Practical Lessons for Job Recommendations in the Cold-Start Scenario

Jianxun Lian
University of Science and Technology of China
Hefei, China
jianxun.lian@outlook.com

Fuzheng Zhang
Microsoft Research
Beijing, China
fuzzhang@microsoft.com

Min Hou
University of Science and Technology of China
Hefei, China
hmhoumin@gmail.com

Hongwei Wang
Shanghai Jiao Tong University
Shanghai, China
wanghongwei55@gmail.com

Xing Xie
Microsoft Research
Beijing, China
xingx@microsoft.com

Guangzhong Sun
University of Science and Technology of China
gzsun@ustc.edu.cn

ABSTRACT

The 2017 ACM RecSys Challenge focuses on the problem of job recommendations on XING in a cold-start scenario. In this paper we describe our solution as well as some practical lessons learned from the competition. We model this task as a binary classification problem. Negative candidate selection is the first key phase in our solution. We design a negative sampling strategy which performs significantly better than taking users' deleted or unclicked items as negative candidates. We then extract comprehensive features to model the relationship between a user-job candidate, including the direct profile similarity between the user and the job, and the profile similarity between the user's historical interested jobs and the target job. To make the whole pipeline scalable and easy to deploy online, we decide to use a single boosting tree model as the final discriminative model, instead of using a stacking ensemble of multiple models. Overall our model ranked 5th on the challenge leaderboard, and our last model has remained in 2nd place during the last two online weeks. We have open-sourced our implementation on https://github.com/Leavingseason/RecsysChallenge2017.

CCS CONCEPTS

• **Information systems → Recommender systems**;

KEYWORDS

job recommendations, cold start, recsys challenge 2017, recommendation systems, content-based filtering

ACM Reference format:
Jianxun Lian, Fuzheng Zhang, Min Hou, Hongwei Wang, Xing Xie, and Guangzhong Sun. 2017. Practical Lessons for Job Recommendations in the Cold-Start Scenario. In *Proceedings of RecSys Challenge '17, Como, Italy, August 27, 2017,* 6 pages.
https://doi.org/10.1145/3124791.3124794

RecSys Challenge '17, August 27, 2017, Como, Italy

ACM ISBN 978-1-4503-5391-5/17/08…$15.00
https://doi.org/10.1145/3124791.3124794

1 INTRODUCTION

The ACM RecSys challenge 2017 [4] focuses on the problem of job recommendations on XING. XING is a social network for business, has attracted more than 18 million users and typically has around 1 million active job posting on the platform. In order to better connect job seekers and recruiters, the RecSys Challenge 2017 asks participants to build recommender systems to solve the following task: given a new job posting, identify those users (a) who may be interested in receiving the job posting as a push recommendation and (b) who that are also appropriate candidates for the given job.

The job recommendation task is very close to traditional CTR prediction tasks such as ad click prediction or app download prediction, with a common key component being to estimate the probability that a user will click on the target item. However, one special thing to job recommendation is that we need to model not only the preference of the user over the item, but also the relevance between the user and the item. For example, some popular apps such as WeChat and Office 365 may be liked and downloaded by many people, while normally almost no jobs can be applied commonly to an overwhelming majority of people. This is due to a job seeker only being interested in the jobs for which he/she is qualified. So the first key step is to model both relevance and preference of a user-item pair. In this paper, we introduce a simple, efficient and effective candidate sampling method to handle this problem, and in the discussion section we further propose an integrated model that can incorporate both relevance and preference.

Just like some related CTR tasks [3, 6, 12, 18], we formulate the job recommendation as a binary classification problem, where a positive label indicates that the user has ever clicked on, bookmarked, or replied to the item, and a negative label indicates the user has deleted or no taken action on the item. Not surprisingly, the most important part is feature engineering. Although some research works [5, 15] aim to develop models which can be trained end-to-end without feature engineering, for this job recommendation task, the use of elaborately crafted features can achieve far better scores. We split the features into three pillars: (1) features derived from the similarity between the user's profile and the target item's profile; (2)features derived from the similarity between the user's historical interested items and the target item; (3) features using words and tags directly from profiles. We find that the most important features come from the second pillar, and it is in line

Table 1: Fields in the user profile file.

