COSI129 Final Projecet: RecSys Challenge 2019

Wei Lu **Kun Li** **Bingyang Ye**
Brandeis University
{luwei,kunli,byye}@brandeis.edu

April 29, 2019

1 Introduction

Trivago is a global hotel search platform focused on reshaping the way travelers search for and compare hotels on our website and app. Due to the nature of their domain, we face specific challenges that make it difficult to build predictive models and recommendation systems that are tailored to the needs of the visitors. So companies like trivago are faced with challenges like users searching for accommodations comparatively infrequently with sometimes long time intervals between their trips, booking accommodation too expensive, and information about the personal preferences of travelers too sparse.

Classical approaches to build recommendation systems are not very well suited to address the challenges mentioned above. Many well-established methods like matrix factorization and collaborative filter variants compute recommendations based on data sets with aggregated information of users and their preferences. Consequently, they have a hard time delivering accurate predictions in extreme cold-start scenarios in which the majority of users can be considered new.

On the other hand, common click prediction techniques suffer from a lack of personalization and have no straightforward way to take into account the time-dependent nature of interaction sequences of users that allows updating recommendations in case of changing preferences. The new field of sequence-aware recommender systems addresses some of these issues, but is in its infancy and can benefit from large-scale datasets for algorithm development and evaluation.

In this challenge, we hope to build a more robust and more domain specific recommendation system that can handle the problems addressed before.

2 Problem Definition

The goal of the challenge is to use user signals within a session to detect the intent of the user and to update the recommendation of accommodations provided to the user. Given a dataset of the interactions of the users on our website and metadata for the items they interacted with, the participants are tasked with predicting what items have been clicked in the later part of a session.

The data provided for this challenge consists of a training and test set, and metadata for accommodations (items). The training set contains user actions up to a specified time (split date). It can be used to build models of user interactions and specifies the type of action that has been per-

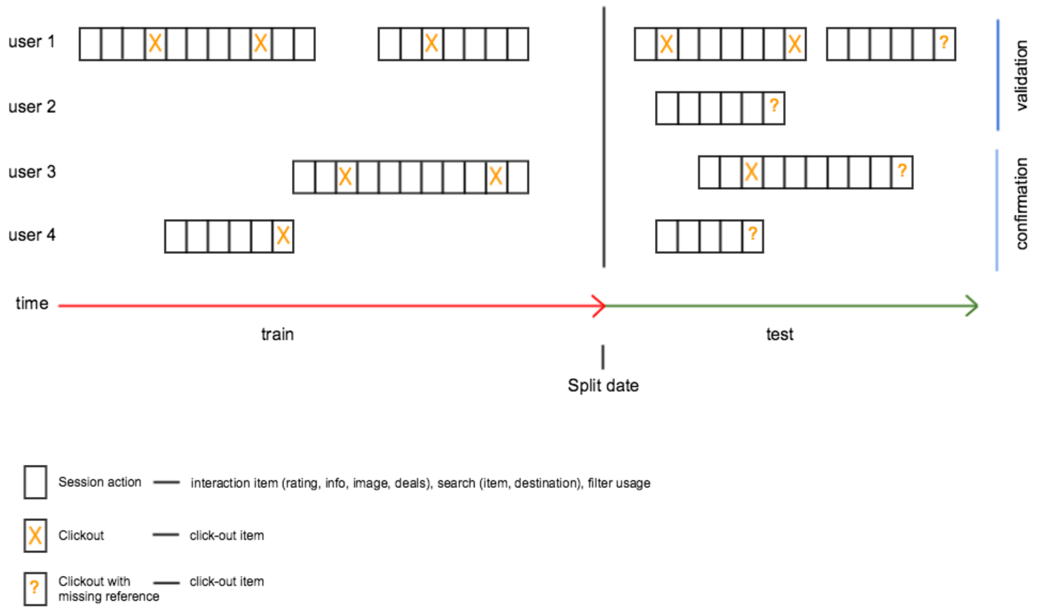

user 1
user 2
user 3
user 4
time
train
test
Split date
validation
confirmation
Session action — interaction item (rating, info, image, deals), search (item, destination), filter usage
Clickout — click-out item
Clickout with missing reference — click-out item

Figure 1: Data setting

formed (filter usage, search refinements, item interactions, item searches, item click-outs) as well as information about impressed items and prices at the time of a click-out.

