Meta Matrix Factorization for Federated Rating Predictions

Yujie Lin
Shandong University
Qingdao, China
yu.jie.lin@outlook.com

Pengjie Ren*
University of Amsterdam
Amsterdam, The Netherlands
p.ren@uva.nl

Zhumin Chen
Shandong University
Qingdao, China
chenzhumin@sdu.edu.cn

Zhaochun Ren
Shandong University
Qingdao, China
zhaochun.ren@sdu.edu.cn

Dongxiao Yu*
Shandong University
Qingdao, China
dxyu@sdu.edu.cn

Jun Ma
Shandong University
Qingdao, China
majun@sdu.edu.cn

Maarten de Rijke
University of Amsterdam & Ahold Delhaize
Amsterdam, The Netherlands
m.derijke@uva.nl

Xiuzhen Cheng
Shandong University
Qingdao, China
xzcheng@sdu.edu.cn

ABSTRACT

Federated recommender systems have distinct advantages in terms of privacy protection over traditional recommender systems that are centralized at a data center. With the widespread use and the growing computing power of mobile devices, it is becoming increasingly feasible to store and process data locally on the devices and to train recommender models in a federated manner. However, previous work on federated recommender systems does not fully account for the limitations in terms of storage, RAM, energy and communication bandwidth in a mobile environment. The scales of the models proposed are too large to be easily run on mobile devices. Also, existing federated recommender systems need to fine-tune recommendation models on each device, which makes it hard to effectively exploit collaborative filtering information among users/devices.

Our goal in this paper is to design a novel federated learning framework for rating prediction (RP) for mobile environments that operates on par with state-of-the-art fully centralized RP methods. To this end, we introduce a federated matrix factorization (MF) framework, named meta matrix factorization (MetaMF), that is able to generate private item embeddings and RP models with a meta network. Given a user, we first obtain a collaborative vector by collecting useful information with a collaborative memory module. Then, we employ a meta recommender module to generate private item embeddings and a RP model based on the collaborative vector in the server. To address the challenge of generating a large number of high-dimensional item embeddings, we devise a rise-dimensional generation strategy that first generates a low-dimensional item embedding matrix and a rise-dimensional matrix, and then multiply them to obtain high-dimensional embeddings. We use the generated model to produce private RPs for the given user on her device.

MetaMF shows a high capacity even with a small RP model, which can adapt to the limitations of a mobile environment. We conduct extensive experiments on four benchmark datasets to compare MetaMF with existing MF methods and find that MetaMF can achieve competitive performance. Moreover, we find MetaMF achieves higher RP performance over existing federated methods by better exploiting collaborative filtering among users/devices.

*Co-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/10.1145/3397271.3401081

CCS CONCEPTS

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

KEYWORDS

Federated recommender system; rating prediction; matrix factorization; meta learning

ACM Reference Format:
Yujie Lin, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Dongxiao Yu, Jun Ma, Maarten de Rijke, and Xiuzhen Cheng. 2020. Meta Matrix Factorization for Federated Rating Predictions. 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/10.1145/3397271.3401081

1 INTRODUCTION

Traditionally, recommender systems are organized in a centralized fashion, i.e., service providers hold all data and models at a data center. As even an anonymized centralized dataset still puts user privacy at risk via combinations with other datasets [43], federated or decentralized recommender systems are increasingly being considered so to realize privacy-aware recommendations [1, 2, 47]. In federated recommender systems, a global model in the server can be trained from user-specific local models on multiple mobile devices (e.g., phones, laptops, smartwatches), ensuring that users' interaction data never leaves their devices. Such recommender systems are capable of reducing the risk of leaking private user data.

Larger recommendation models need more space for storage, more RAM for running programs, more energy for calculation, and more communication bandwidth for downloading or updating. Unlike fully centralized recommender systems at a data center, federated recommender systems that need to run on local devices have stricter requirements on the scale of the model. Previous work on federated recommender systems [1, 7, 32, 47, 55] neglects to fully account for the model scale, so that the proposed federated recommendation approaches need to fine-tune the model on each device. Accordingly, limited device resources (e.g., storage, RAM, energy, and communication bandwidth, etc.) are heavily occupied. Moreover, existing federated approaches cannot effectively exploit collaborative filtering (CF) information among users/device, which limits the performance of existing federated recommendation methods.

To tackle the problems listed above, we focus on a new privacy-aware federated recommendation architecture for the rating prediction (RP) task [24, 30, 31]. For the RP task we aim to predict the rating that a user would give to an item that she has not rated in the past as precisely as possible [19, 25]. In this paper, our target is to design a novel federated learning framework to RP for a federated mobile environment that operates on par with state-of-the-art fully centralized RP methods.

As the method of choice for the RP task, matrix factorization (MF) is used to optimize latent factors to represent users and items by projecting users and items into a joint dense vector space [18, 25]. Today's MF methods consider RP models as well as item embeddings of the same size and shared parameters for all users in order to predict personalized ratings. For fitting all user data, the shared RP model with item embeddings must be large in size. In this paper, we hypothesize that using private item embeddings and models can achieve competitive performance with a small model scale, based on two intuitions. First, different users might have different views and/or angles about the same item: it is not necessary for all users to use shared item embeddings that require many parameters. Second, different users might favor different RP strategies, which means we can use a specific and small model to fit a user's private data. A key challenge is how we can build private RP models on local devices and at the same time effectively utilize collaborative filtering (CF) information on the server as we may not have enough personal data for each user to build her own model.

In this paper, we address this challenge by introducing a novel matrix factorization framework, namely *meta matrix factorization* (MetaMF). Instead of building a model on each local device, we propose to "generate" private item embeddings and RP models with a meta network. Specifically, we assign a so-called indicator vector (i.e., a one-hot vector corresponding to a user id) to each user. For a given user, we first fuse her indicator vector to get a collaborative vector by collecting useful information from other users with a collaborative memory (CM) module. Then, we employ a meta recommender (MR) module to generate private item embeddings and a RP model based on the collaborative vector. It is challenging to directly generate the item embeddings due to the large number of items and the high dimensions. To tackle this problem, we devise a rise-dimensional generation (RG) strategy that first generates a low-dimensional item embedding matrix and a rise-dimensional matrix, and then multiply them to obtain high-dimensional embeddings. Finally, we use the generated RP model to obtain RPs for this user. In a federated recommender system, we deploy the private RP model on the user's device, and the meta network, including CM and MR modules, on the server.

We perform extensive experiments on four benchmark datasets. Despite its federated nature, meta matrix factorization (MetaMF) shows comparable performance with state-of-the-art MF methods on two datasets, while using fewer parameters for item embeddings and RP models. Both the generated item embeddings and the RP model parameters exhibit clustering phenomena, demonstrating that MetaMF can effectively model CF in a federated manner while generating a private model for each user. Moreover, we find that MetaMF achieves higher RP performance than state-of-the-art federated recommendation methods by better exploiting CF among users/devices. To facilitate reproducibility of the results, we are sharing the code.

The main contributions of this paper are as follows:

- We introduce a novel federated matrix factorization (MF) framework, MetaMF, that can reduce the parameters of RP models and item embeddings without loss in performance.
- We devise a meta network, including collaborative memory and meta recommender modules, to better exploit collaborative filtering in federated recommender systems.
- We propose a rise-dimensional generation strategy to reduce the parameters and calculation in generation.
- We conduct extensive experiments and analyses to verify the effectiveness and efficiency of MetaMF.

