Proceedings on Privacy Enhancing Technologies ; 2020 (3):264–283

Seira Hidano*, Takao Murakami, Shuichi Katsumata, Shinsaku Kiyomoto, and Goichiro Hanaoka

Exposing Private User Behaviors of Collaborative Filtering via Model Inversion Techniques

Abstract: Privacy risks of collaborative filtering (CF) have been widely studied. The current state-of-the-art inference attack on user behaviors (e.g., ratings/purchases on sensitive items) for CF is by Calandrino et al. (S&P, 2011). They showed that if an adversary obtained a moderate amount of user's public behavior before some time T, she can infer user's private behavior *after* time T. However, the existence of an attack that infers user's private behavior *before* T remains open. In this paper, we propose the first inference attack that reveals past private user behaviors. Our attack departs from previous techniques and is based on *model inversion* (MI). In particular, we propose the first MI attack on factorization-based CF systems by leveraging data poisoning by Li et al. (NIPS, 2016) in a novel way. We inject malicious users into the CF system so that adversarialy chosen "decoy" items are linked with user's private behaviors. We also show how to weaken the assumption made by Li et al. on the information available to the adversary from the whole rating matrix to only the item profile and how to create malicious ratings effectively. We validate the effectiveness of our inference algorithm using two real-world datasets.

Keywords: inference attacks, model inversion attacks, recommender systems, collaborative filtering, data poisoning, privacy exposure

DOI 10.2478/popets-2020-0052
Received 2019-11-30; revised 2020-03-15; accepted 2020-03-16.

***Corresponding Author: Seira Hidano:** KDDI Research, Inc., E-mail: se-hidano@kddi-research.jp
Takao Murakami: National Institute of Advanced Industrial Science and Technology (AIST), E-mail: takao-murakami@aist.go.jp
Shuichi Katsumata: National Institute of Advanced Industrial Science and Technology (AIST), E-mail: shuichi.katsumata@aist.go.jp
Shinsaku Kiyomoto: KDDI Research, Inc., E-mail: kiyomoto@kddi-research.jp
Goichiro Hanaoka: National Institute of Advanced Industrial Science and Technology (AIST), E-mail: hanaoka-goichiro@aist.go.jp

1 Introduction

Recommender systems have become an imperative component of electronic commerce systems [52, 53, 58] as used by Amazon, YouTube, IMDB, Google, and so on. Given a user's historical behavior such as ratings or purchases, a recommender system suggests a product that meets the user's preference. Typically, recommender systems rely on *collaborative filtering* (CF), which is a method to analyze the patterns of the historical behavior of the users. As recommender systems play an increasing role on the Web, it has become a compelling research topic to understand the exploitation and privacy aspects of recommendation systems [16, 26]. In particular, the dynamic nature of recommender systems (i.e., recommender systems periodically update the internal parameters according to user's new behaviors) leads to a broad attack surface, and we have seen numerous latent risks exposed in the past few decades exploiting this nature [8, 13, 22, 25, 30–32, 38, 49–51, 61, 63, 66, 68].

When discussing privacy risks for recommender systems, we often consider two types of private information: *behavior of users* [8, 51] and *attributes of users* [1, 48, 63, 70]. Here, *behavior* is the purchase or rating histories of users in a recommender system, and can be further divided into *public* and *private* behavior. Public behavior is shared through social networks such as blogs and Facebook [35, 56, 60, 61], or through *item similarity lists*, e.g., Amazon's "Customers who bought this item also bought..." feature [8]. Private behavior is those wished to be kept private such as purchase history of sensitive items; e.g., medical items, sexual movies. On the other hand, *attributes* are private information tied to the users independently of the system; e.g., gender, political views, and sexual orientations.

In the past decade, we have seen significant development regarding inference attacks on user attributes [1, 17, 27, 30, 48, 63, 70]. However, inference attacks on user (private) behaviors have been limited. As far as our knowledge goes, there have only been two works [8, 51] whose focus is on such attacks on recommender systems. For example, Calandrino et al. [8] proposed an algorithm that takes a moderate amount of user's public behavior before some time T and infers the user's private behavior *after* time T via observing the temporal changes in the public outputs of a recommender system (more details are provided in Section 2 and Appendix C). This provides evidence that we can infer user's private behavior *after* some time T. However, the existence of an attack that infers user's private behavior *before* time T remains open. Considering that we have a good understanding of the privacy risks of user *attributes*, this lack in understanding of privacy risks of user *behaviors* is quite unsatisfactory; the privacy of user behaviors is as important as that of user attributes. Therefore, the main question we investigate in this paper is: **to what extent can we infer user's *past* private behavior from recommender systems?**

Our Contributions. In this work, we show that there exists an attack that infers user's *past* private behavior (i.e., past purchase or rating histories) in a CF-based recommender system.

Technically, we propose the first *model inversion* (MI) attack for CF-based recommender systems. The MI attack [14, 15, 21, 64] — recently introduced by [15] in a context different to recommender systems — allows an adversary to predict private information used as input to the model, when given access to the prediction model along with some auxiliary information (See Section 2.3 and Appendix D for details). Thus, at a high level, an MI attack against recommender systems allows an adversary to predict private user behaviors. Our MI attack is based on the recent poisoning attack on CF-based recommender systems by Li et al. [32] in combination with the general framework for MI attacks proposed by Hidano et al. [21] (see "Technical Overview" for details). We investigate the setting of an *active* adversary where she can inject fake (malicious) users using the dynamic nature of recommender systems. We further assume the adversary knows non-sensitive items included in the top-l items recommended to the users by the recommender system, i.e., she gets to see the public output of the recommender system. This has been a widely accepted assumption in MI attacks [14, 15, 21, 64], and is also backed up by the spread of sharing (non-sensitive) personal information/behavior via social networks to improve the effectiveness of recommender systems [35, 56, 60, 61], e.g., sharing personal ratings of items on blogs, Facebook, and IMDB.

We demonstrate a successful user behavior inference attack on two real-world datasets: Foursquare location dataset [67] and MovieLens dataset [19]. For the Foursquare (resp. MovieLens) dataset, our inference attack achieved at most 100% precision and 50% recall (resp. 74.6% precision and 23.5% recall). For example, for the Foursquare dataset, we viewed visits to any locations in the *hospital* category as private behavior, and show that the adversary can infer whether or not a user has visited any hospital. Our MI attack also has the option of adding the functionality of avoiding attack detection. Assuming a detector with the one-class SVM (OCSVM) [9, 54], we show that our MI attack avoids detection, while keeping high precision of the attack.

In summary, our contributions are as follows:

- We propose the first inference attack that infers user's *past* private behavior in a CF-based recommender system.
- We propose the first MI attack for CF-based recommender systems. The technique we develop may be of independent interest since it can weaken the assumption and boost the efficiency of the poisoning attack on factorization-based recommender systems in [32] (see "Technical Overview" for details).
- We conduct experiments using two real-world datasets and provide evidence that our theoretical attack works as intended. We also show that our inference attack can still be effective under standard detectors of malicious/fake users.