The recommendations should be provided for a test set that contains information about sessions after the split date but is missing the information about the accommodations that have been clicked in the last part of the sessions. The required output is a list of maximum 25 items for each click-out ordered by preferences for the specific user. The higher the actually clicked item appears on the list the higher the score.

3 Data Pre-processing

By looking at the the data, we could observe that the data has all the features as following:

- user_id: identifier of the use
- session_id: identifier of each session
- timestamp: UNIX timestamp for the time of the interaction
- step: step in the sequence of actions within the session
- action_type: identifier of the action that has been taken by the user.
  - clickout item: user makes a click-out on the item and gets forwarded to a partner website. The reference value for this action is the item_id. Other items that were displayed to the user and their associated prices are listed under the 'impressions' and 'prices' column for this action.
  - interaction item rating: user interacts with a rating or review of an item. The reference value for this action is the item id.
  - interaction item info: user interacts with item information. The reference value for this action is the item id.

  - interaction item image: user interacts with an image of an item. The reference value for this action is the item id.
  - interaction item deals: user clicks on the view more deals button. The reference value for this action is the item id.
  - change of sort order: user changes the sort order. The reference value for this action is the sort order description.
  - filter selection: user selects a filter. The reference value for this action is the filter description.
  - search for item: user searches for an accommodation. The reference value for this action is the item id.
  - search for destination: user searches for a destination. The reference value for this action is the name of the destination.
  - search for poi: user searches for a point of interest (POI). The reference value for this action is the name of the POI.

- reference: reference value of the action as described for the different action types
- platform: country platform that was used for the search, e.g. trivago.de (DE) or trivago.com (US)
- city: name of the current city of the search context
- device: device that was used for the search
- current_filters: list of pipe-separated filters that were active at the given timestamp
- impressions: list of pipe-separated items that were displayed to the user at the time of a click-out (see action_type = clickout_item)
- prices: list of pipe-separated prices of the items that were displayed to the user at the time of a click-out (see action_type = clickout_item)

Among all these features, clickout item is definitely the most important one because when users actually click on an item, it shows that they have strong interest in it. Therefore, our intuition is that if the more an item is clicked out, the higher the chance it should be clicked out by other users, regardless of other user preferences.

4 Model and Framework

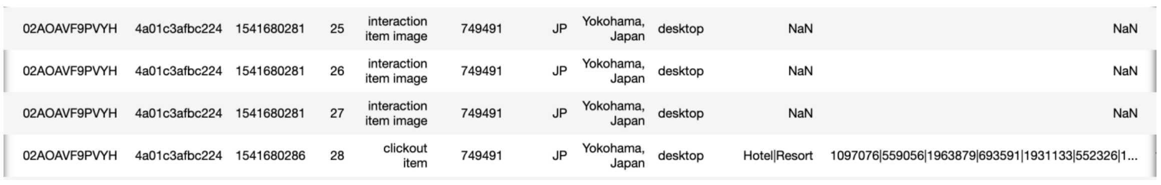
The model we use is the baseline model proviede by the challenge organizer. The model first gets number of clicks each item receiveed in the training dataset. Then it will calculate the recommendation by calculating the popularity of items. The final output data frame will have an impression list sorted according to the number of clicks per item in a reference data frame.

5 Experiments

Our first attempt is to run the baseline model directly without any other feature engineering. The intuition of calculating the popularity of items is based on the times an item has been clicked. But it didn't put personal preference into consideration. We tried several experiments to improve it.