field	type	comments
id	categorical	anonymized ID of the user
job roles	bag of words	list of jobrole terms that were extracted from the user's current job titles
career level	categorical	career level ID , e.g. Beginner or Experienced
discipline	categorical	anonymized IDs represent disciplines such as "Consulting", "HR", etc.
industry	categorical	anonymized IDs represent industries such as "Internet", "Automotive", etc.
country	categorical	describes the country in which the user is currently working.
region	categorical	specified for some users who was in Germany.
experience_01	numerical	the number of CV entries that the user has listed as work experiences
experience_02	numerical	the estimated number of years of work experience that the user has
experience_03	numerical	the estimated number of years that the user is already working in her current job
edu degree	categorical	estimated university degree of the user
edu fields	bag of words	fields of studies that the user studied
wtcj	categorical	estimation regarding the user's willingness to change job
premium	categorical	the user subscribed to XING's payed premium membership

with the intuition: for the cold-start recommendation content-based filtering is the most efficient method and is essential for building a better profile via the user's historical activities.

We compare different classifiers and find that the gradient boosting decision tree (GBDT) model is the best one, with is inline with most of the winning solutions coming from various data mining competitions [8, 10, 16]. Blending the results of multiple models, known as model ensemble, usually further improves scores. However, given that it is necessary for online systems to keep simple and efficient for good scalability and maintainability, we decide to use a single GBDT model through both the offline and online period without model ensemble. Our initial model ranked 6th in the first two weeks of the online challenge, and after we updated our model, we ranked 2nd in the last two consecutive weeks. On average we ranked 5th on the leaderboard.

2 PROBLEM STATEMENT

The 2017 Recsys Challenge [4] consists of an offline phase and an online phase. For the offline phase, target users and items are fixed, participants are asked to recommend no more than 100 users for each target item, the ground-truth labels are made from the historical activities. For the online phase, each team will receive a new group of target users and items daily, and the recommendation submitted by the teams will actually be rolled out to real users on XING's live system. The training dataset includes users' profiles, items' profiles, and historical transactions between users and items. The details of the user profiles are listed in Table 1. Most of the fields in the item profiles are the same as for the fields user profiles, except that items have additional latitude and longitude information for location tagging. The interaction file has four fields, including user id, item id, timestamp, and the type of interaction.

Typically, user response prediction can be regarded as binary classification or ranking problem. In this paper we mainly discuss binary classification models, while we also compare them with one ranking model. A positive label indicates that a user has some explicit positive action on an item, such as clicks or replies; a negative label indicates that a user neglects or explicitly deletes an item.

Evaluation metrics. The official evaluation metrics [1] are calculated according to the user/item's premium level and the interaction type. For better demonstration and comparison of various models, we use the following evaluation metrics throughout the paper:

(1) reward. This is the official evaluation metrics. We predict 10% of the offline test dataset (selected by $item\_id\ mod\ 10 = 1$) and submit it to the platform for judgement[2].
(2) reward@1. Since the online phase requires that each target user can at most receive one recommendation, we follow this constraint and only keep the top 1 prediction for each user.
(3) p@1. Success user rate (which we denote as *precision* in this paper) at top 1 predictions. Calculated by $\frac{|success(user@1)|}{|users|}$, where $success(user@1)$ is the set of users that have positive interactions on the top 1 prediction.
(4) p@5. Success user rate at top 5 predictions. Calculated by $\frac{|success(user@5)|}{|users|}$.
(5) AUC. Area under the ROC curve.

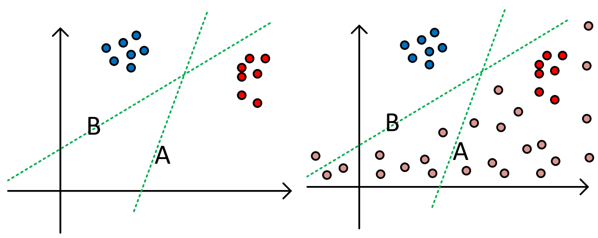
We reserve 1000 items from the training set as the validation set to simulate the cold-start scenario. *reward* and *reward@1* are evaluated on 10% of the official offline test dataset, while *p@1*, *p@5* and *AUC* are evaluated on our validation set.

