Leveraging laziness, Browsing-Pattern Aware Stacked Models for Sequential Accommodation Learning to Rank

Edoardo D'Amico
Politecnico di Milano
edoardo1.damico@mail.polimi.it

Giovanni Gabbolini
Politecnico di Milano
giovanni.gabbolini@mail.polimi.it

Daniele Montesi
Politecnico di Milano
daniele.montesi@mail.polimi.it

Matteo Moreschini
Politecnico di Milano
matteo1.moreschini@mail.polimi.it

Federico Parroni
Politecnico di Milano
federico.parroni@mail.polimi.it

Federico Piccinini
Politecnico di Milano
federico1.piccinini@mail.polimi.it

Alberto Rossettini
Politecnico di Milano
alberto.rossettini@mail.polimi.it

Alessio Russo Introito
Politecnico di Milano
alessio2.russo@mail.polimi.it

Cesare Bernardis
Politecnico di Milano
cesare.bernardis@polimi.it

Maurizio Ferrari Dacrema
Politecnico di Milano
maurizio.ferrari@polimi.it

ABSTRACT

In this paper we provide an overview of the approach we used as team PoliCloud8 for the ACM RecSys Challenge 2019. The competition, organized by Trivago, focuses on the problem of session-based and context-aware accommodation recommendation in a travel domain. The goal is to suggest suitable accommodations fitting the needs of the traveller to maximise the chance of a redirect (click-out) to a booking site, relying on explicit and implicit user signals within a session (clicks, search refinement, filter usage) to detect the users intent. Our team proposes a solution based on several new features, designed to capture specific types of information as well as some well-known models: gradient boosting, neural networks and a stacking-based ensemble.

CCS CONCEPTS

• **Information systems → Recommender systems**; **Learning to rank**.

KEYWORDS

ACM RecSys Challenge 2019, Recommender Systems, Feature Engineering, Stacking Ensemble, Learning To Rank

ACM Reference Format:
Edoardo D'Amico, Giovanni Gabbolini, Daniele Montesi, Matteo Moreschini, Federico Parroni, Federico Piccinini, Alberto Rossettini, Alessio Russo Introito, Cesare Bernardis, and Maurizio Ferrari Dacrema. 2019. Leveraging laziness, Browsing-Pattern Aware Stacked Models for Sequential Accommodation Learning to Rank. In *Proceedings of the ACM Recommender Systems Challenge 2019 (RecSys Challenge '19), September 16–20, 2019, Copenhagen, Denmark.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Recommender systems have become a widespread tool designed to offer personalised and relevant content to users in many different sectors, like e-commerce or entertainment, in such a way to help them identifying relevant content in a wide database. The ACM RecSys Challenge 2019[1] focuses on developing a session-based and context-aware recommender system able to provide a list of accommodations that will match the needs of the user. The task of the competition is to predict, among the search result, which accommodations the user will click during the last part of a session.

The paper is organized as follows. In Section 2 we outline the problem formulation, the dataset structure and the evaluation metrics. We build our solution using two categories of models, first in Section 3.1 we describe our ranking models (which are tasked to predict the clickout position among all), then in Section 3.2 the binary classification models (aimed at solving a simpler classification sub-problem). In Section 3.3 we explain how the individual predictions are combined together via an ensemble. In Section 4 we describe our feature engineering proposals and a selection of the most significant features. In Section 5 we present the results we obtained, along with a set of related experiments. We publicly release our source code with additional documentation and information on our solution.[2]

2 PROBLEM FORMULATION

Trivago provided the participants with a log of the interactions. Any record log is identified by *user_id, session_id, step* and *timestamp*. The step identifies the relative number of the interaction inside a continuous sequence. From now on we will refer to a *sample* as a *session* (i.e. an ordered series of steps characterised by the same user_id and session_id). A clickout is the interaction in which the user picks one of the accommodations that are displayed to him/her. The clickout we aim to predict is the last occurring in the session steps. Our task is to predict which of the shown accommodations the user will select. The evaluation metric is MRR. The dataset provided to the participants is gathered from Trivago results webpage and,

[1]http://www.recsyschallenge.com/2019/
[2]https://github.com/MaurizioFD/recsys-challenge-2019-trivago

as a probable consequence of their own recommendation system and the user's navigation pattern, is heavily unbalanced towards the accommodations in the first positions. Moreover, the company provided us an *item metadata* file, containing all the information about the accommodations (or items) characteristics (rating, stars, comforts...). In the following, item and accommodation are used as synonyms.

