FedRec++: Lossless Federated Recommendation with Explicit Feedback

Feng Liang[a, b], Weike Pan[a, b]* and Zhong Ming[a, b]*

liangfeng2018@email.szu.edu.cn, {panweike, mingz}@szu.edu.cn

[a]National Engineering Laboratory for Big Data System Computing Technology,
Shenzhen University, Shenzhen, China

[b]College of Computer Science and Software Engineering,
Shenzhen University, Shenzhen, China

Problem Definition

Privacy-aware recommendation with explicit feedback

- Input: A set of rating records for each user u, i.e., $\mathcal{R}_u = \{(u, i, r_{ui}); i \in \mathcal{I}_u\}$, where $\mathcal{I}_u$ is a set of items rated by user u and r_{ui} is the rating assigned to item i by user u.
- Goal: Estimate the preference of user u to the unrated items without exposing the rating scores and the rating behaviors of each user u (i.e., $\mathcal{R}_u$ and $\mathcal{I}_u$), which is the main difference between federated recommendation and traditional recommendation.

Motivation

Some very recent works propose to randomly sample some unrated items for each user and then assign some virtual ratings, so that the server can not identify the scores and the set of rated items easily during the server-client interactions. However, the virtual ratings assigned to the randomly sampled items will inevitably introduce some noise to the model training process, which will then cause loss in recommendation performance.

Notations (1/2)

Table: Some notations and their explanations.

Notation	Explanation
n	number of users (i.e., clients)
m	number of items
$\mathfrak{R} = \{1, \ldots, 5\}$	rating range
$r_{ui} \in \mathfrak{R}$	rating of user u to item i
$\mathcal{R} = \{(u, i, r_{ui})\}$	rating records in training data
$\mathcal{R}_u$	rating records w.r.t. user u in $\mathcal{R}$
$\mathcal{R}^{te} = \{(u, i, r_{ui})\}$	rating records in test data
$\mathcal{I}$	the whole set of items
$\mathcal{I}_u$	items rated by user u
$\mathcal{I}'_u$, $\lvert\mathcal{I}'_u\rvert = \rho\lvert\mathcal{I}_u\rvert$	sampled unrated items w.r.t. user u from $\mathcal{I}\backslash\mathcal{I}_u$
$\mathcal{U}$	the whole set of users
$\mathcal{U}_i$	users who rated item i
$\mathcal{U}'_i$	users who virtually rated item i
$\mathcal{U}'^{\tilde{u}}_i$	users who virtually rated item i w.r.t. denoiser $\tilde{u}$
$\tilde{\mathcal{U}} = \{\tilde{u}\}$	denoising clients (denoisers)
$y_{ui} \in \{0, 1\}$	indicator variable

Notations (2/2)

Table: Some notations and their explanations (cont.).

Notation	Explanation
$d \in \mathbb{R}$	number of latent dimensions
$U_{u\cdot}, U'_{u\cdot} \in \mathbb{R}^{1\times d}$	user-specific latent feature vector
$V_{i\cdot} \in \mathbb{R}^{1\times d}$	item-specific latent feature vector
$\hat{r}_{ui}$	predicted rating of user u to item i
γ	learning rate
ρ	sampling parameter
c	number of clients in training
η	number of denoising clients
λ	tradeoff parameter
T	iteration number

Probabilistic Matrix Factorization (PMF)

In PMF [Mnih and Salakhutdinov, 2007], the rating of a user u to an item i is calculated via the inner product of their latent feature vectors, i.e., $\hat{r}_{ui} = U_{u\cdot} V_{i\cdot}^T$, where $U_{u\cdot}, V_{i\cdot} \in \mathbb{R}^{1 \times d}$.

Federated Recommendation with Explicit Feedback (FedRec) (1/2)

In FedRec [Lin et al., 2020], in order to protect a user's rating behaviors, i.e., the set of items $\mathcal{I}_u$ rated by a user u, the authors design an effective hybrid filling (HF) strategy to randomly sample some unrated items. Firstly, it randomly samples $|\mathcal{I}'_u|$ unrated items of user u from $\mathcal{I} \backslash \mathcal{I}_u$, where $|\mathcal{I}'_u| = \rho|\mathcal{I}_u|$ with $\rho \in \{1, 2, 3\}$. Secondly, it uses the average rating or predicted rating of user u to a sampled item i as a virtual rating r'_{ui}. Thirdly, it calculates the gradients of user u to the rated items and the unrated items, i.e., $\nabla V_{i\cdot}, i \in \mathcal{I}_u \cup \mathcal{I}'_u$, and then uploads these gradients to the server. In this way, FedRec with the HF strategy achieves the purpose of protecting the user's original rating records and the rating behaviors in the preference modeling process. In particular, the virtual rating r'_{ui} is as follows,

$$r'_{ui} = \begin{cases} \frac{\sum_{k=1}^{m} y_{uk} r_{uk}}{\sum_{k=1}^{m} y_{uk}}, t < T_{\text{predict}} \\ U'_{u\cdot} V_{i\cdot}^T, t \geq T_{\text{predict}} \end{cases} \quad (1)$$

where t denotes the number of iterations that have been executed in model training, and T_{predict} is a parameter that determines when to start using the predicted rating as a virtual rating to a sampled rated item i.