Technical Overview. Our MI attack takes full advantage of the dynamic nature of recommender systems and deviates from standard MI attacks for static prediction systems [14, 15, 64]. In particular, our methodology is inspired by the recent MI attack on (dynamic) linear regression models by Hidano et al. [21]. In [21], they introduced a general methodology for conducting MI attacks using *poisoning attacks* [4, 31, 32] — a technique widely studied to maximally degrade the effectiveness or credibility of the system by injecting malicious user data into the model. At a high level, their MI attack works in two flows: The adversary first injects carefully crafted malicious data into the dynamic model so that the current model will be updated to a *target* model. Then, the adversary infers private information used as input by only observing the output of the target model.

Two key challenges in proposing an MI attack for CF are: (i) defining an appropriate target model which allows the adversary to infer the user's past private behavior and (ii) formulating an efficient and hard to detect poisoning attack which modifies the model to the target model. To address the first challenge, we define a target model in such a way that a user who buys or rates a particular sensitive item X in the past at any time will be recommended some non-sensitive *decoy* items $\{Y_i\}$ chosen by the adversary. Here, a decoy item should be considered as non-sensitive to the users (e.g., family movies, daily commodities), and should be an item *not* recommended to most users. Notably, when any decoy item Y_i (which is typically not recommended to users) is recommended to a user by the recommender system and the adversary obtains knowledge of that fact, then she will be able to infer that the user had bought or rated the sensitive item X with high probability.

To address the second challenge, we formulate our MI attack based on the recent poisoning attack on factorization-based CF by Li et al. [32]. They cast poisoning attacks as an optimization problem of finding the optimal choice of malicious users to inject that would maximize the adversary's utility (objective), where they considered two well-studied utilities for poisoning attacks: maximizing the error of the recommender system and boosting/reducing the popularity of certain products. However, these utilities do not aim at MI attacks nor can they be directly used for MI attacks. In this paper, we propose a new type of data poisoning attack that *links sensitive items with decoy items while avoiding the detection of data poisoning.*

In addition, one of the shortcomings of the poisoning attack in [32] is that they assume that the adversary has complete knowledge of the rating matrix. That is, full access to the training data (i.e., information on which user bought/rated which item) of the recommender system. Although this may be an acceptable assumption in the poisoning attack setting of [32], this is unacceptable for our setting. The adversary having full knowledge of the training data is equivalent to the adversary having full knowledge of all the user's past behaviors.

Thus we propose a data poisoning attack that *only takes as input an item profile.* Since the item profile does not contain the user's sensitive information, the system may share it with the users (or release it publicly) [40, 46]. The adversary may also estimate the item profile from public web datasets [55], or approximate the item-by-item matrix by using item similarity or "related items" lists publicly available on the web services [8]. Thus, the assumption that the adversary has the item profile is much weaker and practical. Moreover, our algorithm that only uses the item profile can be made more efficient than the algorithm in [32] by exploiting the fact that we do not rely on user information. In Section 4.4, we also conducted experiments where the adversary estimates an item profile using some partial information on the rating matrix to reflect real-world scenarios.

Last but not least, we propose a method to *optimize which items to rate.* In [32], they randomly generated items to rate for each malicious user account. Instead, we propose a method to optimize which items to rate, under the constraint that each malicious users can rate at most b_{max} items. This is done by solving an optimization problem with the constraint on the l_1 norm of the ratings for each malicious user account.

2 Background and Related Works

2.1 Factorization-based CF

Preliminaries. A recommender system based on factorization-based CF predicts unobserved entries in the *rating matrix* $\mathbf{M} \in \mathbb{R}^{m\times n}$ by using matrix factorization [29, 57, 59], and recommends items for a user based on the estimated entries. Here, $m, n \in \mathbb{N}$ denote the numbers of users and items in the recommender system, respectively, and the (i, j)-th element $\mathbf{M}_{i,j}$ of $\mathbf{M}$ represents the rating of the j-th item by the i-th user.

Since a user generally rates only a small number of items, there are many *unobserved* elements in the rating matrix $\mathbf{M}$. We denote the set of observed elements in $\mathbf{M}$ by $\Omega = \{(i, j) : \mathbf{M}_{i,j} \text{ is observed}\}$. Let $\mathcal{R}_\Omega : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$ be a *masking function* that takes as input an $m \times n$ matrix $\mathbf{A}$ and outputs an $m \times n$ matrix $\mathcal{R}_\Omega(\mathbf{A})$ by masking $\mathbf{A}$ with Ω. Specifically, given $\mathbf{A}$, $\mathcal{R}_\Omega$ outputs a matrix $\mathcal{R}_\Omega(\mathbf{A})$ whose (i, j)-th element $[\mathcal{R}_\Omega(\mathbf{A})]_{i,j}$ is defined as follows:

$$[\mathcal{R}_\Omega(\mathbf{A})]_{i,j} = \begin{cases} \mathbf{A}_{i,j} & \text{if } (i,j) \in \Omega \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

We provide a table summarizing the notations used throughout this paper in Appendix A for reference.

Prediction Algorithm. We provide details on how factorization-based CF is implemented. In the initial training phase, we first predict the unobserved elements of the rating matrix $\mathbf{M}$ using matrix factorization as $\widehat{\mathbf{M}} = \mathbf{UV}$, where $\widehat{\mathbf{M}} \in \mathbb{R}^{m\times n}$ is a prediction of $\mathbf{M}$, $\mathbf{U} \in \mathbb{R}^{m\times k}$ and $\mathbf{V} \in \mathbb{R}^{k\times n}$ are called *user profiles* and *item profiles*, respectively, and we assume $k \ll \min(m, n)$ in general. We denote the set of $\mathbf{U}$ and $\mathbf{V}$ by Θ; i.e., $\Theta = \{\mathbf{U}, \mathbf{V}\}$, and call Θ a *model.*

There are multiple ways to compute Θ from $\mathbf{M}$; e.g., alternating least squares (ALS) [29], nuclear-norm minimization [7]. In this paper, we focus on the ALS which is widely used in practice. ALS computes an approximate solution to the following optimization problem:

$$\min_{\Theta} \; ||\mathcal{R}_{\Omega}(\mathbf{M} - \mathbf{UV})||_F^2 + \lambda||\Theta||_F^2, \tag{2}$$

where $||\cdot||_F^2$ represents the Frobenius norm (i.e., square sum of all elements), and λ (> 0) is called a *regularization parameter.* The first term in (2) is the summation of square errors over the *observed* elements in $\mathbf{M}$. The second term in (2) is introduced to avoid overfitting the observed elements, and is called a *regularization term.* λ controls the extent of regularization, and is generally determined by cross-validation [29].

Since the optimization problem (2) is not convex, it is typically infeasible to find an exact solution. However, if we fix either $\mathbf{U}$ or $\mathbf{V}$, the optimization problem (2) becomes convex in the remaining unfixed variable, and can be solved optimally. Thus, given an initial value for Θ, ALS updates $\mathbf{U}$ by solving (2) for $\mathbf{U}$ while fixing $\mathbf{V}$. Then it updates $\mathbf{V}$ by solving (2) for $\mathbf{V}$ while fixing $\mathbf{U}$. It iterates this process until convergence, and uses the converged model Θ as an approximate sub-optimal solution to (2). After computing Θ, we can compute a prediction $\widehat{\mathbf{M}}$ of $\mathbf{M}$ by using $\widehat{\mathbf{M}} = \mathbf{UV}$.

