Content-Based approaches for Cold-Start Job Recommendations

Mattia Bianchi
mattia3.bianchi@mail.polimi.it

Federico Cesaro
federico.cesaro@mail.polimi.it

Filippo Ciceri
filippo.ciceri@mail.polimi.it

Mattia Dagrada
mattia.dagrada@mail.polimi.it

Alberto Gasparin
alberto.gasparin@mail.polimi.it

Daniele Grattarola
daniele.grattarola@mail.polimi.it

Ilyas Inajjar
ilyas.inajjar@mail.polimi.it

Alberto Maria Metelli
alberto.metelli@mail.polimi.it

Leonardo Cella
leonardo.cella@mail.polimi.it

Politecnico di Milano
Milan, Italy

ABSTRACT

This paper provides an overview of the approach we adopted as team Lunatic Goats for the ACM RecSys Challenge 2017 [7]. The competition, organized by XING.com, focuses on a cold start job recommendation scenario. The goal was to design and tune a recommendation system able to predict past users' interactions, for the offline stage, and to provide recommendations pushed every day to real users through the XING portal, for the online stage. Our strategy, which saw models coming from different techniques combined in a multi-layer ensemble, granted us the first place in the offline part and the qualification as second best team in the final leaderboard. All our algorithms mainly resort to content-based approaches, that, thanks to its ability to provide good recommendations even for cold-start items allowed us, quite unexpectedly, to achieve good results in terms of prediction quality and computational time.

CCS CONCEPTS

• **Information systems → Recommender systems**;

KEYWORDS

ACM RecSys Challenge 2017, Recommendation Systems, Job recommendations, Cold-Start recommendations, Content-Based Filtering

ACM Reference Format:
Mattia Bianchi, Federico Cesaro, Filippo Ciceri, Mattia Dagrada, Alberto Gasparin, Daniele Grattarola, Ilyas Inajjar, Alberto Maria Metelli, and Leonardo Cella. 2017. Content-Based approaches for Cold-Start Job Recommendations. In *Proceedings of RecSys Challenge '17, Como, Italy, August 27, 2017,* 5 pages.
https://doi.org/10.1145/3124791.3124793

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

ACM ISBN 978-1-4503-5391-5/17/08…$15.00
https://doi.org/10.1145/3124791.3124793

1 INTRODUCTION

Job recommendation is a continuously growing field of application in the Recommendation Systems area, offering new opportunities and posing new challenges. A typical approach revolves around pushing already seen job postings to users; this is often done through the usage of Collaborative Filtering (CF) techniques [10]. However, in a cold-start scenario, where items have not yet been seen by users, collaborative methods are not applicable. The problem of cold-start items has been known in literature for a while [13], but still remains one of the major challenges in the area of recommendation systems.

When features are available for items, and possibly for users, Content-Based methods (CBF) [4–6, 8, 14] can be used. Our approach is mainly content-based and consists in recommending jobs with features similar to those already interacted with by users, as well as those matching user profiles. We also resort to collaborative information, such as users' past interactions, in an indirect way. We leverage on the predictions of the content-based methods to impute interactions for cold-start items and then we run standard collaborative filtering algorithms on the augmented dataset. In order to exploit the diversity of the individual models, we adopt a multi-level ensemble architecture.

The paper is organized as follows. In Section 2 we outline the competition format and the structure of the dataset. In Section 3 we describe the data preparation process. Section 4 is devoted to algorithms description, while in Section 5 we illustrate the ensemble architecture. Finally, in Section 6 we go over our results and propose possible extensions.

2 PROBLEM FORMULATION

The competition consists in a cold-start recommendation problem, where the goal is to balance a candidate's disposition toward a job posting (item) with the job recruiter's interest in the candidate (user), and at the same time avoiding the recommendation of job postings that a candidate may react negatively to. Clearly, the most challenging facet of the competition is the absence of previous interactions with the target jobs, which impeded the direct application of collaborative approaches.

The competition is divided into two phases: offline and online [7]. In the offline phase, we are given a static dataset and our task is to predict the interactions of target users with target items that occur

in the given time frame, starting from data containing interactions of the users with non-target items. For each target item, we are allowed to recommend at most 100 target users. Achieving a top 20 finish in the offline part allows access to the second phase. The online phase consists in pushing daily recommendations to real users through the XING portal over a five week period. In this phase, we are limited to recommend one item for each target user.