Federated Recommendation with Explicit Feedback (FedRec) (2/2)

In FedRec with PMF [Mnih and Salakhutdinov, 2007] as the backbone model, the gradient of each item i is as follows,

$$\nabla V_{i\cdot} = \frac{\sum_{u \in \mathcal{U}_i \cup \mathcal{U}_i'} \nabla V_{\text{EF}}^{\text{HF}}(u,i)}{|\mathcal{U}_i \cup \mathcal{U}_i'|}. \quad (2)$$

Notice that $\mathcal{U}_i \cup \mathcal{U}_i'$ denotes the users that have rated or virtually rated item i, and

$$\nabla V_{\text{EF}}^{\text{HF}}(u,i) = \begin{cases} (U_{u\cdot} V_{i\cdot}^T - r_{ui}) U_{u\cdot} + \lambda V_{i\cdot}, y_{ui} = 1 \\ (U_{u\cdot} V_{i\cdot}^T - r_{ui}') U_{u\cdot} + \lambda V_{i\cdot}, y_{ui} = 0 \end{cases} \quad (3)$$

where r_{ui} and r_{ui}' are the true observed rating and the virtual rating of user u to item i, respectively.

Although FedRec achieves privacy protection in rating prediction, the randomly sampled items in the hybrid filling strategy introduces some noise to the recommendation model, which inevitably affects the performance. This motivates us to design a lossless version of FedRec, which is critical to be deployed in a real-world application.

Secure Distributed Collaborative Filtering (SDCF)

SDCF [Jiang et al., 2019] uses stochastic gradient langevin dynamics (SGLD) [Welling and Teh, 2011] as a gradient descent method so as to defend differential attacks and prevent users' latent factors being leaked to the server. However, there may still be the leakage of the users' rating behaviors. For this reason, SDCF uses a two-stage randomized response algorithm to perturb the rated items and unrated items of each user, and then uploads the corresponding items' gradients to the server. Finally, SDCF can thus protect the users' rating behaviors similar to that of FedRec by uploading the virtually rated items' gradients.

As another issue, users have no ratings for the unrated items, hence the users can not calculate the values of the loss in the unrated items' gradients via $r_{ui} - U_{u\cdot} V_{i\cdot}^T$ (i.e., e_{ui}), $i \in \mathcal{I} \backslash \mathcal{I}_u$. To solve this problem, SDCF samples some virtual $e_{ui}, i \in \mathcal{I} \backslash \mathcal{I}_u$ from the distribution of $e_{ui}, i \in \mathcal{I}_u$ of the user u. However, this strategy will also introduce some noise to the gradient at each iteration of model training.

Decentralized Distributed Matrix Factorization (1/2)

Decentralized matrix factorization (DMF) [Chen et al., 2018] is a distributed POI recommendation framework for protecting users' rating data and solving the problem of computation and storage in the server. DMF keeps users' rating data in the corresponding local clients and utilizes these rating data to calculate the global items' latent factors and the local items' latent factors. And then each client synchronously sends the global items' gradients to his/her neighboring clients who are chosen by a random walk method.

Although this framework saves the resource of the server and avoids the risk of the rating data of all the users being leaked from the server to malicious attackers, there still exists the leakage of users' rating behaviors. Specifically, each user will receive the items' gradients from their own neighbors at each iteration of model training, and these items' gradients contain the items' IDs. Hence, this user will know the rated items of its neighbors, i.e., rating behaviors of its neighbors.

Decentralized Distributed Matrix Factorization (2/2)

PDMFRec [Duriakova et al., 2019] is also a decentralized distributed POI recommendation framework with a novel, user-centric and privacy-enhanced matrix factorization method. This framework builds a user's adjacency graph in trustworthy clients via co-rated items between users, which solves the privacy problem of user geographic location leaked by DMF [Chen et al., 2018] when constructing the user adjacency graph. Furthermore, PDMFRec also proposes two privacy-protection settings that allow users to control the privacy-protection level. In the first setting, it allows each user to only choose parts of the rated items to take part in the construction of the user's adjacency graph. In the second setting, the rated items hidden by each user in the first setting also do not take part in model training. Compared with DMF, PDMFRec has comparable performance, better privacy-protection level, i.e., protecting the users' rating behaviors and ensuring the anonymity of the sending client. The anonymity of the sending client in PDMFRec inspires us to design an effective noise elimination strategy to prevent the denoising clients from colluding with the server, which will be described in detail later.

Our Solution: FedRec++

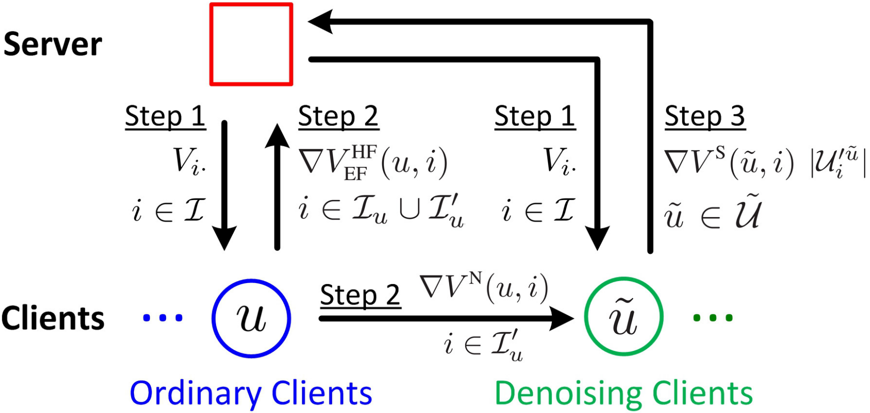

Figure: Illustration of the interactions between the server and each client in our loss federated recommendation (FedRec++). Notice that FedRec [Lin et al., 2020] is a special case of our FedRec++ without the denoising clients.