In the predication phase, the factorization-based CF system recommends new items that each users are potentially interested in based on $\widehat{\mathbf{M}}$. This is often referred to as a user-to-item based (or factorization-based) CF. The user-to-item based CF won the Netflix prize [29] and also have been the focus of many studies including Li et al. [32]. In the user-to-item based CF, the system recommends items for which the corresponding elements in $\mathbf{M}$ are unobserved and the predicted ratings in $\widehat{\mathbf{M}}$ are high. One of the most natural methods for recommending such items for each user $i \in [m]$ is to recommend the *top-l items* (e.g., $l = 1, 5, 10$) with high (predicted) ratings in $\widehat{\mathbf{M}}_i$ among unobserved items. Here $\widehat{\mathbf{M}}_i$ represents the i-th row of $\widehat{\mathbf{M}}$. In practice, the system may recommend these top-l items by showing them on the web page after the user logs in, or by sending users an e-mail or message containing the item information.

Dynamic Updates. It is common for practical recommender systems (regardless of it being factorization-based CF) to update their internal systems when new users and/or items are added or when users make new ratings on items. Notably, recommender systems are *dynamic* where they update the model Θ on a regular basis (e.g., every night, once a week depending on the application). This extra feature of recommender systems has been shown to be exploitable in several ways. In the next subsection, we review some of the attacks.

2.2 Exploitation and Privacy Risks for Recommender Systems

Owing to its broad attack surface, recommender systems have been an attractive subject to understand its exploits and privacy risks. Below we review previous works. Due to page limitation, we briefly describe the outline of the previous works in this section, and describe a more extensive review in Appendix C.

Exploitation. There are many attacks aiming to maliciously modify the recommender system in a way so that an adversary can control how items will be recommended to users. Looking ahead, our work is in the line of *poisoning* attacks (also known as *shilling* attacks) [13, 31, 32, 41, 47, 68] which injects malicious users into the recommender system to control the output behavior of the system. Recently, sophisticated poisoning attacks for specific types of recommender systems aiming to maximize the error of the system or boost/reduce the popularity of particular items using (non-convex) optimization techniques have emerged [13, 32, 68].

Privacy Risks. Inference attacks aim to illegitimately infer sensitive information of the users in recommender systems and are considered to be one of the major privacy risks. There are two types: inference attacks against *user attributes* and *user behaviors.* As already mentioned in Section 1, the latter type is the main target of this paper. So far, there have been two works [8, 51] studying such attacks on recommender systems. Calandrino et al. [8] proposed an attack that tracks the change in the item profile after time T (the time on which the adversary decides to attack the system), hence cannot be used to infer user's private behavior before time T.

2.3 Model Inversion Attacks

Model inversion (MI) attacks are a type of attack aiming to expose sensitive information that was used as input to a prediction model [14, 15, 21, 64]. In MI attacks, an adversary is provided the following inputs: some information on the prediction model f, the output of the prediction model $f(\mathbf{x})$ or the true label y corresponding to the features $\mathbf{x}$ of a *target* user, and some auxiliary information depending on the attack scenario. Then, the goal of the adversary is to predict the sensitive portion (or all) of the features $\mathbf{x}$ of the target user. In this work, we consider the so-called *grey-box* setting where the adversary gets oracle access to f along with the structure of f and some auxiliary information is further known, e.g., the fact that f is a linear regression model and some (not all) regression coefficients. We describe further details on MI attacks including the explanation of the black/grey/white-box in Appendix D.

Hidano et al. [21] proposed a new type of MI attacks which targets *dynamic* prediction systems. They modified the system's model Θ by injecting malicious data. Our work will follow this approach.

2.4 Formalizing Poisoning Attacks for Factorization-based CF

We finally review the poisoning attack for factorization-based CF by Li et al. [32].

Let $m' \in \mathbb{N}$ be the number of malicious users that will be injected to the recommender system by the adversary, and let $\mathbf{M}' \in \mathbb{R}^{m' \times n}$ be a rating matrix of all m' malicious users. We denote the set of observed elements in $\mathbf{M}'$ by $\Omega' = \{(i,j) : \mathbf{M}'_{i,j} \text{ is observed}\}$. In other words, the adversary performs data poisoning by creating m' malicious user accounts, whose rating matrix is given by $\mathbf{M}'$ (as we describe later in detail, the adversary computes $\mathbf{M}'$ in advance for her intended purpose).

As explained above, due to the dynamic feature of recommender systems, after the new (malicious) user information $\mathbf{M}'$ is added to $\mathbf{M}$, the factorization-based CF updates the model Θ. Specifically, it predicts the concatenated rating matrix (including unobserved elements) using matrix factorization as follows:

$$\begin{bmatrix} \widehat{\mathbf{M}}^* \\ \widehat{\mathbf{M}}' \end{bmatrix} = \begin{bmatrix} \mathbf{U}^* \\ \mathbf{U}' \end{bmatrix} \mathbf{V}^*, \tag{3}$$

where $\widehat{\mathbf{M}}^* \in \mathbb{R}^{m \times n}$ (resp. $\widehat{\mathbf{M}}' \in \mathbb{R}^{m' \times n}$) is a prediction of $\mathbf{M}$ (resp. $\mathbf{M}'$), $\mathbf{U}^* \in \mathbb{R}^{m \times k}$ (resp. $\mathbf{U}' \in \mathbb{R}^{m' \times k}$) is a user profile for m users (resp. m' malicious users), and $\mathbf{V}^* \in \mathbb{R}^{k \times n}$ is an item profile. Note that $\widehat{\mathbf{M}}^*$, $\mathbf{U}^*$, and $\mathbf{V}^*$ are now different from $\widehat{\mathbf{M}}$, $\mathbf{U}$, and $\mathbf{V}$, respectively, due to data poisoning (we use a symbol with "$*$" to represent the poisoned data). Let $\Theta^* = \{\mathbf{U}^*, \mathbf{U}', \mathbf{V}^*\}$ be the poisoned model. Θ^* is computed from $\mathbf{M}$ and $\mathbf{M}'$ in the same way as in Section 2.1. For example, if we use ALS, we will find an approximate solution to the following optimization problem:

$$\begin{aligned} \min_{\Theta^*} \; & ||\mathcal{R}_{\Omega}(\mathbf{M} - \mathbf{U}^*\mathbf{V}^*)||_F^2 \\ & + ||\mathcal{R}_{\Omega'}(\mathbf{M}' - \mathbf{U}'\mathbf{V}^*)||_F^2 + \lambda ||\Theta^*||_F^2. \end{aligned} \tag{4}$$

Here, $[\mathcal{R}_{\Omega'}(\mathbf{A})]_{i,j}$ equals to $\mathbf{A}_{i,j}$ if $(i,j) \in \Omega'$ and 0 otherwise. Fig. 1 depicts the rating matrix, its prediction, user profile, and item profile before/after data poisoning. Due to data poisoning, the prediction of $\mathbf{M}$ changes

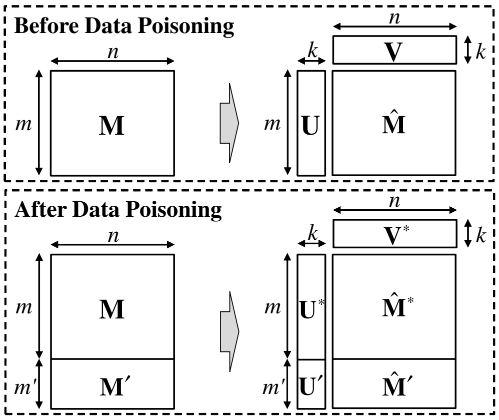


Fig. 1. Rating matrix, its prediction, user profile, and item profile before/after the poisoning attacks.

from $\widehat{\mathbf{M}} = \mathbf{U}\mathbf{V}$ to $\widehat{\mathbf{M}}^* = \mathbf{U}^*\mathbf{V}^*$. Since the system recommends items based on the prediction of $\mathbf{M}$, the recommended items also change owing to data poisoning.