3 NEGATIVE CANDIDATES SELECTION

We find that for job recommendation, the selection of negative candidates is highly important. We start by using the items deleted by the user or receive no interaction after impression as negative candidates. We find the model trained based on these training instances yields very poor performance as shown in the *negative action* row in Table 2. Although the performance can be improved by adding a filter to remove those user-job pairs which do not have common words in the user's and item's titles, the scores (as shown in the *negative action filtered* row) is still far less than a simple random sampling of negative candidates (as shown in the *random small* row). It is not hard to understand the cause: every historical

[1] The details can be found here: http://2017.recsyschallenge.com
[2] The offline submission entrance is still open for experimental evaluation.

item shown to the user is proposed by XING's recommender system, which means the item is at least relevant to the user to some extent. If all the training data come from the user's historical interaction logs provided by XING, the trained model is biased to the user's relevant area. However, in the real world prediction step, we are dealing with an open target item set, which contains far more items that are not relevant to the user. The trained model does not have knowledge in this new area, thus it will produce a lot of false positive cases. We further illustrate this with Figure 1. To sum it up, the classifier usually fails to work if the data distribution of the training and the test sets are different.



(a) distribution of training data **(b) distribution of test data**

Figure 1: Illustration of a classifier failing to work when distributions of training data and test data are different. In the left figure, a max-margin classifier will choose A rather than B as the optimal discriminative plane. However, in the real test set, there are many hidden irrelevant data points, which make B a better splitting plane.

Algorithm 1 Wide & Deep Negative Sampling

Require: user set $\mathbf{U}$, item set $\mathbf{V}$, and interactions $\mathbf{I}$.
Ensure: training candidates $\mathbf{D}$.
$\mathbf{D} \leftarrow \emptyset$
Add pairs from $\mathbf{I}$ with activity {1,2,3,5} to $\mathbf{D}$ as positive candidates
Add pairs from $\mathbf{I}$ with activity {4} to $\mathbf{D}$ as negative candidates
%% the wide part
$\mathbf{V}_1 \longleftarrow$ sample 100000 items from $\mathbf{V}$
for each item $v \in \mathbf{V}_1$ **do**
  for $i = 0 \longrightarrow 500$ **do**
    sample one user $u \in \mathbf{U}$
    Add $< u, v >$ to $\mathbf{D}$ as negative candidate if not exists
  end for
end for
%% the deep part
$\mathbf{V}_2 \longleftarrow$ sample 1000 items from $\mathbf{V}$
for each item $v \in \mathbf{V}_2$ **do**
  for $i = 0 \longrightarrow 30000$ **do**
    sample one user $u \in \mathbf{U}$
    Add $< u, v >$ to $\mathbf{D}$ as negative candidate if not exists
  end for
end for
return $\mathbf{D}$

Since random sampling performs well, we ask two follow up questions: (1) how many negative candidates are enough for sampling? (2) Is there a strategy better than the brute-force sampling over the complete $N \times M$ space? Question (1) is related to the class imbalance problem. Usually we can train a better model if we collect more instances. However, since we have a limited number of positive instances, the increase of negative instances aggravates the unbalance between positive and negative labels. We propose an effective and efficient negative sampling algorithm as shown in Algorithm 1, in order to control the total number of negative instances while keeping as much information as possible in the open space. It consists two parts: the wide part and the deep part. The wide part aims to cover a large number of items, while each item is only connected to a small number of users; the deep part aims to explore exhaustive candidates for one item. The total number of sampled negative candidates is 80 million. To demonstrate the necessity of both the deep and the wide parts, we sample negative candidates using only wide part or deep part, while keeping the total number of negative candidates at 80 million. The performance comparison is shown in Table 2. We find that by using both the wide and deep part we can get the best model.

4 FEATURE EXPLORATION

4.1 Feature Engineering

Our entire feature set can be split into three pillars: profile matching, historical matching, and bag of words.

4.1.1 Profile Matching. Job relevance can be measured by comparing a user's profile and an item's profile. The numerical and categorical fields in the user's profile and the item's profile are used directly as features, such as discipline, industry, number of words in job roles. Next we use some boolean variables to record whether the corresponding field of user's profile and item's profile matches, e.g. the user's industry v.s. the item's industry, and the user's country v.s. the item's country. We further intersect some fields from the user's profile and the item's profile, considering that using each base field separately may not be enough to explain the relationship. For example, intersecting the user's *career_level* field with the item's *career_level* field will yield a new categorical column with $n \times m$ possible values, where n, m stands for the possible values for the user's *career_level* field and the item's *career_level* field, respectively.