2.1 Mathematical Notation

We denote with $\mathcal{U}$ and $\mathcal{I}$ the set of users and items respectively. For each user, we denote the set of items the user u interacted with as $\mathcal{I}(u)$. Symmetrically, we represent the set of users that interacted with an item i as $\mathcal{U}(i)$. To distinguish between positive and negative interactions for a user u, we use the symbols $\mathcal{I}^+(u)$ and $\mathcal{I}^-(u)$ respectively. The set of target items and target users are represented as $\mathcal{I}^{\mathcal{T}}$ and $\mathcal{U}^{\mathcal{T}}$ respectively.

2.2 Dataset description

The dataset for the competition, provided by XING, spans over a 93 day period and contains ~323M interactions, divided into six types:

- *impression* (~315M occurences): also known as type 0 interactions, these are job postings shown to a user by the XING portal. The presence of an impression does not imply the user interacted with the posting;
- *click* (~6.9M occurrences): type 1 interactions, happen when a user clicks on a job posting;
- *bookmark* (~282k occurrences): type 2 interactions, when a user adds a job posting to their favorites;
- *application* (~118k occurrences): type 3 interactions, when a user clicks the apply button on a job posting;
- *delete* (~907k occurrences): type 4 interactions, when a user deletes a recommendation from their homepage;
- *recruiter interest* (~100k occurrences): type 5 interactions, when a recruiter associated to a job posting clicks a user profile.

In the following, we will assume interactions of type 1, 2, 3 and 5 as positive interactions and those of type 4 as negative interactions. The dataset contains ~1.5M users, of which approximately ~689k are active in the 93 day time frame. Out of these users, the target user set is composed of both active and inactive users (cold-start users), and amounts to approximately ~75k users. An active user has on average ~12 interactions in the examined time frame, with a median of 3.

The item set is composed by ~1.3M job postings. Out of these postings, approximately ~768k are cold-start, out of which only ~47k are part of the target item set.

2.3 Evaluation metric

In order to align with the focus of the competition, a custom scoring function is provided for evaluation purposes (Algorithm 1).

Its two components are *user_success* and *item_sucess* scores. Although it might seem that the bulk of the score comes from *user_success*, during local testing we realized that correctly predicting many positive interactions for a large set of target items rather than for a large set of target users yields better scores. This is a result from gaining more points by hitting the *item_success* threshold.

Algorithm 1 Evaluation metric pseudo-code (from [7]).

```
score(item, users) =
    users.map(u => userSuccess(item,u)).sum +
    itemSuccess(item, users)

userSuccess(item, user) =
    (
        if (clicked) 1 else 0
        + if (bookmarked || replied) 5 else 0
        + if (recruiter interest) 20 else 0
        - if (delete only) 10 else 0
    ) * premiumBoost(user)

premiumBoost(user) = if (user.isPremium) 2 else 1

itemSuccess(item, users) =
    if (users.filter(u => success(item, u) > 0).size > 0) {
        if (item.isPaid) 50
        else 25
    } else 0
```

From an optimization standpoint, the evaluation function is discontinuous and therefore non-differentiable. This greatly reduces the set of applicable optimization tools (see 5.2).

2.4 Local validation

In order to locally test our algorithms, we split the dataset into training and validation set. We obtain the best result by *randomly sampling* items (~10k) from those items that have at least one interaction and using them as target items. We populate the validation set with all the interactions of the sampled target items and the training set with the remaining interactions. In this way, we guarantee that the target items are cold-start, as in the competition dataset. The target users (~16k) are sampled among the users that have interactions with items in the validation set. The number of users is selected in order to maintain the same user-item ratio as that of the competition dataset.

One of the keys to our success during the first part of the competition is the fact that the performance of our algorithms on the validation set is similar to that of the offline leaderboard. The absence of a private leaderboard allows us to use the validation set to tune our algorithms, in order to choose whether or not a modification is beneficial, basically granting us an infinite number of submissions.

We also exploit the validation set to find the best weights of each single algorithm for the ensemble (detailed description in 5.2) using a Genetic Algorithm and Gradient-free Optimization Methods.

3 PREPROCESSING

We build the feature space by performing One Hot Encoding of the user and item features and preprocessing them with TF-IDF (Term Frequency - Inverse Document Frequency) [2]. TF-IDF allows to account for the relevance of a feature in the computation of similarity between users and/or items. For instance, for the user

features, before the preprocessing an entry f_{uk} is 1 if user u possesses feature k, 0 otherwise. Thus the term frequency part is binary ($f_{uk} = TF(u,k)$) while the $IDF(k)$ is computed as:

$$IDF(k) = \log \frac{|\mathcal{U}|}{\sum_{v \in \mathcal{U}} f_{vk}}.$$

After the preprocessing each entry f_{uk} is replaced with $f_{uk} IDF(k)$. The same transformation is applied to the item features.

3.1 Feature aggregation

In order to better exploit the information contained in the user and item data we extend the feature space. We define new *aggregated features* obtained as the entry-wise logical conjunction ($\wedge$) between the original binarized feature vectors (before TF-IDF transformation):

$$\boldsymbol{f}_{k_1,k_2,\ldots,k_n} = \boldsymbol{f}_{k_1} \wedge \boldsymbol{f}_{k_1} \wedge \ldots \wedge \boldsymbol{f}_{k_n}, \tag{1}$$

where $\boldsymbol{f}_{k_1,k_2,\ldots,k_n}$ is the aggregated feature and $\boldsymbol{f}_{k_i}$ $i = 1, 2, \ldots, n$ are the original binarized feature. Thus, $f_{u,k_1,k_2,\ldots,k_n} = 1$ if user u possesses simultaneously all the features $k_1, k_2, \ldots, k_n$, the same rationale holds for items. We perform this transformation combining several sets composed of two or three features. Clearly, this transformation creates a sparser feature space since the aggregated feature are rarer and therefore their importance is boosted by TF-IDF reweighting. This results in a significant performance improvement for the content-based algorithms.

3.2 Negative user filtering

Since hitting type 4 interactions has a negative effect on the final score we avoid recommending users with a high predisposition towards negative interactions. We remove all target users with a fraction of negative interactions larger than 0.9 (*negative user filtering*).

4 ALGORITHMS

Our model is composed of five individual algorithms that leverage both content and collaborative methods.

4.1 Content Based Filtering - Item based

We resort to a classic *item-based content based filtering* (CBF) [5], in which the similarity between two jobs i and j is defined as the Tanimoto coefficient between the two feature vectors:[1]

$$s_{ij}^{\text{CBF}} = \text{Tanimoto}(\boldsymbol{f}_i, \boldsymbol{f}_j, \beta) = \frac{\boldsymbol{f}_i^T \boldsymbol{f}_j}{\|\boldsymbol{f}_i\|_2^2 + \|\boldsymbol{f}_j\|_2^2 - \boldsymbol{f}_i^T \boldsymbol{f}_j + \beta}, \tag{2}$$

where β is a shrinkage term whose value is empirically set to 100.

Since the challenge requires accounting for both positive and negative feedback, we build two models for predicting positive (CBF+) and negative (CBF−) interactions[2]. In both cases the score of cold-start item i for user u is given by:

$$\hat{r_{ui}^{\text{CBF}\star}} = \frac{1}{|\mathcal{I}^\star(u)| + \gamma} \sum_{j \in \mathcal{I}^\star(u)} r_{uj} s_{ij}^{\text{CBF}}, \tag{3}$$

where $\star = +$ for CBF+ and $\star = -$ for CBF−. The term $\frac{1}{|\mathcal{I}^\star(u)|+\gamma}$ acts as a regularization term to account for the number of interactions a user has made overall, avoiding a preference towards highly active users (γ is a shrinkage term set to 7).

CBF+ is the best performing individual algorithm. Our approach is based on building distinct models for positive and negative interactions; combining them at ensemble-time results in outperforming the classic explicit feedback approach.

4.2 User-Item Profile Matching

CBF approaches are based on the assumption that if a user liked (or disliked) an item we are confident that s/he is going to like (or dislike) similar items, where the similarity is computed on the item features. However, when a user is cold-start, a CBF approach cannot be used. When users and items share a set of features, to overcome this limitation, we can resort to *user-item profile matching* (PM) methods. Given a user u and a cold-start item i, the score is computed as the cosine coefficient between the corresponding feature vectors:

$$\hat{r_{ui}^{\text{PM}}} = \text{Cosine}(\boldsymbol{f}_u, \boldsymbol{f}_i, \beta) = \frac{\boldsymbol{f}_u^T \boldsymbol{f}_i}{\|\boldsymbol{f}_u\|_2 \|\boldsymbol{f}_i\|_2 + \beta}, \tag{4}$$

where β is empirically set to 25.

4.3 Content based micro-clustering

To exploit the collaborative information contained in the dataset we resort to a *micro-clustering* approach in order to impute fictitious interactions for the target items [1]. For each target item we associate the interactions of the top 5 similar items according to content-based similarity (2). Now, on this augmented dataset, we can run standard collaborative filtering algorithms. We limit the application of collaborative methods to the prediction of positive interactions only.

4.4 Collaborative Filtering

We run an *item-based collaborative filtering* algorithm (CF) [10] on the augmented dataset in which the similarity between two items i and j is defined as the cosine coefficient between the corresponding rating vectors:

$$s_{ij}^{\text{CF}} = \text{Cosine}(\boldsymbol{r}_i, \boldsymbol{r}_j, \beta) = \frac{\boldsymbol{r}_i^T \boldsymbol{r}_j}{\|\boldsymbol{r}_i\|_2 \|\boldsymbol{r}_j\|_2 + \beta}, \tag{5}$$

where β is a shrinkage parameter set empirically to 500. The score prediction of target item i for user u is given by:

$$\hat{r_{ui}^{\text{CF}}} = \sum_{j \in \mathcal{I}^+(u)} r_{uj} s_{ij}^{\text{CF}}. \tag{6}$$

4.5 Matrix Factorization

iALS [9] is a matrix factorization algorithm designed to tackle the problem of implicit feedback. In order to account for the importance of each interaction, a variable representing the *confidence level* in observing an interaction r_{ui} is introduced. In our case, we provide different definitions according to whether the interaction is real or

[1] By extensive testing we discovered that Tanimoto coefficient outperforms the classic cosine coefficient in our dataset.

[2] CBF− is intended to predict type 4 interactions, which yield negative points, thus we trained it in order to get a score as negative as possible.

Table 1: iALS model parameters.

Parameter	Value
α	170
λ	300
factors	800
iterations	30

fictitious:

$$c_{ui} = \begin{cases} 1 + \alpha n_{ui} & \text{if the interaction is real} \\ 1 + \sum_{j \in \text{CBF}-5\text{NN}(i)} s_{ij}^{\text{CBF}} r_{uj} & \text{if the interaction is fictitious} \end{cases},$$

where n_{ui} is the number of times user u made a positive interaction with item i. For imputed interactions the confidence value is obtained as the sum of the ratings given by the same user to the top 5 similar items $\text{CBF} - 5\text{NN}(i)$, weighted by the content-based similarity. Preferences are estimated as the inner product of the user-factors and the item-factors:

$$\hat{r}_{ui}^{\text{iALS}} = \boldsymbol{x}_u^T \boldsymbol{y}_i,$$

that the algorithm has to find by minimizing the following cost function:

$$\sum_{u \in \mathcal{U}, i \in \mathcal{I}} c_{ui}(r_{ui} - \boldsymbol{x}_u^T \boldsymbol{y}_i)^2 + \lambda(\|\boldsymbol{x}_u\|_2^2 + \|\boldsymbol{y}_i\|_2^2).$$

In order to find the vectors $\boldsymbol{x}_u$ for each user u and $\boldsymbol{y}_i$ for each item i an *alternating least square* optimization process is used, i.e. the current iteration user-factors are used to compute the next iteration item-factors and vice versa, until convergence is reached. The best parameters found are reported in Table 1.

5 ENSEMBLE

We chose a *multi-level ensemble* to take advantage of the diversity in the predictions coming from algorithms that use different types of information or different recommendation approaches. This allows us to optimize each set of algorithms in a segmented manner. The ensemble architecture is depicted in Figure 1. Notice that in the ensemble we include the XING baseline, run with the aggregated features.

Each ensemble level receives the scores of the top-n (n is set to 1000) users for each item as input from each connected blocks and processes them in three steps. First, it *normalizes* the scores, then it *weights* differently[3] the predictions of each block and finally it *sums* the weighted scores to output the prediction of the level. This results in a new set of user-item score pairs that are passed to the layer above. The process is repeated until the final level is reached which outputs the final prediction. The choice of the score normalization strategies, as well as the methods to recover the optimized weights are discussed in the following sections.

5.1 Score Normalization methods

Score Normalization is necessary due to the nature and diversity of the algorithms which output incomparable scores. Having to recommend at most 100 users per item, we work toward normalizing each set of scores via *max scaling*, in order to have comparable curves for the scores of each algorithm. We first resort to an item-oriented normalization approach, named I(tem)-NORM, in which scores are scaled according to the target item they refer to:

$$\hat{r}_{ui}^{\text{I-NORM}} = \frac{\hat{r}_{ui}}{\max_{v \in \mathcal{U}^{\mathcal{T}}} \hat{r}_{vi}}. \tag{7}$$

This first normalization tends to produce predictions in which few users are recommended to large sets of items. To better diversify the input for the ensemble, we move to a user-oriented approach, named U(ser)-NORM, focused on normalizing scores for each target user:

$$\hat{r}_{ui}^{\text{U-NORM}} = \frac{\hat{r}_{ui}}{\max_{j \in I^{\mathcal{T}}} \hat{r}_{uj}}. \tag{8}$$

This ensures that each user is recommended to their best matching item. This second normalization method causes scores to vary significantly. While it yields good recommendations for the single user, the quality of the top 100 users (the offline target) is worse. To maintain the diversification of the output while at least partially preserving the regularity of our score curves, we opt for a convex combination:

$$\hat{r}_{ui}^{\text{UI-NORM}} = \alpha \hat{r}_{ui}^{\text{I-NORM}} + (1 - \alpha)\hat{r}_{ui}^{\text{U-NORM}}, \tag{9}$$

where $\alpha \in [0, 1]$ is the weight assigned to the normalized scores. This parameter is set to 0.6 after extensive testing.

It is worth noting that cold-start users may receive lower scores, and a result come out as underrepresented, due to their recommendations coming from only one of the base algorithms (PM).

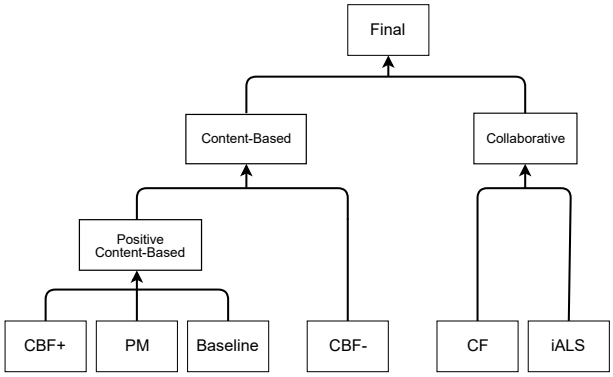

Figure 1: Ensemble architecture.

5.2 Parameter optimization

A major issue in the ensemble process is finding optimal weights for the different ensemble layers. As described above (Section 2.3), the evaluation metric is non-differentiable, rendering most optimization techniques impracticable. We approach the issue by applying two different techniques.

We first resort to Genetic Algorithms [11]. We use a basic implementation, trying ranking and tournament selection for the crossover phase and resorting to the evaluation function itself as fitness function. The learning procedure is stopped once the score remains constant over 100 successive iterations. Tournament selection gives better results, outperforming manual weight selection.

[3] Clearly, CBF− comes along with a negative weight.

In our second attempt, we use Powell's Conjugate Direction Method [12]. It consists in a gradient-free optimization algorithm that uses conjugate direction sets to minimize non-differentiable functions. Although slower than genetic algorithms, its output weights give a better score on our local validation set, and its performance was also the best in the offline part of the challenge. As a result, we re-apply it on the local validation set to select the weights of the ensemble for the online part of the challenge.

5.3 Offline Ensemble vs Online Ensemble

While the offline ensemble focused on scoring all around, the online ensemble focuses on maximizing scores for each user. This results in a maximization of item scores for the offline portion, and user scores for the online portion (see 2.3). Therefore, in the offline part we use the hybrid UI – NORM normalization, while in the online portion we prefer just U – NORM normalization. Another key in distinguishing between the two parts of the competition is the actual portion of the ensemble's final prediction which gets submitted. In the offline portion, we simply take the top-100 final scores for each item, whereas in the online portion, we are limited to one item per user, so we focus on submitting only the best item for each user. As a consequence, not all target items are recommended every time. Furthermore, we exclude iALS and the XING baseline from the online ensemble, due to their slower computation time and high memory usage. A key part of the online competition was pushing recommendations as soon as possible. Since recommendation pushes were instantaneous, making them available to users faster gave a higher chance for a user positive interaction.

6 RESULTS AND CONCLUSIONS

Our recommendation architecture allowed us to win the offline portion of the competition and to reach a second place finish in the online portion.

The scores of each individual algorithm and the ensemble, as well as the execution time, are reported in Table 2. The computational times do not include preprocessing and content-based micro-clustering (∼ 60 min). The predictions of most of the algorithms in the ensemble are heavily correlated, especially CBF+ with CF and iALS, due to the underlying micro-clustering approach adopted as a preliminary step for running collaborative methods. Nevertheless, the ensemble managed to extract the differing predictions from each algorithm, which was beneficial for the evaluation score.

The major strength of our architecture, especially with reference to the recommender used for the online portion of the competition, is its speed and ease of implementation. In contrast to classic machine learning approaches, online execution is seamless and does not require one to retrain the entire model as new users/items come.

Since the application domain is related to cold-start recommendations, where collaborative approaches are not applicable straightforward, improvements should be almost strictly related to working with features. Possible extensions to our model could look into user-personalized feature weighting, such as User-Specific Feature-based similarity models [6], or focus on deriving feature relevance from collaborative information, such as Feature-based factorized bilinear similarity model [14] and [3]. The cold-start problem has also been tackled by means of Matrix Factorization approaches in [8].

Referring specifically to the online portion of the competition, applying a time-decay model, where newer information is more heavily weighted than older information, could be beneficial. Another interesting extension to the model would look into using conversion rate analysis, i.e. modelling the causal relation between impressions and interactions.

Table 2: Score and execution time of each algorithm and the ensemble. Execution time refers to a VM with 8 cores and 16GB RAM except for iALS - 8 core 64GB RAM.

Algorithm	Local score	Leaderboard score	Execution time
CBF+	57852	60257	13 min
CBF–	-1330	-8529	4 min
PM	17260	16777	7 min
CF	42213	39250	12 min
iALS	48081	52411	150 min
XING Baseline	14742	14395	40 min
Ensemble	60625	71372	2 min

ACKNOWLEDGMENTS

The authors would like to thank Prof. Paolo Cremonesi for the continuous support and for having provided the computational infrastructure used in this challenge.

REFERENCES

[1] Qilong Ba, Xiaoyong Li, and Zhongying Bai. 2013. Clustering collaborative filtering recommendation system based on SVD algorithm. In *ICSESS, 2013 4th IEEE International Conference on*. IEEE, 963–967.
[2] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*. Vol. 463. ACM press New York.
[3] Leonardo Cella, Stefano Cereda, Massimo Quadrana, and Paolo Cremonesi. 2017. Derive item features relevance from past user interactions. In *UMAP*.
[4] Wei Chu and Seung-Taek Park. 2009. Personalized recommendation on dynamic content using predictive bilinear models. In *Proceedings of the 18th International Conference on World Wide Web*. ACM, 691–700.
[5] Marco de Gemmis, Pasquale Lops, Cataldo Musto, Fedelucio Narducci, and Giovanni Semeraro. 2015. Semantics-aware content-based recommender systems. In *Recommender Systems Handbook*. Springer, 119–159.
[6] Asmaa Elbadrawy and George Karypis. 2013. Feature-based similarity models for top-n recommendation of new items. *Department of Computer Science, University of Minnesota, Minneapolis, Minnesota, Tech. Rep* (2013), 14–016.
[7] 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.
[8] Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, Steffen Rendle, and Lars Schmidt-Thieme. 2010. Learning attribute-to-feature mappings for cold-start recommendations. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 176–185.
[9] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *Data Mining (ICDM), 2008 8th IEEE International Conference on*. IEEE, 263–272.
[10] Yehuda Koren and Robert Bell. 2015. Advances in collaborative filtering. In *Recommender systems handbook*. Springer, 77–118.
[11] Melanie Mitchell. 1998. *An introduction to genetic algorithms*. MIT press.
[12] Michael JD Powell. 1978. A fast algorithm for nonlinearly constrained optimization calculations. In *Numerical analysis*. Springer, 144–157.
[13] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. 2002. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 253–260.
[14] Mohit Sharma, Jiayu Zhou, Junling Hu, and George Karypis. 2015. Feature-based factorized bilinear similarity model for cold-start top-n item recommendation. In *SIAM International Conference on Data Mining*. SIAM, 190–198.