Li et al. [32] proposed a method to optimize the malicious rating matrix $\mathbf{M}'$ for the adversary's purpose, given that the adversary has *full access* to the original rating matrix $\mathbf{M}$. As for the adversary's purpose, they considered an *availability attack* and an *integrity attack* (and their combination). They then defined, for each attack, a *utility function* $R : \mathbb{R}^{m \times n} \times \mathbb{R}^{m \times n} \to \mathbb{R}$, which takes as input $\widehat{\mathbf{M}}^*$ and $\widehat{\mathbf{M}}$, and outputs a *utility score* that measures the goodness of $\widehat{\mathbf{M}}^*$ for the adversary respective to the adversary's purpose. Informally, in an availability attack an adversary tries to maximize the recommendation error of the recommendation system and in an integrity attack the adversary tries to boost/reduce the popularity of specific items.

For both utility functions, Li et al. [32] proposed an algorithm to find $\mathbf{M}'$ that maximizes the utility score $R(\widehat{\mathbf{M}}^*, \widehat{\mathbf{M}})$ under the assumption that the adversary has full access to $\mathbf{M}$. They use projected gradient descent where the gradient computation is based on first-order KKT conditions to solve the optimization problem. For both utility functions, the time complexity of their algorithm can be expressed as $O((m+m')nk + (|\Omega| + |\Omega'|)k)$, where the former term is dominant in typical systems since $|\Omega| + |\Omega'| \ll (m+m')n$. Here, we note that to estimate the time complexity, we exploit the fact that derivation of specific parameters contain many zero values. Some details on the mechanics of the algorithm will become more clear in Section 3 and for the exact details on the algorithm of Li et al., see [32].

3 Model Inversion Attacks for Collaborative Filtering Systems

In this section, we propose an inference attack for factorization-based CF that can infer past private behavior of users. To this end, we propose the first model inversion (MI) attack for the factorization-based CF.

3.1 Threat Model

Environment of recommender system. We first explain our threat model. We consider a recommender systems based on factorization-based CF. As is the case for standard deployed recommender systems, we assume the system is dynamic, i.e., updates the model parameter periodically, and that an adversary can maliciously inject fake users into the system by generating new user accounts. Moreover, an adversary can rate arbitrary items using the fake user accounts. We note that our MI is grey-box. Namely, the adversary knows that the recommender system is a factorization-based CF and further knows the value of the internal parameters such as number of users m and items n. Finally, we consider user-to-item based recommendation (See Section 2.1).

Attack goal. The goal of the adversary is to infer past sensitive behaviors (i.e., rating histories) of users. Since we consider dynamic recommender systems, a user may rate items at any point during the lifetime of the system. In other words, the goal of the adversary is to infer any past sensitive behavior of the users. Moreover, since in practice, an adversary may not be able to inject arbitrary malicious users due to detection of the system, we consider adversaries that try to subvert such detections.

Adversary's input. Firstly, we assume the adversary is given the item profile (denoted as $\mathbf{V}$ in Section 2.1) of the recommender system as auxiliary information.In practice, there are multiple of ways to collect such information. In the simplest case, since the item profile does not apparently contain user's sensitive information, the system may share it with the users or release it publicly [40, 46]. It has also become standard practice for recommender systems to publish item similarity lists (e.g., Amazon's "Customers who bought this item also bought..." feature) [8], so the adversary can approximate the item-by-item matrix, hence, the item profile from such information.

The adversary can also estimate the item profile from public web datasets [55] or some partial information on the rating matrix. For example, in some real-world services such as Amazon and Google Maps, we can observe all of the public ratings for some users by accessing the users' profile pages[1]. In Section 4.4, we also conducted experiments where the adversary estimates an item profile using such information.

Next, the adversary is given the non-sensitive part of the top-l items recommended to the target user as input. Recall that in an MI attack, the adversary is given the output of the prediction model or the true label corresponding to the features of a target user as input (See Section 2.3). This is arguably a harder to acquire information compared to the item profile, however, we do not consider this as a significant impediment to our attack. Namely, due to the wide spread of social networks and its increasing role in recommendation systems [35, 56, 60, 61], it has become common for users to share their personal (non-sensitive) recommendations through social networks: for example most websites running recommender systems provide the users the option to embed the information to blogs or to share it via Twitter and Facebooks. We can also consider alternative indirect approaches. For instance, on many websites, users publicly rate or comment on (non-sensitive) items, revealing a high likelihood of being recommended that item by the system. Other sites which are not directly tied to the user's rating histories, but leak partial information of the user may be taken advantage as well [24, 37, 44, 45]. For examples, "liked" movies and books listed in a Facebook user profile may reveal partial information of recommended items on Amazon.

3.2 Overview

In our attack, the adversary creates m' malicious user accounts whose rating matrix is given by $\mathbf{M}'$. The goal of the attack is to *link the sensitive items with some adversarialy chosen "decoy" items which are seemingly innocuous to users.* Fig. 2 shows an overview of the proposed attack. We provide an in-depth overview below.

Preparation. Let $\mathcal{X} \subseteq [n]$ be the set of sensitive items (e.g., adult movies, medical items) and $\mathcal{Y} \subseteq [n] \setminus \mathcal{X}$ be a set of "decoy" items. The decoy items are seemingly in-

1 Note that unlike social networks, the user names on the profile pages are *nicknames* in most cases, and hence the adversary does not know their real names. However, if the users disclose decoy items via social networks (as described in Section 3.2), their past sensitive behaviors are revealed along with their real names.

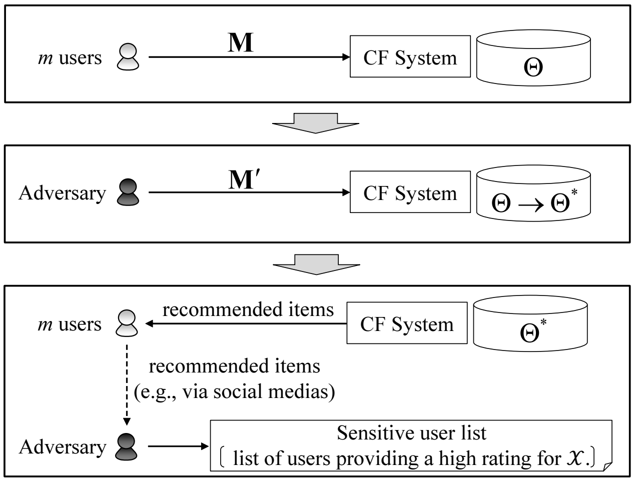


Fig. 2. Overview of our MI attack. Using them, the adversary creates a *sensitive user list*, which is a list of users who provided a high rating for a sensitive item in $\mathcal{X}$.

