Job Recommender System: Cold Start and Constrained Setting

ABSTRACT

We present our solution to the job recommendation task for *RecSys Challenge 2017*. Our solution focuses on addressing cold start and constrained recommendation challenges. In offline round, we build a system based on a hybrid matrix factorization model to make use of user and item profiles for recommendations to new job postings. We demonstrate various schemes that are effective for recommendations in cold start scenarios like removal of item identity for better generalization to new items. Our solution achieved 7^{th} place in the offline round of the challenge. In online round, we study the problem of recommendation with constraints on the number of times items and users can appear in the recommendation pairs. We propose a novel algorithm based on stable matching to address the problem. Our local evaluation demonstrates the effectiveness of the method.

KEYWORDS

Recommendation Systems, RecSys Challenge, Cold Start, Matrix Factorization, Stable Matching, Constrained Targeting

ACM Reference format:
. 2019. Job Recommender System: Cold Start and Constrained Setting. In *Proceedings of ACM Conference, Como, Italy, Aug 2017 (RecSys Challenge'17),* 5 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

We present our solution to the job recommendation task in *RecSys Challenge 2017*. In our solution we focus on addressing two recommendation challenges: cold start and recommendation under constraints. We first state the challenges as follows:

Cold start: The system is asked to recommend users to "new" job posts based on observed interactions (like post clicks, bookmarks, replies, etc.) on older job posts. The features of the new items are available but the items have never appeared in the interaction history. The system has to understand the features and how they contribute to interactions of interests, and make recommendation based solely on these features.

Recommendation under constraints: When making recommendation in the form of a set of user-item pairs, the system has the constraint that no single user appears more than n times in total; similarly, no single item appears more than m times. With the constraint, the system cannot simply make recommendation independently for each item (or user). Instead, it has to carefully balance user and item preferences and make comprehensive recommendations. In the challenge, we have n=1, m=250 but we are interested in the general cases.

Both challenges are common in real scenario applications. Cold start has been a long standing challenge due to the fact there are a large portion of inactive users and new items. Constrained recommendations are useful for smart targeting by restricting the number of recommendation pairs pushed, so that relevant recommendations can get maximum visibility. In this challenge, we should balance interests of both users and recruiters, while providing constrained recommendations.

RecSys Challenge'17, Aug 2017, Como, Italy
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: Dataset and Feature Description.

User Feature types	**Features**
Categorical	career_lever, discipline_id, industry_id, id, country, region, exp_years, edu_degree, exp_in_entries_class, exp_in_current, premium
Multi-hot	job_roles, field_of_studies
Item Feature types	**Features**
Categorical	id, career_level, discipline_id, country, region, employment
Numerical	latitude, longitude, created_at
Multi-hot	title, tags

2 CHALLENGE AND DATA: RECSYS 2017

The *RecSys Challenge 2017* provides interaction data between job postings and users sampled from the professional network and job search website *Xing.com*. [1] The task is to recommend users for each target item (job posting) that would maximize the user interest in the item as well as the recruiter interest in the user. The target items are new or relatively new job postings that are not present or sparsely present in the interactions data. The target users are a subset of users provided by RecSys, who are allowed to receive recommendations. A maximum number of recommendations are allowed for each item.

Features. Along with interactions, the challenge provides the features defining the user profiles as well as the items. Users and items are described by a rich set of features like their career level, job roles and others. The features can take on one value or multiple values depending on their type. The values that they take are encoded as numeric ids for the purpose of the task. The interaction data comprises the different ways in which a user interacts with a job posting, such as clicks, bookmarks, reply, and delete. Interactions also include another type of interaction to capture the recruiter interest in a user. Additionally, interaction data also includes impressions which is that the job posting is shown to the user. The detailed feature information of this dataset is shown in Table 1.

Evaluation Metric. The evaluation metric has two components, user success and item success scores defined as follows:

UserSuccess = ((clicked) * 1 + (bookmarked or replied) * 5 + (recruiter interest) * 20 + (delete only) * -10) * premium

where premium is the factor of 2 for prioritizing users who pay for the services of the website.

[1] https://www.xing.com/

ItemSuccess = 25 * success * paid

where success is 1 if there is atleast one user recommended for that item with a UserSuccess score > 0, otherwise 0 and paid is the factor of 2 for prioritizing job postings that are paid from the ones that are not. The scores are summed across recommended target users and over all the target items.