2 RELATED WORK

We group related work into federated recommender systems, matrix factorization, and meta learning.

2.1 Federated Recommender Systems

For RPs in a federated environment, it is impractical to only rely on local data to train a model for each device, due to data sparsity. Thus, previous work for the federated environment focuses on how to collaboratively train models on distributed data using existing recommendation methods. Ziegler [55] propose to build a graph among computers based on trust, then to use collaborative filtering to do recommendations. Kermarrec et al. [22] further present a user-based random walk approach with CF across devices to predict ratings. Wang et al. [47] introduce a parallel and distributed matrix factorization algorithms to cooperatively learn user/item latent factors across multiple devices. Barbosa et al. [2] propose that smartphones exchange data between devices and calculate their own recommendation via collaborative filtering. Beierle and Eichinger [3] further present a mobile architecture consisting of data collection, data exchange, and a local recommender system; the data collection component gets data about the user from local device, data exchange gets data about other users from other devices, and the local recommender system utilizes all available data for recommending items to the user.

Several studies have introduced federated learning [33] into the realm of recommendation, which provides a way to realize

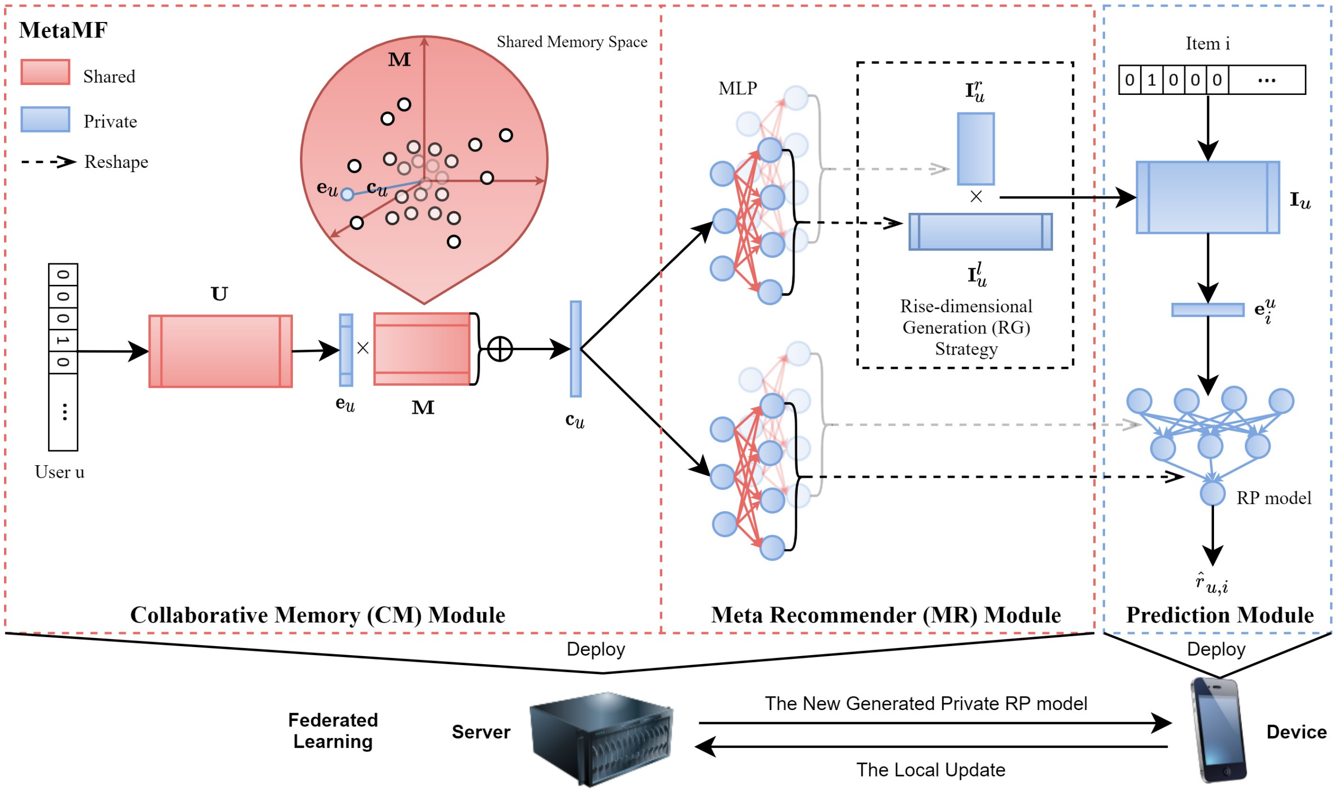


Figure 1: An overview of MetaMF. It consists of three modules. The CM module and the MR module with the RG strategy tend to generate private item embeddings and RP models for different users, which are deployed into the server. The prediction module aims to predict private ratings based on the generated item embeddings and RP models for each user, which is deployed into the device.

federated recommender systems. Chen et al. [7] propose a recommendation framework based on federated meta learning, which maintains a shared model in the cloud. To adapt it for each user, they download the model to the local device and fine-tune the model for personalized recommendations. Ammad-ud din et al. [1] formulate federated collaborative filtering (FCF) methods and adapt WRMF [20] to demonstrate the applicability of FCF.

Unlike us, Ammad-ud din et al. [1] do not focus on the size of the local models while maintaining performance; importantly, they focus on the ranking task, not the rating prediction task that we focus on. In previous federated learning methods, the global model in the server and the local model in the device have the same size, the local model is a copy of the global model. No previous work uses the type of architecture that we design for MetaMF that deploys a big meta network into the server to exploit CF while deploying a small RP model into the device to predict ratings.

2.2 Matrix Factorization

Matrix factorization (MF) has attracted a lot of attention since it was proposed for recommendation tasks. Early studies focus mainly on how to achieve better rating matrix decomposition. Sarwar et al. [39] employ singular value decomposition (SVD) to reduce the dimensionality of the rating matrix, so that they can get low-dimensional user and item vectors. Goldberg et al. [13] apply principal component analysis (PCA) to decompose the rating matrix, and obtain the principal components as user or item vectors. Zhang et al. [54] propose non-negative matrix factorization (NMF), which decomposes the rating matrix by modeling each user's ratings as an additive mixture of rating profiles from user communities or interest groups and constraining the factorization to have non-negative entries. Mnih and Salakhutdinov [34] propose probabilistic matrix factorization (PMF) to model the distributions of user and item vectors from a probabilistic point of view. Koren [24] proposes SVD++, which enhances SVD by including implicit feedback as opposed to SVD, which only includes explicit feedback.

The matrix decomposition methods mentioned above estimate ratings by simply calculating the inner product between user and item vectors, which is not sufficient to capture their complex interactions. Deep learning has been introduced to MF to better model user-item interactions with non-linear transformations. Sedhain et al. [40] propose AutoRec, which takes ratings as input and reconstructs the ratings by an autoencoder. Later, Strub et al. [42] enhance AutoRec by incorporating side information into a denoising autoencoder. He et al. [18] propose the neural collaborative filtering (NCF), which employs a multi-layer perceptron (MLP) to model user-item interactions. Xue et al. [52] present the deep matrix factorization (DMF) which enhances NCF by considering both explicit and implicit feedback. He et al. [17] use convolutional neural networks (CNNs) to improve NCF and present the ConvNCF, which uses the outer product to model user-item interactions. Cheng et al. [9] introduce an attention mechanism into NCF to differentiate the importance of different user-item interactions. Recently, a number of studies have investigated the use of side information or implicit feedback to enhance these neural models [30, 49, 50, 53].

