in the impression. It improved the score marginally.

Other hyper parameters. We tested XGBoost dart mode and decreased the minimum sum hessian in leaf.

5.2 Model blending

We used 37 LightGBM models with ranking objectives with a simple linear model which was directly optimizing the MRR objective on the validation dataset. The coefficients of the final ensemble are available in the source code.

5.3 Technical requirements

All our computations were made on virtual machines with the 96 vCPUs and 624 GB memory on GCP (Google Cloud Platform).

6 FUTURE RESEARCH

To use this approach in the production environment we would like to change the problem formulation. This means we would remove the timestamp of the last session click-out from the training and validation data. Furthermore, we believe there should be a minimum time difference between the last interaction and the first prediction. It means that the model needs some time to recalculate to take into account the most recent interactions, e.g., we cannot change the ranking of the current impressions even though the user interacted with the results. The models could work in a cascade fashion when some simple rules and heuristics are applied instantaneously and more complex models predictions are available after X minutes.

7 CODE

The source code is available at https://github.com/logicai-io/recsys2019

REFERENCES

[1] Christopher J. C. Burges, Krysta Marie Svore, Paul N. Bennett, Andrzej Pastusiak, and Qiang Wu. 2010. Learning to Rank Using an Ensemble of Lambda-Gradient Models. In *Yahoo! Learning to Rank Challenge (YLRC'10)*. JMLR.org, Haifa, Israel, 25–35. http://dl.acm.org/citation.cfm?id=3045754.3045758

[2] Ashkan Ebadi and Adam Krzyzak. 2016. A Hybrid Multi-Criteria Hotel Recommender System Using Explicit and Implicit Feedbacks. *World Academy of Science, Engineering and Technology* 10, 8 (2016), 1450 – 1458. http://waset.org/Publications?p=116

[3] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. Curran Associates Inc., USA, 3149–3157. http://dl.acm.org/citation.cfm?id=3294996.3295074

[4] Quoc Le and Alexander Smola. 2007. Direct optimization of ranking measures. *CoRR* abs/0704.3359 (2007), 856–864. arXiv:0704.3359 http://arxiv.org/abs/0704.3359

[5] Microsoft. 2019. LightGBM. https://buildmedia.readthedocs.org/media/pdf/lightgbm/latest/lightgbm.pdf

[6] K.V. Rashmi and Ran Gilad-Bachrach. 2015. DART: Dropouts meet Multiple Additive Regression Trees. *CoRR* abs/1505.01866 (April 2015), 489–497. arXiv:1505.01866 https://www.microsoft.com/en-us/research/publication/dart-dropouts-meet-multiple-additive-regression-trees/

[7] trivago, TU Wien, Politecnico di Milano, and Karlsruhe Institute of Technology. 2019. ACM RecSys Challenge 2019. http://www.recsyschallenge.com/2019/. Accessed: 2019-07-22.

[8] Maksims Volkovs, Guang Wei Yu, and Tomi Poutanen. 2017. Content-based Neighbor Models for Cold Start in Recommender Systems. In *Proceedings of the Recommender Systems Challenge 2017 (RecSys Challenge '17)*. ACM, New York, NY, USA, Article 7, 6 pages. https://doi.org/10.1145/3124791.3124792

Table 3: Feature importance (top 20 features)

Feature name	Description	Feature importance
session_start_ts	Time since session start	2431
price_vs_mean_price	The price of the hotel relative to the average price of all hotels in query	2147
clickout_step_rev	How many steps until the last step	2049
clickout_item_item_last_timestamp	Time since last clickout click	1963
rank	Rank of the hotel	1927
clickout_prob_time_position_offset	Probability of a click given the time difference and position of the last click	1915
last_price_diff	What is the price difference between this hotel and the previous click	1913
mean_rank_counter_mean	Mean across historical mean rank for hotels on the impression list	1772
avg_price_similarity	What is the difference between the price and average prices of clicked hotels	1390
clickout_max_step	Maximum step	1218
mean_rank_counter_min	Minimum across historical mean rank for hotels on the impression list	1062
user_start_ts	Time since the first user event	981
clickout_item_ctr_rank_weighted	CTR of the hotel weighted by the rank	968
clickout_item_ctr_corr	CTR of the hotel (impressions are corrected)	960
clickout_counter_vs_interaction_counter_pure	Hotel's total number of clicks divided by the total number of interactions	953
item_clickouts_intersection	How similar is the impression the the previous impression	892
item_id	Item ID (original numerical value)	881
clickout_item_uniq_prob	How unique are the hotels that are clicked by the user	879
interaction_item_image_item_last_timestamp	Time since last interaction with the hotel images	856
price_pct_by_platform	What is the price of the hotel relative to the platform prices	780
interact_item_uniq_prob	How unique are the hotels that the user interacted with	752
clickout_item_ctr_corr_by_platform	CTR of the hotel (impressions are corrected) – grouped by the platform	744

Table 4: Hyper parameters of the models

Boosting type	Objective function	Learning rate	Number of trees	Max leaves	Min data in leaf	Drop rate	Bagging fraction	Feature fraction	Max position	XGBoost dart mode	MRR	Blending coefficient
dart	NDCG	0.2	5000	64	5	0.015	0.8	0.7	20	False	69.068	0.51223
dart	NDCG	0.2	5000	64	5	0.03	0.8	0.7	20	False	69.026	0.76749
dart	NDCG	0.2	3200	64	5	0.1	1	1	20	False	69.018	0.37716
dart	NDCG	0.2	5000	64	5	0.03	0.8	0.7	20	False	69.011	0.47778
dart	NDCG	0.2	4800	32	5	0.1	1	1	20	False	69.010	0.33917
dart	NDCG	0.2	3200	64	5	0.1	1	1	20	False	69.006	0.25198
dart	NDCG	0.2	3200	64	5	0.1	1	1	20	True	69.002	0.58178
dart	NDCG	0.2	5000	64	5	0.015	0.8	0.7	20	False	69.001	0.79173
dart	NDCG	0.2	3200	64	5	0.1	1	1	25	False	68.995	0.00569
dart	NDCG	0.2	3200	64	3	0.1	1	1	25	False	68.985	0.00161
dart	MRR3	0.2	3200	64	5	0.1	1	1	20	False	68.970	-0.12063
dart	NDCG	0.2	1600	128	5	0.1	1	1	20	False	68.949	0.15376
dart	MRR3	0.2	1600	128	5	0.1	1	1	20	False	68.912	-0.30801
dart	NDCG	0.2	1600	128	5	0.1	1	1	20	False	68.894	0.13886
dart	NDCG	0.2	1000	256	5	0.03	1	0.5	20	False	68.860	-0.32652
dart	NDCG	0.2	1600	128	5	0.1	1	1	20	False	68.825	-0.10392
gbdt	NDCG	0.1	5000	62	5	-	1	1	20	False	68.783	0.17662
dart	NDCG	0.4	3200	64	5	-	1	1	25	False	68.782	0.40968
gbdt	NDCG	0.2	1600	62	20	-	1	1	20	False	68.773	0.32795
gbdt	NDCG	0.1	1600	62	20	-	1	1	20	False	68.768	-0.27577
gbdt	NDCG	0.1	5000	62	5	-	1	1	20	False	68.766	-0.10544
gbdt	Ranker3	0.1	1600	62	20	-	1	1	20	False	68.755	-0.46916