Offline and online rounds. *RecSys Challenge 2017* is divided into two parts, the offline round and the online round. For the offline round, submission of participants are evaluated based on the next two weeks online interactions of the target items (used as ground truth). For the online round, an initial dataset of users and items with interaction data is provided. New target items are released on a daily basis and the task is to recommend a maximum of 250 users for every item within that day. The item, as in the case of the offline challenge, is a new job posting that has not appeared in the initial data provided. The target users for each day also change but are a subset of the users provided in the initial data. An additional constraint imposed for smart targeting, is to allow only one item to be pushed to any target user. Therefore recommendations across all the target items can contain every target user at most once.

3 APPROACH: RECOMMEND NEW ITEMS

In this section we present our approach to deal with cold start challenge. We build our system based on hybrid matrix factorization models and explore different techniques to adapt to our problem.

3.1 Hybrid Matrix Factorization

It is crucial to make use of contents of users and items for recommendation in cold start situations. Meanwhile, it is often not sufficient to use content alone. To combine both, we choose hybrid matrix factorization technique [2], to model both identities and features of users and items in the same space as vector representation and infer their parameters by fitting to the observed interactions.

Particularly, we represent each user and item as a d-dimensional vector computed as a sum of the vector representations of its associated features. Let $\vec{x}_j^U/\vec{x}_j^I$ denote the embedding (i.e., vectors of factors) of the user/item feature j, $\vec{q}_u/\vec{q}_i$ denote the embedding of user u /item i, and b_j^U/b_j^I denote the user/item bias for feature j. Then the representations of users, items, including their biases are

$$\vec{q}_u = \sum_{j \in f_u} \vec{x}_j^U, \vec{q}_i = \sum_{j \in f_i} \vec{x}_j^I; \qquad b_u = \sum_{j \in f_u} b_j^U, b_i = \sum_{j \in f_i} b_j^I. \quad (1)$$

The model prediction score for pair $\{u, i\}$ is then given by

$$S(u, i) = \vec{q}_u \cdot \vec{q}_i + b_u + b_i \quad (2)$$

The model is trained by minimizing the sum of a loss on $S(u, i)$ and the observed ground truth $t(u, i)$,

$$L = \sum_{\{u,i\} \in I} \ell(S(u, i), t(u, i)) \quad (3)$$

where I is set of interactions between user u and item i, ℓ is a loss function such as Cross-Entropy loss (HMF-CE).

3.2 Feature Combination

The features associated with users and items are heterogeneous. In order to mode them more efficiently, we divide features into three distinct kinds: *categorical*, *multi-hot*, and *numerical*, as shown in Table 1. Different from *Categorical* and *Numerical* features, *Multi-hot* features are often descriptive, and an object is accompanied by one or more feature values of a kind. In feature combination in (1), we first average embedding of values within each *Multi-hot* category before averaging across all feature types. This hierarchy improves model performance.

3.3 Identity Embedding Schemes

It is in general important to model the identity of a user or an item in addition to features as the identity often captures unique characteristics beyond content information (e.g., popularity). It is often suboptimal to only model content for recommendation. However, in cold start situations, new item identities are new and do not carry useful information.

We empirically explore different schemes whether the identity is modeled or not. In case when identity is still modeled, we add "unk" identity to represent rare (or low frequency) ones and use its vector for new items. Our results show the model variant without modeling identity is better. We also find it is still important to retrain user identities as it helps differentiate users.

3.4 Negative User Post-processing

The evaluation discourage negative interaction: "deletes". In our model training (3), we do not incorporate such discourage mechanism. Instead, we identify target users that have had a behavior of negative interactions in the past and filter them out from the final recommendations. We find this simple heuristic is very helpful and complement our training criterion (3). More sophisticated heuristics as well as training criterion for feature selection should be explored and could benefit the model.

3.5 Other Techniques

We briefly describe other techniques we explored. They have helped our system to different extents although are not the major model components.

Different loss functions. We have tried a *rank-based loss*. We implemented a modified version of Weighted Approximately Ranked Pairwise (WARP) loss [3] (HMF-WARP). It computes a high-rank flavored loss based a pseudo-rank of an item and updates parameters in a batch training manner. The detail of the loss is not in the scope of this work. It helps achieve better prediction accuracy than Cross-Entropy loss (HMF-CE). We have also tried *weighted loss*. Uniform training penalty in (3) is replace by a weighted loss that considers different interaction rewards. We times each single loss by its reward.