5.1 Experiment 1: Consider preceding actions

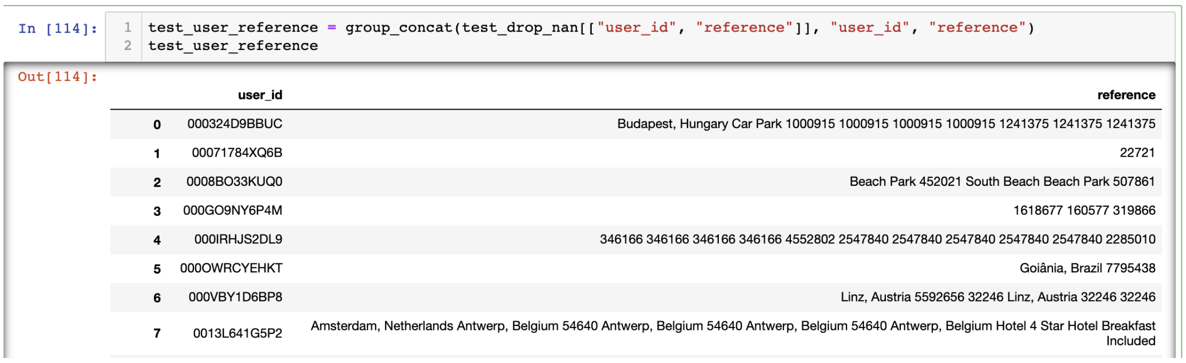
This is based on the assumption that if a user has taken actions (viewing images, ratings, deals ..) on a hotel, it's very likely that he/she will click out that hotel, as shown in the figure below:



02AOAVF9PVYH	4a01c3afbc224	1541680281	25	interaction item image	749491	JP	Yokohama, Japan	desktop	NaN	NaN
02AOAVF9PVYH	4a01c3afbc224	1541680281	26	interaction item image	749491	JP	Yokohama, Japan	desktop	NaN	NaN
02AOAVF9PVYH	4a01c3afbc224	1541680281	27	interaction item image	749491	JP	Yokohama, Japan	desktop	NaN	NaN
02AOAVF9PVYH	4a01c3afbc224	1541680286	28	clickout item	749491	JP	Yokohama, Japan	desktop	Hotel|Resort	1097076|559056|1963879|693591|1931133|552326|1...

Figure 2: A user viewed images of a hotel then clicked it out

We put all the hotels a user has taken action on in a set, using the group_concat function in the baseline (each row represent a user and his/her actions):



```
In [114]:
test_user_reference = group_concat(test_drop_nan[["user_id", "reference"]], "user_id", "reference")
test_user_reference
```

Out[114]:

	user_id	reference
0	000324D9BBUC	Budapest, Hungary Car Park 1000915 1000915 1000915 1000915 1241375 1241375 1241375
1	00071784XQ6B	22721
2	0008BO33KUQ0	Beach Park 452021 South Beach Beach Park 507861
3	000GO9NY6P4M	1618677 160577 319866
4	000IRHJS2DL9	346166 346166 346166 346166 4552802 2547840 2547840 2547840 2547840 2547840 2285010
5	000OWRCYEHKT	Goiânia, Brazil 7795438
6	000VBY1D6BP8	Linz, Austria 5592656 32246 Linz, Austria 32246 32246
7	0013L641G5P2	Amsterdam, Netherlands Antwerp, Belgium 54640 Antwerp, Belgium 54640 Antwerp, Belgium 54640 Antwerp, Belgium Hotel 4 Star Hotel Breakfast Included

Figure 3: Users and their actions

For each hotel in the set of actions, if the hotel is in the impression list, move that hotel to the top. Using this method we got the score 0.498.