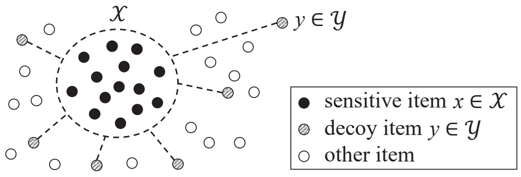


Fig. 3. Visualization of items after data poisoning. The dashed lines represent that there is a high similarity between sensitive items $\mathcal{X}$ and decoy items $\mathcal{Y}$.

nocuous for users (e.g., non-erotic novels, commodities) and are chosen by the adversary in advance. One way to do so is to randomly choose $\mathcal{Y}$ from the set of non-sensitive items $[n] \setminus \mathcal{X}$. This is simple but effective, as shown in our experiments. Let $\mathbf{M}$ be the rating matrix which represents the ratings provided by the m users for some items *before* data poisoning. We assume that the rating matrix $\mathbf{M}$ includes high ratings for sensitive items in $\mathcal{X}$ provided by some users. Our goal is to learn $\mathbf{M}$ or put differently to model invert the recommender system.

Malicious user injection. The adversary performs data poisoning by creating m' malicious user accounts and providing ratings for items, which are represented as $\mathbf{M}'$. The rating matrix $\mathbf{M}'$ is designed so that *sensitive items $\mathcal{X}$ will be linked with decoy items $\mathcal{Y}$* by adding $\mathbf{M}'$ to $\mathbf{M}$ (we describe desirable properties for $\mathbf{M}'$ in more detail in Section 3.3, and explain how to compute $\mathbf{M}'$ in Section 3.4). Fig. 3 shows a visualization of the items after data poisoning.

Algorithm 1: Model Inversion

```
1 Create $m'$ malicious user accounts and add $\mathbf{M}'$
    to $\mathbf{M}$ (How to compute $\mathbf{M}'$ is provided in
    Section 3.5).
2 Obtain the recommended items for the $m$ users.
3 Output a sensitive user list by adding a user to
    the list if her recommended items include $l'$
    decoy items.
```

Inference of sensitive user behavior. After data poisoning, the system updates the model from Θ to Θ^*, and recommends m users a new list of items (e.g., top-l items as described in Section 2.1), e.g., by showing them on the web page after log-in or by sending personalized e-mail/message. Since after poisoning, there is a high similarity between $\mathcal{X}$ and $\mathcal{Y}$, the recommended items should include one or more decoy items with high probability for those who provided a high rating for a sensitive item in $\mathcal{X}$. In our threat model (See Section 3.1), the adversary observes the non-sensitive part of the recommended items. Using them, the adversary creates a list of users who provided a high rating for an item in $\mathcal{X}$, which we call a *sensitive user list*. Specifically, the adversary adds a user to the sensitive user list if her recommended items include $l' \in \mathbb{N}$ decoy items, where l' is a pre-determined threshold (e.g., $l' = 1, 2, \cdots, 5$). Since there is a high similarity between $\mathcal{X}$ and $\mathcal{Y}$, the adversary would be confident that a user rated an item in $\mathcal{X}$ if her recommended items include l' decoy items. **Algorithm 1** summarizes our model inversion algorithm.

Remarks on our MI attack. Our MI attack has no special requirements on the sizes $|\mathcal{X}|$ and $|\mathcal{Y}|$. When there are multiple items in $\mathcal{X}$, the adversary cannot specify, for each user in the list, which item among $\mathcal{X}$ she rated. However, if $\mathcal{X}$ is a set of items of the same type (e.g., adult movies, books having the same political orientation, or medical items), the adversary can infer her sexual inclination, political views, or health condition that she wants to keep as secret. Therefore, the adversary can violate a user's privacy just by disclosing the fact that she provided a high rating for some (unspecified) item in $\mathcal{X}$. It is also possible to extend our attack so that different items in $\mathcal{Y}$ correlate differently with items in $\mathcal{X}$. In this case, the adversary can specify which item in $\mathcal{X}$ users evaluated/purchased.

In addition, note that the sensitive user list contains two types of errors: *precision* and *recall*, and that the trade-off between them can be controlled by changing the value of l'. When l' is large, the adversary can decide with very high confidence that users in the sensitive user list provided a high rating for a sensitive item in $\mathcal{X}$. However, she fails to include users who rated a sensitive item in $\mathcal{X}$ if the recommended items include less than l' decoy items. Thus, the adversary can increase precision at the cost of recall by increasing l'. In general, if the adversary is targeting to attack some unspecified group of users, then precision is considered to be more desirable than recall. This is because even if recall is low, those users recommended the decoy items will be known by the adversary with high confidence that they have rated sensitive items in the past. Therefore, in real-life scenarios, an adversary being able to mount such attacks even with, say 10% recall, may be quite damaging.

3.3 Utility Function for MI Attacks

To formally cast our MI attack as an optimization problem, we first propose a utility function R^{mi} which captures the motive of the adversary. As described in the overview (Section 3.2), we would like to design the maliciously injected rating matrix $\mathbf{M}'$ so that sensitive items $\mathcal{X}$ will be linked with decoy items $\mathcal{Y}$. Namely, the similarity between $\mathcal{X}$ and $\mathcal{Y}$ should increase after injecting $\mathbf{M}'$ to the system. On the other hand, although not mandatory, we would also like to avoid detection of malicious user accounts as much as possible.

A natural way to define a utility function R^{mi} which captures the above intent would be to define R^{mi} so that it takes the rating matrices $\mathbf{M}$ and $\mathbf{M}^*$ as input: this is the approach taken by Li et al. [32] for their poisoning attack (See Section 2.4). However, unfortunately, as mentioned in Section 3.1, in an MI attack the adversary does not have knowledge of the rating matrix $\mathbf{M}$ and is only given the item profile $\mathbf{V}$ as input. Therefore, we approximate the utility function that we really want with the following utility function $R^{\mathsf{mi}} : \mathbb{R}^{n\times k} \times \mathbb{R}^{n\times k} \to \mathbb{R}$ which takes as input *item profiles* $\mathbf{V}^*$ *and* $\mathbf{V}$:

$$R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V}) = \sum_{i=1}^{k} \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} (\mathbf{V}_{i,x} - \mathbf{V}^*_{i,y})^2 + \sum_{i=1}^{k} \sum_{x\in\mathcal{X}} (\mathbf{V}_{i,x} - \mathbf{V}^*_{i,x})^2 + \mu||\mathbf{V}^* - \mathbf{V}||_F^2. \quad (5)$$

Below, we explain the terms in (5) in detail and justify that this approximates the utility function we want.

