AR-CF: Augmenting Virtual Users and Items in Collaborative Filtering for Addressing Cold-Start Problems

Dong-Kyu Chae
Georgia Institute of Technology
Atlanta, USA
cdongkyu3@gatech.edu

Jihoo Kim
Hanyang University
Seoul, South Korea
datartist@hanyang.ac.kr

Duen Horng Chau
Georgia Institute of Technology
Atlanta, USA
polo@gatech.edu

Sang-Wook Kim*
Hanyang University
Seoul, South Korea
wook@hanyang.ac.kr

ABSTRACT

Cold-start problems are arguably the biggest challenges faced by *collaborative filtering* (CF) used in recommender systems. When few ratings are available, CF models typically fail to provide satisfactory recommendations for *cold-start users* or to display *cold-start items* on users' top-*N* recommendation lists. *Data imputation* has been a popular choice to deal with such problems in the context of CF, filling empty ratings with inferred scores. Different from (and complementary to) data imputation, this paper presents **AR-CF**, which stands for *Augmented Reality CF*, a novel framework for addressing the cold-start problems by generating virtual, but plausible neighbors for cold-start users or items and augmenting them to the rating matrix as additional information for CF models. Notably, AR-CF not only directly tackles the cold-start problems, but is also effective in improving *overall* recommendation qualities. Via extensive experiments on real-world datasets, AR-CF is shown to (1) significantly improve the accuracy of recommendation for cold-start users, (2) provide a meaningful number of the cold-start items to display in top-*N* lists of users, and (3) achieve the best accuracy as well in the basic top-*N* recommendations, all of which are compared with recent state-of-the-art methods.

CCS CONCEPTS

• **Information systems** → **Recommender systems**.

KEYWORDS

Recommender systems, collaborative filtering, cold-start problems, data sparsity, generative adversarial nets

ACM Reference Format:
Dong-Kyu Chae, Jihoo Kim, Duen Horng Chau, and Sang-Wook Kim. 2020. AR-CF: Augmenting Virtual Users and Items in Collaborative Filtering for Addressing Cold-Start Problems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401038

*Corresponding author

SIGIR '20, July 25–30, 2020, Virtual Event, China

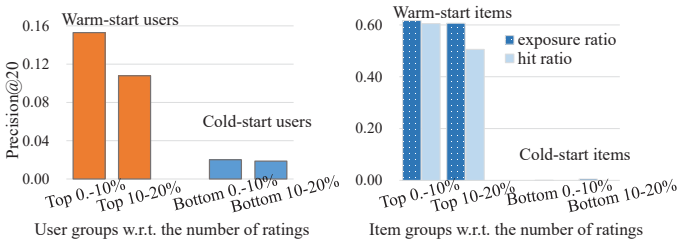
ACM ISBN 978-1-4503-8016-4/20/07…$15.00
https://doi.org/10.1145/3397271.3401038

1 INTRODUCTION

Collaborative filtering (CF) is one of the most successful methods in recommender systems [6, 8]. It aims at providing a personalized top-*N* list of items to a *target user* based on user-item interaction data [3], usually represented by a form of a *rating matrix* where a row indicates a user, a column indicates an item, and a value does a rating given by a user to an item, mostly in a specific range (e.g., [1, 5]) [4, 5]. In the literature, a large number of CF models have been studied, where popular examples include those based on *matrix factorization* (MF) [10, 18] or *deep neural networks* (DNN) [3, 4, 32, 36]. Despite their success, however, CF models tend to become less effective when facing the so-called *cold-start problems* [16, 28] due to the sparsity of the rating matrix. More specifically, it is difficult for CF models to understand the preferences of those *users* having insufficient ratings, thereby resulting in unsatisfactory recommendation to them (a.k.a. *cold-start user* problem). Similarly, CF models cannot fully capture the latent features of *items* that have insufficient ratings, thereby having difficulty in recommending those items to any user's top-*N* list (a.k.a. *cold-start item* problem).

To highlight these two types of cold-start problems, we performed a preliminary experiment with AutoRec [32] on the Movielens 1M dataset and tested its performance on the cold-start users and items w.r.t. top-20 recommendation. The results are shown in Figure 1. Figure 1(a) shows the accuracy (in terms of *precision*) of recommendation to the cold-start users is significantly lower than that of recommendation to the users with sufficient ratings available (i.e., *warm-start users*). Figure 1(b) shows that the warm-start items are very frequently recommended (denoted as *exposure ratio*) and many of them are actually chosen (denoted as *hit ratio*) by users, but the cold-start items are rarely recommended nor chosen by users.

To address the cold-start problems, most work so far have tried to exploit *additional information* such as users' demographic data [2] or social/trust networks [28], and auxiliary content of items such as review text [40] or images [13]. However, their methods are applicable only when such additional data is available [7]. Some studies employed *human crowd workers* in a crowdsourcing platform to



(a) Cold-start user problem (b) Cold-start item problem

Figure 1: Illustration of cold-start problems.

find relevant neighbors of the cold-start items [16] or developed frameworks based on *active learning* [34] that asks target users to provide more ratings. However, such human-machine interactive systems are known to be very time-consuming and costly to develop [16]. Our focus is on *pure CF*, i.e., a recommendation task given only with the rating matrix *without requiring any additional information and the human-in-loop framework*. In this context, *data imputation* has been the most popular choice in dealing with the cold-start problems of CF [7, 17, 30, 31, 41]. Its main idea is to resolve data sparsity in the rating matrix by inferring the values for missing ratings and imputing them to the rating matrix [20, 21, 23]. However, it is still not tailored to tackling directly the cold-start problems for users and items; rather it mainly reduces the overall sparsity of a dataset. In our comprehensive experiments (in Section 5), data imputation successfully improved the overall accuracy of recommendation but was not that much effective to alleviate the cold-start problems.

In this paper, we propose a novel CF framework that *directly tackles the cold-start problems and is effective as well in improving the overall accuracy of recommendation*. Different from existing imputation strategies that fill in the missing cells *inside* the rating matrix, our main idea is to *generate virtual, but plausible users and items* and then to *augment* the rating matrix with them as *additional rows* (i.e., users) and *columns* (i.e., items). More specifically, our **AR-CF** framework, which stands for *Augmented Reality CF*, consists of the following two steps: *AR-step* and *CF-step*. In the AR-step, we train *four* different *Conditional Generative Adversarial Nets* (CGANs) [26], where two CGANs collaborate with each other to generate virtual users (i.e., *user row vectors*); similarly, the other two CGANs focus on generating virtual items (i.e., *item column vectors*). Since user (resp. item) data is very sparse, which is quite different from image data typically considered by GANs [12, 29], we propose to use two CGANs where one focuses on learning *rating data distribution* and the other capturing *missingness distribution*. Then, we condition the trained CGANs with the cold-start users (resp. items) to generate their virtual "*neighbors*", and augment the rating matrix with them. In the CF-step, we train *any* existing CF model with the augmented rating matrix rather than the original one and then infer each user's (including the generated virtual users) predicted scores on her unrated items (including the generated virtual items). Finally, we refine each user's (resp. each item's) rating predictions by aggregating their virtual neighbors' inferred ratings and produce recommendation based on the refined ratings.

Contributions. To the best of our knowledge, our work is the first attempt to exploit GANs to generate virtual, but realistic neighbors of users and items to alleviate the cold-start problems in the context of *pure CF*. Notably, our framework presents the following important advantages: (1) from the viewpoint of CF models, the plausible and similar neighbors of the cold-start users and items can be seen as additional training data that could be helpful in better understanding the cold-start users and items. Thanks to these virtual neighbors, CF models can provide the cold-start users with higher-quality recommendations, and allow cold-start items to be more frequently recommended to users. (2) Not only for cold-start users and items, our GANs can also generate neighbors for any users and items, which could help improve the overall accuracy of CF. (3) Once our CGANs are completely trained, they can provide the desired number of virtual users and items to create a richer rating matrix, which is beneficial for any CF models, without requiring any cost or human workers. (4) Our framework is compatible with existing imputation methods, working on top of them. This is a very important property since the data imputation methods have been very successful in improving the accuracy of CF models so our ideas and imputation methods are synergistic, amplifying each other's positive effects. (5) Our experimental results on four real-world datasets demonstrate the benefits of AR-CF: thanks to the generated virtual neighbors, CF models provide the cold-start users with more-accurate recommendations and enable the cold-start items to be recommended more frequently to appropriate users. Moreover, our framework generally improves the overall accuracy of recommendation in existing state-of-the-art top-N recommenders.

Organization. The remainder of this paper is organized as follows. In Sections 2 and 3, we present the details of the AR-step and the CF-step of our AR-CF, respectively. In Section 4, we briefly review the current literature related to our work. In Section 5, we show the experimental settings and the results of our extensive experiments. In Section 6, we finally summarize and conclude our work. The notations frequently used throughout this paper are summarized in Table 1.

2 AR-STEP

This section introduces how we generate virtual users and items to be augmented into $\mathbb{R}$ based on the *Generative Adversarial Nets* (GAN) [12], a framework to train generative models with complicated, high-dimensional real-world data such as images. GAN consists of two models competing with each other: one is a *generative* model (shortly, $\mathcal{G}$) and the other is a *discriminative* model (shortly, $\mathcal{D}$). During learning, $\mathcal{G}$ tries to generate realistic data and passes it to $\mathcal{D}$; $\mathcal{D}$ evaluates the possibility that the data came from the ground truth rather than from $\mathcal{G}$ [7, 9]. Formally, the objective function of $\mathcal{G}$ and $\mathcal{D}$, $V(\mathcal{D},\mathcal{G})$, is defined as follows:

$$\min_{\mathcal{G}}\max_{\mathcal{D}} V(\mathcal{D},\mathcal{G}) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\ln\mathcal{D}(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\ln\mathcal{D}(\mathcal{G}(\mathbf{z}))] \quad (1)$$

where $\mathbf{x}$ is a ground-truth data from the data distribution p_{data} and $\mathbf{z}$ is a low-dimensional latent vector sampled from known prior p_z. $\mathcal{G}(\mathbf{z})$ is a synthetic data from the generator distribution p_g [7]. Ideally, the Jensen-Shannon divergence between p_{data} and p_g is minimized as a result of optimizing Eq. (1) when $\mathbf{z} \sim p_z$ [11, 12].

Table 1: Notations

Notation	Description
$\mathbb{R}$	An original rating matrix
$\mathbb{R}^+$	An augmented rating matrix
$\mathbf{r_u}$, $\mathbf{r_i}$	Rating vectors of user u and item i
$\mathbf{m_u}$, $\mathbf{m_i}$	Missingness vectors of user u and item i
$\mathbf{c_u}$, $\mathbf{c_i}$	Condition vectors of user u and item i
$\mathcal{G}$, $\mathcal{D}$	A generator and a discriminator in the GAN framework
$\mathcal{G}_\mathcal{U}$, $\mathcal{D}_\mathcal{U}$	$\mathcal{G}$ and $\mathcal{D}$ learning rating distribution w.r.t. users
$\mathcal{G}_\mathcal{I}$, $\mathcal{D}_\mathcal{I}$	$\mathcal{G}$ and $\mathcal{D}$ learning rating distribution w.r.t. items
$\mathcal{G}^m_\mathcal{U}$, $\mathcal{D}^m_\mathcal{U}$	$\mathcal{G}$ and $\mathcal{D}$ learning missingness distribution w.r.t. users
$\mathcal{G}^m_\mathcal{I}$, $\mathcal{D}^m_\mathcal{I}$	$\mathcal{G}$ and $\mathcal{D}$ learning missingness distribution w.r.t. items
$\mathbf{x} \parallel \mathbf{y}$	Concatenation of two vectors $\mathbf{x}$ and $\mathbf{y}$
$\mathbf{x} \odot \mathbf{y}$	Element-wise multiplication of two vectors $\mathbf{x}$ and $\mathbf{y}$

Hence, the completely trained $\mathcal{G}$ is expected to generate realistic data.

2.1 Model Description and Training

Based on the original GAN aforementioned, we build our framework with taking the following properties into consideration:

- **Cold-start awareness (P1)**: One of our important goals is to alleviate the problems with the cold-start users and items. Therefore, it would be desirable for our model to be able to generate realistic neighbors of *specific* (i.e., cold-start) users and items in the original dataset if they are given to the model, rather than just randomly generating without any control under which users and items are generated.
- **Sparsity awareness (P2)**: Unlike the image data considered by the original GAN, which is represented as a *dense* vector, the users and items in a recommender system is typically a *sparse vector* having a large number of missing elements inside. This is because most of users typically rate only a small number of chosen items. In light of this fact, GAN in our framework should be able to consider such sparsity nature (i.e., *missingness*) when generating users and items.

Regarding property P1, we build our framework on the basis of *Conditional GAN* (CGAN) [26]: this extension to the original GAN allows $\mathcal{G}$ to produce data related to a given specific condition by feeding $\mathcal{G}$ a desired condition as an additional input in conjunction with the random noise input $\mathbf{z}$. Thus, its objective function is:

$$V(\mathcal{D},\mathcal{G}) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\ln\mathcal{D}(\mathbf{x}|\mathbf{c})] - \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\ln\mathcal{D}(\mathcal{G}(\mathbf{z}|\mathbf{c}))], \quad (2)$$

where $\mathbf{c}$ corresponds to a condition vector such as a one-hot vector of a specific class label (e.g., "cats" or "dogs" in image data) [19]. For property P2, we build *a pair of collaborative CGANs* to generate *users*, where one learns the *rating data distribution* and the other learns the *missingness distribution*. Also, we train another pair of CGANs to generate *items* in the same way.

Figure 2(a) describes the training of two pairs of CGANs where the above two CGANs and the rest two CGANs focus on users and items, respectively. For the sake of simplicity, this subsection mainly elaborates the details of our models focusing on users, and then will briefly explain how the user-based formulations change when focusing on items. Formally, based on Eq. (2), the objective function of our two CGANs for users, denoted as $V(\mathcal{D}_\mathcal{U}, \mathcal{D}^m_\mathcal{U}, \mathcal{G}_\mathcal{U}, \mathcal{G}^m_\mathcal{U})$, can be written as:

$$\begin{aligned} &V(\mathcal{D}_\mathcal{U}, \mathcal{D}^m_\mathcal{U}, \mathcal{G}_\mathcal{U}, \mathcal{G}^m_\mathcal{U}) \\ &\simeq \frac{1}{|\mathcal{B}_\mathcal{U}|}\Big(\sum_{u\in\mathcal{B}_\mathcal{U}} \big(\ln\mathcal{D}_\mathcal{U}(\mathbf{r_u} \parallel \mathbf{c_u}) + \ln\mathcal{D}^m_\mathcal{U}(\mathbf{m_u} \parallel \mathbf{c_u})\big) \quad (3) \\ &- \sum_{u\in\mathcal{B}_\mathcal{U}} \big(\ln(\mathcal{D}_\mathcal{U}(\mathcal{G}_\mathcal{U}(\mathbf{z}_\mathcal{U} \parallel \mathbf{c_u}) \odot \mathbf{m_u}) + \ln(\mathcal{D}^m_\mathcal{U}(\mathcal{G}^m_\mathcal{U}(\mathbf{z}_\mathcal{U} \parallel \mathbf{c_u}))))\big)\Big) \end{aligned}$$

where $\mathbf{z}_\mathcal{U}$ denotes a random noise vector for users, $\mathbf{r_u}$ denotes a real rating vector of a user u, $\mathbf{c_u}$ denotes u's condition vector, and $\mathcal{B}_\mathcal{U}$ does a minibatch of users to be conditioned. Most importantly, $\mathbf{m_u}$ corresponds to u's real *missingness vector* specifying whether a rating of u on item i is observed ($e_{ui} = 1$) or missing ($e_{ui} = 0$). Fed with the same input $\mathbf{z}_\mathcal{U}||\mathbf{c_u}$, $\mathcal{G}_\mathcal{U}$ and $\mathcal{G}^m_\mathcal{U}$ generate u's synthetic rating and missingness vectors, denoted as $\hat{\mathbf{r}}_\mathbf{u}$ and $\hat{\mathbf{m}}_\mathbf{u}$, respectively. Likewise, conditioned by the user-specific condition $\mathbf{c_u}$, both $\mathcal{D}_\mathcal{U}$ and $\mathcal{D}^m_\mathcal{U}$ are trained to discriminate u's real rating vector $\mathbf{r_u}$ and real missingness vector $\mathbf{m_u}$ from the fake data $\hat{\mathbf{r}}_\mathbf{u}$ and $\hat{\mathbf{m}}_\mathbf{u}$, respectively. All the generators and discriminators in our CGANs are implemented by DNN where the stochastic gradient descent (SGD) with minibatch and the back-propagation algorithm are employed to train them. Algorithm 1 shows the adversarial process among the generators $\mathcal{G}_\mathcal{U}$ and $\mathcal{G}^m_\mathcal{U}$ and the discriminators $\mathcal{D}_\mathcal{U}$ and $\mathcal{D}^m_\mathcal{U}$. As a result of the training, we expect $\mathcal{G}_\mathcal{U}$ and $\mathcal{G}^m_\mathcal{U}$ would capture the true distribution of ratings and missingness of users on items.

Another point worth mentioning is that, during the training of the two CGANs simultaneously, we multiply u's true missingness vector $\mathbf{m_u}$ with $\mathcal{G}_\mathcal{U}$'s output by $\mathcal{G}_\mathcal{U}(\mathbf{z}_\mathcal{U}||\mathbf{c_u}) \odot \mathbf{m_u}$, as shown in Eq. (3). The reason for this multiplication (i.e., "*masking*" $\mathcal{G}_\mathcal{U}$'s output) is to deal with the *sparse* nature of rating vectors that $\mathcal{G}_\mathcal{U}$ tries to mimic. By doing so, $\mathcal{G}_\mathcal{U}$'s output only on the observed ratings can contribute to the entire learning process of GAN and the gradients from the unobserved ratings are safely ignored. Such a *masking scheme* has been commonly employed by most of GANs for the data with missing values (e.g., in [5], [7], and [44]).

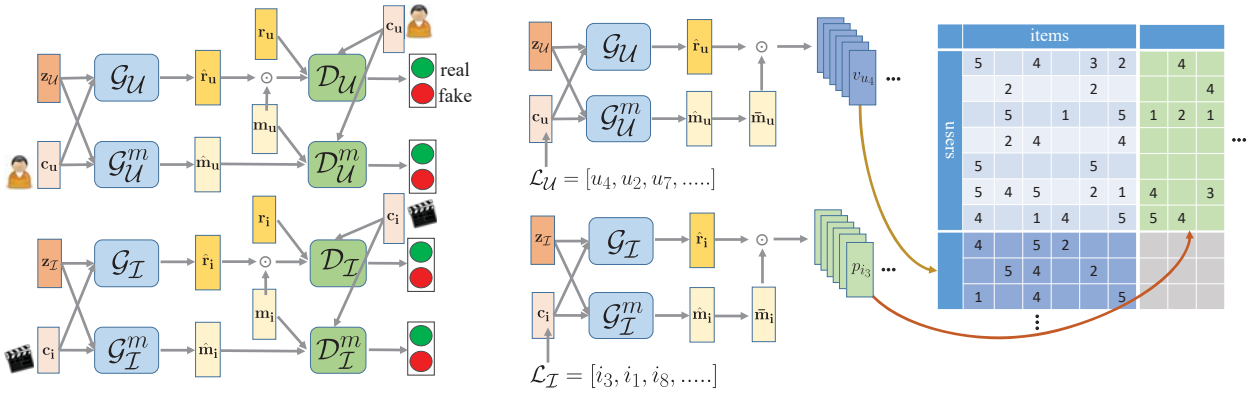
Until now, we have described our CGANs for learning the rating distribution and missingness distribution of *users*. We note that the aforementioned formulations can be easily extended to learning those of *items*. Similarly, we formulate the objective function of our CGANs for items, $V(\mathcal{D}_\mathcal{I}, \mathcal{D}^m_\mathcal{I}, \mathcal{G}_\mathcal{I}, \mathcal{G}^m_\mathcal{I})$, as follows:

$$\begin{aligned} &\frac{1}{|\mathcal{B}_\mathcal{I}|}\Big(\sum_{i\in\mathcal{B}_\mathcal{I}} \big(\ln\mathcal{D}_\mathcal{I}(\mathbf{r_i} \parallel \mathbf{c_i}) + \ln\mathcal{D}^m_\mathcal{I}(\mathbf{m_i} \parallel \mathbf{c_i})\big) \quad (4) \\ &- \sum_{i\in\mathcal{B}_\mathcal{I}} \big(\ln(\mathcal{D}_\mathcal{I}(\mathcal{G}_\mathcal{I}(\mathbf{z}_\mathcal{I} \parallel \mathbf{c_i}) \odot \mathbf{m_i}) + \ln(\mathcal{D}^m_\mathcal{I}(\mathcal{G}^m_\mathcal{I}(\mathbf{z}_\mathcal{I} \parallel \mathbf{c_i}))))\big)\Big), \end{aligned}$$

where $\mathbf{z}_\mathcal{I}$ denotes a random noise vector for items, $\mathcal{B}_\mathcal{I}$ denotes a minibatch of items, $\mathbf{c_i}$ denotes the condition vector that specifies an item i, and $\mathbf{r_i}$/$\mathbf{m_i}$ does i's true rating/missingness vector.

2.2 Generation and Augmentation

After the training of our CGANs is completed, we now ready to generate virtual neighbors of the real users and items in a dataset. Figure 2(b) describes the overview of the generation and augmentation process. As the first step, we need to choose whose neighbors will be generated among the entire users and items. Formally, let



(a) Training of our CGANs (b) Generation and augmentation

Figure 2: Overview of the AR-step. Two pairs of CGANs are trained, where the upper and the bottom pairs are tailored to generating users and items, respectively. The generated users and items are augmented to the rating matrix as additional rows and columns. Note that we do not consider interactions between the generated users and items.

Algorithm 1 LearnUserCGANs

Input: Rating matrix $\mathbb{R}$
Output: $\mathcal{G}_{\mathcal{U}}$ and $\mathcal{G}^m_{\mathcal{U}}$
Initialize the model parameters of $\mathcal{D}_{\mathcal{U}}$, $\mathcal{D}^m_{\mathcal{U}}$, $\mathcal{G}_{\mathcal{U}}$, and $\mathcal{G}^m_{\mathcal{U}}$
while not converged **do**
  for $\mathcal{G}$-step **do** ($\mathcal{D}_{\mathcal{U}}$ and $\mathcal{D}^m_{\mathcal{U}}$ are fixed here)
    Sample minibatch of users $\mathcal{B}_{\mathcal{U}}$
    Generate fake rating vectors $\{\hat{\mathbf{r}}_{(u)}\}, \forall u \in \mathcal{B}_{\mathcal{U}}$ by $\mathcal{G}_{\mathcal{U}}$ and fake missingness vectors $\{\hat{\mathbf{m}}_{(u)}\}, \forall u \in \mathcal{B}_{\mathcal{U}}$ by $\mathcal{G}^m_{\mathcal{U}}$
    Update $\mathcal{G}_{\mathcal{U}}$ and $\mathcal{G}^m_{\mathcal{U}}$ by *descending* the stochastic gradients of $V(\mathcal{D}_{\mathcal{U}}, \mathcal{D}^m_{\mathcal{U}}, \mathcal{G}_{\mathcal{U}}, \mathcal{G}^m_{\mathcal{U}})$
  end for
  for $\mathcal{D}$-step **do** ($\mathcal{G}_{\mathcal{U}}$ and $\mathcal{G}^m_{\mathcal{U}}$ are fixed here)
    Sample minibatch of users $\mathcal{B}_{\mathcal{U}}$
    Get their real rating vectors $\{\mathbf{r}_{(u)}\}, \forall u \in \mathcal{B}_{\mathcal{U}}$ and real missingness vectors $\{\mathbf{m}_{(u)}\}, \forall u \in \mathcal{B}_{\mathcal{U}}$
    Generate fake rating vectors $\{\hat{\mathbf{r}}_{(u)}\}, \forall u \in \mathcal{B}_{\mathcal{U}}$ by $\mathcal{G}_{\mathcal{U}}$ and fake missingness vectors $\{\hat{\mathbf{m}}_{(u)}\}, \forall u \in \mathcal{B}_{\mathcal{U}}$ by $\mathcal{G}^m_{\mathcal{U}}$
    Update $\mathcal{D}_{\mathcal{U}}$ and $\mathcal{D}^m_{\mathcal{U}}$ by *ascending* the stochastic gradients of $V(\mathcal{D}_{\mathcal{U}}, \mathcal{D}^m_{\mathcal{U}}, \mathcal{G}_{\mathcal{U}}, \mathcal{G}^m_{\mathcal{U}})$
  end for
end while
return $\mathcal{G}_{\mathcal{U}}$ and $\mathcal{G}^m_{\mathcal{U}}$

$\mathcal{L}_{\mathcal{U}}$ and $\mathcal{L}_{\mathcal{I}}$ be lists of users and items whose neighbors will be generated by the trained generators. Now, our question is who/what will be included in $\mathcal{L}_{\mathcal{U}}/\mathcal{L}_{\mathcal{I}}$. Naturally, the cold-start users and items that have a few ratings would be primarily considered. Meanwhile, the normal users and items having relatively sufficient ratings need to be considered as well since we also aim to improve entire users' overall satisfaction with recommendations.

To this end, we define *non-uniform probability distribution* for sampling over users and items as:

$$p(u) \propto \frac{1}{|\mathcal{I}_u|}, \quad p(i) \propto \frac{1}{|\mathcal{U}_i|}, \tag{5}$$

where $\mathcal{I}_u$ and $\mathcal{U}_i$ denote the sets of items rated by u and users having rated i, respectively. In other words, the users/items having less ratings available are more likely to be sampled and included in $\mathcal{L}_{\mathcal{U}}/\mathcal{L}_{\mathcal{I}}$. We believe this approach makes our framework mainly focus on the cold-start users and items while not completely ignoring normal users and items. We sample users and items *with replacement*, hence the same users and items can be listed multiple times. We denote δ_u and δ_i as pre-defined sizes of $\mathcal{L}_{\mathcal{U}}$ and $\mathcal{L}_{\mathcal{I}}$ (i.e., the numbers of virtual users and items to be generated), and the sampling is terminated when the numbers of users and items sampled reach to δ_u and δ_i, respectively. The values of δ_u and δ_i are chosen empirically; the impact of those hyper-parameters on the performance will be further analyzed in Section 5.

After the sampling, for each user u in $\mathcal{L}_{\mathcal{U}}$, we feed $\mathbf{z}_{\mathcal{U}} \| \mathbf{c_u}$ for $\mathcal{G}_{\mathcal{U}}$ and $\mathcal{G}^m_{\mathcal{U}}$ to generate $\hat{\mathbf{r}}_\mathbf{u}$ and $\hat{\mathbf{m}}_\mathbf{u}$. Owing to the random noise $\mathbf{z}_{\mathcal{U}}$ and u's specific condition $\mathbf{c_u}$, the generated $\hat{\mathbf{r}}_\mathbf{u}$ is expected to contain the predicted ratings on all the items that u's neighbor might have given if she did exist. Likewise, we expect $\hat{\mathbf{m}}_\mathbf{u}$ to contain the probability that each item might have been used by u's neighbor. Finally, we complete the user generation by making $\hat{\mathbf{r}}_\mathbf{u}$ as a sparse vector having ratings only on the items on which the neighbor might be the most likely to give ratings. This is done by (1) converting $\hat{\mathbf{m}}_\mathbf{u}$ to zero-one vector, denoted as $\bar{\mathbf{m}}_\mathbf{u}$, such that the top-$k$ highest values in $\hat{\mathbf{m}}_\mathbf{u}$ become 1 and the rest 0, and (2) multiplying $\hat{\mathbf{r}}_\mathbf{u}$ with $\bar{\mathbf{m}}_\mathbf{u}$. Generating the virtual items can be performed in a very similar way with the aforementioned process by using the generators $\mathcal{G}_{\mathcal{I}}$ and $\mathcal{G}^m_{\mathcal{I}}$, and we will skip its details for simplicity.

Here, the k value controls the number of ratings that the virtual users and items will have. We empirically observed that both too small and too large k values are not effective in improving the accuracy of recommendations: if k is too small, the generated neighbors would also have the cold-start problems, thereby not that much helpful in understanding the real cold-start users and items. In addition, the overall sparsity would become higher if such sparse rows and columns are augmented in the rating matrix. In contrast, if it is too large, the generated neighbors would not be similar with the real cold-start users and items, so CF models may

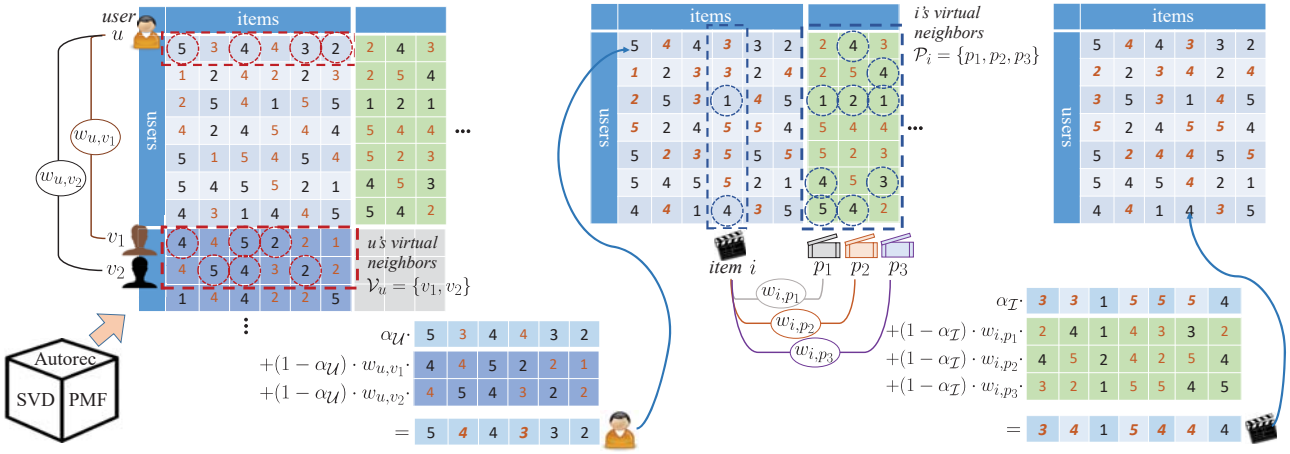


Figure 3: Illustration of the CF-step. Colored, and relatively small integers indicate the ratings predicted by CF. Bold-Italic integers indicate the refined ratings. The similarities are computed based only on the original ratings (circled with dotted lines), not those predicted nor refined.

have difficulty in learning the characteristics of the real cold-start users and items based on their neighbors. As a compromise, we empirically choose to set k as the average number of ratings of real users when generating virtual users, and the average number of ratings of real items when generating virtual items.

3 CF-STEP

Figure 3[1] presents the overview of this step. Firstly, we train an arbitrary CF model by using the augmented rating matrix $\mathbb{R}^+$ as training data, and then use the trained model to infer the empty cells in $\mathbb{R}^+$. Note that we now have the augmented rating matrix with the unknown ratings being predicted, including not only the ratings of real users on real items, but also those of real users on virtual items, and those of virtual users on real items. We do not predict the ratings of virtual users on virtual items since these ratings will not be used to produce the final recommendation results.

Next, we *refine* the predicted ratings of real users by exploiting their virtual neighbors' ratings. The implication behind this idea is similar to that of the traditional memory-based CF methods [37]. Since the virtual neighbors of a user (say, u) may have similar taste to that of u and the ratings of those neighbors on each target item (say, i) predicted by CF would be helpful in inferring u's rating on i more accurately. Similarly, u's predicted ratings on the virtual neighbors of i may also be beneficial to the inference since i and its neighbors may share lots of similarities in users' taste.

We first adjust the predicted ratings of *real users* by aggregating their corresponding neighbors' ratings, and then adjust those of *real items* by aggregating their corresponding neighbors' ratings. However, we will mainly describe our idea with focusing on a real user 'u' and a real item 'i' for simplicity. It can be easily generalized to any real user and any real item. First, we collect u's neighbors, denoted as $\mathcal{V}_u$, and compute the similarity between u and each neighbor v in $\mathcal{V}_u$ by using the *Pearson correlation coefficient* [37] below[2]:

$$w_{u,v} = \frac{\sum_{t \in I_{u,v}} (r_{ut} - \bar{r}_u)(r_{vt} - \bar{r}_v)}{\sqrt{\sum_{t \in I_{u,v}} (r_{ut} - \bar{r}_u)^2} \sqrt{\sum_{t \in I_{u,v}} (r_{vt} - \bar{r}_v)^2}}, \quad (6)$$

where $I_{u,v}$ indicates intersection, i.e., a set of items that both u and v have rated, r_{ui} does user u's rating to item i, and $\bar{r}_u$ does the average rating of u on the items included in $I_{u,v}$. Then, u's refined rating on i is computed by the weighted average of u's predicted rating $\hat{r}_{ui}$ and her neighbors' ratings, as follows:

$$\tilde{r}_{ui} = \alpha_{\mathcal{U}} \cdot \hat{r}_{ui} + (1 - \alpha_{\mathcal{U}}) \cdot \sum_{v \in \mathcal{V}_u} w_{u,v} \cdot \hat{r}_{vi} \quad (7)$$

where $\alpha_{\mathcal{U}}$ is a tunable parameter that controls the importance of virtual users' ratings in this refinement process. If $\alpha_{\mathcal{U}}$=1 as an extreme case, the above equation actually does not consider virtual users' ratings at all; if $\alpha_{\mathcal{U}}$=0, it only relies on virtual users' ratings, ignoring the real user's rating.

After adjusting the predicted ratings of all the real users, we now focus on adjusting real items' predicted ratings. Similarly, we collect i's neighbors, denoted as $\mathcal{P}_i$, and then refine $\tilde{r}_{ui}$ again based on the virtual items in $\mathcal{P}_i$, as follows:

$$\tilde{\tilde{r}}_{ui} = \alpha_{\mathcal{I}} \cdot \tilde{r}_{ui} + (1 - \alpha_{\mathcal{I}}) \cdot \sum_{p \in \mathcal{P}_i} w_{i,p} \cdot \hat{r}_{up} \quad (8)$$

where $\alpha_{\mathcal{I}}$ controls the importance of virtual items' ratings and $w_{i,p}$ denotes the similarity between real item i and each of i's neighbors, p. After adjusting all the real items' predicted ratings, we finally provide each real user with a subset of real items whose ratings refined through the aforementioned processes are the highest.

4 RELATED WORK

Data imputation has been the most successful technique to solve the cold-start problems in the area of pure CF. Among many others, *Zero-Injection* [17, 22] has shown the state-of-the-art performance.

[1] Actually, all the generated and predicted ratings are with floating-point values. However, in Figures 2 and 3, we represent them with the integer values for simplicity.

[2] Other similarity metrics such as *Cosine* [37] can be applied here.

It carefully finds so-called *uninteresting items* that each user has not rated yet but is unlikely to prefer even if recommended to her. Then, it injects *zero ratings* to the identified uninteresting items as her negative preferences. Very recently, several imputation methods employ the generator in GAN as an imputation model for missing data. Chae et al. [7] proposed *Rating Augmentation GAN* (RAGAN) that aims at exploiting GAN to generate plausible ratings to be imputed while resolving the inherent *selection bias* of the rating data. *Generative Adversarial Imputation Nets* (GAIN) [44] and MisGAN [24] are also well-designed data imputation methods based on GAN, even though their domain is not recommender systems. They assume that the data is *missing completely at random* (MCAR) [15, 35] and build their models on the basis of this philosophy.

Among the contents-based or the hybrid recommender systems, AugCF [41] and RSGAN [45] are relevant to our work. In the AugCF framework, $\mathcal{G}$ outputs the most plausible item for a given user and a given class (e.g., *like*/*dislike*) with the help of *side information* associated with the users and items, and $\mathcal{D}$ distinguishes whether the generated item is fake or real. RSGAN aims at generating a target user's social friends who would be reliable and able to bring better recommendation performance. In our future work, we plan to extend our AR-CF to deal with various types of side information and compare it with hybrid recommenders.

5 EVALUATION

This section reports and analyzes the results of our extensive experiments, which are carefully designed to answer the following key questions:

- **Q1**: Are the generated neighbors beneficial for resolving the cold-start problems?
- **Q2**: How does AR-CF perform compared with the state-of-the-arts for all users and items?
- **Q3**: How does AR-CF perform according to different values of key hyper-parameters?
- **Q4**: How much do the generated users and items influence the scalability of AR-CF?

5.1 Experimental Settings

5.1.1 Datasets. We used four real-world datasets: Movielens 100K, Movielens 1M[3], Watcha[4], and CiaoDVD [38] datasets, whose detailed statistics are summarized in Table 2. For each dataset, following [14, 39], we chronologically split the ratings into two subsets: the first 80% for training and the remaining 20% for testing [5]. Among the ratings in the test set, we considered the ratings of 4 and 5 as our ground truth.

5.1.2 Implementation details. As mentioned, one of the most important advantages of AR-CF is that it can be performed on top of any existing data imputation methods. Once an arbitrary imputation method produces the rating matrix with a specific amount of missing cells imputed, which we denote by $\check{\mathbb{R}}$, then we can simply run AR-CF on $\check{\mathbb{R}}$ as if we run it on the original rating matrix $\mathbb{R}$. Since the data imputation methods have been very successful in improving the accuracy of CF models, our AR-CF can enjoy such performance gains without requiring any modification. We are aware of several successful imputation methods summarized in Section 4. Among them, we chose Zero-Injection (ZI in short, hereafter) and applied it to $\mathbb{R}$ before running AR-CF, since ZI is simple but fast and has achieved state-of-the-art accuracy in the literature. It is also less sensitive to its hyper-parameters [17, 22] and its code is publicly available. However, we note any other imputation methods can be applied here, and we remain AR-CF's collaborating with various imputation methods for our future work.

[3]https://grouplens.org/datasets/movielens/
[4]http://watcha.net
[5]However, since Watcha does not have the timestamp information, we split only its ratings in a random manner.

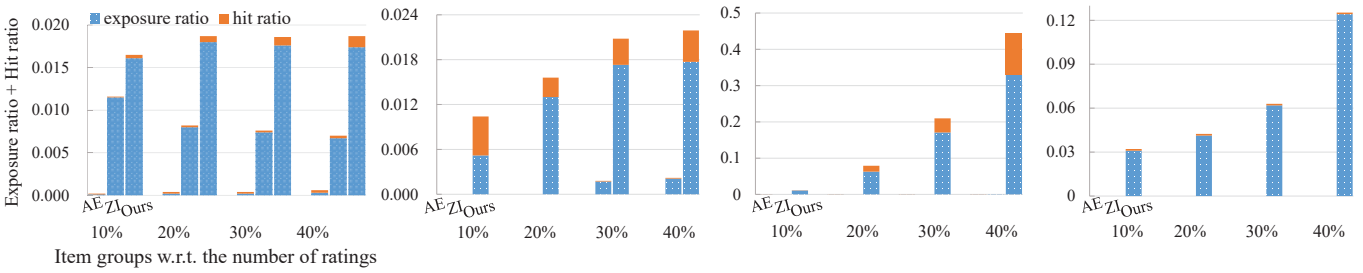
Table 2: Dataset statistics

Datasets	# users	# items	# ratings	Sparsity
Movielens 100K	943	1,682	100,000	93.69%
Watcha	1,391	1,927	101,073	96.98%
Movielens 1M	6,039	3,883	1,000,209	95.72%
CiaoDVD	7,628	15,536	62,358	99.94%

We performed the grid search to find the optimal values of hyper-parameters in our AR-CF. For the generators and discriminators in our four CGANs, we tested a different number of hidden layers with {1, 2, 3, 4, 5} and a different number of hidden nodes per hidden layer with {300, 400, 500, 600, 700}. We fixed the dimensionality of the random noise vector $\mathbf{z}$ as 128. For training the CGANs, the size of the minibatch was varied with {64, 128, 192, 256} and a learning rate with {0.005, 0.001, 0.0005, 0.0001}. We chose the numbers of virtual users and items, $\delta_{\mathcal{U}}$ and $\delta_{\mathcal{I}}$, by using the ratio to the entire number of real users (say, m) and real items (say, n) in a dataset, and varied them with {0.25, 0.5, 0.75, 1.0, 1.5, 2.0}; if $\delta_{\mathcal{U}} = 0.5$, for example, (int) $m \times 0.5$ users will be generated. We also varied the importance parameters, $\alpha_{\mathcal{U}}$ and $\alpha_{\mathcal{I}}$, with {0, 0.2, 0.4, 0.6, 0.8, 1.0}. We employed SVD [18] and AutoRec [32] as our CF models; we tested AutoRec's number of hidden layers with {1, 2, 3, 4} and its number of hidden nodes per hidden layer with {200, 300, 400, 500}; we tested SVD's number of latent factors with {20, 40, 60, 80, 100}. For training the CF models, we used the size of the minibatch with {64, 128, 256, 512} and the learning rate with {0.001, 0.0005, 0.0001}. For all the CGANs and CF models, we fixed their L2 regularization coefficients as 0.001. Following [7], we use a user's (or, an item's) rating vector as user-specific (or, item-specific) condition.

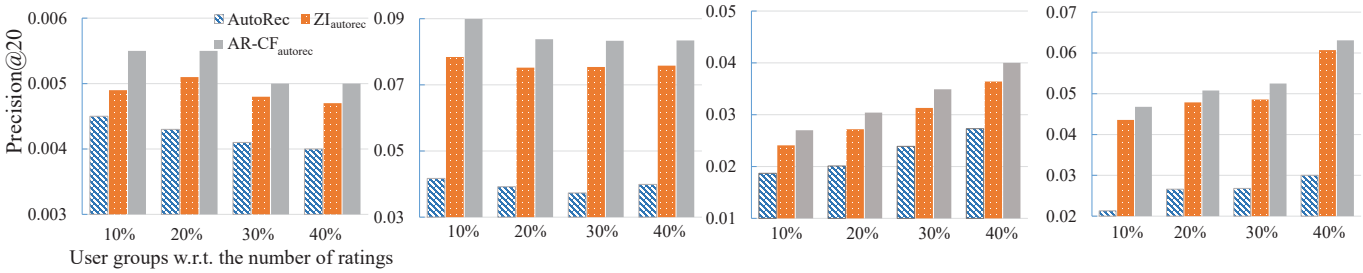
5.1.3 Competitors. We employed (1) two widely-used baseline CF methods of SVD [18] and AutoRec [32], (2) four CF methods based on state-of-the-art data imputation algorithms of Zero-Injection (ZI) [17], RAGAN [7], GAIN [44], and MisGAN [24], and (3) four state-of-the-art top-N recommenders of PureSVD [10], CDAE [43], CFGAN [5], and NGCF [42]. SVD is a standard MF-based model along with user and item bias terms [18]. AutoRec is an Autoencoder-based, non-linear latent factor model for CF. We already explained ZI, RAGAN, GAIN, and MisGAN in Section 4. For these imputation methods, we also employed SVD and AutoRec to provide the final recommendations after they completed their own imputation algorithms on $\mathbb{R}$. PureSVD, CDAE (*Collaborative De-noising Autoencoder*), CFGAN, and NGCF (*Neural Graph CF*) are the state-of-the-art recommender models based on SVD, Autoencoder, GAN, and GCN (*Graph Convolutional Networks*) [1], respectively.

For each competing method, we tested various values for its hyper-parameters (e.g., the numbers of hidden nodes and hidden



(a): CiaoDVD (b): Watcha (c): Movielens 1M (d): Movielens 100K

Figure 4: Exposure/hit ratio of cold-start items.



(a): CiaoDVD (b): Watcha (c): Movielens 1M (d): Movielens 100K

Figure 5: Accuracy of recommendation to cold-start users.

layers of $\mathcal{G}$ and $\mathcal{D}$ in the GAN-based models and the number of latent factors in the MF-based models) and selected a set of values that provide the highest accuracy. Since some of our competitors such as CFGAN and NGCF are tailored to perform the *one-class collaborative filtering* task [27, 33], which can be regarded as the special case of this paper's multi-class setting, we binarized $\mathbb{R}$ only when running them.

5.1.4 Evaluation metrics. We employed *precision*, *recall*, *normalized discounted cumulative gain* (nDCG), and *mean reciprocal rank* (MRR) for evaluating our model's recommendation accuracy [5, 7, 21]. *Precision* and *recall* focus on how many correct items are included in the recommendation list while *nDCG* and *MRR* account for the ranked positions of correct items in the recommendation list.

5.2 Experimental Results

5.2.1 Q1: Effectiveness on the cold-start problems. The problem in the cold-start items is that they are very difficult to be recommended, and thus rarely selected by users as well, as illustrated in Figure 1(b). We quantitatively measured such problems by defining the notions of "*exposure ratio*" and "*hit ratio*": when a CF model recommends top-N items to users, the exposure ratio reveals how many cold-start items are exposed (i.e., included in the top-N recommendation list) to users, and the hit ratio does how many those items are actually chosen by users. Formally, the exposure ratio is computed by B/A where B is the number of cold-start items which are exposed to at least one user, and A is the number of the entire cold-start items. The hit ratio is computed by C/A where C is the number of cold-start items which are recommended and *actually chosen* by at least one user (i.e., included in at least one user's ground truth).

Figure 4 reports each method's results of exposure ratio and hit ratio on each dataset. In each graph, $X\%$ in the x-axis indicates that the bottom-X percent of items according to the number of ratings were defined as cold-start items. AutoRec (AE in short, here) and $\text{ZI}_{\text{autorec}}$ (ZI in short, here) were compared with ours in this experiment. Since the results on this task provided by other methods are similar with or not on par with those of $\text{ZI}_{\text{autorec}}$, we omit their results in Figure 4 due to space limitations. We observed that AutoRec has difficulty in understanding the cold-start items, thereby recommending none or very few of them to users on all the four datasets. Even after the data imputation with ZI, $\text{ZI}_{\text{autorec}}$ still failed to exhibit at least one cold-start item to at least one user on many cases (e.g., dealing with bottom-10% and 20% cold-start items on Watcha, and all cases on Movielens 100K). However, after AR-CF generated their neighbors, we can observe that AR-$\text{CF}_{\text{autorec}}$ pushed a meaningful number of cold-start items to the top-N recommendation lists to users. Moreover, we can see that some of the recommended items are correct recommendation, i.e., they are actually chosen and highly rated by users. These results demonstrate the effectiveness of our AR-CF in resolving the cold-start item problems: by generating their realistic neighbors, AR-CF can "promote" the cold-start items and even result in purchases *without requiring any cost or human powers*.

Next, Figure 5 highlights the recommendation accuracy of our AR-CF to the cold-start users in terms of *precision@20*; the other metrics exhibited very similar tendency, hence they are omitted due to space limitations. We defined the cold-start users as the bottom-X percent of users according to their number of ratings, and then tried different X values; AutoRec, $\text{ZI}_{\text{autorec}}$, and AR-$\text{CF}_{\text{autorec}}$ are compared as well. The experimental results demonstrate that our AR-CF is really effective in solving the cold-start user problem. For example, AR-$\text{CF}_{\text{autorec}}$ performs well to the bottom-10% cold-start users on Watcha compared to $\text{ZI}_{\text{autorec}}$ (12.8% improved) and on Ciao (11% improved). It is also worth mentioning that AR-CF

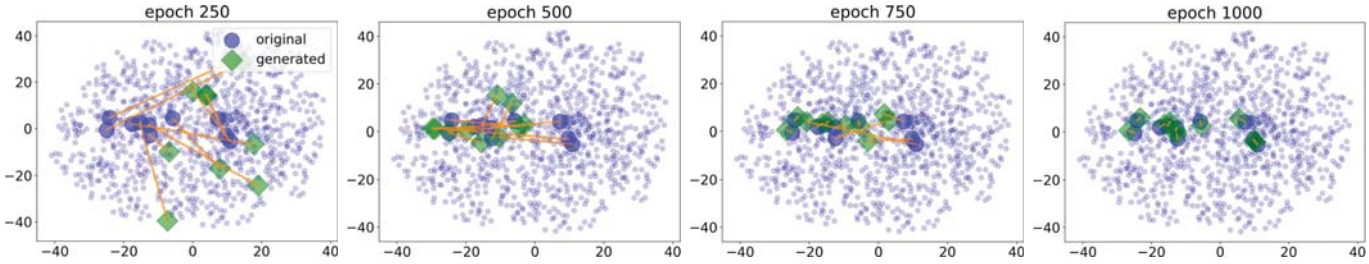


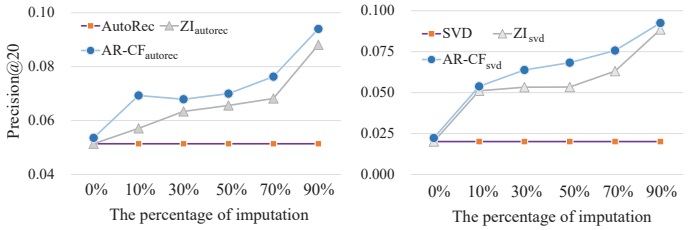
Figure 6: Visualization of the real users and virtual neighbors at specific epochs. Some selected users and their corresponding virtual neighbors are connected each other and are represented by the large circles and rhombuses, respectively.

Table 3: Comparison results. The best accuracy on each metric among competing methods is highlighted with bold face.

datasets	Movielens 100K				Movielens 1M				Watcha				CiaoDVD			
metrics	Pre.	Rec.	nDCG	MRR	Pre.	Rec.	nDCG	MRR	Pre.	Rec.	nDCG	MRR	Pre.	Rec.	nDCG	MRR
SVD	.0529	.0284	.0644	.1389	.0507	.0142	.0466	.0777	.0664	.0536	.0777	.1490	.0068	.0274	.0180	.0169
AutoRec	.0897	.0432	.0999	.1876	.0568	.0206	.0651	.1380	.0636	.0539	.0754	.1433	.0069	.0278	.0181	.0169
PureSVD	.1453	.1019	.1673	.2855	.0953	.0311	.0997	.1765	.1187	.0893	.1387	.2610	.0081	.0330	.0230	.0217
CDAE	.1411	.0843	.1561	.2608	.0856	.0410	.0922	.1722	.0968	.0685	.1057	.1981	.0069	.0286	.0188	.0176
CFGAN	.1451	.0826	.1610	.2755	.0904	.0398	.0972	.1785	.1015	.0696	.1100	.2005	.0072	.0296	.0188	.0173
NGCF	.1462	.0924	.1630	.2762	.0911	.0433	.0967	.1742	.1056	.0735	.1177	.2169	.0086	.0352	.0242	.0230
$\text{MisGAN}_{\text{svd}}$	.1643	.1032	.1891	.3131	.1028	.0390	.1096	.1953	.1231	.0922	.1428	.2634	.0071	.0295	.0188	.0173
$\text{MisGAN}_{\text{autorec}}$	.1627	.1034	.1872	.3125	.0994	.0389	.1059	.1898	.1269	.0951	.1456	.2633	.0071	.0318	.0210	.0189
GAIN_{svd}	.1614	.1050	.1864	.3109	.0991	.0372	.1054	.1895	.1210	.0908	.1398	.2551	.0089	.0355	.0245	.0229
$\text{GAIN}_{\text{autorec}}$	.1592	.0987	.1839	.3108	.0943	.0331	.1016	.1895	.1273	.0973	.1459	.2636	.0085	.0350	.0239	.0221
ZI_{svd}	.1689	.1124	.1936	.3181	.1039	.0378	.1103	.1969	.1248	.0938	.1463	.2700	.0093	.0371	.0261	.0250
$\text{ZI}_{\text{autorec}}$	.1656	.1084	.1900	.3173	.1037	.0397	.1098	.1957	.1260	.0924	.1472	.2681	.0097	.0405	.0274	.0252
$\text{RAGAN}_{\text{svd}}$	.1596	.1003	.1855	.3141	.1015	.0424	.1091	.1951	.1272	.0946	.1479	.2725	.0099	.0377	.0269	.0263
$\text{RAGAN}_{\text{autorec}}$	.1663	.1045	.1911	.3140	.1043	.0470	.1116	.1973	.1231	.0927	.1452	.2681	.0096	.0392	.0257	.0236
$\text{AR-CF}_{\text{svd}}$	.1678	.1077	**.1971**	**.3332**	**.1052**	**.0482**	**.1133**	**.2038**	.1319	.0973	.1527	.2738	.0103	.0417	.0288	**.0274**
$\text{AR-CF}_{\text{autorec}}$	**.1707**	**.1127**	.1954	.3254	.1047	.0455	.1120	.2005	**.1358**	**.1016**	**.1548**	**.2767**	**.0106**	**.0448**	**.0290**	.0267

is especially beneficial for the 10% and 20% cold-start users (i.e., "extremely" cold-start users). We believe such results came from the idea of our AR-CF that carefully generates the realistic neighbors of cold-start users to help the CF models understand them better, rather than simply filling the missing cells in $\mathbb{R}$ as other imputation methods did.

Here, we further examined whether our CGANs were well-trained so that the generated neighbors are highly plausible. Figure 6 visualizes the users (i.e., user rating vectors) in Movielens 100K by employing UMAP (*Uniform Manifold Approximation and Projection*) [25], one of the most popular visualization tools. In each plot, all the real users are projected on a shared 2D space at a specific epoch. Also, we chose 10 real users and visualized them and their virtual neighbors as well. As shown in the plot at epoch 250, our CGANs seemed to be under-trained because the generated virtual neighbors are quite far from their corresponding real users. However, we can observe that the distance between a pair of users tends to get closer as our CGANs are trained more. Finally, at 1000 epoch, we see that each pair of real and virtual users are placed very closely to each other in the space. Owing to these virtual (but plausible) neighbors, the CF models could understand the cold-start users much better and provide satisfactory recommendation to them. We note that we obtained very similar visualizations when training the CGANs



(a) AutoRec, $\text{ZI}_{\text{autorec}}$, and $\text{AR-CF}_{\text{autorec}}$ (b) SVD, ZI_{svd}, and $\text{AR-CF}_{\text{svd}}$

Figure 7: Accuracy w.r.t. the amount of imputation.

for items and also on the other datasets, but omit them due to the limited space.

5.2.2 Q2: Accuracy comparisons with state-of-the-arts for all users and items. The results shown in Table 3 aim to confirm that AR-CF is also effective in the general recommendation setting. We only show top-5 recommendation results and omit the top-20 results due to space limitation; they exhibited very similar trend. We observed that (1) even though NGCF, CFGAN, and CDAE relied on the limited information (i.e., binarized $\mathbb{R}$), they performed better than expected; especially NGCF outperformed several imputation-based methods

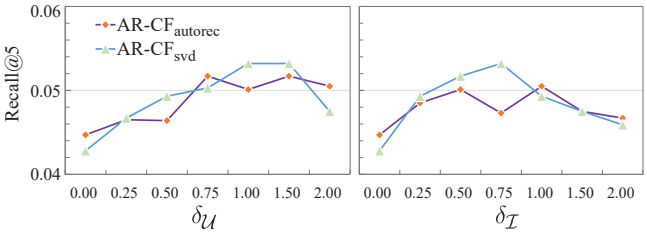


Figure 8: Sensitivity analysis on $\delta_{\mathcal{U}}$ and $\delta_{\mathcal{I}}$.

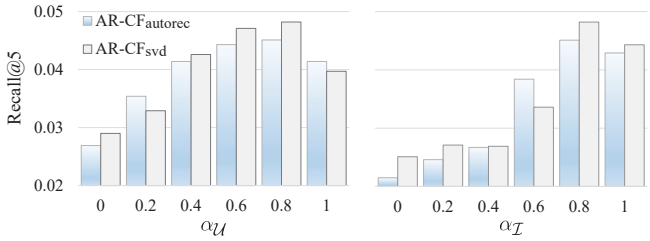


Figure 9: Impact of $\alpha_{\mathcal{U}}$ and $\alpha_{\mathcal{I}}$ on accuracy.

on Ciao. (2) The imputation-based methods performed much better than the CF models without imputation. This is because the data imputation relieves the data sparsity problem in $\mathbb{R}$ so that it makes CF models provide more qualified recommendation results. (3) Our AR-CF universally and consistently provided the best accuracy compared with all the imputation methods as well as the basic CF models, and we believe that the benefits are credited to AR-CF's characteristics that it can take the advantage of the performance gains coming from the data imputation and can exploit the virtual (but plausible) users and items as (additional) qualified training data. Through this observation, we can confirm that AR-CF is not only tailored to remedying the cold-start users and items, but is also effective in improving entire users' overall satisfaction with recommendations.

Also, we further examined the performance of ours with ZI in more detail, since we ran our framework on top of it. Figure 7 reports the accuracy (the y-axis) of ZI and ours equipped with AutoRec and SVD, according to the different percentages of missing cells in $\mathbb{R}$ imputed by ZI (the x-axis). As shown in the figure, ZI significantly improves original AutoRec and SVD; while enjoying this significant improvement by ZI, our AR-CF further improves the accuracy by augmenting additional users and items to $\mathbb{R}$. These results are consistent regardless of how may missing cells are imputed and which CF models are equipped.

5.2.3 Q3: Impact of key hyper-parameters. All the hyper-parameters used in AR-CF are shown in Section 5.1.2, each of which would affect the recommendation accuracy. Among them, this subsection investigates the impact of the following hyper-parameters, which are unique in our AR-CF and the most important in each of AR-step and CF-step: (1) $\delta_{\mathcal{U}}$ and $\delta_{\mathcal{I}}$, the numbers of virtual users and items to be generated, and (2) $\alpha_{\mathcal{U}}$ and $\alpha_{\mathcal{I}}$, the importance parameters used in the rating refinement process. The results of only *recall@5* on Movielens 1M are shown due to space limitations: those of the other metrics and those on the other datasets showed similar trend.

Figure 8 reports the accuracy of AR-CF$_{\text{autorec}}$ with regards to varying $\delta_{\mathcal{U}}$ and $\delta_{\mathcal{I}}$. Here, we fixed $\alpha_{\mathcal{U}} = \alpha_{\mathcal{I}} = 0.8$. We empirically found that the moderate range of $\delta_{\mathcal{U}}$ and $\delta_{\mathcal{I}}$ values would be in $[0.5, 1.5]$. If they are set with smaller values (e.g., less than 0.5), there won't be enough numbers of virtual users and items, so they are not that much influential. In contrast, if they are set with larger values (e.g., more than 1.5), CF models would pay too much attention to understanding the virtual users and items, rather than achieving its original goal of capturing preferences of real users on real items. Hence, $\delta_{\mathcal{U}}$ and $\delta_{\mathcal{I}}$ are set to 1.0 and 0.75, respectively; we believe it would provide a good balance in focus between virtual and real users (resp. items).

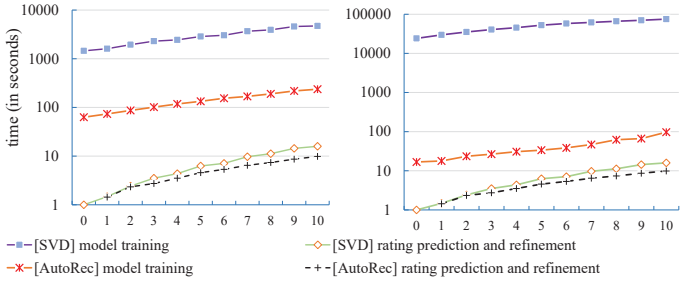
Figure 9 shows the accuracy over varying $\alpha_{\mathcal{U}}$ and $\alpha_{\mathcal{I}}$. Here, the two extreme cases (i.e., $\alpha_{\mathcal{U}} = \alpha_{\mathcal{I}} = 1$ and $\alpha_{\mathcal{U}} = \alpha_{\mathcal{I}} = 0$) provided lower accuracy, which implies the necessity of using both the real ratings as well as the generated ratings in our rating refinement step. We observe that the accuracy grows until $\alpha_{\mathcal{U}}$ and $\alpha_{\mathcal{I}}$ reach around 0.8. These results show that focusing more on a real (target) user's ratings is the most important in recommending items to her, but her neighbors' ratings as well as each target item's neighbors' ratings are still important and are worth referring to.

5.2.4 Q4: Scalability analysis. We analyze the scalability of our AR-CF because it involves the generated users and items that require additional computations in model training, rating prediction, and rating refinement. We tested it on CiaoDVD and Movielens 1M since CiaoDVD has the largest numbers of users and items and Movielens 1M has the largest number of ratings. For each dataset, we generated 500 users and 500 items and augmented them to $\mathbb{R}$. Then, we trained AutoRec and SVD to predict the unknown ratings and finally went through the rating refinement process. We repeated this experiment 10 times, where each iteration *doubles* the number of users and items to be generated and augmented. We measured the time taken by the two major computations (i.e., (1) model training and (2) rating prediction and refinement after the model training) in each iteration. Our experimental results are summarized in Figure 10. In each graph, the x-axis indicates each iteration, and the y-axis does computation time taken by the aforementioned tasks (in log scale). We ran all our experiments on a single machine equipped with an i9 7700K Intel CPU, 64GB RAM, and NVIDIA TITAN XP GPU.

We can observe that the rating prediction and refinement were performed very fast compared with the model training. In both datasets, the model training taken by each iteration increases *linearly* according to the increasing number of users and items augmented. Meanwhile, AutoRec performed much faster than SVD because AutoRec's training can be easily parallelized by fast GPU. The training of SVD took much more time in Movielens 1M since it has a much more number of ratings than CiaoDVD. We can conclude that AR-CF is *scalable* to the augmented users and items, requiring linear time in the number of users and items generated. Also, using AutoRec or any DNN-based model as CF would require a smaller time for additional computations thanks to the effective parallelism by using the GPU.

6 CONCLUDING REMARKS

In this paper, we proposed AR-CF, a novel framework for addressing the well-known cold-start problems. The AR-step in our framework



(a) CiaoDVD (b) Movielens 1M

Figure 10: Scalability analysis.

carefully trains four different CGANs to generate virtual, but realistic neighbors of cold-start users and items. The generated neighbors are augmented to the original rating matrix as additional rows (i.e., users) and columns (i.e., items). Then, in our CF-step, we proposed a rating refinement process that exploits virtual neighbors' ratings to adjust their corresponding real users/items' ratings. Through our extensive experiments, we demonstrated that AR-CF is really effective in dealing with the cold-start problems: it significantly improved the accuracy of recommendation to the cold-start users, and also made a meaningful number of the cold-start items to be displayed in top-N lists of users. Moreover, it performed the best among the recently proposed, state-of-the-art data imputation methods. Its sensitivity to several key-hyper parameters and scalability to the generated users and items are also analyzed accordingly. Indeed, AR-CF opens up a novel way to deal with the cold-start problems effectively via going outside the box of conventional data imputation approaches and working on top of them.

ACKNOWLEDGMENTS

This research was supported by (1) Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (NRF-2017M3C4A7069440), (2) the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2020R1A2B5B03001960), and (3) Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (No. NRF-2017M3C4A7083678).

REFERENCES

[1] R. Berg, T. N. Kipf, and M. Welling. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* (2017).
[2] R. Burke. 2002. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction* 12, 4 (2002), 331–370.
[3] D.-K. Chae, S.-W. Kim, and J.-T. Lee. 2019. Autoencoder-based personalized ranking framework unifying explicit and implicit feedback for accurate top-N recommendation. *Knowledge-Based Systems* 176 (2019), 110–121.
[4] D.-K. Chae, J. A. Shin, and S.-W. Kim. 2019. Collaborative adversarial autoencoders: An effective collaborative filtering model under the GAN framework. *IEEE Access* 7 (2019), 37650–37663.
[5] D.-K. Chae et al. 2018. CFGAN: A generic collaborative filtering framework based on generative adversarial networks. In *ACM CIKM*. 137–146.
[6] D.-K. Chae et al. 2018. On identifying k-nearest neighbors in neighborhood models for efficient and effective collaborative filtering. *Neurocomputing* 278 (2018), 134–143.
[7] D.-K. Chae et al. 2019. Rating augmentation with generative adversarial networks towards accurate collaborative filtering. In *WWW*. 2616–2622.
[8] K.-J. Cho et al. 2019. No, that's not my feedback: TV show recommendation using watchable interval. In *IEEE ICDE*. 316–327.
[9] E. Choi et al. 2017. Generating multi-label discrete electronic health records using generative adversarial networks. *arXiv preprint arXiv:1703.06490* (2017).
[10] P. Cremonesi, Y. Koren, and R. Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *ACM RecSys*. 39–46.
[11] C. Donahue, J. McAuley, and M. Puckette. 2018. Synthesizing audio with generative adversarial networks. *arXiv preprint arXiv:1802.04208* (2018).
[12] I. Goodfellow et al. 2014. Generative adversarial nets. In *NIPS*. 2672–2680.
[13] R. He and J. McAuley. 2016. VBPR: visual bayesian personalized ranking from implicit feedback. In *AAAI*.
[14] X. He et al. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *ACM SIGIR*. 549–558.
[15] J. M. Hernández-Lobato, N. Houlsby, and Z. Ghahramani. 2014. Probabilistic matrix factorization with non-random missing data. In *ICML*. 1512–1520.
[16] D.-G. Hong et al. 2019. CrowdStart: Warming up cold-start items using crowdsourcing. *Expert Systems with Applications* 138 (2019).
[17] W.-S. Hwang et al. 2016. "Told you i didn't like it": Exploiting uninteresting items for effective collaborative filtering. In *IEEE ICDE*. 349–360.
[18] Y. Koren, R. Bell, and C. Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).
[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.
[20] Y.-C. Lee, S.-W. Kim, and D. Lee. 2018. gOCCF: Graph-theoretic one-class collaborative filtering based on uninteresting items. In *AAAI*.
[21] J. Lee et al. 2016. Improving the accuracy of top-N recommendation using a preference model. *Information Sciences* 348 (2016), 290–304.
[22] J. Lee et al. 2019. l-Injection: Toward effective collaborative filtering using uninteresting items. *IEEE TKDE* 31, 1 (2019), 3–16.
[23] Y. Lee et al. 2018. How to impute missing ratings?: Claims, solution, and its application to collaborative filtering. In *WWW*. 783–792.
[24] S. C.-X. Li, B. Jiang, and B. Marlin. 2019. MisGAN: Learning from incomplete data with generative adversarial networks. *arXiv preprint arXiv:1902.09599* (2019).
[25] L. McInnes et al. 2018. UMAP: Uniform manifold approximation and projection. *Journal of Open Source Software* 3, 29 (2018), 861.
[26] M. Mirza and S. Osindero. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784* (2014).
[27] R. Pan et al. 2008. One-class collaborative filtering. In *IEEE ICDM*. 502–511.
[28] C. Park et al. 2016. Improving top-k recommendation with truster and trustee relationship in user trust network. *Information Sciences* 374 (2016), 100–114.
[29] A. Radford, L. Metz, and S. Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).
[30] Y. Ren et al. 2012. The efficient imputation method for neighborhood-based collaborative filtering. In *ACM CIKM*. 684–693.
[31] Y. Ren et al. 2013. AdaM: Adaptive-maximum imputation for neighborhood-based collaborative filtering. In *IEEE/ACM ASONAM*. 628–635.
[32] S. Sedhain et al. 2015. Autorec: Autoencoders meet collaborative filtering. In *WWW*. 111–112.
[33] S. Sedhain et al. 2016. On the effectiveness of linear models for one-class collaborative filtering.. In *AAAI*. 229–235.
[34] B. Settles. 2009. *Active learning literature survey*. Technical Report. University of Wisconsin-Madison Department of Computer Sciences.
[35] H. Steck. 2010. Training and testing of recommender systems on data missing not at random. In *KDD*. 713–722.
[36] F. Strub and J. Mary. 2015. Collaborative filtering with stacked denoising autoencoders and sparse inputs. In *NIPS workshop on machine learning for eCommerce*.
[37] X. Su and T. M. Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in artificial intelligence* 2009 (2009), 4.
[38] J. Tang, H. Gao, and H. Liu. 2012. mTrust: Discerning multi-faceted trust in a connected world. In *WSDM*. 93–102.
[39] M. Wan et al. 2017. Modeling consumer preferences and price sensitivities from large-scale grocery shopping transaction logs. In *WWW*. 1103–1112.
[40] H. Wang, N. Wang, and D.-Y. Yeung. 2015. Collaborative deep learning for recommender systems. In *KDD*. 1235–1244.
[41] Q. Wang et al. 2019. Enhancing collaborative filtering with generative augmentation. In *KDD*. 548–556.
[42] X. Wang et al. 2019. Neural graph collaborative filtering. In *ACM SIGIR*. 165–174.
[43] Y. Wu et al. 2016. Collaborative denoising auto-encoders for top-N recommender systems. In *WSDM*. 153–162.
[44] J. Yoon, J. Jordon, and M. Schaar. 2018. GAIN: Missing data imputation using generative adversarial nets. In *ICML*. 5675–5684.
[45] J. Yu et al. 2019. Generating reliable friends via adversarial training to improve social recommendation. In *IEEE ICDM*. 768–777.