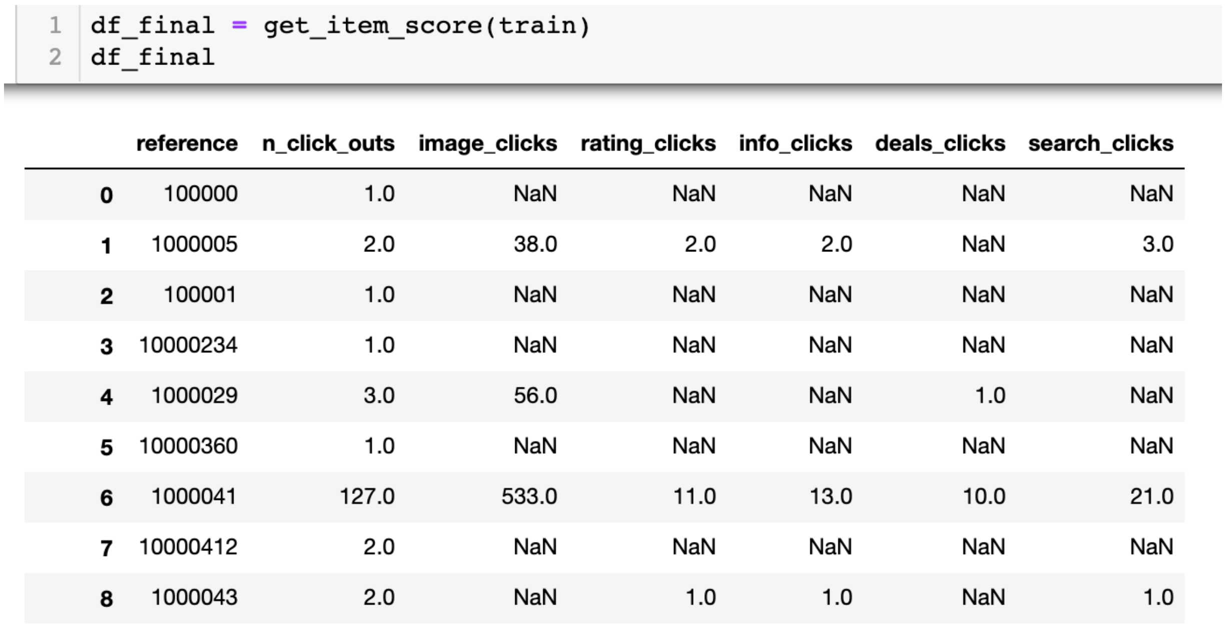
The benefit of using set to store actions is that duplicate items will be removed and it takes only $O(1)$ time to check if an item is in the set. But the drawback is that it ignores the order of actions (since there is no order in a set). This leads to our second experiment.

5.2 Experiment 2: Consider the order of actions

The assumption for this experiment is that actions taken in the last part of the sessions play more important roles. To refine the method used above, instead of using a set to store actions, we use a list to store all actions, the order of the items in the list is based on time order. This little change improved our score to 0.532.

5.3 Experiment 3: New definition of popularity

In the methods above, the popularity for each item is just based on the number of "click_out" it received. However, we assume that other actions (like click images, ratings, info, deals ...) should also be considered. Now we consider the times of all these action for each item. We have a dataframe as below.

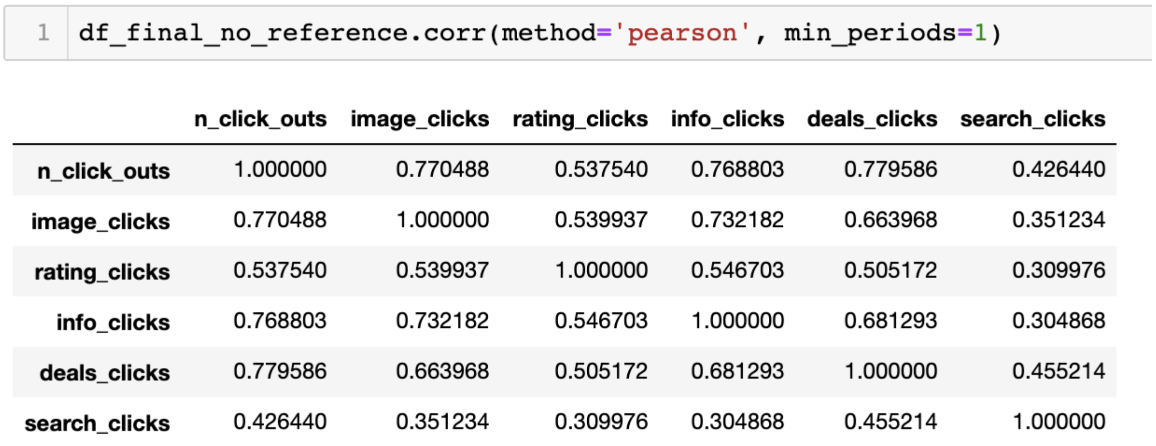


```python
df_final = get_item_score(train)
df_final
```