Eliminate the Gradient Noise in the Server (1/3)

In the beginning, the server initializes the model parameters $V_{i\cdot}, i \in \mathcal{I}$, and sends them to each client, i.e., step 1 in Figure 1. When each ordinary client $u \in \mathcal{U} \backslash \tilde{\mathcal{U}}$ completes the computation of the item gradients, i.e., $\nabla V_{\text{EF}}^{\text{HF}}(u, i)$, $i \in \mathcal{I}_u \cup \mathcal{I}'_u$, by using the local rating data, the server will receive the item gradients from each ordinary client $u \in \mathcal{U} \backslash \tilde{\mathcal{U}}$, i.e., step 2 in Figure 1. Then the server can calculate the summation of the gradients of each item $i \in \mathcal{I}$ as follows,

$$\nabla V_{i\cdot} = \sum_{u \in \mathcal{U} \backslash \tilde{\mathcal{U}}} \nabla V_{\text{EF}}^{\text{HF}}(u, i), \tag{4}$$

where $\mathcal{U} \backslash \tilde{\mathcal{U}}$ denotes the ordinary clients (excluding the denoising clients). Since $\nabla V_{\text{EF}}^{\text{HF}}(u, i)$ with $i \in \mathcal{I}_u \cup \mathcal{I}'_u$ contain the gradients of client u to unrated items $\mathcal{I}'_u$, the server can not identify each client u's rated items $\mathcal{I}_u$ easily. Hence, the rating behaviors of each client u are protected. Notice that $\nabla V_{i\cdot}$ in Eq.(4) is not immediately divided by $|\mathcal{U}_i \cup \mathcal{U}'_i|$ as that in FedRec [Lin et al., 2020] via Eq.(2), but is divided by the number of users that have rated item i, i.e., $|\mathcal{U}_i|$, after noise elimination for more accurate preference modeling. We will show the details in Eq.(5) and Eq.(6).

Eliminate the Gradient Noise in the Server (2/3)

The gradient noise (i.e., $\nabla V^{\mathrm{HF}}_{\mathrm{EF}}(u,i), i \in \mathcal{I}'_u$) from each ordinary client $u \in \mathcal{U} \backslash \tilde{\mathcal{U}}$ will inevitably bias the modeling of the user's preferences, which will be more serious when a larger value of ρ is used [Lin et al., 2020]. In order to eliminate the gradient noise in $\nabla V_{i\cdot}$ in Eq.(4), we design a specific algorithm for noise elimination in the server. Firstly, the server randomly selects some denoising clients $\tilde{\mathcal{U}} \subset \mathcal{U}$ as denoisers, which will be used to collect the gradient noise, i.e., $\nabla V^{\mathrm{HF}}_{\mathrm{EF}}(u,i)$, $i \in \mathcal{I}'_u$, $u \in \mathcal{U} \backslash \tilde{\mathcal{U}}$, from the ordinary clients. Secondly, each denoising client $\tilde{u}$ sends the summation of the noisy gradients of the ordinary clients (i.e., $\nabla V^{\mathrm{S}}(\tilde{u},i)$) to the server in order to eliminate the noise in $\nabla V_{i\cdot}$ in Eq.(4), which corresponds to step 3 in Figure 1. Notice that the server will also receive the number of users who virtually rated item i, i.e., $|\mathcal{U}'^{\tilde{u}}_i|$, from each denoising client $\tilde{u} \in \tilde{\mathcal{U}}$ in step 3 in Figure 1. Thirdly, once the server has received $\nabla V^{\mathrm{S}}(\tilde{u},i)$ from all the denoisers $\tilde{\mathcal{U}}$, the server can eliminate the gradient noise in $\nabla V_{i\cdot}$ in Eq.(4) as follows,

$$\nabla V_{i\cdot} \leftarrow \nabla V_{i\cdot} - \sum_{\tilde{u} \in \tilde{\mathcal{U}}} \nabla V^{\mathrm{S}}(\tilde{u},i), \tag{5}$$

where $\nabla V^{\mathrm{S}}(\tilde{u},i)$ also contains the gradients for item i of the denoiser $\tilde{u}$ itself, i.e., $\nabla V^{\mathrm{HF}}_{\mathrm{EF}}(\tilde{u},i), i \in \mathcal{I}_{\tilde{u}}$. We will describe the details in Eq.(9).

Eliminate the Gradient Noise in the Server (3/3)

After the server has eliminated the gradient noise in $\nabla V_{i\cdot}$, the server can then calculate the number of users who rated item i, i.e., $|\mathcal{U}_i| = |\mathcal{U}_i \cup \mathcal{U}'_i| - \sum_{\tilde{u} \in \tilde{\mathcal{U}}} |\mathcal{U}'^{\tilde{u}}_i|$, and update $V_{i\cdot}$ as follows,

$$V_{i\cdot} \leftarrow V_{i\cdot} - \gamma \frac{\nabla V_{i\cdot}}{|\mathcal{U}_i|}, \quad (6)$$

where γ denotes the learning rate. We depict the whole process of eliminating the gradient noise in the server in Algorithm 1.