All these models provide personalized RPs by learning user representations to encode differences among users, while sharing item embeddings and models. In contrast, MetaMF provides private RPs by generating non-shared and small models as well as item embeddings for individual users.

2.3 Meta Learning

Meta learning, also known as "learning to learn," has shown its effectiveness in reinforcement learning [51], few-shot learning [35], image classification [37].

Jia et al. [21] propose a network to dynamically generate filters for CNNs. Bertinetto et al. [4] introduce a model to predict the parameters of a pupil network from a single exemplar for one-shot learning. Ha et al. [15] propose hypernetworks, which employ a network to generate the weights of another network. Krueger et al. [26] present a Bayesian variant of hypernetworks that learns the distribution over the parameters of another network. Chen et al. [8] use a hypernetwork to share function-level information across multiple tasks. Few of them target recommendation, which is a more complex task with its own unique challenges.

Recently, some studies have introduced meta learning into recommendations. Vartak et al. [45] study the item cold-start problem in recommendations from a meta learning perspective. They view recommendation as a binary classification problem, where the class labels indicate whether the user engaged with the item. Then they devise a classifier by adapting a few-shot learning paradigm [41]. Lee et al. [27] propose a meta learning-based recommender system called MeLU to alleviate the user cold-start problem. MeLU can estimate new users' preferences with a few consumed items and determine distinguishing items for customized preference estimation by an evidence candidate selection strategy. Du et al. [10] unify scenario-specific learning and model-agnostic sequential meta learning into an integrated end-to-end framework, namely Scenario-specific Sequential Meta learner (s^2Meta). s^2Meta can produce a generic initial model by aggregating contextual information from a variety of prediction tasks and effectively adapt to specific tasks by leveraging learning-to-learn knowledge.

Different from these publications, we learn a hypernetwork (i.e., MetaMF) to directly generate private MF models for each user for RPs.

3 META MATRIX FACTORIZATION

3.1 Overview

Given a user u and an item i, the goal of *rating prediction* (RP) is to estimate a rating $\hat{r}_{u,i}$ that is as accurate as the true rating $r_{u,i}$. We denote the set of users as $\mathcal{U}$, the set of items as $\mathcal{I}$, the set of true ratings as $\mathcal{R}$, which will be divided into the training set D_{train}, the validation set D_{valid}, and the test set D_{test}.

As shown in Fig. 1, MetaMF has three components: a *collaborative memory module* (see Section 3.3), a *meta recommender module* (see Section 3.4 and a *prediction module* (i.e., a RP model; see Section 3.5), where the CM and MR modules constitute a meta network shared by all users, and the prediction module is private. In CM module, we first obtain the user embedding $\mathbf{e}_u$ of u from the user embedding matrix $\mathbf{U}$ and take it as the coordinates to obtain the collaborative vector $\mathbf{c}_u$ from a shared memory space that fuses information from all users. Then we input $\mathbf{c}_u$ to the MR module to generate the parameters of a private RP model for u. The RP model can be of any type. In this work, the RP model is a multi-layer perceptron (MLP). We also generate the private item embedding matrix $\mathbf{I}_u$ of u with a *rise-dimensional generation strategy*. Finally, the prediction module takes the item embedding $\mathbf{e}_{ui}$ of i from $\mathbf{I}_u$ as input and predicts $r_{u,i}$ using the generated RP model.

3.2 Federated Rating Predictions

Before we detail each module, we first detail how to use MetaMF to decentralize data to build a federated recommender system. Because MetaMF can be divided into a meta network, including CM and MR modules, and a RP model, i.e., the prediction module, making it suitable to combine with federated learning to realize a federated recommender system.

Specifically, we can deploy the CM and MR modules into a data center, i.e., the server, and deploy the prediction module locally into mobile devices. The centralized server first generates and delivers different parameters to different mobile devices. Next, each mobile device calculates the loss and the gradients of the parameters in the prediction module based on its private data, and uploads the gradients to the server. Then the server can calculate the gradients of the parameters in the CM and MR modules based on the gradients gathered from each device, and update the parameters. Finally, the server generates and delivers new parameters to each mobile device. Like federated machine learning methods, MetaMF can protect user privacy to a certain extent, because user data does not need to be uploaded to the server. Naturally, the strength of the privacy protection depends on the content of the updates; see Section 7.

MetaMF provides a solid trade-off between exploiting CF for higher RP performance and protecting users' personal information. It places the meta network with the most parameters in the server and places the prediction module of a small scale in devices, which is more suitable to a mobile environment with limited storage, RAM, energy and communication bandwidth.

3.3 Collaborative Memory Module

In order to facilitate collaborative filtering, we propose the CM module to learn a collaborative vector for each user, which encodes both the user's own information and some useful information from other users.

Specifically, we assign each user u and each item i the indicator vectors, $\mathbf{i}_u \in \mathbb{R}^m$ and $\mathbf{i}_i \in \mathbb{R}^n$ respectively, where m is the number of users and n is the number of items. Note that $\mathbf{i}_u$ and $\mathbf{i}_i$ are one-hot vectors with each dimension corresponding to a particular user or item. For the given user u, we first get the user embedding $\mathbf{e}_u$ by Eq. 1:

$$\mathbf{e}_u = \mathbf{U}\mathbf{i}_u, \tag{1}$$

where $\mathbf{e}_u \in \mathbb{R}^{d_u}$, $\mathbf{U} \in \mathbb{R}^{d_u \times m}$ is the user embedding matrix, and d_u is the size of user embeddings. Then we proceed to obtain a collaborative vector for u. Specifically, we use a shared memory matrix $\mathbf{M} \in \mathbb{R}^{d_u \times k}$ to store the basis vectors which span a space of all collaborative vectors, where k is the dimension of basis vectors and collaborative vectors. And we consider the user embedding $\mathbf{e}_u$ as the coordinates of u in the shared memory space. So the collaborative vector $\mathbf{c}_u \in \mathbb{R}^k$ for u is a linear combination of the basis vectors in $\mathbf{M}$ by $\mathbf{e}_u$, as shown in Eq. 2:

$$\mathbf{c}_u = \sum_i \mathbf{M}(i,:)\mathbf{e}_u(i), \tag{2}$$

where $\mathbf{M}(i,:)$ is the i-th vector of $\mathbf{M}$ and $\mathbf{e}_u(i)$ is the i-th scalar of $\mathbf{e}_u$. Because the memory matrix $\mathbf{M}$ is shared among all users, the shared memory space will fuse information from all users. MetaMF can flexibly exploit collaborative filtering among users by assigning them with similar collaborative vectors in the space defined by $\mathbf{M}$, which is equivalent to learning similar user embeddings as in existing MF methods.

3.4 Meta Recommender Module

We propose the MR module to generate the private item embeddings and RP model based on the collaborative vector from the CM module.