3 MODELS

The clicked-impression distribution is very unbalanced towards the first positions, with the first two accounting for around 50% of the clicked impressions. To address this, we relied on two sets of models: *ranking models*, tasked to predict the clicked impression among all 25, and *binary classification models*, tasked with a simpler binary classification problem, i.e. whether the clicked impression is in the first position or not, see Section 3.2.

3.1 Ranking models

3.1.1 Boosted Trees. We strongly rely on Gradient Boosting for decision trees and combine different variations of it.

XGBoost: [2] with Rank Pairwise loss. We perform one-hot encoding of the categorical features.

CatBoost: [6] is a recent library effective in handling categorical features. We use as loss YetiRank [3].

LightGBM: [4] which is effective in handling categorical features and can optimise the LambdaRank loss.

We trained the models using all the hand-crafted features available, but without the *item metadata*. The metadata alone is used to train another version of *XGBoost*, that we call *XGBoost impressions tags*.

3.1.2 Recurrent Neural Networks (RNN). In order to exploit the sequential structure of the problem we adopt a recurrent neural network. Each session is represented by a fixed number of interactions and fed to the network. Since ranking metrics are not yet available in Tensorflow/Keras, we chose to optimize the binary crossentropy loss with target vectors of 25 (maximum possible length of the impressions list). We tried with both LSTM and GRU cells and we found that the most performing architecture is the following: 2 recurrent layers (GRU, hidden size 64, dropout and recurrent dropout rate of 0.2) and 2 dense layer (with 64 and 25 neurons, relu and softmax activation functions, respectively), each of them followed by a dropout layer (with rate 0.1). We trained three versions of the network: one with sample weights designed to balance the distribution of the 25 classes, one without any weighting, and one using a training set without the first class (in order to completely ignore the majority of the samples with the first class).

3.1.3 Tensorflow Ranking (TFranking). Tensorflow ranking[5] is a ranking algorithm released by Google and is capable to optimise list-wise losses for ranking. We use a *feed-forward* neural network with the following structure: two dense hidden layers with 256 and 128 neurons respectively, each of them is followed by a batch normalisation layer. Batch normalisation is applied to the input data as well. A dropout layer with rate equal to 0.5 is applied to the last hidden layer. The output layer is dense with a number of neurons equal to the number of items to score. We tried various loss functions described in [5]: Sigmoid cross-entropy loss (pointwise), logistic and hinge losses (pairwise) and SoftmaxCross-Entropy loss (listwise). The latter one achieved the best result with a $\Delta MRR = +0.002$ w.r.t the best pairwise loss (pairwise hinge loss).

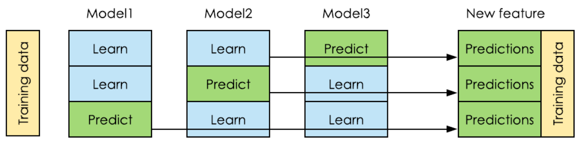

Figure 1: Creation of the stacking ensemble training set using 3-fold cross-validation and out-of-fold prediction

3.2 Binary classification

In order to mitigate the effects of a highly unbalanced dataset, we decided to introduce simpler and more balanced sub-problems. Their goal is to perform a binary classification to distinguish whether the clickout occurs on the first impression of the list or not. The results reported in Table 2 support our hypothesis of the resulting task to be easier. We used two models: XGBoost and RNN similarly to what described in Section 3.

3.3 Ensemble

We ensemble all the models previously described with *stacking ensemble*. The key idea is to use the predictions/classifications of all models and treat them as features to build another higher-level model. Due to its high accuracy, we choose XGBoost as ensemble algorithm. In particular we use our handcrafted features in combination with the scores coming from the algorithms. We have to apply a *k-fold cross-validation* on training data and *out-of-fold* prediction to retrieve the scores of each fold, as shown in Figure 1.

4 FEATURES

Feature engineering proved to be crucial to achieve good results. We overall developed about two hundred features among which we describe the most important or peculiar ones.

4.1 Context Features

The Context Features exploit the interactions with accommodations occurring within a single session.

4.1.1 Lazy User. The goal of this feature is to model the navigation pattern of the user along the displayed impressions. The underlying idea is that, since the clickout happens on an accommodation which is displayed by the Trivago interface (i.e. on user's monitor), it is more likely that a user will click on an impression which is near to his browsing position which can be deduced from his previous interactions, when available. In all the other cases, we assume that the user is at the top of the page when the clickout happens. As shown in Figure 2, the effect of using the *Lazy User* representation for the position of the clicked accommodation is that the problem becomes even more unbalanced towards the nearest accommodation with respect to the assumed position of the user. In addition, it is interesting to notice that it almost never happens that a user performs a clickout over an impression which is in a position above his inferred position (i.e. there are a negligible number of times in

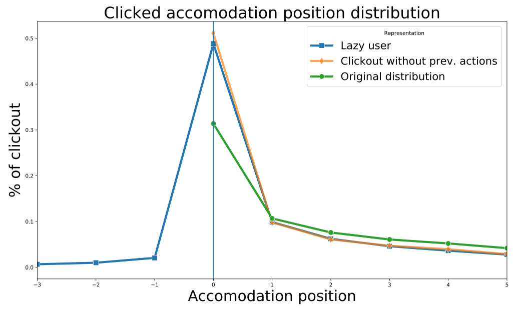

Figure 2: Comparison between original distribution of clickouts and the one represented by the Lazy User feature. It shows the percentage of clickout per accommodation position in the list. Negative positions, related to the Lazy User representation, mean that the clickout happens on an accommodation which is above the inferred position of the user (using that position as index zero).

which the user "scrolls up" the page before the clickout). Finally, another unexpected result shown by the plot is that the distribution given by the *Lazy User* representation is almost identical to the one of the sessions in which the clickouts happen as the first known interaction with the user (i.e. clickout at step one).

4.1.2 User2Item. This feature represents a *user* as a *vector* of the accommodation properties we estimate to be the most relevant for him/her. The feature builds, for each user/session pair, a vector containing the number of occurrences of each item metadata in the interacted items. For each impression, the final feature value is the similarity between the session feature value and the impression metadata, computed via both cosine similarity and Manhattan distance.

4.2 Content Features

The Content Features take into account interactions spanning across multiple sessions or other non-session related information.

4.2.1 User Features. Since around *20%* of the users has more than one session, we add some feature to represent this *global* user navigation behaviour. Among those there are: *number of user interactions with the item*, *seconds before closest interaction*, *action type of closest interaction*, *number of times the item appeared in impressions list*, *average price interacted items*, *average position of clicked items in impressions list*, and many others. Since a user could have searched more than one city along his/her history, we compute those features only including sessions associated to the city referenced in the last clickout. Moreover, it was extremely useful the separation between past sessions and future sessions. Features were crafted distinguishing both the case of past sessions and future sessions occurrences.

4.2.2 Popularity Features. The following features are some of those built around the concept of popularity:

Session Clickout/Interaction Count: number of occurrences of an item (clicked or interacted) within that session;

Platform Popularity: percentage of the number of times that an item has been clicked over the total number of clicks, within a specific platform. The idea is that there can be items with a small overall popularity that exhibit significant variance across platform.

City Location Popularity: highlights the most popular accommodations *for each city*, it is computed in the same way of the Platform Popularity, but for cities;

Percentage Clicked / Appeared: the percentage of impressions of that item that resulted in clicks, as a measure of the *user interest* in that impression.

4.2.3 Platform Characterising Features. This group of features attempts to model the different relevance that users browsing the accommodations from different platforms may attribute to different metadata. Among them, the most important is *PlatformFeatures-Similarity*. The feature builds a vector containing the number of occurrences of each metadata in the clicked accommodations for each platform. The vector is then normalised and the feature value is thecosine similarity computed with each impression metadata.

4.2.4 Price Quality. This feature models how an impression is *cost-effective* in terms of stars. It is computed as the weighted summation of the impression rating (weight 1.5) and the impression stars (weight 1.0), divided by the impression price. In case of missing values, we use the average of all the current impressions.

4.2.5 Filters satisfaction. Due to their different semantics, we distinguish the filters in two categories:

Change of sort order filters: they sort the impressions based on 7 criteria (price, distance, rating, price and recommended, rating and recommended, distance and recommended, our recommendations).

Tags filters: they filter on the impressions tags (metadata).

The *change-of-sort-order filters* have been one-hotted and used directly as features. The *tag filters*, instead, have been considered calculating, for each impression, the fraction of tags that are present in both the impression tags and the filters.

Table 1: Best performing models evaluated on our private split

	XGBoost	LightGBM	CatBoost	TFranking	RNN
MRR	0.6791	0.668	0.673	0.667	0.654

Table 2: Evaluation of our two classifiers in the binary classification scenario (clickout in first impression or not) and XGBoost ranker.

Algorithm	Precision	Recall	F1 Score
XGBoost ranker	0.51	0.91	0.65
XGboost classifier	0.74	0.74	0.74
RNN classifier	0.68	0.78	0.73

5 EXPERIMENTAL EVALUATION

For each model we report in Table 1 the best MRR achieved on the private validation dataset. The hyperparameters tuning is done via Random and Bayesian search [1].

5.1 Cluster-wise analysis

In order to provide a more in-depth analysis of our solution we also evaluate the algorithms on subsets of *sessions*, we refer to as *clusters*. Since all the ranking models have similar performance on each cluster, the following evaluation refers to the final ensemble.

Table 3: Evaluation based on the presence of interactions with items. The clusters in *italic* represent the union of the previous clusters

Id	Cluster Name	Size	MRR
1.1	Item Interaction one step before clickout	59%	0.7282
1.2	Item Interaction more steps before clickout	5%	0.5452
1	*At least one item interaction*	*64%*	*0.7137*
2.1	One step	21%	0.6712
2.2	More than one step	15%	0.5034
2	*No item interactions*	*36%*	*0.6018*

Table 4: Evaluation based on number of actions in session

Cluster Name	Size	MRR
One action	21%	0.6712
Two actions	14%	0.6283
From Three to Ten actions	37%	0.6814
More than ten actions	28%	0.7153

5.1.1 Device. Sessions associated to the same device used by the user are clustered together. The quota of sessions associated to each device and the corresponding MRR are the following: Mobile, 55%, 0.6899; Desktop, 37%, 0.6739; Tablet, 8%, 0.6564. These results highlight very good performance in case of mobile devices. Our hypothesis is that on mobile devices the navigation is more constrained resulting in less flexibility to compare several options, therefore users pay attention mainly to the first items in the page or closely follow the results ordering. It is also interesting to notice how different are the performances when a tablet is used.

5.1.2 Presence of interactions with items. Sessions can be divided in those having at least one interaction with an item, and those which do not. We notice from Table 3 that interactions with items are a very rich source of information. The MRR is high in case such interaction is right before the last clickout (Id 1.1). In the opposite case (Id 1.2), the performance drops, most likely because the item interactions are not meaningful for the clickout that we have to predict. When there is a lack of item interactions due to sessions composed of just the last clickout (Id 2.1), the performance does not degrade much. In the former cluster almost half of the clickouts are on the first impression, as shown in Figure 2. We believe this to be the reason of the good performance the model manages to achieve, even without having any information on the interacted items.

5.1.3 Number of actions. As shown in Table 4, we cluster the *sessions* based on how many interactions they are composed by. We observe that in very long sessions (i.e. more than ten interactions) the results are very good. This further confirms the importance of interactions in understanding the user's interest. We also observe that in sessions with exactly two actions the results are much worse.

5.1.4 Clusters analysis. From Table 3 we identify sets of sessions that heavily penalise the final score. In particular, we struggle to model sessions where the user has not interacted with any accommodation (Id 1.2) and where the interactions are not meaningful for the *clickout* that we have to predict (Id 2.2). Content Features help in those cases: with the *XGBoost* model described in Section 3.1.1, we obtain an MRR increment of 0.035 on those clusters against an increase of only 0.011 on the others.

Table 5: Permutation importance for the top 15 features of the ensemble model, we report mean and standard deviation on 5 shuffle

Feature name	Average Δ MRR	Standard Δ MRR
XGBoost	0.4177	0.0013
TFRranking (softmax loss)	0.0029	0.0003
RNN (balanced)	0.0027	0.0002
RNN classifier	0.0025	0.0003
XGBoost impressions tags	0.0009	0.0003
No. of past clickouts on impression	0.0009	0.0002
Personalised popularity	0.0008	0.0003
Length of impressions list	0.0005	0.0002
Last position interacted	0.0004	0.0002
TFRranking (pairwise hinge loss)	0.0003	0.0001
Number of clickouts	0.0003	0.0000
Session duration (seconds)	0.0003	0.0003
Past time from last interaction impression	0.0003	0.0002
RNN (without first class)	0.0002	0.0002
Impression position	0.0002	0.0001

5.2 Ranking Performance analysis

We discuss the result of the stacking ensemble, by comparing the distribution of the positions of the correct clickouts before and after the ranking algorithm has been applied. A full diagram is available in the online material. We can observe that from position 3 to position 25 the difference is negative on average i.e. the total percentage of correct clickouts in this positions have been reduced. The mean of the decrement is $\mu = -0.0085$ and the standard deviation is $\sigma = 0.0018$. In contrast we can see that the only two positions with a positive increment are the *first* and the *second* with an increment of $\Delta 1 = +0.19$ and $\Delta 2 = +0.01$ respectively. From the analysis it turns out that the final ranking algorithm was able to detect, with equal effort, correct clickouts from position 3 to 25.

6 CONCLUSION

The final ensemble contained the following models: XGBoost Ranker (Section 3.1.1), TensorflowRanking, (Section 3.1.3), RNN, (Section 3.1.2), RNN Classifier (Section 3.2) and XGBoost Impression Tags (Section 3.1.1). In addition, the features adopted to train the models were all added again to the ensemble. To have an idea of the contribution each feature we report the *permutation importance*[3] of the top 15 features on the final ensemble, see Table 5. The permutation importance is computed as the difference between the ensemble score and the score obtained by randomly shuffling a single feature in the dataset at a time, in order to see how much the model relies on that feature and is therefore sensitive to its changes. Finally, note that not all the algorithms described in the previous sections were used due to the lack of time for the cross-validation dataset creation (Section 3.3). However, experiments suggest that the performance could have been further improved. Overall the ensemble succeeded in improving the overall performance as confirmed by the final MRR of 0.6765 in the private leaderboard and 0.6813 in our private validation split, guaranteeing us the tenth position in the final leaderbord, with a ΔMRR of 0.0033 from the fifth position and 0.0091 from the first.

ACKNOWLEDGEMENT

We would like to thank Prof. Paolo Cremonesi for his support.

[3]https://eli5.readthedocs.io/en/latest/blackbox/permutation_importance.html

REFERENCES

[1] S. Antenucci, S. Boglio, E. Chioso, E. Dervishaj, K. Shuwen, T. Scarlatti, and M. Ferrari Dacrema. 2018. Artist-driven layering and user's behaviour impact on recommendations in a playlist continuation scenario. In *Proceedings of the ACM Recommender Systems Challenge 2018 (RecSys 2018)*. https://doi.org/10.1145/3267471.3267475 Source: https://github.com/MaurizioFD/spotify-recsys-challenge.

[2] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[3] Andrey Gulin, Igor Kuralenok, and Dimitry Pavlov. 2011. Winning The Transfer Learning Track of Yahoo!'s Learning To Rank Challenge with YetiRank. In *Proceedings of the Learning to Rank Challenge*.

[4] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and T. M. Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *NIPS*.

[5] Rama Kumar Pasumarthi, Sebastian Bruch, Xuanhui Wang, Cheng Li, Michael Bendersky, Marc Najork, Jan Pfeifer, Nadav Golbandi, Rohan Anil, and Stephan Wolf. 2019. TF-Ranking: Scalable TensorFlow Library for Learning-to-Rank. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. (to appear).

[6] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: Unbiased Boosting with Categorical Features. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*.

A PREPROCESSING

A.1 Cleaning phase

Following an analysis of the original dataset, we decide to apply some data cleaning. In particular we decide to remove the interactions occurring after the last *clickout* in a *session*, the accommodations in *item metadata* file that do not appear neither in *train* nor *test* and the erroneous interactions having missing accommodation. For Recurrent Neural Networks we decide to delete consecutive duplicates of interactions within the same *session* keeping only the number of repetitions, to lower the computational complexity.

A.2 Additional phase after the Cleaning Phase

We attempted to further process output of the cleaning phase transforming it more radically. In particular we carried out those experiments:

Unrolling: we crafted new *sessions* from existing ones by considering any set of steps from the beginning to a *clickout* as a new session. We aim this way to augment the data available

User Based: we merged sessions of the same user in a unique session, hoping to replicate the effect of the User Features

Oversampling: we resampled sessions where the *clickout* was different from the first element in the list, aiming at mitigating the balancing problem.

However, the MRR obtained by using one of those methods proved to be much lower than the ones resulted from the use of Section A.1. Most likely it is because changing the dataset radically as described above breaks structures in the data that were useful for the algorithm to learn. So, in the end, we use data as they are after the Cleaning Phase.

A.3 Dataset creation phase

From cleaned data described in Section A.1, we create a dataset having the format required to use each model adopted for the challenge. After the data has been cleaned, we build two different datasets depending on the type of model we apply:

Ranker: We treat each session as a *query* and each accommodation as a *document*. The dataset contains the features associated to session, accommodation and their combination.

RNN: Each session is truncated/padded to a fixed number of 12 interactions, as experimental evidences suggest that higher values result in negligible improvements. The label is a binary one-hot vector of 25 elements representing the position of the clicked impression.

B RANKING PERFORMANCE ANALYSIS

We discuss the result brought from the final model we used (stacking ensemble), bringing a comparison on the distribution of the positions of the correct clickouts before and after the ranking algorithm has been applied, showing the percentage difference in Figure 3. We can observe that from position 3 to position 25 the difference is negative on average i.e. the total percentage of correct clickouts in this positions have been reduced. The mean of the decrement is $\mu = -0.0085$ and the standard deviation is $\sigma = 0.0018$. In contrast we can see that the only two positions with a positive increment are the *first* and the *second* with an increment of $\Delta 1 = +0.19$ and $\Delta 2 = +0.01$ respectively. From the analysis it turns out that the final ranking algorithm was able to detect, with equal effort, correct clickouts from position 3 to 25.

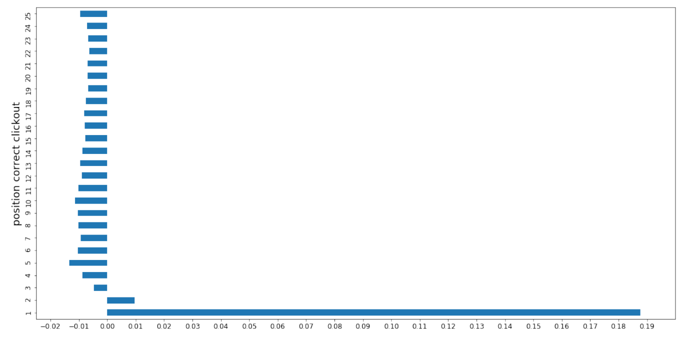

Figure 3: Comparison between distribution of correct clickouts before and after ranking.