Eliminate the Gradient Noise in the Client (1/3)

After each client $u \in \mathcal{U}$ has received the item-specific latent feature vectors $V_{i\cdot}, i \in \mathcal{I}$ from the server, i.e., step 1 in Figure 1, each client $u \in \mathcal{U}$ can use its own local rating data to calculate the gradient $\nabla U_{u\cdot}$,

$$\nabla U_{u\cdot} = \frac{\sum_{i \in \mathcal{I}_u}(-e_{ui} V_{i\cdot} + \lambda U_{u\cdot})}{|\mathcal{I}_u|}, \tag{7}$$

where $e_{ui} = r_{ui} - \hat{r}_{ui}$. Moreover, we can calculate the gradients $\nabla V_{\text{EF}}^{\text{HF}}(u, i)$, $i \in \mathcal{I}_u \cup \mathcal{I}'_u$ via Eq.(3), which is the same as that in FedRec [Lin et al., 2020].

Notice that $\nabla U_{u\cdot}$ is used to update $U_{u\cdot}$ locally in each client, and $\nabla V_{\text{EF}}^{\text{HF}}(u, i)$ are sent to the server to update $V_{i\cdot}$ with the denoising information, i.e., step 2 in Figure 1.

Notice that if a client u is a denoising client, it only needs to calculate the gradients $\nabla V_{\text{EF}}^{\text{HF}}(u, i)$ with $i \in \mathcal{I}_u$ rather than the gradients $\nabla V_{\text{EF}}^{\text{HF}}(u, i)$ with $i \in \mathcal{I}_u \cup \mathcal{I}'_u$, because the gradients $\nabla V_{\text{EF}}^{\text{HF}}(u, i)$ with $i \in \mathcal{I}_u$ would be mixed with the gradient noise of the ordinary clients before being sent to the server. Hence, the privacy of the denoising client u does not leak towards the server, i.e., the rating behaviors of client u are protected.

Eliminate the Gradient Noise in the Client (2/3)

When each ordinary client $u \in \mathcal{U} \backslash \tilde{\mathcal{U}}$ sends $\nabla V_{\text{EF}}^{\text{HF}}(u, i)$ with $i \in \mathcal{I}_u \cup \mathcal{I}'_u$ to the server, they need to send the gradient noise (i.e., $\nabla V^{\text{N}}(u, i)$, $i \in \mathcal{I}'_u$) to a denoising client $\tilde{u} \in \tilde{\mathcal{U}}$ at the same time, i.e., from the ordinary clients to the denoising clients of step 2 in Figure 1. We have the gradients $\nabla V^{\text{N}}(u, i)$ as follows,

$$\nabla V^{\text{N}}(u, i) = \nabla V_{\text{EF}}^{\text{HF}}(u, i), i \in \mathcal{I}'_u, \tag{8}$$

where $\mathcal{I}'_u$ denotes the sampled unrated items w.r.t. user u. Notice that transferring information between clients is a prominent specialty of the decentralized distributed framework. In our FedRec++, we adopt this specialty by sending gradient noise to the denoising clients, which can help eliminate the gradient noise.

For each denoising client $\tilde{u} \in \tilde{\mathcal{U}}$, it does not need to send $\nabla V_{\text{EF}}^{\text{HF}}(\tilde{u}, i)$, $i \in \mathcal{I}_{\tilde{u}}$ to the server immediately, because they can send $\nabla V^{\text{S}}(\tilde{u}, i)$ containing $\nabla V_{\text{EF}}^{\text{HF}}(\tilde{u}, i)$, $i \in \mathcal{I}_{\tilde{u}}$ to the server after collecting the gradient noise from the ordinary clients. We have $\nabla V^{\text{S}}(\tilde{u}, i)$ as follows,

$$\nabla V^{\text{S}}(\tilde{u}, i) = \sum_{u \to \tilde{u}} \nabla V^{\text{N}}(u, i) - \nabla V_{\text{EF}}^{\text{HF}}(\tilde{u}, i), i \in \mathcal{I}_{\tilde{u}}, \tag{9}$$

where the first term denotes the summation of the gradient noise sent to the denoising client $\tilde{u}$. Notice that the server can then use Eq.(5) to completely remove the noise, and the resulting $\nabla V_{i\cdot}$ on the left side of Eq.(5) contains the pure gradient of the rated item by the corresponding ordinary and/or denoising clients (i.e., users).

Eliminate the Gradient Noise in the Client (3/3)

Notice that each ordinary client $u \in \mathcal{U} \backslash \tilde{\mathcal{U}}$ only sends its own gradient noise to one denoising client. And each denoising client $\tilde{u}$ does not need to send its own gradient noise (i.e., $\nabla V^{\mathrm{N}}(\tilde{u}, i)$) to other denoisers, because each denoiser $\tilde{u}$ does not need to send the gradient noise $\nabla V_{\mathrm{EF}}^{\mathrm{HF}}(\tilde{u}, i), i \in \mathcal{I}'_{\tilde{u}}$ to the server and thus the server does not need to eliminate the gradient noise in $\nabla V_{\mathrm{EF}}^{\mathrm{HF}}(\tilde{u}, i)$, $\tilde{u} \in \tilde{\mathcal{U}}$. When each denoiser $\tilde{u}$ calculates the summation of the gradient noise from the ordinary clients, they can also calculate the number of users who virtually rated item i w.r.t. the denoiser $\tilde{u}$ at the same time, i.e., $|\mathcal{U}_i'^{\tilde{u}}|$. After each denoiser $\tilde{u} \in \tilde{\mathcal{U}}$ obtains $\nabla V^{\mathrm{S}}(\tilde{u}, i)$, they send $\nabla V^{\mathrm{S}}(\tilde{u}, i)$ and $|\mathcal{U}_i'^{\tilde{u}}|$ to the server, i.e., step 3 in Figure 1. Notice that $\nabla V^{\mathrm{S}}(\tilde{u}, i)$ in Eq.(9) will not leak the privacy of the denoising clients towards the server, because $\nabla V^{\mathrm{S}}(\tilde{u}, i)$ contains the gradient noise information of the ordinary client $u \in \mathcal{U} \backslash \tilde{\mathcal{U}}$. We describe the whole process of eliminating the gradient noise in the client in Algorithm 2.

Algorithm(1/2)

Algorithm 1 The algorithm of FedRec++ in the server.

Randomly select some clients as denoisers, i.e., $\tilde{\mathcal{U}}$.
Initialize the model parameters $V_{i\cdot}, i = 1, 2, \ldots, m$ and send them to each client $u \in \mathcal{U}$.
for $t = 1, 2, \ldots, T$ **do**
  for each client $u \in \mathcal{U}$ in parallel **do**
    ClientTraining($V_{i\cdot}, i = 1, 2, \ldots, m$; TRAINING; u; $\tilde{u}$; $\tilde{\mathcal{U}}$; t).
  end for
  Synchronize(). /*Wait for the clients to complete calculation.*/
  for $i = 1, 2, \ldots, m$ **do**
    Calculate the gradient $\nabla V_{i\cdot}$ via Eq.(4) and also $|\mathcal{U}_i \cup \mathcal{U}'_i|$.
  end for
  for each client $\tilde{u} \in \tilde{\mathcal{U}}$ in parallel **do**
    ClientTraining(NULL; COLLECTING; 0; $\tilde{u}$; NULL; 0).
  end for
  Synchronize(). /*Wait for all the clients to complete.*/
  for $i = 1, 2, \ldots, m$ **do**
    Eliminate the gradient noise in $\nabla V_{i\cdot}$ via Eq.(5).
    Calculate the number of users who rated item i, i.e., $|\mathcal{U}_i| = |\mathcal{U}_i \cup \mathcal{U}'_i| - \sum_{\tilde{u} \in \tilde{\mathcal{U}}} |\mathcal{U}'^{\tilde{u}}_i|$.
    Update $V_{i\cdot}$ via Eq.(6).
  end for
  Decrease the learning rate $\gamma \leftarrow 0.9\gamma$.
end for

Algorithm(2/2)

Algorithm 2 ClientTraining($V_{i\cdot}, i = 1, 2, \ldots, m$; OPERATION; u; $\tilde{u}$; $\tilde{\mathcal{U}}$; t), i.e., the algorithm of FedRec++ in the client.

```
if OPERATION == TRAINING then
    Sample items I'_u from I\I_u with |I'_u| = ρ|I_u|.
    Assign U_u· to U'_u·, and then update U'_u· via U'_u· ← U'_u· − γ∇U'_u· in T_lcoal iterations.
    Assign a virtual rating for item i, i ∈ I'_u via Eq.(1).
    Calculate the gradient ∇U_u· via Eq.(7) and then update U_u· via U_u· ← U_u· − γ∇U_u·.
    for i ∈ I_u ∪ I'_u do
        Calculate ∇V^HF_EF(u, i) via Eq.(3).
    end for
    Upload ∇V^HF_EF(u, i) with i ∈ I_u ∪ I'_u to the server.
      if u is not a denoising client then
          Calculate ∇V^N(u, i) with i ∈ I'_u via Eq.(8), and send ∇V^N(u, i) with i ∈ I'_u to a denoiser ũ ∈ Ũ.
      else
          Calculate ∇V^S(u, i) via Eq.(9).
      end if
else if OPERATION == COLLECTING then
      Receive the gradient noise ∇V^N(u, i) with i ∈ I'_u from user u ∈ U\Ũ.
      Calculate the summation of the gradient noise Σ_{u→ũ} ∇V^N(u, i) and the number of users who virtually rated item i
    w.r.t. the denoiser ũ, i.e., |U'^ũ_i|.
      Send the sum of gradient noise ∇V^S(ũ, i) and |U'^ũ_i| to the server.
end if
```

Discussions

The designed noise elimination strategy in the server and in the client are actually quite generic and can be applied to other recommendation methods. For example, as introduced before, SDCF [Jiang et al., 2019] uses a two-stage random response algorithm to perturb the rated items $\mathcal{I}_u$ and the unrated items $\mathcal{I}''_u$ of each user, and then calculates the gradients $\nabla V_{i\cdot}^{\text{SDCF}}$ with $i \in \mathcal{I}''_u$ of each user to the unrated items. The gradients $\nabla V_{i\cdot}^{\text{SDCF}}$ with $i \in \mathcal{I}''_u$ can be rewritten as follows,

$$\nabla V_{i\cdot}^{\text{SDCF}} = \gamma(e_{ui} U_{u\cdot} + V_{i\cdot}\mathbf{\Lambda}) - N(0, \gamma \mathbf{I}), i \in \mathcal{I}''_u, \tag{10}$$

where γ is the learning rate, e_{ui} with $i \in \mathcal{I}''_u$ are sampled from the distribution of e_{ui} with $i \in \mathcal{I}_u$, $\mathbf{\Lambda}$ is a diagonal matrix related to the Gamma distribution of the regularization term of $V_{i\cdot}$, and $\mathbf{I}$ is an identity matrix with appropriate dimension. We can see that the gradients $\nabla V_{i\cdot}^{\text{SDCF}}$ with $i \in \mathcal{I}''_u$ are similar to the gradient noise $\nabla V^{\text{N}}(u, i)$ with $i \in \mathcal{I}'_u$ in Eq.(8) in our FedRec++. Hence, we can eliminate the noise introduced by the two-stage random response algorithm in SDCF via the denoising strategy in our FedRec++.

Privacy Analysis

- Firstly, each user's original rating records are always kept locally in the client in the whole process, which ensures the security of the original data.
- Secondly, our FedRec++ adopts a hybrid filling strategy [Lin et al., 2020] to assign a virtual rating to each randomly sampled unrated item, which protects the users' rating behaviors.
- Thirdly, each ordinary client u transfers the gradients of the sampled unrated items (i.e., the gradient noise $\nabla V^{\mathrm{N}}(u, i), i \in \mathcal{I}'_u$) to a denoising client, which again does not reveal the user's rating behaviors (i.e., $\mathcal{I}_u$).
- Fourthly, the denoising client cannot identify the source (i.e., the sender) of the gradient, because the gradient does not contain the sensitive information of user ID, which guarantees the anonymity of each ordinary client [Duriakova et al., 2019].
- Finally, even if the server colludes with the denoising clients, the server cannot obtain the rating behaviors of a specific user according to the gradient noise of the ordinary clients as collected by the denoising clients because of the anonymity of the clients (i.e., the denoising clients do not know which ordinary client the gradient noise belongs to). Hence, the server cannot obtain a user u's rating behaviors $\mathcal{I}_u$ via comparing the item ID of $\nabla V^{\mathrm{HF}}_{\mathrm{EF}}(u, i), i \in \mathcal{I}_u \cup \mathcal{I}'_u$ uploaded to the server by user u and $\nabla V^{\mathrm{N}}(u, i), i \in \mathcal{I}'_u$ sent to a denoising client by user u.

Datasets and Evaluation Metrics

- Besides using the two datasets in FedRec [Lin et al., 2020], i.e., MovieLens 100K (ML100K) and MovieLens 1M (ML1M), we also include a subset from Netflix (NF5K5K). Specifically, ML100K contains $100,000$ ratings of $1,682$ movies from 943 users; ML1M contains $1,000,209$ ratings of $3,952$ movies from $6,040$ users; and NF5K5K contains $7,944,473$ ratings of $5,000$ most popular movies from $5,000$ most active users.
- We use two commonly used evaluation metrics, i.e., RMSE and MAE, for performance evaluation. Notice that the losslessness of our denoising strategy is independent of the datasets and the evaluation metrics.

Parameter Settings

For parameter configurations, we mainly follow FedRec [Lin et al., 2020]. In particular, we fix the number of latent features $d = 20$ and the number of iterations $T = 100$. We search the best value of the learning rate $\gamma \in \{0.7, 0.8, \ldots, 1.4\}$, and have $\gamma = 0.8$, $\gamma = 0.8$ and $\gamma = 1.0$ on ML100K, ML1M and NF5K5K, respectively. We search the best value of the tradeoff parameter on the regularization terms $\alpha \in \{0.1, 0.01, 0.001\}$, and have $\alpha = 0.001$ on all the three datasets. We use different values of the sampling parameter $\rho \in \{0, 1, 2, 3\}$. We choose the best value of the iteration number T_{predict} for starting filling the sampled unrated items via Eq.(1) and the iteration number T_{local} for locally training $U'_{u\cdot}$ both from $\{5, 10, 15\}$, and have $(T_{\text{predict}}, T_{\text{local}}) = (10, 10)$, $(T_{\text{predict}}, T_{\text{local}}) = (5, 15)$ and $(T_{\text{predict}}, T_{\text{local}}) = (5, 15)$ on ML100K with $\rho = 1$, $\rho = 2$ and $\rho = 3$, respectively; and have $(T_{\text{predict}}, T_{\text{local}}) = (10, 15)$, $(T_{\text{predict}}, T_{\text{local}}) = (10, 15)$ and $(T_{\text{predict}}, T_{\text{local}}) = (10, 15)$ on ML1M with $\rho = 1$, $\rho = 2$ and $\rho = 3$, respectively; and have $(T_{\text{predict}}, T_{\text{local}}) = (5, 10)$, $(T_{\text{predict}}, T_{\text{local}}) = (5, 10)$ and $(T_{\text{predict}}, T_{\text{local}}) = (5, 15)$ on NF5K5K with $\rho = 1$, $\rho = 2$ and $\rho = 3$, respectively. All the hyper parameters are searched according to the MAE performance on the first copy of each dataset.

Baselines

In order to study the effectiveness of our FedRec++, in particular of the merit of losslessness, we compare our FedRec++ with the most closely related work, i.e., FedRec [Lin et al., 2020].