3.4.1 Private Item Embeddings. We propose to generate the private item embedding matrix $\mathbf{I}_u \in \mathbb{R}^{d_i \times n}$ for each user u, where d_i is the size of item embeddings. However, it is a challenge to directly generate the whole item embedding matrix when there are a large number of items with relatively high-dimensional item embeddings (instead of extremely small ones). Therefore, we propose a rise-dimensional generation (RG) strategy to decompose the generation into two parts: a low-dimensional item embedding matrix $\mathbf{I}_u^l \in \mathbb{R}^{s \times n}$ and a rise-dimensional matrix $\mathbf{I}_u^r \in \mathbb{R}^{d_i \times s}$, where s is the size of low-dimensional item embeddings and $s \ll d_i$. Specifically, we first follow Eq. 3 to generate $\mathbf{I}_u^l \in \mathbb{R}^{sn}$ and $\mathbf{I}_u^r \in \mathbb{R}^{d_i s}$ (in the form of vectors):

$$\begin{aligned} \mathbf{h}_i^l &= \mathrm{ReLU}(\mathbf{W}_i^l \mathbf{c}_u + \mathbf{b}_i^l), \quad \mathbf{I}_u^l = \mathbf{U}_i^l \mathbf{h}_i^l, \\ \mathbf{h}_i^r &= \mathrm{ReLU}(\mathbf{W}_i^r \mathbf{c}_u + \mathbf{b}_i^r), \quad \mathbf{I}_u^r = \mathbf{U}_i^r \mathbf{h}_i^r, \end{aligned} \tag{3}$$

where $\mathbf{W}_i^l$ and $\mathbf{W}_i^r \in \mathbb{R}^{o \times k}$, $\mathbf{U}_i^l \in \mathbb{R}^{sn \times o}$ and $\mathbf{U}_i^r \in \mathbb{R}^{d_i s \times o}$ are weights; $\mathbf{b}_i^l$ and $\mathbf{b}_i^r \in \mathbb{R}^o$ are biases; $\mathbf{h}_i^l$ and $\mathbf{h}_i^r \in \mathbb{R}^o$ are hidden states; o is the hidden size. Then we reshape $\mathbf{I}_u^l$ to a matrix whose shape is $s \times n$, and reshape $\mathbf{I}_u^r$ to a matrix whose shape is $d_i \times s$. Finally, we multiply $\mathbf{I}_u^l$ and $\mathbf{I}_u^r$ to get $\mathbf{I}_u$:

$$\mathbf{I}_u = \mathbf{I}_u^r \mathbf{I}_u^l. \tag{4}$$

Compared to directly generating $\mathbf{I}_u$, which needs $O(d_i \times n)$ parameters, the RG strategy needs $O(s \times n + d_i \times s)$ parameters which reduces the cost of generating $\mathbf{I}_u$. For different users, the generated item embedding matrices are different.

3.4.2 Private RP Model. We also propose to generate a private RP model for each user u. We use a MLP as the RP model, so we need to generate the weights and biases for each layer of MLP. Specifically, for layer l, we denote its weights and biases as $\mathbf{W}_l^u \in \mathbb{R}^{f_{out} \times f_{in}}$ and $\mathbf{b}_l^u \in \mathbb{R}^{f_{out}}$ respectively, where f_{in} is the size of its input and f_{out} is the size of its output. Then $\mathbf{W}_l^u$ and $\mathbf{b}_l^u$ are calculated as follows:

$$\begin{aligned} \mathbf{h}_g &= \mathrm{ReLU}(\mathbf{W}_g^h \mathbf{c}_u + \mathbf{b}_g^h), \\ \mathbf{W}_l^u &= \mathbf{U}_g^w \mathbf{h}_g + \mathbf{b}_g^w, \\ \mathbf{b}_l^u &= \mathbf{U}_g^b \mathbf{h}_g + \mathbf{b}_g^b, \end{aligned} \tag{5}$$

where $\mathbf{W}_g^h \in \mathbb{R}^{o \times k}$, $\mathbf{U}_g^w \in \mathbb{R}^{f_{out} f_{in} \times o}$ and $\mathbf{U}_g^b \in \mathbb{R}^{f_{out} \times o}$ are weights; $\mathbf{b}_g^h \in \mathbb{R}^o$, $\mathbf{b}_g^w \in \mathbb{R}^{f_{out} f_{in}}$ and $\mathbf{b}_g^b \in \mathbb{R}^{f_{out}}$ are biases; $\mathbf{h}_g \in \mathbb{R}^o$ is hidden state. Finally, we reshape $\mathbf{W}_l^u$ to a matrix whose shape is $f_{out} \times f_{in}$. Note that $\mathbf{W}_g^h$, $\mathbf{b}_g^h$, $\mathbf{U}_g^w$, $\mathbf{b}_g^w$, $\mathbf{U}_g^b$ and $\mathbf{b}_g^b$ are not shared by different layers of the RP model. And f_{in} and f_{out} also vary with different layers. Detailed settings can be found in the experimental setup. Also, MetaMF returns different parameters of the MLP to each user.

3.5 Prediction Module

The prediction module estimates the user's rating for a given item i using the generated item embedding matrix $\mathbf{I}_u$ and RP model from the CM module.

First, we get the private item embedding $\mathbf{e}_i^u \in \mathbb{R}^{d_i}$ of i from $\mathbf{I}_u$ by Eq. 6:

$$\mathbf{e}_i^u = \mathbf{I}_u \mathbf{i}_i. \tag{6}$$

Then we follow Eq. 7 to predict $r_{u,i}$ based on the RP model:

$$\begin{aligned} \mathbf{h}_1 &= \mathrm{ReLU}(\mathbf{W}_1^u \mathbf{e}_i^u + \mathbf{b}_1^u), \\ \mathbf{h}_2 &= \mathrm{ReLU}(\mathbf{W}_2^u \mathbf{h}_1 + \mathbf{b}_2^u), \\ &\vdots \\ \mathbf{h}_{L-1} &= \mathrm{ReLU}(\mathbf{W}_{L-1}^u \mathbf{h}_{L-2} + \mathbf{b}_{L-1}^u), \\ \hat{r}_{u,i} &= \mathbf{W}_L^u \mathbf{h}_{L-1} + \mathbf{b}_L^u, \end{aligned} \tag{7}$$

where L is the number of layers of the RP model. The weights $\{\mathbf{W}_1^u, \mathbf{W}_2^u, \ldots, \mathbf{W}_{L-1}^u, \mathbf{W}_L^u\}$ and biases $\{\mathbf{b}_1^u, \mathbf{b}_2^u, \ldots, \mathbf{b}_{L-1}^u, \mathbf{b}_L^u\}$ are generated by the CM module. The last layer L is the output layer, which returns a scalar as the predicted rating $\hat{r}_{u,i}$.

3.6 Loss

In order to learn MetaMF, we formulate the RP task as a regression problem and the loss function is defined as:

$$L_{rp} = \frac{1}{|D_{train}|} \sum_{r_{u,i} \in D_{train}} (r_{u,i} - \hat{r}_{u,i})^2. \tag{8}$$

To avoid overfitting, we add the L2 regularization term:

$$L_{reg} = \frac{1}{2} \|\Theta\|_2^2, \tag{9}$$

where Θ represents the trainable parameters of MetaMF. Note that unlike existing MF methods, the item embeddings and the parameters of RP models are not included in Θ, because they are also the outputs of MetaMF, not trainable parameters.

The final loss L is a linear combination of L_{rp} and L_{reg}:

$$L = L_{rp} + \lambda L_{reg}, \tag{10}$$

where λ is the weight of L_{reg}. The whole framework of MetaMF can be efficiently trained using back propagation with federated learning on decentralized data, as showed in Algorithm 1.