First, we observe that the item profile $\mathbf{V}$ can be used as an approximation of the rating matrix $\mathbf{M}$. In factorization-based CF, the item profile $\mathbf{V}$ is a low-dimensional representation of the property or concept that the items have. In other words, each row of $\mathbf{V}$ can be regarded as a cluster (also known as latent factor) [36]. In the poisoning phase, since we want to boost the similarity between the sensitive items $\mathcal{X}$ and decoy items $\mathcal{Y}$, we can instead use the item profile $\mathbf{V}$ which is a succinct representation of the relation between all items. In more detail, since each row of the predicted rating matrix $\widehat{\mathbf{M}}$ is a linear combination of the item profile $\mathbf{V}$ (See Fig. 1), in order for a decoy item in $\mathcal{Y}$ to have a high prediction in case the rating for a sensitive item in $\mathcal{X}$ was high, it suffices to consider the item profile.

Now that we saw we can approximate $\mathbf{M}$ by $\mathbf{V}$, we proceed to explain each term in (5). We begin with the first and second terms. At a high level, the accuracy of our MI attack increases with decrease in the two terms. The first (resp. second) term in (5) is the summation of the square errors between the entries of sensitive items $\mathcal{X}$ in $\mathbf{V}$ and decoy items $\mathcal{Y}$ (resp. sensitive items $\mathcal{X}$) in $\mathbf{V}^*$. The first term captures intent of "increase the similarity between $\mathcal{X}$ and $\mathcal{Y}$" and the second term captures intent of "maintain high ratings for sensitive items $\mathcal{X}$ unchanged". The former is self-explanatory. The latter term is required since even if we increase the similarity between sensitive items $\mathcal{X}$ and decoy items $\mathcal{Y}$, our attack would be void unless the decoy items are recommended to the users.

More specifically, if we decrease the square errors between sensitive items $\mathcal{X}$ *after poisoning* and decoy items $\mathcal{Y}$ *after poisoning* (i.e., if we decrease $\sum_{i=1}^{k} \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} (\mathbf{V}^*_{i,x} - \mathbf{V}^*_{i,y})^2$) instead of the first and second terms, then all of the entries in $\mathcal{X}$ and $\mathcal{Y}$ might be very small after poisoning (i.e., we might find a trivial solution such as: $\mathbf{V}^*_{i,x} = \mathbf{V}^*_{i,y} = 0$ for any $i \in [k]$, any $x \in \mathcal{X}$, and any $y \in \mathcal{Y}$). In this case, decoy items would not be recommended to the users who had rated sensitive items. We introduce the first and second terms to avoid this issue. By the second term, if a user had rated a sensitive item high, then the updated model will predict that the user has high affinity with the sensitive item. Then by the first term, it will also predict (i.e., recommend) a decoy item to the user.

Finally, we explain the meaning of the third term in (5). At a high level, this is included to make the attack less detectable. The term is the summation of square errors between elements in $\mathbf{V}^*$ and $\mathbf{V}$. When this term is small, the item profile $\mathbf{V}$ is not changed much by data poisoning. Therefore, since $\mathbf{V}^*$ and $\mathbf{V}$ approximate the behaviors of $\widehat{\mathbf{M}}^*$ and $\widehat{\mathbf{M}}$, respectively, the two predicted rating matrices $\widehat{\mathbf{M}}^*$ and $\widehat{\mathbf{M}}$ do not change much by data poisoning. Furthermore, taking into consideration that $\widehat{\mathbf{M}}$ is a prediction of the actual rating matrix $\mathbf{M}$, we can expect that the injected malicious users $\mathbf{M}'$ which is used to generate the item profile $\mathbf{V}^*$ do not deviate much from the actual user behavior $\mathbf{M}$. Specifically, as the third term decreases, the injected malicious users by our attack will become less detectable. Since this has the opposite effect from the first two terms, we include a hyper parameter μ (≥ 0) to control the trade-off between the precision of our MI attack and the difficulty of detection. The accuracy of our MI attack will increase by moving μ close to 0. Conversely, the detection of our attack becomes more difficult by increasing μ.

3.4 Casting Our MI Attack as an Optimization Problem

In this section, we formalize our MI attack as an optimization problem using the utility function R^{mi} defined in (5). As a recap, the variable we want to optimize is the rating matrix $\mathbf{M}'$ that we inject into the system and we assume that the item profile $\mathbf{V}$ is available to the adversary. We also assume that each rating is bounded in the range $[-\Lambda, \Lambda]$ or $[0, \Lambda]$, where Λ is a positive number known to the adversary. Since the description of our algorithm is essentially the same for both ranges, in the following we only explain our algorithm for the former range $[-\Lambda, \Lambda]$. We also let $\mathbb{M}_1$ be the set of all rating matrices $\mathbf{M}'$ such that each rating in $\mathbf{M}'$ is bounded in the range $[-\Lambda, \Lambda]$. Finally, we assume that each malicious user account only rates at most $b_{max} \in \mathbb{N}$ items. We make this assumption to reduce the computational resource required for the adversary to rate items of malicious users. Moreover, this is useful for avoiding detection of our attack; intuitively, if the malicious user rates too many items, since it is an uncommon behavior for standard users, the adversary may easily detect.

In our MI attack, we optimize the b ($\leq b_{max}$) items each injected malicious users rate. We note that in the poisoning attack of Li et al. [32], the items are randomly chosen. To summarize so far, the optimization problem of our MI attack can be expressed as solving the following optimal malicious rating matrix $\mathbf{M}'_{opt,l_0} :=$

$$\underset{\mathbf{M}' \in \mathbb{M}_1}{\arg\min} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V}) \text{ s.t. } \forall i \in [m'], ||\mathbf{M}'_i||_0 \leq b_{max}, \quad (6)$$

where $\mathbf{M}'_i$ is the i-th row of $\mathbf{M}'$, and $||\cdot||_0$ is the l_0 norm; i.e., $||\mathbf{M}'_i||_0$ is the number of non-zero elements in $\mathbf{M}'_i$. Thus, $\mathbf{M}'_{opt,l_0}$ in (6) minimizes the utility function $R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})$ among $\mathbb{M}_1$ under the constraint that each malicious user rates at most b_{max} items. However, unfortunately, the optimization problem in (6) is non-convex and cannot be solved efficiently.

To circumvent the problem, we relax the l_0 norm constraint with the l_1 norm constraint, and solve an approximated version of (6). Approximating the l_0 norm with the l_1 norm is a well-known technique in sparse modeling [36]. The actual optimization problem we consider is expressed as solving the following optimal malicious rating matrix $\mathbf{M}'_{opt,l_1} :=$

$$\underset{\mathbf{M}' \in \mathbb{M}_1}{\arg\min} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V}) \text{ s.t. } \forall i \in [m'], ||\mathbf{M}'_i||_1 \leq b_{max}\Lambda, \quad (7)$$

where $||\cdot||_1$ is the l_1 norm; i.e., $||\mathbf{M}'_i||_1 = \sum_{j=1}^{n} |\mathbf{M}'_{i,j}|$. $||\mathbf{M}'_i||_1$ equals to $b_{max}\Lambda$ when the malicious user provides the highest rating ($= \Lambda$) to b_{max} items and does not rate any other items. Since the l_1 norm constraint promotes sparsity [20], the optimization problem in (7) provides a sparse solution $\mathbf{M}'_{opt,l_1}$, in which each malicious user rates only a small number of items, and hence, can be used as an approximation to $\mathbf{M}'_{opt,l_0}$.