In both FedRec and our FedRec++, we use PMF [Mnih and Salakhutdinov, 2007] as the backbone model and the hybrid filling strategy for virtual ratings [Lin et al., 2020].

Results(1/6)

Table: Recommendation performance of FedRec and our FedRec++ with different values of $\rho \in \{1, 2, 3\}$. Notice that we fix $c = n$ and $\eta = 1$ in our FedRec++, and copy the results of FedRec on ML100K and ML1M from [Lin et al., 2020] for reference and direct comparison.

Data	Algorithm	MAE	RMSE	ρ
ML100K	FedRec	0.7418± 0.0048	0.9424± 0.0064	0
	FedRec	0.7440± 0.0043	0.9432± 0.0056	1
	FedRec++	**0.7417**± 0.0049	**0.9422**± 0.0063	
	FedRec	0.7445± 0.0045	0.9431± 0.0057	2
	FedRec++	**0.7422**± 0.0047	**0.9430**± 0.0061	
	FedRec	0.7447± 0.0043	0.9431± 0.0054	3
	FedRec++	**0.7416**± 0.0049	**0.9421**± 0.0064	
ML1M	FedRec	0.7193± 0.0012	0.9106± 0.0015	0
	FedRec	0.7217± 0.0011	0.9129± 0.0012	1
	FedRec++	**0.7198**± 0.0011	**0.9113**± 0.0013	
	FedRec	0.7239± 0.0011	0.9152± 0.0011	2
	FedRec++	**0.7195**± 0.0011	**0.9109**± 0.0013	
	FedRec	0.7263± 0.0010	0.9178± 0.0010	3
	FedRec++	**0.7196**± 0.0013	**0.9109**± 0.0015	
NF5K5K	FedRec	0.7139± 0.0007	0.9090± 0.0008	0
	FedRec	0.7148± 0.0004	0.9102± 0.0005	1
	FedRec++	**0.7137**± 0.0008	**0.9088**± 0.0012	
	FedRec	0.7152± 0.0005	0.9104± 0.0005	2
	FedRec++	**0.7137**± 0.0002	**0.9089**± 0.0002	
	FedRec	0.7160± 0.0007	0.9110± 0.0005	3
	FedRec++	**0.7138**± 0.0005	**0.9090**± 0.0004	

Results(2/6)

Observations:

- The performance of our FedRec++ with $\rho \in \{1, 2, 3\}$ is almost the same with that of FedRec with $\rho = 0$ (i.e., without introducing gradient noise), which means that our FedRec++ is able to completely eliminate the noise introduced when assigning virtual ratings to the sampled unrated items (i.e., the denoising strategy is lossless). The results are very promising and clearly show that our FedRec++ ensures privacy in model training without sacrificing the recommendation accuracy.
- The performance of FedRec decreases with a larger value of ρ for higher security, which is expected because the volume of noise is proportional to the value of ρ. On the contrary, the performance of our FedRec++ does not decrease accordingly, which means that it can well eliminate the noise regardless of a small or large value of ρ.

Results(3/6)

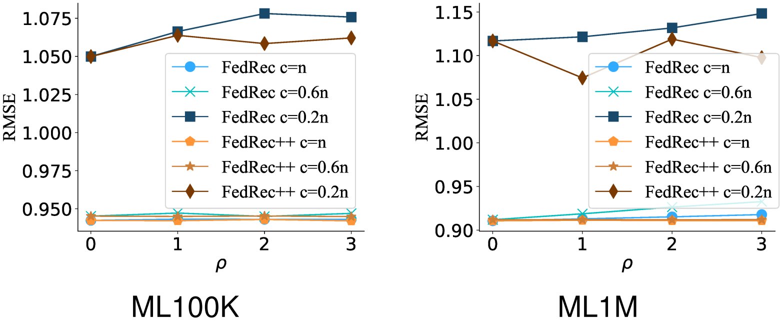

Figure: Recommendation performance of FedRec and our FedRec++ with different values of $c \in \{0.2n, 0.6n, n\}$ and $\rho \in \{1, 2, 3\}$. Notice that we fix $\eta = 1$ and the results on NF5K5K are similar to that on ML100K.

Results(4/6)

Observations:

- When $c = 0.2n$, the performance of FedRec decreases fast as ρ increases, and the overall performance of our FedRec++ is significantly better than that of FedRec on each corresponding value of ρ, which again shows the effectiveness of the noise elimination strategy in our FedRec++. Because there are fewer clients participating in model training when $c = 0.2n$, the model training is insufficient and the error is higher as expected. Hence, when $c = 0.2n$, the performance of both FedRec and our FedRec++ with $\rho = 1, 2, 3$ are worse than that in Table 3.
- When $c \in \{0.6n, n\}$, the performance of FedRec on ML100K does not decrease much with the increased values of ρ, while its performance on ML1M decreases. This means that we may choose to use the noise elimination strategy in our FedRec++ appropriately for different datasets. Importantly, the performance of our FedRec++ with $\rho \in \{1, 2, 3\}$ is almost the same with that in Table 3, which means that we may only use 60% clients in model training to achieve comparable performance as that of using all the clients.

Results(5/6)

Table: Average communication cost per iteration of each denoising client, each ordinary client and each (denoising or ordinary) client in our FedRec++ with different numbers of denoising clients $\eta \in \{0, 1, n/4, n/2\}$. Notice that we fix $c = n$ and $\rho = 1$. The unit of each cost is 80 bytes occupied by one latent vector.