4.1.2 Historical Matching. The user's profile is filled explicitly by the user. Actually, a fixed number of fields is not enough to introduce a user exhaustively. What is more, sometimes the user does not enter many messages in the online platform and his/her profile is very brief, or some fields are even fake. By contrast, the user's historical activities are an ideal implicit supplement to the profile, which is why we try to build the user's secondary profile from his/her historical interacted items. We design several numerical variables by matching the user's secondary profile and the target item's profile, covering title similarity, industry similarity, career level similarity, employment similarity, and location similarity. Typically, *similarity* here means the percentage of items in the secondary profile that share the same corresponding field with the target item. For location similarity, we additionally calculate

Table 2: Evaluation of different negative candidates selection methods. The classification model is GBDT.

method	reward	reward@1	p@1	p@5	AUC
negative action	1502	8	0.0050	0.0193	0.6659
negative action (filtered)	2474	430	0.1546	0.2978	0.6128
random (small)	5089	1805	0.4148	0.6572	0.9656
random(small) + negative action	3015	1205	0.2648	0.4172	0.8686
random (wide)	6013	2804	0.4755	0.7056	0.9736
random (deep)	5025	945	0.1251	0.3030	0.9291
random (wide & deep)	**7286**	**3005**	**0.6419**	**0.8264**	**0.9825**

Table 3: Evaluation of different feature set. The classification model is GBDT.

feature set	reward	change	reward@1	p@1	p@5	AUC
ALL	**7286**	-	**3005**	**0.6419**	**0.8264**	**0.9825**
- profile	5999	↓ 17.6%	1795	0.4428	0.6363	0.9479
- history	1883	↓ 74.2%	848	0.2816	0.5391	0.9190
- BOW	6790	↓ 6.81%	2919	0.5347	0.7501	0.9764
- car & loc	5603	↓ 23.1%	2306	0.5195	0.7453	0.9724
- location	6640	↓ 8.87%	2734	0.5421	0.7573	0.9758

the min/avg/max distance between the target item and items in the user's secondary profile via latitude and longitude.

4.1.3 Bag of Words. Up to now we have compacted the profile into some numerical variables to describe the similarity. Usually it is necessary to treat the profile as a document and learn its latent representation to retain as much information as possible. Some popular methods are latent topic models [1] and deep learning techniques [17]. However, due to limitations, we do not have enough time to build and tune parameters for these models, so we adopt the most simple bag-of-words feature to retain the raw document. To reduce dimension, we only use the top 20k most frequent words.

4.2 Feature Evaluation

To study the importance of each feature pillar, we remove one of them and see how the performance changes. Results are shown in Table 3, where *- profile* indicates removal of the entire *profile matching* pillar; *- history* indicates removal of the entire *historical matching* pillar; *- BOW* indicates removal of the entire *bag of words* pillar. We can observe that three pillars contribute to the best model, and the *historical matching* features are the most important features, without which the performance drops severely by 74.2%. Thus we are curious to further explore the features within the *historical matching* pillar. Row *- car & loc* in Table 3 means for the *historical matching* pillar we only keep similarity variables extracted from titles and tags, and remove those similarity variables extracted from career level, employment, and location. Row *- location* in Table 3 indicates that we exclude all location related variables in the *historical matching* pillar. By doing this we want to verify whether only one or two historical fields are enough to explain the data. Results demonstrate that every field is important, and the more fields we have in the profile, the more precise the model can be.

Table 4 lists the top 10 most important features ranked by GBDT. The vast majority of the top features come from the *historical matching* pillar, which again demonstrates that implicit profiles better describe the user. The best feature is *sum_clicked_item_sim*, which means the sum of similarities between the user's clicked items and the target item. Intuitively, the more similar the user's historical interested items are with the target item, the more likely it is that the user will click on the target item.