3.5 Solving the Optimization Problem

3.5.1 Projected Gradient Descent

In this section, we explain how to solve the optimization problem in (7). Since the l_1 norm is convex, we can find an approximate solution to (7) by using the projected gradient descent (PGD) method. Given some initial matrix $\mathbf{M}'^{(0)}$, we solve for $\mathbf{M}'_{opt,l_1}$ by sequentially updating $\mathbf{M}'$ as follows:

$$\mathbf{M}'^{(t)} = \mathrm{Proj}_{\mathbb{M}_2}\left(\mathbf{M}'^{(t-1)} - s_t \cdot \nabla_{\mathbf{M}'} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})\right), \quad (8)$$

where $\mathbf{M}'^{(t)}$ is $\mathbf{M}'$ computed at the t-th iteration, s_t is the step size at the t-th iteration, $\nabla_{\mathbf{M}'} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V}) \in \mathbb{R}^{m' \times n}$ is the direction to descend, and $\mathrm{Proj}_{\mathbb{M}_2}$ is the projection operator onto $\mathbb{M}_2$, where $\mathbb{M}_2$ is defined as the set of all feasible matrices $\mathbf{M}'$ defined as

$$\begin{aligned}\mathbb{M}_2 = \{\mathbf{M}' | &\forall i \in [m'], \forall j \in [n], \\ &\mathbf{M}'_{i,j} \in [-\Lambda, \Lambda], ||\mathbf{M}'_i||_1 \leq b_{max}\Lambda\}. \end{aligned} \quad (9)$$

We postpone the discussion on how to compute $\nabla_{\mathbf{M}'} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})$ in (8) to Section 3.5.2 and proceed with how to compute $\mathrm{Proj}_{\mathbb{M}_2}$.

Specifically, after computing $\nabla_{\mathbf{M}'} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})$ in (8), we subtract $s_t \cdot \nabla_{\mathbf{M}'} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})$ from $\mathbf{M}'^{(t-1)}$, and project the result value onto $\mathbb{M}_2$. Since $\mathbb{M}_2$ is convex, the projection onto $\mathbb{M}_2$ can be written as a projection onto a point in $\mathbb{M}_2$ that minimizes the Frobenius norm:

$$\mathrm{Proj}_{\mathbb{M}_2}(\mathbf{A}) = \underset{\mathbf{B} \in \mathbb{M}_2}{\arg\min} ||\mathbf{A} - \mathbf{B}||_F^2. \quad (10)$$

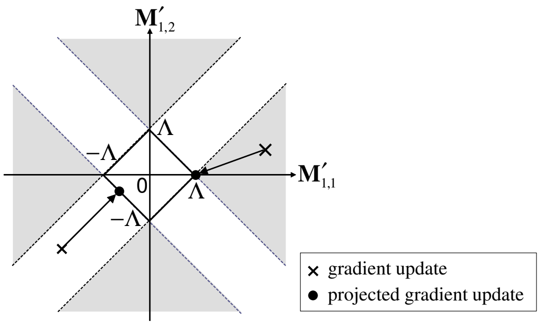


Fig. 4. Projected gradient update ($m' = 1$, $n = 2$, $b_{max} = 1$). A gradient update ($= \mathbf{M}'^{(t-1)} + s_t \cdot \nabla_{\mathbf{M}'} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})$)) is projected onto $\mathbb{M}_2 = \{\mathbf{M}' \mid |\mathbf{M}'_{1,1}| + |\mathbf{M}'_{1,2}| \le \Lambda\}$.

Fig. 4 shows two examples of the projected gradient updates computed via (8) and (10) in the case where $m' = 1$, $n = 2$, and $b_{max} = 1$. In this figure, the gradient update in the shaded area is projected onto a corner of $\mathbb{M}_2 = \{\mathbf{M}' \mid |\mathbf{M}'_{1,1}| + |\mathbf{M}'_{1,2}| \le \Lambda\}$, which permits a sparse solution.

We update $\mathbf{M}'$ by computing a projected gradient update via (8) and (10) until convergence, and use the converged value as an approximation of $\mathbf{M}'_{opt,l_1}$. If the number of non-zero elements in $\mathbf{M}'_i$ of the converged matrices exceeds b_{max}, we choose b_{max} items with the highest absolute ratings $|\mathbf{M}'_{i,j}|$ and set ratings of the remaining items to zero. In this way, we satisfy the constraint that each malicious user account rates at most b_{max} items.

3.5.2 Computing the Gradient $\nabla_{\mathbf{M}'} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})$

It remains to show how to compute $\nabla_{\mathbf{M}'} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V}) \in \mathbb{R}^{m' \times n}$ in (8). Firstly, by using the chain rule, $\nabla_{\mathbf{M}'} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})$ can be written as follows:

$$\nabla_{\mathbf{M}'} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V}) = (\nabla_{\mathbf{M}'} \mathbf{V}^*)(\nabla_{\mathbf{V}^*} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})), \quad (11)$$

where $\nabla_{\mathbf{M}'} \mathbf{V}^* \in \mathbb{R}^{(m' \times n) \times (n \times k)}$ and $\nabla_{\mathbf{V}^*} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V}) \in \mathbb{R}^{n \times k}$. Below we describe how to compute the individual terms $\nabla_{\mathbf{M}'} \mathbf{V}^*$ and $\nabla_{\mathbf{V}^*} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})$ in detail.

Computation of $\nabla_{\mathbf{M}'} \mathbf{V}^*$. We compute $\nabla_{\mathbf{M}'} \mathbf{V}^*$ through the KKT (Karush-Kuhn-Tucker) condition [6] similarly to the approach took in [32]. Specifically, a model Θ^* that minimizes (4) satisfies the following KKT condition for $j \in [n]$:

$$\begin{aligned}\lambda \mathbf{v}^*_j = &\textstyle\sum_{i \in \Omega_j} (\mathbf{M}_{i,j} - \mathbf{u}^{*\top}_i \mathbf{v}^*_j) \mathbf{u}^*_i \\ &+ \textstyle\sum_{i \in \Omega'_j} (\mathbf{M}'_{i,j} - \mathbf{u}'^{\top}_i \mathbf{v}^*_j) \mathbf{u}'_i, \end{aligned} \quad (12)$$

where $\mathbf{u}^*_i \in \mathbb{R}^k$ (resp. $\mathbf{u}'_i \in \mathbb{R}^k$) is the i-th row of $\mathbf{U}^*$ (resp. $\mathbf{U}'$), $\mathbf{v}^*_j \in \mathbb{R}^k$ is the j-th row of $\mathbf{V}^*$, and $\Omega_j \subseteq [m]$ (resp. $\Omega'_j \subseteq [m']$) is the set of observed rows in the j-th column of $\mathbf{M}$ (resp. $\mathbf{M}'$); i.e., $\Omega_j = \{i : \mathbf{M}_{i,j} \text{ is observed}\}$, $\Omega'_j = \{i : \mathbf{M}'_{i,j} \text{ is observed}\}$.