Algorithm 1 MetaMF

Input: All trainable parameters Θ, which are stored in the server; The user set $\mathcal{U}$, where one user per device; For user u, her local data D_u stored in her device, and $D_{train} = \sum_{u \in \mathcal{U}} D_u$; † means the code is executed in the device.

Output: Θ; For user u, the parameters of her RP model and item embeddings Φ_u, which are stored in her device;

Initialize Θ randomly in the server;
for u in $\mathcal{U}$ **do**
  Generate Φ_u based on Θ;
  Send Φ_u to u's device;
end for
while not convergent **do**
  Sample a batch S from $\mathcal{U}$;
  for u in S **do**
    Sample a batch B_u from D_u;†
    Calculate the gradient of Φ_u based on B_u;†
    Upload the gradient to the server;†
    Calculate the gradient of Θ based on the gradient of Φ_u;
  end for
  Accumulate the gradients of Θ gathered from S;
  Update Θ based on the accumulated gradient;
  for u in $\mathcal{U}$ **do**
    Regenerate Φ_u based on new Θ;
    Send Φ_u to u's device;
  end for
end while

4 EXPERIMENTAL SETUP

Table 1: Statistics of the datasets, where *#avg* means the average number of user ratings, Hetrec-ML is the short name of Hetrec-movielens.

Datasets	*#users*	*#items*	*#ratings*	*#avg*	*#sparsity (%)*
Douban	2,509	39,576	894,887	357	0.9
Hetrec-ML	2,113	10,109	855,599	405	4
Movielens1M	6,040	3,706	1,000,209	166	4.5
Ciao	7,375	105,096	282,619	38	0.04

We seek to answer the following research questions. (RQ1) How does the proposed method MetaMF for federated rating predictions perform compared to state-of-the-art MF methods for the RP task? Does the federated nature of MetaMF come at cost in terms of performance on the RP task? (RQ2) What is the contribution of generating private item embeddings and RP models?

4.1 Datasets

We conduct experiments on four widely used datasets: **Douban** [19], **Hetrec-movielens** [6], **Movielens1M** [16] and **Ciao** [14]. We list the statistics of these four datasets in Table 1. For each user on each dataset, we randomly separate her data into three chunks: 80% as the training set, 10% as the validation set and 10% as the test set.

4.2 Baselines

We compare MetaMF with the following conventional, deep learning-based and federated MF methods. It is worth noting that in this paper we focus on predicting ratings based on rating matrices, thus for fairness we neglect MF methods that need side information.

- **Conventional methods:**
  - **NMF** [54]: uses non-negative matrix factorization to decompose rating matrices.
  - **PMF** [34]: applies Gaussian distributions to model the latent factors of users and items.
  - **SVD++** [24]: extends SVD by considering implicit feedback for modeling latent factors.
  - **LLORMA** [28]: uses a number of low-rank sub-matrices to compose rating matrices.
- **Deep learning-based methods:**
  - **RBM** [38]: employs restricted Boltzmann machine (RBM) to model the generation process of ratings.
  - **AutoRec** [40]: proposes autoencoders (AEs) to model interactions between users and items. AutoRec has two variants, with one taking users' ratings as input, denoted by AutoRec-U, and the other taking items' ratings as input, denoted by AutoRec-I.
  - **NCF** [18]: the state-of-the-art MF method that combines generalized matrix factorization and MLP to model user-item interactions. We adapt NCF for the RP task by dropping the sigmoid activation function on its output layer and replacing its loss function with Eq. 8.
- **Federated methods:**
  - **FedRec** [7]: a federated recommendation method, which employs MAML [11] to learn a shared RP model in the server and update the model for each device. In our experiments, the shared RP model is a MLP with two layers (layer sizes are 16 and 1 respectively), and its user/item embedding size is 64.

4.3 Evaluation Metrics

To evaluate the performance of rating prediction methods, we employ two evaluation metrics, i.e., Mean Absolute Error (MAE) and Mean Square Error (MSE). Both of them are widely applied for the RP task in recommender systems. Given the predicted rating $\hat{r}_{u,i}$ and the true rating $r_{u,i}$ of user u on item i in the test set D_{test}, MAE is calculated as:

$$\text{MAE} = \frac{1}{|D_{test}|} \sum_{r_{u,i} \in D_{test}} |r_{u,i} - \hat{r}_{u,i}|. \tag{11}$$

MSE is defined as:

$$\text{MSE} = \frac{1}{|D_{test}|} \sum_{r_{u,i} \in D_{test}} (r_{u,i} - \hat{r}_{u,i})^2. \tag{12}$$

Statistical significance of observed differences is tested for using a two-sided paired t-test for significant differences ($p < 0.01$).

4.4 Implementation Details

The user embedding size d_u and the item embedding size d_i are set to 32. The size of the collaborative vector k is set to 128. The size of the low-dimensional item embedding s is set to 8. The hidden

Table 2: Comparison results of MetaMF and baselines on the four datasets. A superscript $\approx$ indicates that there is no statistically significant difference between MetaMF and NCF (two-sided paired t-test, $p < 0.01$).

Method	Douban		Hetrec-movielens		Movielens1M		Ciao	
	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
NMF	0.602	0.585	0.625	0.676	0.727	0.848	0.750	1.039
PMF	0.639	0.701	0.617	0.644	0.703	0.788	1.501	3.970
SVD++	0.593	0.570	0.579	0.590	0.671	0.740	0.738	0.963
LLORMA	0.610	0.623	0.588	0.603	0.675	0.748	1.349	3.396
RBM	1.058	1.749	1.124	1.947	1.122	2.078	1.132	2.091
AutoRec-U	0.709	0.911	0.660	0.745	0.678	0.775	1.673	5.671
AutoRec-I	0.704	0.804	0.633	0.694	0.663	**0.715**	0.792	1.038
NCF	**0.583**	**0.547**	0.572	**0.575**	0.675	0.739	**0.735**	**0.937**
FedRec	0.760	0.927	0.846	1.265	0.907	1.258	0.865	1.507
MetaMF	$0.584^{\approx}$	0.549	$\mathbf{0.571}^{\approx}$	$0.578^{\approx}$	0.687	0.760	0.774	1.043

Table 3: Rating prediction results of MetaMF, MetaMF-SI and MetaMF-SM on the four datasets. MetaMF-SI shares item embeddings for all users; MetaMF-SM shares the parameters of prediction module for all users.

Method	Douban		Hetrec-movielens		Movielens1M		Ciao	
	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
MetaMF	**0.584**	**0.549**	**0.571**	**0.578**	**0.687**	**0.760**	0.774	1.043
MetaMF-SI	0.586	0.552	0.590	0.615	0.696	0.784	**0.732**	**0.925**
MetaMF-SM	0.595	0.571	0.595	0.622	0.697	0.788	0.789	1.061

size o is set to 512. And the RP model in the prediction module is an MLP with two layers (one hidden layer and one output layer) whose layer sizes are 8 and 1. During training, we initialize all trainable parameters randomly with the Xavier method [12]. We choose Adam [23] to optimize MetaMF, set the learning rate to 0.0001, and set the regularizer weight λ to 0.001. Our framework is implemented with Pytorch [36]. In our experiments, we implement NCF based on the released code of the author.[1] We use the code released by the respective authors[2] for AutoRec. We use LibRec[3] for the remaining baselines.

5 EXPERIMENTAL RESULTS

5.1 What Is the Cost of Federation?

We start by addressing RQ1 and compare our federated rating prediction model MetaMF with state-of-the-art MF methods. Table 2 lists the RP performance of all MF methods.

Our main observations are as follows. First, on the Douban and Hetrec-movielens datasets, MetaMF outperforms most baselines despite the fact that it is federated while most baselines are centralized. And MetaMF is slightly inferior to NCF, but this difference is not significant. So we can draw the conclusion that the performance of MetaMF is comparable to NCF on these two datasets. See Section 5.2 and Section 6.1 for further analysis.

Second, on the Movielens1M and Ciao datasets, MetaMF does not perform well, in some cases worse than some traditional methods, such as SVD++. The most important reason is that the average numbers of user ratings on these two datasets are small. As shown in Table 1, the statistics *#avg* on these four datasets are 357, 405, 166 and 38 respectively. Because the Douban and Hetrec-movielens datasets provide more private data for each user, MetaMF is able to capture the differences among users for learning private item embeddings and RP models. However, the Movielens1M and Ciao datasets lack sufficient data, which limits the performance of MetaMF.

Third and finally, MetaMF significantly outperforms FedRec on all datasets with smaller user/item embedding size and RP model scale. And FedRec performs worse than most baselines on most datasets. The reason may be that FedRec cannot effectively exploit CF information among users/devices. Although FedRec maintains a shared model in the server, it needs to fine-tune the model on each device, which prevents some useful information from being shared among devices. However, MetaMF can flexibly take advantage of CF among users/devices by the meta network. See Section 6.2 for further analysis.

Although federated recommender systems can protect user privacy by keeping data locally, it is harder for them to exploit collaborative filtering among users than for centralized approaches, which affects their performance. So how to share information among multiple devices in a privacy-aware manner is still a core problem in federated recommender systems. We can observe that MetaMF does not outperform NCF, and the performance of FedRec is also worse than of most centralized baselines. Although the federated nature of MetaMF makes it trade performance for privacy, it can still achieve comparable performance with NCF on two datasets, which shows

[1] https://github.com/hexiangnan/neural_collaborative_filtering
[2] https://github.com/gtshs2/Autorec
[3] https://www.librec.net/

MetaMF can get a better balance between privacy protection and RP performance.

5.2 What Does the Privatization of MetaMF Contribute?

Next we address RQ2 to analyze the effectiveness of generating private item embeddings and RP models to the overall performance of MetaMF. First, we compare MetaMF to MetaMF-SI, which only generates private RP models for different users while sharing a common item embedding matrix among all users. As shown in Table 3, MetaMF outperforms MetaMF-SI on most datasets, except for the Ciao dataset. We conclude that generating private item embeddings for each user can improve the performance of MetaMF. It is possible that the Ciao dataset lacks sufficient private data for learning private item embeddings, so that MetaMF performs worse than MetaMF-SI. And if we compare MetaMF-SI with NCF, we find that MetaMF-SI also outperforms NCF on the Ciao dataset, which indicates that generating private RP models can improve RP on the Ciao dataset.

Next, we compare MetaMF with MetaMF-SM, which generates different item embeddings for different users and shares a common RP model among all users. From Table 3, we can see that MetaMF consistently outperforms MetaMF-SM on all datasets. Thus, generating private RP models for users is able to improve the performance of MetaMF too.

Furthermore, by comparing MetaMF-SI and MetaMF-SM, we observe that MetaMF-SI outperforms MetaMF-SM on all datasets. This shows that item embeddings have a greater impact on the performance of MetaMF than RP models.

Finally, in response to RQ2 we conclude that generating private item embeddings and RP models in MetaMF contributes to overall performance of MetaMF.

6 ANALYSIS

Next, we want to understand whether MetaMF shows a high capacity at a small RP model scale compared to NCF. And to which degree does MetaMF generate different item embeddings and RP models for different users while exploiting collaborative filtering?

6.1 Model Scale Analysis

We examine whether MetaMF shows a high model capacity at a small RP model scale by comparing MetaMF with NCF at different model scales on the Douban and Hetrec-movielens datasets. Because MetaMF and NCF are both MLP-based methods, we represent each model scale as a combination of the item embedding size and the list of layer sizes,[4] which are the key hyper-parameters to affect the number of parameters. For MetaMF, we also list the collaborative vector size and the hidden layer size in the CM and MR modules for each model scale, however we only care about the generated parameters in the prediction module of MetaMF because in federated recommender systems we only need to deploy the prediction module (i.e., the RP model) on local devices. Note that the number of parameters we consider is independent from the number of users/devices.

[4]The specific number of parameters is $d_u \times m + d_i \times n + \sum_{l=1}^{L} f_{in}^{l} \times f_{out}^{l}$, here we use l to differentiate different layers.

Table 4: The performance of MetaMF with different model scales, where each model scale is represented as a tuple (*item embedding size*, [*layer sizes in the Prediction Module*], [*collaborative vector size, hidden layer size in the MR Module*]).

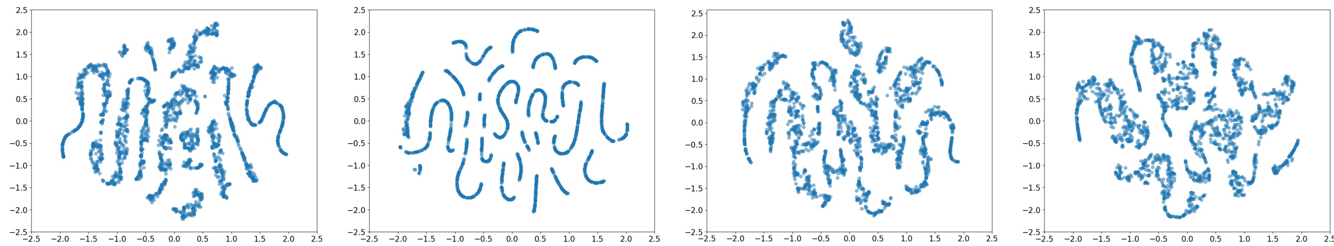
Model scale	Douban		Hetrec-movielens	
	MAE	MSE	MAE	MSE
(8, [2, 1], [32, 128])	0.584	**0.548**	0.575	0.584
(16, [4, 1], [64, 256])	0.587	0.552	0.573	0.582
(32, [8, 1], [128, 512])	**0.584**	0.549	**0.571**	**0.578**
(64, [16, 1], [256, 1024])	—	—	0.578	0.591

Table 5: The performance of NCF with different model scales, where each model scale is represented as (*item embedding size*, [*layer sizes*]).

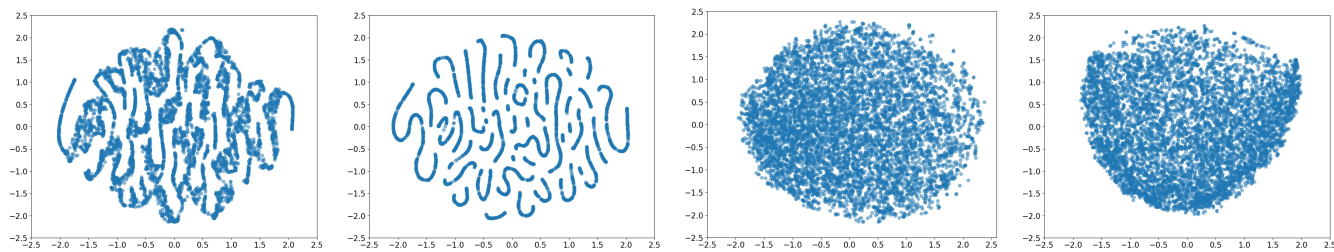
Model scale	Douban		Hetrec-movielens	
	MAE	MSE	MAE	MSE
(16, [16, 8, 4, 1])	0.587	0.552	0.585	0.603
(32, [32, 16, 8, 1])	0.587	0.552	0.583	0.600
(64, [64, 32, 16, 1])	0.584	0.549	0.579	0.595
(128, [128, 64, 32, 1])	0.585	0.549	0.574	0.579
(256, [256, 128, 64, 1])	**0.583**	**0.547**	**0.572**	**0.575**
(512, [512, 256, 128, 1])	0.584	0.547	0.572	0.581
(1024, [1024, 512, 256, 1])	0.586	0.549	0.574	0.582

In Table 4 and Table 5, we list the performance of MetaMF and NCF for different model scales. By comparing the best settings of MetaMF (32, [8], [128, 512]) and NCF (256, [256, 128, 64]), we see that MetaMF achieves a comparable performance with NCF with a smaller item embedding size, fewer layers and smaller layer sizes. Importantly, at small model scales, MetaMF significantly outperforms NCF. This is because the item embeddings and the RP model generated by MetaMF are private, using a small scale is sufficient to accurately encode the preference of a specific user and predict her ratings. However for NCF, the shared RP model with item embeddings has to be large in size since it needs to incorporate the information of all users and predict ratings for all users. We conclude that generating private item embeddings and RP models helps MetaMF to keep a relatively high capacity with fewer parameters in item embeddings and RP models, and improve the performance of rating prediction at the small model scale. Furthermore, when deploying the recommendation system on mobile devices, this advantage of MetaMF can save storage space, RAM, energy and communication bandwidth.

As the model scale increases, the performance of MetaMF deteriorates earlier than NCF. Because there is not sufficient private data to train too many parameters for each user, larger model scales easily lead MetaMF to overfit. And even though we have the RG strategy to alleviate the memory and computational requirements for generating private item embeddings, it is unrealistic to generate larger embeddings for too many items. For example, we do not list the performance of MetaMF with the model scale of (64, [16], [256, 1024]) on the Douban dataset, because there are too many items in the dataset, making generation too difficult.



(a) The weights of the hidden layer on the Douban dataset. (b) The embeddings of item 16716 on the Douban dataset. (c) The weights of the hidden layer on the Hetrec-movielens dataset. (d) The embeddings of item 1931 on the Hetrec-movielens dataset.



(e) The weights of the hidden layer on the Movielens1M dataset. (f) The embeddings of item 482 on the Movielens1M dataset. (g) The weights of the hidden layer on the Ciao dataset. (h) The embeddings of item 8271 on the Ciao dataset.

Figure 2: The generated weights and item embeddings reduced dimension by t-SNE and normalized by mean and standard deviation on the four datasets, where one point corresponds to one user.

6.2 Weights and Embeddings

In order to verify that MetaMF generates private item embeddings and RP models for users while efficiently exploiting CF, we visualize the generated weights and item embeddings after reducing their dimension by t-SNE [44] and normalizing them by mean and standard deviation,[5] where each point represents a user's weights or item embeddings. Because there are many items, we randomly select one item from each dataset for visualization. As shown in Fig. 2, MetaMF generates different weights and item embeddings for different users on most datasets, which indicates that MetaMF has the ability to capture private factors for users. We also notice the existence of many non-trivial clusters in most visualizations, which shows that MetaMF is able to share information among users to take advantage of collaborative filtering in the meta network.

The only exception is on the Ciao dataset, where MetaMF seems unable to learn distinguishable weights and item embeddings. The Ciao dataset does not provide sufficient private data for learning effective weights and item embeddings for each user. It also illustrates why MetaMF does not perform well on the Ciao dataset.

7 CONCLUSION AND DISCUSSION

In this paper, we studied the federated rating prediction problem. In particular, we investigated how to reduce the model scale of matrix factorization methods in order to make them suitable for a federated environment. To achieve this, we proposed a novel matrix factorization framework, named MetaMF, that can generate private RP models as well as item embeddings for each user with a meta network. We conducted extensive experiments to compare and analyze the performance of MetaMF. MetaMF performs competitively, at the level of state-of-the-art RP methods despite using a significant smaller RP model and embedding size for items. In particular, by using collaborative filtering in a federated environment, MetaMF outperforms the federated recommendation method FedRec by a large margin. Thus, we hope MetaMF can advance future research on federated recommendation by presenting a new framework and a scheme.

Next, we discuss some limitations of MetaMF and related future work. First, MetaMF still has a risk of leaking private information. MetaMF uses a meta network to directly generate private item embeddings and RP models on the server. Although the meta network can efficiently exploit CF to improve RP performance, it may leak personal information about users in private item embeddings, RP models and their updates. As future work, we plan to design a more privacy-aware generation network that preserves the high RP performance at the same time. Second, currently MetaMF cannot handle cold-start users well [5]. Although MetaMF incorporates a CM module to collect collaborative information to alleviate this issue, it still needs a certain amount of data for each user to achieve satisfactory performance. As a result, it does not perform well when there is not enough personalized data for each user. A possible solution direction is to reduce the data requirements of MetaMF using few-shot [46] or zero-shot learning [48]. Finally, because the ranking prediction task [29] is also important in the area of recommendation system, we will also evaluate the performance of MetaMF on the ranking prediction.

DATA AND CODE

To facilitate reproducibility of our work, we are sharing all resources used in this paper at https://bitbucket.org/HeavenDog/metamf/src/master/.

[5] Here, $\mathrm{norm}(x) = \frac{x-\mu}{\sigma}$, where μ is the mean and σ is the standard deviation.

ACKNOWLEDGMENTS

We thank our anonymous reviewers for their helpful comments.

This work is supported by National Key R&D Program of China with grant No. 2019YFB2102600, the Natural Science Foundation of China (61832012, 61972234, 61902219), the Foundation of State Key Laboratory of Cognitive Intelligence, iFLYTEK, P.R. China (COGOSC-20190003), the Key Scientific and Technological Innovation Program of Shandong Province (2019JZZY010129), the Fundamental Research Funds of Shandong University, and the Innovation Center for Artificial Intelligence (ICAI).

All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

REFERENCES

[1] Muhammad Ammad-ud din, Elena Ivannikova, Suleiman A Khan, Were Oyomno, Qiang Fu, Kuan Eeik Tan, and Adrian Flanagan. 2019. Federated Collaborative Filtering for Privacy-Preserving Personalized Recommendation System. *arXiv preprint arXiv:1901.09888* (2019).

[2] Lucas Nunes Barbosa, Jonathan Gemmell, Miller Horvath, and Tales Heimfarth. 2018. Distributed User-Based Collaborative Filtering on an Opportunistic Network. In *AINA*. IEEE, 266–273.

[3] Felix Beierle and Tobias Eichinger. 2019. Collaborating with Users in Proximity for Decentralized Mobile Recommender Systems. In *UIC*. IEEE.

[4] Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. 2016. Learning Feed-forward One-shot Learners. In *NeurIPS*. 523–531.

[5] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Jesús Bernal. 2012. A Collaborative Filtering Approach to Mitigate the New User Cold Start Problem. *Knowledge-based systems* 26 (2012), 225–238.