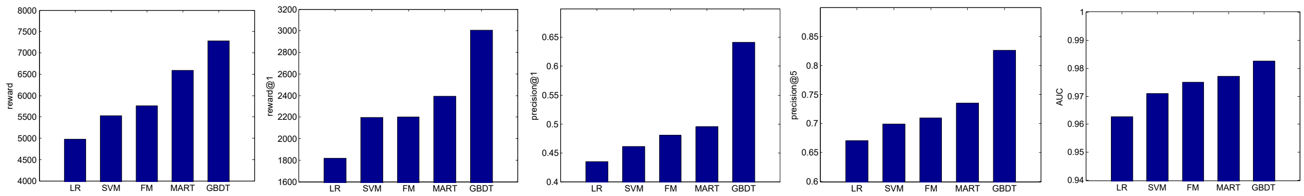
Table 4: Feature importance from GBDT.

feature	importance	pillar
sum_clicked_item_sim	1	historical matching
nearest_clicked_item	0.34	historical matching
avg_city_match	0.32	historical matching
sum_city_match	0.31	historical matching
nearest_clicked_item	0.28	historical matching
avg_clicked_item_sim	0.21	historical matching
region_match	0.21	profile matching
sum_distance	0.19	historical matching
item_is_paid	0.19	profile matching
tag_similarity	0.12	profile matching

5 MODEL SECTION

With all features, we train models using logistic regression (LR), support vector machine (SVM), factorization machine (FM), gradient boosting decision trees (GBDT), and LambdaMART. LR, SVM, FM, and GBDT are classification model, while LambdaMART [2] is a ranking model. We find the best parameters for each model using grid-searching and report their best scores. Figure 2 shows that under all evaluation metrics, GBDT performs best. Another interesting observation is that although GBDT generates only slightly improvement over the other models in terms of AUC, it performs far better in terms of reward@1 and precision@1. Reward@1 and P@1 are actually more realistic metrics because real users usually view only a small number of items.

We also try two ways ensembling several models to further improve performance. The first way is blending the best output from LR/SVM/FM/MART/GBDT by harmonic average or stacking ensemble. The other way is to train GBDT with different parameters and different bags of features, and then blend the results.



(a) reward (b) reward@1 (c) Precision@1 (d) Precision@5 (e) AUC

Figure 2: Performance comparison among logistic regression (LR), support vector machine (SVM), factorization machine (FM), LambdaMARK (MARK), and gradient boosted decision tree (GBDT).

Table 5: Results of complexity reduction.

	reward@1	AUC	test size	test time
initial	3005	0.9825	4 TB	2.6 h
LR preprocess	2998	0.9824	250 GB	0.2 h

We can get about 1.5% improvement through the model ensemble. However, this step makes the running pipeline more complex and time-consuming. To make the pipeline efficient, we decide to give up model ensemble for the challenge.

6 COMPLEXITY REDUCTION

In Section 3 we generate about 80 million candidates for training set, and the entire test set for the offline stage contains 46k $\times$ 74k possible candidates. After extracting features the size of test file (feature vectors) is about 4 TB, and the training and test processes together cost about 12 hours. The data is so huge that we are curious about reducing the data volume. We notice that in fact many items proposed by Algorithm 1 are not relevant to the user. Inspired by [8], we decide to use a weak classification model to filter out low quality candidates. Since we have found that LR is the relatively weakest learner and GBDT is the best one, we use LR to filter instances with probability lower than 0.01 in both the training set and test set, then train a GBDT model using the filtered training set. Finally we make prediction on the filtered test set. Table 5 indicates that with LR preprocessing, there is no significant loss in evaluation metrics, while the file size and running time are reduced significantly.

7 RESULTS OF CHALLENGE

Since the final leaderboard is evaluated according to the online stage, here we only report our growth history in the online stage[3]. At the first two weeks we only use the model without historical career and location similarity features (as shown in row - *car & loc* of Table 3), and we manually start the programs and submit the prediction file. We soon realize that manually repeat the whole pipeline is inefficient and easy to generate bugs (such as forget to update a parameter). Thus we decide to build an automatic pipeline to daily pull data via API, extract features, make predictions, and then submit the prediction file. This automatic pipeline make us able to submit files 12 hours earlier than the manually way we used before. We enabled the pipeline in week 3, along with some new features related to historical career matching. As shown in Figure

[3]Until the time we submit this paper, our best single GBDT model ranks 3rd in the latest offline leaderboard with score 62110.