Data	Denoising Client	Ordinary Client	Client	η
ML100K	0	170	170	0
	81,550	254	258	1
	571	256	350	$n/4$
	251	256	294	$n/2$
ML1M	0	265	265	0
	804,058	397	399	1
	903	397	551	$n/4$
	393	397	460	$n/2$
NF5K5K	0	2535	2535	0
	6,324,227	3799	3801	1
	7309	3800	4608	$n/4$
	3537	3799	4172	$n/2$

Results(6/6)

Observations:

- Using more denoising clients (i.e., a larger value of η) can more efficiently process the gradient noise in parallel, which thus reduces the cost of each denoising client.
- The cost of each ordinary client is almost the same when $\eta \in \{1, n/4, n/2\}$, which is expected since it is independent of the number of denoising clients.
- When $\eta \in \{n/4, n/2\}$, the cost of each client is higher than that when $\eta \in \{0, 1\}$, because there are more denoising clients sending gradients to the server. Quantitatively, the cost of each client is at most $350 - 170 = 180$, $551 - 265 = 286$, and $4608 - 2535 = 2073$ more on ML100K, ML1M and NF5K5K, respectively. Because there are 100 iterations in model training, the averaged additional communication costs for each client are $180 \times 80 \times 100$ bytes $= 1.37$ MB, $286 \times 80 \times 100$ bytes $= 2.18$ MB, and $2073 \times 80 \times 100$ bytes $= 15.8$ MB, on ML100K, ML1M and NF5K5K, respectively. We can see that the noise elimination strategy in our FedRec++ only consumes a small amount of communication cost of clients, which shows another merit of our FedRec++.

Conclusions

- In this paper, we study an emerging problem, i.e., privacy-aware recommendation with explicit feedback. In particular, we propose a novel and lossless federated recommendation method called FedRec++, for which we use some denoising clients to completely eliminate the noise caused by the assigned virtual ratings to some randomly sampled items.
- We also conduct privacy analysis to show that our FedRec++ is able to protect the user privacy well. Moreover, our FedRec++ is a generic solution, which embodies FedRec [Lin et al., 2020] as a special case, and the denoising strategy can also be used in the other privacy-aware recommendation method such as SDCF [Jiang et al., 2019].
- Experimental results on three public datasets show the effectiveness (i.e., losslessness) and efficiency (i.e., low communication cost) of our FedRec++.

Future Work

We are interested in generalizing our FedRec++ to some models with ranking losses (e.g., pairwise loss [Rendle et al., 2009] or listwise loss [Wu et al., 2018]), neural network models, and some vertical federated machine learning settings [Yang et al., 2019, Zhang et al., 2020]. We are also interested in federating some more advanced recommendation models such as those based on deep learning techniques [He et al., 2017, Liang et al., 2018, Sun et al., 2019]. Moreover, we will further explore the applicability of the denoising strategy to other works.

Thank you!

- We thank the support of National Natural Science Foundation of China Nos. 61872249 and 61836005.
- Q & A: If you have any questions and/or suggestions, welcome sending us an email: liangfeng2018@email.szu.edu.cn.

Chen, C., Liu, Z., Zhao, P., Zhou, J., and Li, X. (2018).
Privacy preserving point-of-interest recommendation using decentralized matrix factorization.
In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 257–264.

Duriakova, E., Tragos, E. Z., Smyth, B., Hurley, N., Peña, F. J., Symeonidis, P., Geraci, J., and Lawlor, A. (2019).
Pdmfrec: A decentralised matrix factorisation with tunable user-centric privacy.
In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 457–461.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T. (2017).
Neural collaborative filtering.
In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182.

Jiang, J., Li, C., and Lin, S. (2019).
Towards a more reliable privacy-preserving recommender system.
Information Sciences, 482:248–265.

Liang, D., Krishnan, R. G., Hoffman, M. D., and Jebara, T. (2018).
Variational autoencoders for collaborative filtering.
In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 689–698.

Lin, G., Liang, F., Pan, W., and Ming, Z. (2020).
Fedrec: Federated recommendation with explicit feedback.
IEEE Intelligent Systems.

Mnih, A. and Salakhutdinov, R. R. (2007).
Probabilistic matrix factorization.
In *Proceedings of the 21st International Conference on Neural Information Processing Systems*, pages 1257–1264.

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2009).
BPR: Bayesian personalized ranking from implicit feedback.
In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, pages 452–461.

Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., and Jiang, P. (2019).

Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer.
In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1441–1450.

Welling, M. and Teh, Y. W. (2011).
Bayesian learning via stochastic gradient langevin dynamics.
In *Proceedings of the 28th International Conference on Machine Learning*, pages 681–688.

Wu, L., Hsieh, C., and Sharpnack, J. (2018).
SQL-Rank: A listwise approach to collaborative ranking.
In *Proceedings of the 35th International Conference on Machine Learning*, ICML '18, pages 5311–5320.

Yang, Q., Liu, Y., Chen, T., and Tong, Y. (2019).
Federated machine learning: Concept and applications.
ACM Transactions on Intelligent Systems and Technology, 10(2):12:1–12:19.

Zhang, C., Liu, Y., Wang, L., Liu, Y., Li, L., and Zheng, N. (2020).
Joint intelligence ranking by federated multiplicative update.
IEEE Intelligent Systems, 35(4):15–24.