Model training with more users. We also try to observe the effect of training on all users and all items (HMF-all-users) as compared to the training on target users with all items (HMF-target-users) and find that the scores are not benefited by the increased training data and suffer from longer training time requirements.

4 APPROACH: CONSTRAINED RECOMMENDATION

In this section we present our approach to deal with recommendations under constraints. We propose a bi-directional recommendation model with matching to provide constrained recommendations that balance the user and item preferences across all target users and items.

4.1 Overview

Constraints impose a maximum limit on the number of times users and items can appear in the recommendation pairs. This poses additional challenges to recommendation problem. We can easily deal with a single constraint on the maximum number of users that can be recommended for a given item (i.e., m, the maximum number of times a single item can appear), as in the offline challenge. In such situations, we can limit the number of recommended users for the item by their recommendation ranks. However, the problem becomes more challenging when we additionally impose a limit on the number of times a user can appear across the recommendations for items (i.e., n, the maximum number of times a single user can appear). In this situation, if the user appears multiple times across the recommendations of different items, there is no clear way of deciding for which items to retain the user without exceeding the user appearance limit.

One strategy to decide would be to choose arbitrarily between the items. Another strategy is to greedily discard items from the set of competing items based on the ranks of the user in the competing items' recommendations. However both these solutions will result in suboptimal pairings. The rank based strategy suffers from the fact that user ranks are not normalized across items during model training. Our solution overcomes tries to overcome this problem by using an optimal matching strategy.

Our solution to this constrained recommendation problem consists of two components 1) a bi-directional recommendation model, and 2) a matching algorithm as post-processing to find optimal recommendation pairs. We train two instances of the recommendation model, one to provide user recommendations and the other to provide item recommendations. The matching algorithm uses these ranked recommendation lists as user and item preference lists, in order to find the best matches across all pairs of users and items, under the imposed constraints. Other greedy or arbitrary strategies to form pairings would most often result in suboptimal pairings.

4.2 Stable Matching Algorithm

We first provide a brief review of the Gale-Shapely stable matching algorithm [1], followed by our extension to the algorithm for the constrained recommendation problem in the following subsection 4.3. The stable matching algorithm finds an optimal matching between users and items by considering their preference lists. Each user ranks the items according to his preferences. Similarly each item ranks the users by its preferences. In each iteration of the algorithm, a user makes a proposal to an item. If the item is available, they get paired. If the item is already paired, then the item's ranking for the two users is used to decide which user the item should be paired with. The user that is left unpaired can continue to make proposals to items in decreasing order of his preferences until he finds a partner. The conditional pairings can be broken based on the item preferences, and the user's proposals are ordered by the user's preferences. This ensures an optimal non-greedy strategy to find a stable matching between the users and items based on user and item preferences across all users and items.

Table 2: Data statistics.

	target users	all users	target items	all items
counts	75K	1.5M	46K	1.3M

4.3 Extension to Stable Matching Algorithm

We propose extensions to the Gale-Shapley stable matching algorithm to address the constrained recommendation problem.

Preference Lists. First, we consider the user and item preferences as the ranked recommendation lists generated by the bi-directional recommendation model.

Generalization to replicated instances. Second, we generalize the matching problem to allow pairing of multiple users with an item and multiple items with a user. For this purpose we handle replicated instances of users and items that share preferences. The constraints dictate the number of replicated instances required. For instance if the limit on user appearance in recommendation pairs is N, then each user manifests as N replicated instances. Considering that the users make proposals to items, replicating the users to multiple instances is trivial. However, we need to handle the replication of item instances differently. We tie together the replicated items to the original item. So the proposals are still made to the original item. If any replicated instance of that item is available, we pair it to the user instance that made the proposal. If no instance of that item is available, we compare and break pairings of the item's least preferred paired user, across all its replicated instances, in order to get better pairings.

4.4 Bi-directional Recommendation Model

We describe the User-Item Model (u-i) as an instance of the HMF model described in section 3.1 to recommend items to users. Similarly, the Item-User Model (i-u) is used to recommend users to items. We can use the User-Item model to generate ranked recommendation lists of items for the target users and the Item-User model to generate ranked recommendation lists of users for the target items. The matching algorithm will then decide the optimal set of constrained recommendation pairs based on the ranked lists generated by the two models.

5 EXPERIMENTS

In this section we first describe our experimental setup and then report the evaluation results on our approaches.

5.1 Setup

Dataset. In both offline and online rounds, data contains a large set of users and items as well as a subset of users and items as targets. The data statistics for the offline dataset are shown in Table 2. We report interaction distribution over the different types for the offline dataset in Table 3. Note that we do not use impression (Imp) data.

Table 3: Interaction distribution in offline round data. BM: bookmark; Rep: reply; RI: recruiter interests; Del: delete; Imp: impression.

types	click	BM	Rep	RI	Del	Imp
counts	6.9M	282K	118K	100K	907K	314M

Table 4: Results compared to baseline models. "Score" is leaderboard score and LS-x are local evaluation detailed scores.

model	score	LS-4	LS-3	LS-2	LS-1
Active	1316	2725	2755	1900	1930
XGBoost	10004	NA	NA	NA	NA
Ours (7th place)	50069	56457	56457	13907	13907

Local evaluation. In addition to leaderboard score (score), we also split the last two weeks in the training data as validation weeks to perform local evaluations. We select items from these two weeks that have not appeared in the previous weeks as target items. These items are regarded as "new" items in our local evaluation.

We use the scoring metric in the challenge and break down to four local scores (LS): LS-1: only consider user success without deletion penalty; LS-2: consider user success with deletion penalty; LS-3: consider user-success and item-success scores without deletion penalty; LS-4: consider user-success and item-success scores and deletion penalty.

In online round, we do receive interaction feedback of users daily. However, we find it unreliable to treat those daily interactions as ground truth to evaluate different models. This is because the daily interactions are feedback of users on participants' recommendation and thus highly biased. Therefore we choose to still evaluate different approaches in a similar way as in offline round.

Models Our model achieve 7th place in offline round challenge. We also compare it to two simple baselines: Active and XGBoost. **Active** simply recommends users based on their activeness and prefers users who are more active recently. **XGBoost** is provided by the organizer and uses gradient boosting trees to predict if a user would be interested in the item based on the features.

5.2 Results in cold start recommendation

5.2.1 Compared with baseline models. We first report the results of our final model in Table 4. XGBoost does significantly better than Active, which suggests that features are discriminative. Our model outperforms XGBoost baseline significantly by utilizing feature as well as user identities.

5.2.2 Compared with different identity embedding schemes. In Table 5, we compare our model variants with different identity embedding schemes. We observe "no item ids" are consistently better than "with item ids". Also, the results affirm that "user ids" are necessary to get good results.

5.2.3 Negative user filtering. In Table 6, we show the effectiveness of the heuristic of removal of "negative users". In the experiments, we simply remove users that ever have one deletion. This performance gain is observed across all our models.

Table 5: Results of our model with different identity embedding schemes. "Score" is the leaderboard score and LS-x are detailed breakdown of local evaluation scores.

item id	user id	score	LS-4	LS-3	LS-2	LS-1
no	yes	50069	56457	56457	13907	13907
no	no	26484	27892	27892	8217	8217
yes	yes	46870	52534	52534	13434	13434
yes	no	22576	22055	22055	6780	6780

Table 6: Results comparison between whether or not to remove negative users.

model	score	LS-4	LS-3	LS-2	LS-1
HMF-basic	47480	59750	76350	1350	17950
HMF-filtered	50069	56457	56457	13907	13907

Table 7: Results comparison of other techniques. "All-users" refer to training with entire user set.

model	HMF-WL	HMF-CE	HMF-WARP	all-users
score	47471	48925	50069	44100

5.2.4 Other techniques. In Table 7, we report results of several model variants. Weighted loss does not help. I believe it is due to highly peaked event distribution — most events are clicks. WARP loss function clearly helps. Training with all users actually hurts. Probably it distracts the model from focusing on target users.

5.3 Results in constrained recommendation

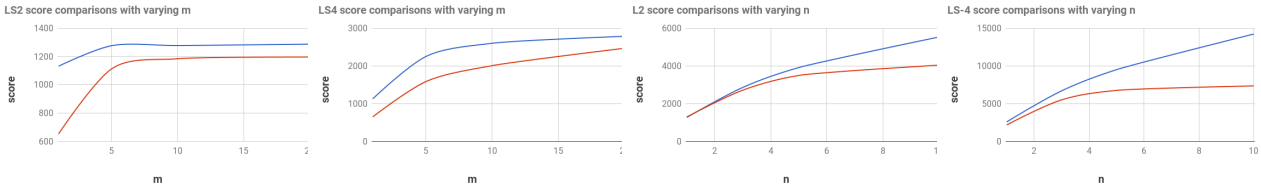
To validate our algorithm for constrained recommendation, we compare our model (u-i matching) to two post-processing strategies mentioned in section 4.1 . We use the HMF model that we used in the offline round as the underlying item-user model (i-u model) for these strategies.