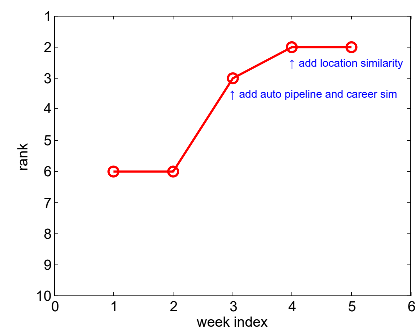


Figure 3: The trend chart of our rank during online phase.

3 our rank increased to 3rd that week. We further added historical location matching features in week 4, and our model remained ranked 2nd since we used the latest model.

8 DISCUSSION

In Section 3 we have demonstrated that random sampling negative candidates performs far better than using the user's none-click impressions as negative candidates. However, removing the latter may to some extent cause information loss. If we regard the user's clicked items as positive candidates, random sampling items as negative candidates, and the user's none-click impressions as negative ranking candidates (for which we do not know their true label, however, we know that their score should not be higher than positive candidates), now we can incorporate classification and ranking into one unified model, which we call *relevance & preference model*. Let $\mathbf{D}_+$ denotes the positive candidates in $\mathbf{D}$ from Section 3, $\mathbf{D}_-$ denotes the negative candidates in $\mathbf{D}$, $\mathbf{R}$ denotes the interactions history, and $\mathbf{\Theta}$ denotes the model parameters. We want to maximize the following posterior:

$$Pr(\mathbf{\Theta}; \mathbf{D}_+, \mathbf{D}_-, \mathbf{R}) = P(\mathbf{D}_+|\mathbf{\Theta})P(\mathbf{D}_-|\mathbf{\Theta})P(\mathbf{R}|\mathbf{\Theta})P(\mathbf{\Theta}) \tag{1}$$

where $P(\mathbf{D}_+|\mathbf{\Theta})$ classifies positive candidates:

$$P(\mathbf{D}_+|\mathbf{\Theta}) = \prod_{i \in \mathbf{D}_+} f(i|\mathbf{\Theta}) \tag{2}$$

and $P(\mathbf{D}_-|\mathbf{\Theta})$ classifies negative candidates:

$$P(\mathbf{D}_-|\mathbf{\Theta}) = \prod_{i \in \mathbf{D}_-} (1 - f(i|\mathbf{\Theta})) \tag{3}$$

Table 6: Score comparison between the *relevance & preference* model and the base LR model.

	P@1	P@5	AUC
LR	0.3541	0.4801	0.8672
relevance & preference	0.3581	0.4991	0.8683

and $P(\mathbf{R}|\Theta)$ models the preference among the user's interaction history, c is a hyper-parameter controlling the weight of the preference module:

$$P(\mathbf{R}|\Theta) = \prod_{(+,-)\in\mathbf{R}} \frac{c}{1 + e^{-(f_+ - f_-)}} \tag{4}$$

$f(,;\Theta)$ can be arbitrary discriminative learner. Due to time limit, we only implement it with LR and conduct experiments over a small subset. Table 6 demonstrates that the proposed *relevance & preference* model is promising. In the future we will implement the model with GBDT and conduct experiment in the complete dataset.

9 RELATED WORKS

Compared to the 2016 Recsys Challenge, the 2017 Recsys Challenge [4] is more focused on online scenarios. All recommendations submitted by teams are pushed to real users daily, which requires us to build efficient pipeline for making timely recommendations. Different from the winning solutions from last year [13, 16, 19], our pipeline is simple and does not include any model ensemble, which make the online logic efficient and easy to maintain.

The most popular method for recommender systems is collaborative filtering (CF). However, since the 2017 Recsys Challenge aims at making recommendations for new items, which is known as the cold-start problem, CF is not applicable in this task. Thus we have to abandon a lot of good models related to CF [7, 9, 17], and mainly consider models with content-based filtering [11, 14]. We find the job recommendation task is very close to click-through rate prediction tasks, where the most popular method is to model it as a binary classification problem [6, 12]. Generalized linear algorithms such as FTRL [12] proves efficient and effective in practice, and we also exploit it for the LR and FM experiments. Recently, some researchers have tried to enhance the non-linear ability of the models with deep neural networks [5, 15, 18]. However, these deep learning-based models require more effort on training (e.g., parameter tuning and pre-training), and at the same time they are still not as fast as the traditional versions which do not use neural networks.

10 CONCLUSIONS

In this paper, we introduce our pipeline for the 2017 Recsys Challenge [4]. There are mainly three key components, i.e. negative candidates selection, feature engineering, and model selection. We have demonstrated that in each component there are some elaborate designs which improve performance significantly. For cold-start jobs recommendation, the user's historical activities are the best features for profiling the user. We also propose a unified model to incorporate both relevance and preference together. For the future work, we will conduct more comprehensive experiments for the proposed relevance & preference model.

REFERENCES

[1] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.
[2] Chris J.C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview.* Technical Report. https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/
[3] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems.* ACM, 7–10.
[4] Abel Fabian, Yashar Deldjoo, Mehdi Elahi, and Daniel Kohlsdorf. 2017. RecSys Challenge 2017: Offline and Online Evaluation. In *Proceedings of the 11th ACM Conference on Recommender Systems (RecSys '17).* ACM, Como, ITALY, 2.
[5] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. *arXiv preprint arXiv:1703.04247* (2017).
[6] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising.* ACM, 1–9.
[7] Yehuda Koren. 2008. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08).* ACM, New York, NY, USA, 426–434. https://doi.org/10.1145/1401890.1401944
[8] Jianxun Lian and Xing Xie. 2016. Cross-Device User Matching Based on Massive Browse Logs: The Runner-Up Solution for the 2016 CIKM Cup. *arXiv preprint arXiv:1610.03928* (2016).
[9] Jianxun Lian, Fuzheng Zhang, Xing Xie, and Guangzhong Sun. 2017. CCCFNet: A Content-Boosted Collaborative Filtering Neural Network for Cross Domain Recommender Systems. In *Proceedings of the 26th International Conference on World Wide Web Companion, Perth, Australia, April 3-7, 2017.* 817–818. https://doi.org/10.1145/3041021.3054207
[10] Guimei Liu, Tam T. Nguyen, Gang Zhao, Wei Zha, Jianbo Yang, Jianneng Cao, Min Wu, Peilin Zhao, and Wei Chen. 2016. Repeat Buyer Prediction for E-Commerce. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16).* ACM, New York, NY, USA, 155–164. https://doi.org/10.1145/2939672.2939674
[11] Zhongqi Lu, Zhicheng Dou, Jianxun Lian, Xing Xie, and Qiang Yang. 2015. Content-based Collaborative Filtering for News Topic Recommendation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI'15).* AAAI Press, 217–223. http://dl.acm.org/citation.cfm?id=2887007.2887038
[12] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 1222–1230.
[13] Andrzej Pacuk, Piotr Sankowski, Karol Wegrzycki, Adam Witkowski, and Piotr Wygocki. 2016. RecSys Challenge 2016: Job Recommendations Based on Preselection of Offers and Gradient Boosting. In *Proceedings of the Recommender Systems Challenge (RecSys Challenge '16).* ACM, New York, NY, USA, Article 10, 4 pages. https://doi.org/10.1145/2987538.2987544
[14] Seung-Taek Park and Wei Chu. 2009. Pairwise Preference Regression for Cold-start Recommendation. In *Proceedings of the Third ACM Conference on Recommender Systems (RecSys '09).* ACM, New York, NY, USA, 21–28. https://doi.org/10.1145/1639714.1639720
[15] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. *arXiv preprint arXiv:1611.00144* (2016).
[16] Wenming Xiao, Xiao Xu, Kang Liang, Junkang Mao, and Jun Wang. 2016. Job Recommendation with Hawkes Process: An Effective Solution for RecSys Challenge 2016. In *Proceedings of the Recommender Systems Challenge (RecSys Challenge '16).* ACM, New York, NY, USA, Article 11, 4 pages. https://doi.org/10.1145/2987538.2987543
[17] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative Knowledge Base Embedding for Recommender Systems. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16).* ACM, New York, NY, USA, 353–362. https://doi.org/10.1145/2939672.2939673
[18] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep learning over multi-field categorical data. In *European Conference on Information Retrieval.* Springer, 45–57.
[19] Dávid Zibriczky. 2016. A Combination of Simple Models by Forward Predictor Selection for Job Recommendation. In *Proceedings of the Recommender Systems Challenge (RecSys Challenge '16).* ACM, New York, NY, USA, Article 9, 4 pages. https://doi.org/10.1145/2987538.2987548