	reference	n_click_outs	image_clicks	rating_clicks	info_clicks	deals_clicks	search_clicks
0	100000	1.0	NaN	NaN	NaN	NaN	NaN
1	1000005	2.0	38.0	2.0	2.0	NaN	3.0
2	100001	1.0	NaN	NaN	NaN	NaN	NaN
3	10000234	1.0	NaN	NaN	NaN	NaN	NaN
4	1000029	3.0	56.0	NaN	NaN	1.0	NaN
5	10000360	1.0	NaN	NaN	NaN	NaN	NaN
6	1000041	127.0	533.0	11.0	13.0	10.0	21.0
7	10000412	2.0	NaN	NaN	NaN	NaN	NaN
8	1000043	2.0	NaN	1.0	1.0	NaN	1.0

Figure 4: Number of actions for each item

For example, item 1000005 has been clicked out twice, its images have been viewed 38 times, both ratings and information have been viewed 2 times respectively, and no one has viewed its deals. Next we consider the importance of each action. We run the pearson correlation for this dataframe and obtain the following result.



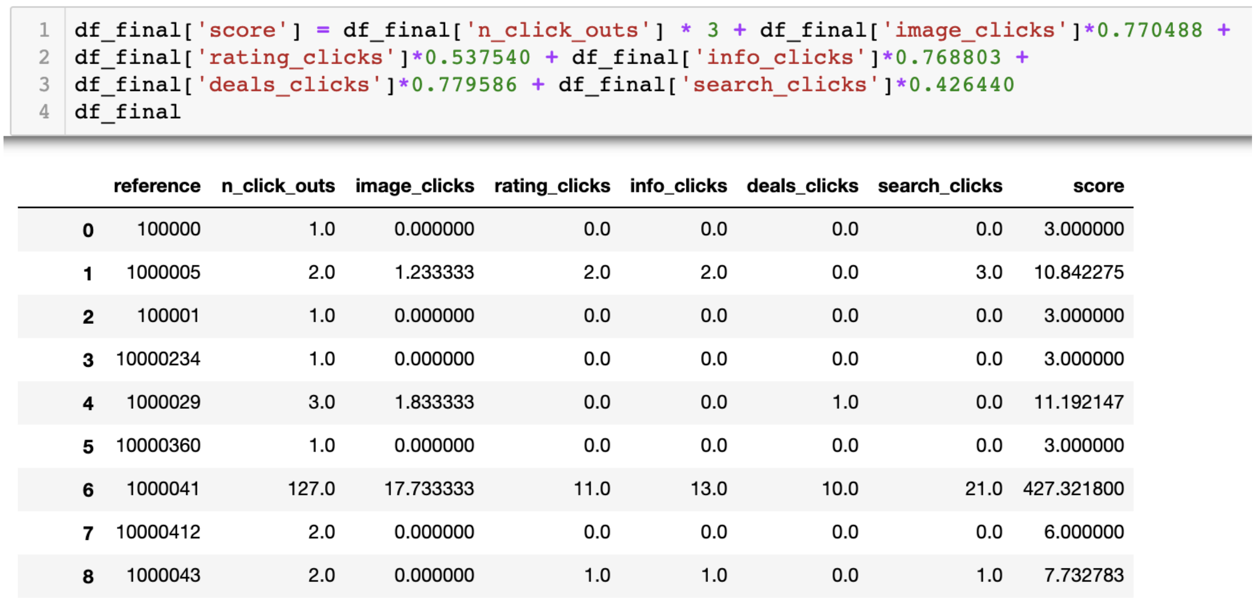
```python
df_final_no_reference.corr(method='pearson', min_periods=1)
```

	n_click_outs	image_clicks	rating_clicks	info_clicks	deals_clicks	search_clicks
n_click_outs	1.000000	0.770488	0.537540	0.768803	0.779586	0.426440
image_clicks	0.770488	1.000000	0.539937	0.732182	0.663968	0.351234
rating_clicks	0.537540	0.539937	1.000000	0.546703	0.505172	0.309976
info_clicks	0.768803	0.732182	0.546703	1.000000	0.681293	0.304868
deals_clicks	0.779586	0.663968	0.505172	0.681293	1.000000	0.455214
search_clicks	0.426440	0.351234	0.309976	0.304868	0.455214	1.000000

Figure 5: Correlation of actions

Our focus is the first row of the table. It shows the correlation of all actions against ”click_out”. We regard these correlations as weights of the corresponding actions. Using these weights we can calculate a weighted sum as the score for each item.

Using these scores, combined with the personal actions as shown in the second experiment, we can rerank the impression list. In the end we get score 0.529. Although this is a more sophisticated method, the result is not as good as we expect.



```
df_final['score'] = df_final['n_click_outs'] * 3 + df_final['image_clicks']*0.770488 +
df_final['rating_clicks']*0.537540 + df_final['info_clicks']*0.768803 +
df_final['deals_clicks']*0.779586 + df_final['search_clicks']*0.426440
df_final
```

	reference	n_click_outs	image_clicks	rating_clicks	info_clicks	deals_clicks	search_clicks	score
0	100000	1.0	0.000000	0.0	0.0	0.0	0.0	3.000000
1	1000005	2.0	1.233333	2.0	2.0	0.0	3.0	10.842275
2	100001	1.0	0.000000	0.0	0.0	0.0	0.0	3.000000
3	10000234	1.0	0.000000	0.0	0.0	0.0	0.0	3.000000
4	1000029	3.0	1.833333	0.0	0.0	1.0	0.0	11.192147
5	10000360	1.0	0.000000	0.0	0.0	0.0	0.0	3.000000
6	1000041	127.0	17.733333	11.0	13.0	10.0	21.0	427.321800
7	10000412	2.0	0.000000	0.0	0.0	0.0	0.0	6.000000
8	1000043	2.0	0.000000	1.0	1.0	0.0	1.0	7.732783

Figure 6: Score for each item

5.4 Experiment 4: A surprisingly simple approach

This is due to a hint we received from classmate Wenxiao: don't consider popularity, just use the raw impression list. With the raw impression list in the test dataset and combined with our method in experiment 2, our score goes surpringly to 0.64. We deduce that the intuition behind this approach is that users are more likely to click out hotels which appear on the top part of the page. Since they have no idea of the popularity we defined for each hotel, reranking hotels based on popularity will not do any good.

6 Conclusion and Future Work

In this report, we introduce a recommendation system to detect the intent of the user and to update the recommendation of accommodations provided to the user. We have run several experiments based on both popularity of items and users' personal preference.

Due to the limited time, our result is not among the best. Our next step is to think about how to use filters and the metadata which provides the properties of each hotel, and we hope it can give us a better result.

A Appendix: Submission history

2019-05-02 22:24:54 (submission_raw_consider_ordered_actions.csv)	valid	0.643641
2019-05-01 16:38:19 (submission_item_score_9times_click_out.csv)	valid	0.284259
2019-05-01 00:36:06 (submission_consider_ordered_actions.csv)	valid	0.532406
2019-04-29 20:52:13 (submission_consider_actions_rerank_images.csv)	valid	0.507478
2019-04-15 03:33:07 (submission_consider_images.csv)	valid	0.431809
2019-04-11 07:38:05 (submission_consider_actions.csv)	valid	0.498586
2019-04-10 05:34:10 (submission_popular.csv)	valid	0.288448
2019-05-02 06:06:13 (submission_no_nomarlize_score_consider_ordered_actions.csv)	valid	0.529558