The first strategy (i-u rand) we compare with, chooses arbitrarily between the competing items when satisfying constraints on the user appearance limits. i-u rand takes recommendation from the i-u model and chooses arbitrarily between the items when more than one item chooses the user. The second strategy (i-u rank) we compare with, greedily discards items from the set of competing items based on the ranks of the user in the competing items' recommendations generated from the i-u model.

The results are shown in Table 8. We report LS-2 and LS-4 on these three models. First, i-u rank is significantly better than i-u rand, which suggests that user ranks do contain preference information although they are not normalized across different items during training. Second, u-i matching outperforms i-u rank. The improvement are significant on LS-4, which verifies that u-i matching better balances interests of users and items by taking preferences of both sides into account.

5.3.1 Results with varying constraints. We further analyze the algorithm by experimenting with varying value of m, i.e., the maximum allowed number of items in all recommendations, and n, i.e., the maximum allowed number of users. We want to demonstrate

Figure 1: Constrained recommendation. Local evaluation score (LS-2 and LS-4) comparisons as constraints change. All blue curves represent u-i matching model. In (a) and (b), n = 1 and red curves represent u-i model without matching; In (c) and (d) m = 10 and red curves represent the i-u rank model.



(a) LS-2 scores, n=1 (b) LS-4 scores, n=1 (c) LS-2 scores, m=10 (d) LS-4 scores, m=10

Table 8: Recommendation under constraints. Result comparisons for different strategies. i-u rand assigns items randomly; i-u rank assigns items based on user rank, u-i matching is based on optimal matching strategy

model	i-u rand	i-u rank	u-i matching
LS-2	231	1306	**1314**
LS-4	506	2206	**3089**

how the matching algorithm works when constraints change and also compare it to other models under varying constraints.

First we compare our u-i matching model with a pure u-i model that does not use i-u model recommendations and hence has no information about the items' preferences. This would give us an idea whether the recommendations benefit from the use of both u-i and i-u models taken together. For the experiments, we fix n to be 1 and vary m. When m is large, the constraint is loose; when m is small, the constraint is tight. The results are shown in Fig 1(a) and 1(b). We observe consistent improvements for the u-i matching over the pure u-i model. The improvement increases as m becomes smaller (the constraint becomes tighter). It suggests the model benefits more from i-u model instance when the constraint is tight.

Second, we compare u-i matching with i-u rank model. i-u rank performs decently when n=1 and handles m constraint easily (by cutting off recommendations beyond m). Now, we fix m to be 10 and vary n. When n is 1, the constraint is trivial; when n is greater than 1, i-u rank needs to maintain the best n user ranks (and items). As seen in Fig 1(c) and 1(d), u-i matching clearly outperforms i-u rank. The improvement becomes more significant as n becomes larger.

6 CONCLUSION AND DISCUSSION

In this work we present a job recommender system that works in cold start and constrained situations. We base our model on hybrid matrix factorization that jointly models contents and observed interactions. To deal with the constraints on the number of times items and users can appear in the recommendation pairs, we propose a novel algorithm that takes into account preferences from both users and items.

Our experiments show the matching algorithm is more effective when constraints are tight (and non-trivial). In the online round of the challenge, we have n = 1 and m = 250, which does not fully exercise our model's capability since m = 250 might be too loose a constraint and n = 1 is trivial.

The current algorithm is a two stage approach which consists of separate recommendation and matching step. In future work, we aim to develop models that combine the two steps as one. Meanwhile, the matching algorithm complexity depends on the number of items (or users) and m and n. This can be high in large scale systems. We would like to look for more scalable alternatives.

REFERENCES

[1] David Gale and Lloyd S Shapley. 1962. College admissions and the stability of marriage. *The American Mathematical Monthly* 69, 1 (1962), 9–15.
[2] Maciej Kula. 2015. Metadata Embeddings for User and Item Cold-start Recommendations. *arXiv preprint arXiv:1507.08439* (2015). http://arxiv.org/abs/1507.08439
[3] Nicolas Usunier, David Buffoni, and Patrick Gallinari. 2009. Ranking with ordered weighted pairwise classification. In *Proceedings of the 26th annual international conference on machine learning*. ACM, 1057–1064.