[6] Ivan Cantador, Peter L Brusilovsky, and Tsvi Kuflik. 2011. Second Workshop on Information Heterogeneity and Fusion in Recommender Systems. In *HetRec2011*.

[7] Fei Chen, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. 2018. Federated Meta-learning for Recommendation. *arXiv preprint arXiv:1802.07876* (2018).

[8] Junkun Chen, Xipeng Qiu, Pengfei Liu, and Xuanjing Huang. 2018. Meta Multi-task Learning for Sequence Modeling. In *AAAI*.

[9] Zhiyong Cheng, Ying Ding, Xiangnan He, Lei Zhu, Xuemeng Song, and Mohan S Kankanhalli. 2018. Aˆ 3NCF: An Adaptive Aspect Attention Model for Rating Prediction. In *IJCAI*. 3748–3754.

[10] Zhengxiao Du, Xiaowei Wang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Sequential Scenario-Specific Meta Learner for Online Recommendation. In *SIGKDD*. ACM.

[11] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic Meta-learning for Fast Adaptation of Deep Networks. In *ICML*. ACM, 1126–1135.

[12] Xavier Glorot and Yoshua Bengio. 2010. Understanding the Difficulty of Training Deep Feedforward Neural Networks. *JMLR* 9 (2010), 249–256.

[13] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. 2001. Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information retrieval* 4, 2 (2001), 133–151.

[14] Guibing Guo, Jie Zhang, Daniel Thalmann, and Neil Yorke-Smith. 2014. ETAF: An Extended Trust Antecedents Framework for Trust Prediction. In *ASONAM*. IEEE, 540–547.

[15] David Ha, Andrew Dai, and Quoc V Le. 2016. Hypernetworks. *arXiv preprint arXiv:1609.09106* (2016).

[16] F Maxwell Harper and Joseph A Konstan. 2016. The Movielens Datasets: History and Context. *ACM TIIS* 5, 4 (2016), 19.

[17] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat Seng Chua. 2018. Outer Product-based Neural Collaborative Filtering. In *IJCAI*.

[18] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *WWW*. ACM, 173–182.

[19] Longke Hu, Aixin Sun, and Yong Liu. 2014. Your Neighbors Affect your Ratings: On Geographical Neighborhood Influence to Rating Prediction. In *SIGIR*. ACM, 345–354.

[20] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *ICDM*. IEEE, 263–272.

[21] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. 2016. Dynamic Filter Networks. In *NeurIPS*. 667–675.

[22] Anne-Marie Kermarrec, Vincent Leroy, Afshin Moin, and Christopher Thraves. 2010. Application of Random Walks to Decentralized Recommender Systems. In *OPODIS*. Springer, 48–63.

[23] Diederik P Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* (2014).

[24] Yehuda Koren. 2008. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In *SIGKDD*. ACM, 426–434.

[25] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer* 42, 8 (2009), 30–37.

[26] David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville. 2017. Bayesian Hypernetworks. *arXiv preprint arXiv:1710.04759* (2017).

[27] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. 2019. MeLU: Meta-Learned User Preference Estimator for Cold-Start Recommendation. In *SIGKDD*. ACM, 1073–1082.

[28] Joonseok Lee, Seungyeon Kim, Guy Lebanon, Yoram Singer, and Samy Bengio. 2016. LLORMA: Local Low-rank Matrix Approximation. *JMLR* 17, 1 (2016), 442–465.

[29] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural Attentive Session-based Recommendation. In *CIKM*. ACM, 1419–1428.

[30] Xiaopeng Li and James She. 2017. Collaborative Variational Autoencoder for Recommender Systems. In *SIGKDD*. ACM, 305–314.

[31] Benjamin M Marlin. 2004. Modeling User Rating Profiles for Collaborative Filtering. In *NeurIPS*. 627–634.

[32] Paolo Massa and Paolo Avesani. 2007. Trust-aware Recommender Systems. In *RecSys*. ACM, 17–24.

[33] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*. 1273–1282.

[34] Andriy Mnih and Ruslan R Salakhutdinov. 2008. Probabilistic Matrix Factorization. In *NeurIPS*. 1257–1264.

[35] Alex Nichol, Joshua Achiam, and John Schulman. 2018. On First-order Meta-learning Algorithms. *arXiv preprint arXiv:1803.02999* (2018).

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An Imperative Style, High-performance Deep Learning Library. In *NeurIPS*. 8024–8035.

[37] Sachin Ravi and Hugo Larochelle. 2017. Optimization as a Model for Few-shot Learning. In *ICLR*.

[38] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann Machines for Collaborative Filtering. In *ICML*. ACM, 791–798.

[39] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2000. Application of Dimensionality Reduction in Recommender System – A Case Study. In *WebKDD*.

[40] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders Meet Collaborative Filtering. In *WWW*. ACM, 111–112.

[41] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical Networks for Few-shot Learning. In *NeurIPS*. 4077–4087.

[42] Florian Strub, Romaric Gaudel, and Jérémie Mary. 2016. Hybrid Recommender System Based on Autoencoders. In *DLRS*. ACM, 11–16.

[43] Latanya Sweeney. 2000. *Uniqueness of Simple Demographics in the U.S. Population*. Technical Report LIDAP-WP4. Carnegie Mellon University, School of Computer Science, Data Privacy Laboratory, Pittsburgh.

[44] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data Using t-SNE. *JMLR* 9, Nov (2008), 2579–2605.

[45] Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. 2017. A Meta-learning Perspective on Cold-start Recommendations for Items. In *NeurIPS*. 6904–6914.

[46] Yaqing Wang and Quanming Yao. 2019. Few-shot Learning: A Survey. *CoRR* (2019).

[47] Zhangyang Wang, Xianming Liu, Shiyu Chang, Jiayu Zhou, Guo-Jun Qi, and Thomas S. Huang. 2015. Decentralized Recommender Systems. *arXiv preprint arXiv:1503.01647* (2015).

[48] Yongqin Xian, Christoph H. Lampert, Bernt Schiele, and Zeynep Akata. 2019. Zero-Shot Learning - A Comprehensive Evaluation of the Good, the Bad and the Ugly. *TPAMI* 41, 9 (2019), 2251–2265.

[49] Teng Xiao, Shangsong Liang, Hong Shen, and Zaiqiao Meng. 2019. Neural Variational Hybrid Collaborative Filtering. In *WWW*. ACM.

[50] Teng Xiao, Shangsong Liang, Weizhou Shen, and Zaiqiao Meng. 2019. Bayesian Deep Collaborative Matrix Factorization. In *AAAI*.

[51] Zhongwen Xu, Hado P van Hasselt, and David Silver. 2018. Meta-gradient Reinforcement Learning. In *NeurIPS*. 2396–2407.

[52] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep Matrix Factorization Models for Recommender Systems.. In *IJCAI*. 3203–3209.

[53] Baolin Yi, Xiaoxuan Shen, Hai Liu, Zhaoli Zhang, Wei Zhang, Sannyuya Liu, and Naixue Xiong. 2019. Deep Matrix Factorization with Implicit Feedback Embedding for Recommendation System. *IEEE TII* (2019).

[54] Sheng Zhang, Weihong Wang, James Ford, and Fillia Makedon. 2006. Learning from Incomplete Ratings Using Non-negative Matrix Factorization. In *SDM*. SIAM, 549–553.

[55] Cai-Nicolas Ziegler. 2004. Semantic Web Recommender Systems. In *EDBT*. Springer, 78–89.