Since $(\mathbf{a}^\top \mathbf{b})\mathbf{a} = (\mathbf{a}\mathbf{a}^\top)\mathbf{b}$ for any vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^k$, the equation (12) can be expressed as follows:

$$\begin{aligned}&\left(\lambda \mathbf{I}_k + \textstyle\sum_{i \in \Omega_j} \mathbf{u}^*_i \mathbf{u}^{*\top}_i + \sum_{i \in \Omega'_j} \mathbf{u}'_i \mathbf{u}'^{\top}_i\right) \mathbf{v}_j \\ &= \textstyle\sum_{i \in \Omega_j} \mathbf{M}_{i,j} \mathbf{u}^*_i + \sum_{i \in \Omega'_j} \mathbf{M}'_{i,j} \mathbf{u}'_i \quad (\text{for } j \in [n]), \end{aligned} \quad (13)$$

where $\mathbf{I}_k$ is the $k \times k$ identity matrix. By differentiating both sides of (13) with respect to $\mathbf{M}'_{i,j}$, we can compute each entry of $\nabla_{\mathbf{M}'} \mathbf{V}^*$ as follows:

$$\frac{\partial \mathbf{v}_j}{\partial \mathbf{M}'_{i,j}} = \left(\lambda \mathbf{I}_k + \sum_{i \in \Omega_j} \mathbf{u}^*_i \mathbf{u}^{*\top}_i + \sum_{i \in \Omega'_j} \mathbf{u}'_i \mathbf{u}'^{\top}_i\right)^{-1} \mathbf{u}'_i, \quad (14)$$

It should be noted, however, that we cannot actually compute (14). The adversary who runs this algorithm only has knowledge of the item profile $\mathbf{V}$ and therefore does not have knowledge of the user profile $\mathbf{U}^*$ or $\mathbf{U}'$. In the setting of Li et al. [32], this was not a big issue since they assumed knowledge of the rating matrix $\mathbf{M}$. Specifically, after they injected $\mathbf{M}'$ to the system, they were able to compute $\mathbf{U}^*$ and $\mathbf{U}'$ using matrix factorization.

To address this issue, we aim to simulate the real rating matrix $\mathbf{M}$ only using the item profile $\mathbf{V}$. If we can do this, then we can use the method of [32] to proceed with computing (14). To this end, we randomly generate $\mathbf{U}$ and construct a predicted rating matrix $\widehat{\mathbf{M}}$ (see Fig. 1). Here we assume that the adversary knows the domain of elements in $\mathbf{U}$ (e.g., $[0, 1]$ as commonly assumed in non-negative matrix factorization [12, 23]), and randomly generates $\mathbf{U}$ from a uniform distribution within this range. The adversary can also estimate the domain of $\mathbf{U}$ from $\mathbf{V}$ and the domain of $\mathbf{M}$. Since $\widehat{\mathbf{M}}$ is a prediction of the actual rating matrix $\mathbf{M}$, $\widehat{\mathbf{M}}$ is an approximation of $\mathbf{M}$. Therefore, we can compute $\mathbf{U}^*$ and $\mathbf{U}'$ via matrix factorization from the approximated $\mathbf{M}$ and $\mathbf{M}'^{(t-1)}$. Finally, we compute $\nabla_{\mathbf{M}'} \mathbf{V}^*$ by computing $\partial \mathbf{v}_j / \partial \mathbf{M}'_{i,j}$ for all $i \in [m']$ and $j \in [n]$ via (14).

Computation of $\nabla_{\mathbf{V}^*} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})$. By differentiating both sides of (5) with respect to $\mathbf{V}^*_{i,j}$, we obtain

$$\frac{\partial R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})}{\partial \mathbf{V}^{*\top}_{i,j}} = \begin{cases} 2(\mu + 1)(\mathbf{V}^*_{i,j} - \mathbf{V}_{i,j}) & \text{if } j \in \mathcal{X} \\ \sum_{x \in \mathcal{X}} 2(\mathbf{V}^*_{i,j} - \mathbf{V}_{i,x}) + 2\mu(\mathbf{V}^*_{i,j} - \mathbf{V}_{i,j}) & \text{if } j \in \mathcal{Y} \\ 2\mu(\mathbf{V}^{*\top}_{i,j} - \mathbf{V}^\top_{i,j}) & \text{otherwise.} \end{cases} \quad (15)$$

Algorithm 2: Computing $\mathbf{M}'$

```
1  Input: $\mathbf{V}$, $k$, $m'$, $\mu$, $b_{max}$, $\Lambda$, $\lambda$
2  Output: $\mathbf{M}'$
3  Initialize $\mathbf{M}'^{(0)}$. $t \leftarrow 1$.
4  repeat
5      Compute $\nabla_{\mathbf{M}'} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})$ by (11), (14), (15).
6      Compute $\mathbf{M}'^{(t)}$ by (8)–(10).
7      $t \leftarrow t + 1$.
8  until $\mathbf{M}'^{(t)}$ converges.
9  Choose $b_{max}$ items with the highest absolute ratings $|\mathbf{M}'_{i,j}|$ and set ratings of the remaining items to zero for each row of $\mathbf{M}'^{(t)}$.
10 return $\mathbf{M}' := \mathbf{M}'^{(t)}$.
```

Thus, we can compute $\nabla_{\mathbf{V}^*} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})$ by computing $\partial R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})/\partial \mathbf{V}^*_{i,j}$ for all $i \in [k]$ and $j \in [n]$ via (15). Recall that we can compute this because the adversary is given $\mathbf{V}$ as input.

Summary of Algorithm. The proposed algorithm to compute $\mathbf{M}'$ can be summarized in **Algorithm 2**. Note that in Section 4, we initialize $\mathbf{M}'^{(0)}$ by a zero matrix.

3.6 Discussions on Our Algorithm and Further Optimization

We finally provide some discussions on our proposed algorithm and provide simple techniques to boost the efficiency of our algorithm. The time complexity of our algorithm is dominated by the time complexity of computing the gradient in (11) in each iteration of the PGD method. The time complexity of computing (11) can be further broken down into time it takes to compute the two gradients $\nabla_{\mathbf{M}'} \mathbf{V}^*$ and $\nabla_{\mathbf{V}^*} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})$, and the time to multiply them together.

First, we analyze the simplest term $\nabla_{\mathbf{V}^*} R^{\mathsf{mi}}(\mathbf{V}^*, \mathbf{V})$. Since we compute (15) for all $i \in [k]$ and $j \in [n]$, the time complexity to compute this gradient is $O(nk)$. On the other hand, the computation of $\nabla_{\mathbf{M}'} \mathbf{V}^*$ is more heavy. Since we simulate the rating matrix $\mathbf{M}$ by $\widehat{\mathbf{M}} \in \mathbb{R}^{m \times n}$ and perform matrix factorization on the concatenated matrix of $\widehat{\mathbf{M}}$ and $\mathbf{M}'$ (See Fig. 1), the time complexity to compute this gradient is dominated by $O((mn + \Omega')k)$. Here, note that $\widehat{\mathbf{M}}$ is a dense matrix which is a prediction of the (sparse) rating matrix $\mathbf{M}$, and $\mathbf{M}'$ is a sparse matrix which is injected to the recommender system with $\Omega' \ll m'k$ observed entries. