Content-based Neighbor Models for Cold Start in Recommender Systems

Maksims Volkovs
layer6.ai
maks@layer6.ai

Guang Wei Yu
layer6.ai
guang@layer6.ai

Tomi Poutanen
layer6.ai
tomi@layer6.ai

ABSTRACT

Cold start remains a prominent problem in recommender systems. While rich content information is often available for both users and items few existing models can fully exploit it for personalization. Slow progress in this area can be partially attributed to the lack of publicly available benchmarks to validate and compare models. This year's ACM Recommender Systems Challenge'17 aimed to address this gap by providing a standardized framework to benchmark cold start models. The challenge organizer XING released a large scaled data collection of user-job interactions from their career oriented social network. Unlike other competitions, here the participating teams were evaluated in two phases – offline and online. Models were first evaluated on the held-out offline test set. Top models were then A/B tested in the online phase where new target users and items were released daily and recommendations were pushed into XING's live production system. In this paper we present our approach to this challenge, we used a combination of content and neighbor-based models winning both offline and online phases. Our model produced the most consistent online performance wining four of the five online weeks, and showed excellent generalization in the live A/B setting.

CCS CONCEPTS

• **Information systems → Recommender systems**;

KEYWORDS

Collaborative filtering, Cold start, Gradient boosting

ACM Reference format:
Maksims Volkovs, Guang Wei Yu, and Tomi Poutanen. 2017. Content-based Neighbor Models for Cold Start in Recommender Systems. In *Proceedings of RecSys Challenge '17, Como, Italy, August 27, 2017,* 6 pages.
https://doi.org/10.1145/3124791.3124792

1 INTRODUCTION

Popularity of online content services, e-commerce and social web has highlighted an important challenge of surfacing relevant information to consumers. Recommender systems have proven to be an effective tool for this task receiving increasingly more attention. One common approach to building accurate recommender models is collaborative filtering (CF). CF is a method of making predictions about an individual's preferences based on the preference information from other users. CF has been shown to work well across various domains [8], and many successful web-services such as Netflix, Amazon and YouTube use CF to deliver highly personalized recommendations to their users.

Despite significant progress in CF over the years a number of challenges remain. One of the important challenges is *cold start*, or the ability of the recommender system to generate accurate recommendations when no preference information is available. Common approach to cold start is to utilize user/item content information and correlate it with preferences. However, while rich content information is available in many CF applications, few approaches have been developed for this problem. Part of the reason for the slow progress is the absence of standardized benchmarks where researchers can validate and compare their models. Very few CF datasets with content information have been publicly released, and many published models only report results on proprietary data making direct comparisons impossible.

This year's ACM Recommender Systems Challenge'17 [1] was aimed at providing a framework for evaluating cold start CF models. The challenge organizer XING (European analog of LinkedIn) released a large-scale data collection of user-job interactions from their career oriented social network. The goal was to use this data to develop a recommender model and then benchmark it against other teams using a standardized evaluation procedure. Two notable aspect of the challenge were that, first, both cold stats users and items were included in the data with rich content information. All models were evaluated on cold start items only, allowing the participants to directly compare their models in this setting. Second, the challenge was partitioned into two phases. In the offline phase the models were evaluated against a held out set of interactions that occurred after the released training set. Then in the online phase top performing teams were given an opportunity to submit live recommendations daily for 35 days into XING's production system. Submitted recommendations were delivered to users in real-time and the interaction feedback from the users was released to the teams the following day. This two-phase framework allowed the participants to first tune their models on the offline test set and then A/B test them live, simulating the development process of a production-level recommender system.

2 CHALLENGE FRAMEWORK

Offline. In the offline phase of the challenge a large-scale dataset of interactions was released for a subset of 1.5M users and 1.3M items. This data included over 322M user-item interactions over a span of four months. Each interaction was generated by one of six

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

ACM ISBN 978-1-4503-5391-5/17/08…$15.00
https://doi.org/10.1145/3124791.3124792

actions: {*impression, click, bookmark, reply, delete, recruiter*}. The first five actions were user-generated, whereas the last action was from an inbound recruiter interest (e.g. click) that represented item's company. In addition to interactions, content information was also provided for both users and items. For users this data included career profile information such as education, work experience, location and current position. Similarly, for items job information was provided such as title/tags, industry, location and career level; see [1] for a detailed description of this dataset.

The goal was to use all of this data to generate top-100 user rankings for the 46K target items from a set of 74K target users. All target items were cold start simulating the production scenario for XING where newly added jobs need to be recommended to relevant users. Rankings produced by each model were evaluated against a held-out test set of interactions between target users and items. Note that the test set was not visible to the teams throughout the competition and was sampled forward in time. Teams were scored according to the following formula:

$$score = \sum_{item} \left(b(item) + \sum_{user \in \pi(item)} b(user) \times score(item, user) \right)$$

$$score(item, user) = \begin{cases} 1 & \text{if click} \\ 5 & \text{if bookmark or reply} \\ 20 & \text{if recruiter} \\ -10 & \text{if only delete} \\ 0 & \text{otherwise} \end{cases}$$

$$b(user) = \begin{cases} 2 & \text{if user is premium} \\ 1 & \text{otherwise} \end{cases}$$

$$b(item) = \begin{cases} 50 & \text{if any score(item, user) > 0 and item is paid} \\ 25 & \text{if any score(item, user) > 0 and item is not paid} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where $\pi(item)$ is the top-100 user ranking for $item$. For each recommended user-item pair the score is based on the interaction type with click receiving a score of 1, bookmark or reply score of 5, and recruiter score of 20. When the only interaction is delete model is penalized with a negative score of -10. In addition, there are two boosts one for correctly recommending items to premium users with paid accounts; and another for correctly recommending at least one user to a paid item where the posting company paid to promote the item. The scoring objective thus encourages recommendations that result in stronger interactions such as bookmark and reply, and gives priority to paid users and items to ensure that the business objectives of XING are met.

Online. In the online phase similar training dataset was released for a new set of users and items. There were a total of 1M users, 853K items and over 92M interactions. Each team received daily batches of target users and items, and was asked to recommend relevant target items to target users with the constraints that (1) maximum one item can be recommended to each user and (2) each item can be recommended to at most 250 users. Target users were always sampled from the 1M training user set but all target items were new and cold start. Upon receiving recommendations XING incorporated them into its live system sending push notifications and recording resulting interactions. This feedback was released to the teams the following day together with a new batch of target users and items. The scoring function for this phase was the same as in the offline phase and scores were updated daily for each team. Full dataset statistics for both offline and online phases are shown in Table 1.

	Offline	Online
users	1,497,020	1,000,000
items	1,306,054	853,058
target users	74,840	variable
targets items	46,559	variable
interactions	322,776,002	92,949,361
impression	314,501,101	88,719,060
click	6,867,579	3,552,973
bookmark	281,672	184,061
reply	117,843	96,907
delete	906,836	360,645
recruiter	100,971	35,715

Table 1: Dataset statistics.

3 OUR APPROACH

In this section we describe our model together with learning and inference procedures. Given that all target items in both online and offline phases were cold start, we opted to develop a content-based recommender that also incorporated user interaction data when it was available. Our aim was to develop a relevance model that would accurately predict interaction probability for a given user-item pair. Considering the sparsity of interaction data we treat {*impression, click, bookmark, reply, recruiter*} interactions as one "positive" interaction type, and predict whether user will generate *any* of these interactions for a given item.

3.1 Framework

In both phases we have a set of N users $\mathcal{U} = \{u_1, ..., u_N\}$ and a set of M items $\mathcal{V} = \{v_1, ..., v_M\}$, N and M varied in each phase. The users' interactions with the items can be represented by an $N \times M$ interaction matrix $\mathbf{R}$ where $\mathbf{R}_{uv}$ is the preference for item v by user u; when no preference information is available $\mathbf{R}_{uv} = 0$. Note that we have a separate $\mathbf{R}$ for each of the six interaction types, to avoid notation clutter we drop the interaction index and explicitly state interaction type. In this representation multiple interactions for the same user-item pair are aggregated together and $\mathbf{R}_{uv}$ is the total interaction count for (u, v). Together with count we also store date of the most recent interaction for (u, v). After aggregation we found that the number of unique user-item pairs for each interaction type was significantly reduced from the original count in Table 1. For example, the number of unique impressions was reduced to only 21M from the original set of 314M in the offline dataset. This made the problem extremely sparse with the most dense impression matrix having density of only 1e-5%. We use $\mathcal{U}(v) = \{u \in \mathcal{U} \mid \mathbf{R}_{uv} \neq 0\}$ to denote the set of users that interacted with v, and $\mathcal{V}(u) = \{v \in \mathcal{V} \mid \mathbf{R}_{uv} \neq 0\}$ to denote the set of items that u interacted with.

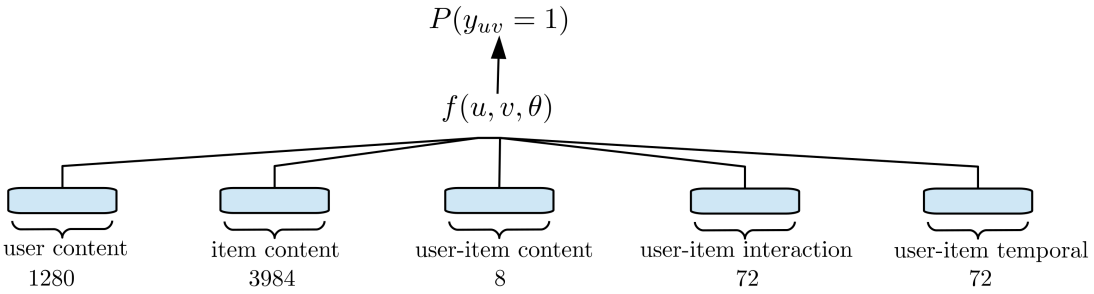

Figure 1: Model architecture diagram. Five feature categories (see Section 3.2) are fed into the classifier $f(u, v, \theta)$ which outputs the probability $P(y_{uv} = 1)$ that user u will "positively" interact with item v. Number of features for the five categories are shown below each feature input, in total 5416 features were used.

3.2 Input Features

We spent considerable effort on feature engineering and after several rounds selected five feature categories that were used in all models. Many of the features were inspired by previous work in this area, in particular [6]. Given user-item pair (u, v) we extract the following features:

- **User content**: all content features for u. We apply 1-of-n transformation to all categorical features and remove categories that appear in fewer than 500 users.
- **Item content**: all content features for v. Similarly to user content, we apply 1-of-n transformation to all categorical features and remove categories that appear in fewer than 500 items.
- **User-item content**: overlap between user and item content. We focus on job-specific overlap such as discipline, industry and career level, as well as location-specific overlap such as country and region:
  - `intersect(jobroles(`u`), tags(`v`))`
  - `intersect(jobroles(`u`), title(`v`))`
  - `career_level(`u`) - career_level(`v`)`
  - `I[career_level(`u`) == career_level(`v`)]`
  - `I[discipline_id(`u`) == discipline_id(`v`)]`
  - `I[industry_id(`u`) == industry_id(`v`)]`
  - `I[country(`u`) == country(`v`)]`
  - `I[region(`u`) == region(`v`)]`

  Technically all of these features can be computed by directly modeling user and item content. However, representing complex predicates such as `same discipline AND same industry AND same country AND same region`, would require very deep models that are able to simultaneously consider all possible values of each sub-predicate. Consequently we found that adding these features significantly accelerated learning achieving comparable or better accuracy in fraction of time and with simpler models.
- **User-item interaction**: similarity between items that u interacted with and v. We compute average similarity for all items that u interacted with excluding v:

$$\frac{1}{|\mathcal{V}(u) \backslash v|} \sum_{\substack{v' \in \mathcal{V}(u) \\ v' \neq v}} \texttt{sim}(v, v')$$

  Here, `sim`(v, v') is a vector with the following 12 features:
  - `intersect(tags(`v`), tags(`v'`))`
  - `intersect(title(`v`), tags(`v'`))`
  - `intersect(tags(`v`), title(`v'`))`
  - `intersect(title(`v`), title(`v'`))`
  - `career_level(`v`) - career_level(`v'`)`
  - `I[career_level(`v`) == career_level(`v'`)]`
  - `I[discipline_id(`v`) == discipline_id(`v'`)]`
  - `I[industry_id(`v`) == industry_id(`v'`)]`
  - `I[employment(`v`) == employment(`v'`)]`
  - `I[country(`v`) == country(`v'`)]`
  - `I[region(`v`) == region(`v'`)]`
  - `distance(`v`, `v'`)`

  Separate average similarities are computed for each of the six interaction types and are concatenated to form the user-item interaction feature vector. These features are analogous to the item-based neighbor collaborative filtering model [7], but with content similarity replacing interaction similarity. Analogous user-based neighbor features can be extracted by computing content similarity between users who interacted with v and u. However, these features would not be applicable to item cold start so we omit them here.
- **User-item temporal**: same features as in user-item interaction but using only the *last* item that u interacted with for each interaction type. The motivation for adding this feature is that the last interaction provides most up-to-date information on user intent, and can signal significant preference altering events such as career/industry change earlier. To reduce noise we experimented with smoother versions of this feature by applying time dependent weighted average, but found that the smoothing factor was difficult to tune and did not produce significant improvements. A number of other approaches have been proposed to incorporate temporal information [5, 9, 10] into CF models. However, after experimenting with several of these models we found that they were difficult to optimize on highly sparse binary data, and offered no straightforward way to incorporate content information.

For all features `intersect` is set intersection size, `I` is a binary indicator variable and `distance` is distance in kilometers between longitude and latitude coordinates. When any information required to compute a given feature was missing we set that feature to 0. We deliberately kept all features conceptually simple and straightforward to calculate. With optimized data structures most of these features can be calculated on the fly and we were able to

get sub-millisecond inference run times for each user-item pair on multicore architectures (see Section 3.5 for more details). Figure 1 shows number of features in each of the five categories, in total 5416 features were used.

3.3 Model

Using the input features, our goal was to create a relevance model that would accurately predict if a user would positively interact with a given item. Recall that positive interaction refers to any of *{impression, click, bookmark, reply, recruiter}*. We take a probabilistic approach and use y_{uv} to denote a binary random variable where $y_{uv} = 1$ if u positively interacted with v and 0 otherwise. We then define a relevance model $f : \mathbb{R}^p \rightarrow \mathbb{R}^1$ as a mapping from input features to probability of positive interaction $P(y_{uv} = 1)$. Full architecture of this model is shown in Figure 1.

Common choices for f include deep neural networks (DNNs) and tree-based approaches such as gradient boosting machines (GBMs) [4]. We experimented with both DNN (1 to 5 hidden layers) and GBM models, and found GBMs to be significantly easier to train. Most difficulties with DNNs came from input sparsity and feature range. On average only several hundred features were on for most user-item pairs resulting in highly sparse gradient that made learning slow. Moreover, functions such as `distance` can produce feature values in the thousands creating a disproportionately large gradient. To deal with this problem we had to apply strict input normalization and/or gradient clipping. However, normalizing highly sparse input is a non-trivial task and we found DNN models to be very sensitive to the choice of normalization. GBMs on the other hand did not require feature normalization and worked well without any input pre-processing. For these reasons we used GBM models in all experiments and trained them using the excellent XGBoost library [2].

3.4 Training

During training our goal was to ensure that the model would generalizes well to both offline and online test data. To simulate test evaluation we partitioned the interaction data forward in time using the last two weeks of interactions as validation set and the rest as training set. Within the validation set we randomly selected a subset of 20K unique items that had at least one positive interaction. These items became the target items, and the joint set of users (approximately 50K) that interacted with these items during the validation time period became the target users. To simulate cold start, all interactions for target items were removed from the training set. Validation set was used to guide both feature and model selection, and the best performing model was submitted for test evaluation to XING.

After partitioning the data we applied a similar approach to [3], and sampled positive and negative user-item pairs to train the model. Specifically, to stay consistent with the target scoring function in Equation 1 we took an item-oriented approach and sampled sets of unique positive and negative users for each item. Positive users were sampled from users that positively interacted with the item and are selected according to the scoring weight. For example, recruiter interaction is 4x more important than bookmark/reply and 20x more important than click. Similarly, negative users were sampled from users that only deleted the item. For items that didn't have enough deletes we randomly sampled negative users from the entire user set (excluding users that already interacted with the item). Random negative sampling is appropriate here since both user and item sets are large enough to ensure that any randomly selected user-item pair is not going to be relevant with very high probability. In all cases cold start items and users were removed from the data before sampling to avoid biasing the model.

Similar procedure was applied to the validation set by including all positive user for each target item and sampling negative users from the target validation users. This allowed us to quickly validate models throughout learning using metrics such as AUC without having to compute the full score from Equation 1. We found that validation AUC correlated well with the test score and most improvements translated to the offline leader board. Note that we chose the AUC metric here due to its ranking interpretation and insensitivity to class imbalance.

The sampling procedure was designed to balance the training data. First, by selecting the same number of positive and negative samples for each item we reduce bias towards popular items that receive significantly more interactions. Empirically we found that balancing item interactions significantly improved generalization with relative gains of over 10%. Reducing popularity bias was also found to be important by other studies [3], leading to better metrics on live A/B tests. Second, controlling the sample sizes for each class allowed us to in turn control the positive/negative class distribution during training. Training with highly imbalanced classes is well known to be difficult and often leads to over/under-fitted models. We avoid these problems by explicitly selecting the number of examples for each class. Finally, sampling positive examples according to the scoring function emphasizes stronger interactions such as bookmark/reply and recruiter interest that indicate higher degree of preference.

After selecting positive and negative samples we minimize the following classification objective for each item v:

$$\begin{aligned} O(v,\theta) &= -\sum_{u \in \mathcal{U}(v)^{POS}} \log P(y_{uv} = 1) - \sum_{u \in \mathcal{U}(v)^{NEG}} \log(1 - P(y_{uv} = 1)) \\ &= -\sum_{u \in \mathcal{U}(v)^{POS}} \log \frac{1}{1 + e^{-f(u,v,\theta)}} - \sum_{u \in \mathcal{U}(v)^{NEG}} \log \left(1 - \frac{1}{1 + e^{-f(u,v,\theta)}}\right) \end{aligned} \tag{2}$$

where $\mathcal{U}(v)^{POS}$ and $\mathcal{U}(v)^{NEG}$ are positive and negative user samples for v, and θ is the set of free parameters. By minimizing O we raise the probability for positive samples and lower it for negative samples, indirectly achieving the desired effect where relevant users are ranked higher than the irrelevant users.

It is important to note here that $f(u, v, \theta)$ is applied to features extracted for each (u, v) pair using the approach outlined in Section 3.2. To simulate forward in time prediction and prevent data leakage it is common practice to only include interactions that occurred before (u, v) (if (u, v) is a positive pair) to compute interaction-based features. However, given the sparsity of interaction data we found that there wasn't enough data to partition interactions forward in time and still get meaningfully dense features. Moreover, considering that the overall timeframe of the data is only four months we can assume that the users' intent didn't change significantly during

that time period. Consequently, we don't partition the interaction data forward in time during feature extraction, and extract features using all available training data including interactions after (u, v). Empirically we found this to generalize well consistently giving us the best accuracy for both offline and online phases.

3.5 Inference

Once the model is trained we use the predicted relevance probabilities $P(y_{uv} = 1)$ to select recommendations. We experimented with various approaches to reduce the number of candidate pairs that needed to be ranked, including job-specific (same industry, discipline etc.) and location-specific (same region, country etc.) filters. However, we found that these filters were too restrictive and dropped many relevant candidates significantly hurting recall. To generate predictions for the offline phase, we thus opted to sort all target users for each item and select top-100 highest scoring users. Similarly, in the online phase, to satisfy the constraints of maximum 1 item per user and 250 users per item, we first computed lists of top-100 most relevant items for each user. Each list was pruned by applying a probability threshold and concatenated together to form one global list of top user-item pairs. We then sorted this global list by relevance probability and iterated in sorted order adding each (u, v) pair into the submission if (1) u hasn't been added already and (2) if v didn't already have 250 users. This procedure greedily maximized the sum of probabilities of all selected pairs while also satisfying the target constraints. Note that the same relevance model was used for both phases with the only difference that in the online phase the model was re-trained on the online training set.

Both online and offline inference procedures required scoring and ranking all available user-item pairs. Given that there are $74,840 \times 46,559 \approx 3.5$B pairs in the offline phase and between 50M and 400M pairs daily in the online phase, we implemented a highly optimized inference routine with feature caching and full multi-core support. User content and item content features were cached while all other feature categories were extracted on the fly using optimized data structures with efficient access to content and interactions for any given user and item. Running this procedure on a server with 20 Intel Xeon E5-2630 cores we were able to obtain sub-millisecond inference runtimes (feature extraction and scoring) for individual user-item pairs, and generate full predictions in under 12 hours for the offline phase and in 0.5-1.5 hours for the online phase.

Finally, we also experimented with various post-processing approaches to maximize the target score function in Equation 1. Specifically, we tried to prioritize paid users and items, as well as apply diversity metrics to ensure broader user and item coverage. However, we found that the gains from these procedures were too small to warrant the additional hyper parameter tuning.

4 RESULTS

We conducted extensive experiments to validate the training/validation framework and model parameters. Through these experiments we found that sampling up to 40 positive and 40 negative users for each item produced good results. The 40-40 sampling scheme provided a large training set with over 20M user-item-target triplets, allowing to train deeper and more complex GBM models without over-fitting. For GBM the typical parameter settings (using XGBoost notation) were: `eta = 0.1`, `max_depth = 15` and `colsample_bytree = 0.6`. We used the logistic objective and trained 500 to 1000 trees. Throughout learning we didn't observe any over-fitting, so it is possible that increasing `max_depth` would lead to better models. However, since inference time is directly proportional to depth × number of trees, we opted to sacrifice model complexity for inference speed.

Offline

model	validation auc	leaderboard
1-2	0.8873	17,708
1-2-3	0.8951	21,605
1-2-3-4	0.9497	70,072
1-2-3-4-5	0.9522	71,995
1. layer6.ai		**75,782**
2. Lunatic Goats		71,372
3. Hushpar		61,427
4. rho		59,461
5. Get all the data		57,043

Online

team	best two weeks	total
1. layer6.ai	**10,963**	**17,979**
2. Lunatic Goats	9,741	15,651
3. chome	9,648	15,290
4. rho	9,536	14,791
5. leavingseason	9,173	15,021

Table 2: Results for offline and online phases of the competition. Top-5 teams are shown for each phase; our team was called layer6.ai. In the offline section we also show both validation auc accuracy and corresponding leaderboard score for different combinations of input feature categories. The models are labeled using the corresponding feature categories: 1=user content, 2=item content, 3=user-item content, 4=user-item interaction and 5=user-item temporal.

Table 2 shows results for both offline and online phases of the competition. For each offline model we show the corresponding feature categories from 1 to 5 that were used as input. From the offline results we see a significant improvement in both validation auc and leaderboard score when content feature categories 1, 2 and 3 are augmented with interaction categories 4 and 5. Validation auc improves by almost 6 points from 0.8951 to 0.9522, and leaderboard score more than triples. This indicates that interaction neighbor-based content features are highly useful and should always be included when available. Table 3 shows feature importances for the best offline model trained with all five feature categories using depth 15 and 500 trees. We show both overall category importance as well as top-5 features for each category. From this table we see a similar pattern where user-item interaction category is nearly three times more important than the next category item content. We also see a common pattern where location features such as distance between items that user interacted with and current item, as well as content features such as overlap between user's job roles and item's

Feature	Importance
user-item interaction	**1048528**
cli distance(v, v')	57565
cli intersect(tags(v), tags(v'))	51671
cli career_level(v) - career_level(v')	47504
imp intersect(tags(v), tags(v'))	45135
imp distance(v, v')	39735
item content	**337514**
latitude(v)	34423
longitude(v)	31158
is_payed(v)	6586
industry_id(v)	4376
career_level(v)	3986
user-item temporal	**226226**
cli distance(v, v')	50925
imp distance(v, v')	42798
cli intersect(tags(v), tags(v'))	17515
imp intersect(tags(v), tags(v'))	14531
del distance(v, v')	11197
user content	**118374**
experience_n_entries(u)	4343
industry_id(u)	4271
career_level(u)	4211
discipline_id(u)	3721
experience_years(u)	3185
user-item content	**36506**
career_level(u) - career_level(v)	9602
intersect(job_roles(u), tags(v))	7476
intersect(job_roles(u), title(v))	6460
I[region(u) == region(v))]	3792
I[career_level(u) == career_level(v)]	3156

Table 3: Feature importance statistics for a GBM model with 500 trees of depth 15. For each of the five feature categories we show the cumulative category importance together with top-5 most important features. Features from user-item interaction and user-item temporal categories are extracted separately for each interaction type, and we show the corresponding interaction type (cli=click, imp=impression, del=delete) for each feature from these categories.

tags, are the most important. This leads to an expected conclusion that recommended jobs should be both relevant and in the right location. Finally, user-item temporal is the third most important category significantly outperforming both user content and user-item content categories. Temporal modeling has repeatedly been found to be important for recommender systems [5, 9, 10] and these results further support that. Throughout our experiments we saw small but consistent gain by incorporating temporal features into the model.

Results for the top-5 teams in each phase are also shown in Table 2. Our team was called `layer6.ai` and our best submission used a blend of two GBM models each trained with different parameter settings. Blending the models improved the offline leaderboard score from 71,995 to 75,782 outperforming the next best team by over 6%. However, even a single model would have been enough to win this phase. In the online phase teams were ranked by the sum of the two best week scores out of a total of five weeks. The lower half of Table 2 shows top-5 team results for the online phase. From these results it is seen that we outperformed all other teams achieving the highest best two week score beating the next best team by over 12%. Our model also had the most consistent performance winning four of the five weeks and achieving the highest total score over the five weeks. To put this into perspective no other team placed 2'nd for more than two weeks. Consistency is highly important for recommender systems since unstable performance can lead to significant user dissatisfaction and churn. Given that the same features and model were used for both phases, these results confirm that our approach generalizes well with consistently strong online A/B performance.

5 CONCLUSION

We have presented our solution to the 2017 ACM Recommender Systems Challenge. Our team placed first in both offline and online phases of the challenge achieving the most consistent performance in the online phase winning four of the five weeks. For future work, a promising area of research is to further explore temporal aspects of user interactions. Users typically have specific intents when it comes to career-related services, and intent can change quickly over time. Modeling temporal patterns can lead to a model that better anticipates and reacts to these changes. Another interesting area for future work is to explore a training framework that more closely aligns with the target scoring function.

REFERENCES

[1] F. Abel, Y. Deldjoo, M. Elahi, and D. Kohlsdorf. 2017. RecSys Challenge 2017: Offline and Online Evaluation. In *ACM Recommender Systems Conference*.

[2] T. Chen and C. Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Knowledge Discovery and Data Mining*.

[3] P. Covington, J. Adams, and E. Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *ACM Recommender Systems Conference*.

[4] J. H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. *Annals of statistics* (2001).

[5] Y. Koren. 2009. Collaborative filtering with temporal dynamics. In *Knowledge Discovery and Data Mining Conference*.

[6] A. Pacuk, P. Sankowski, K. Węgrzycki, A. Witkowski, and P. Wygocki. 2016. RecSys Challenge 2016: Job recommendations based on preselection of offers and gradient boosting. In *ACM Recommender Systems Conference*.

[7] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *International World Wide Web Conference*.

[8] X. Su and T. M. Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence* (2009).

[9] C.-Y. Wu, A. Ahmed, A. Beutel, A. Smola, and H. Jing. 2017. Recurrent recommender networks. In *ACM International Conference on Web Search and Data Mining*.

[10] L. Xiong, X. Chen, T.-K. Huang, J. Schneider, and J. G. Carbonell. 2010. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *IEEE International Conference on Data Mining*.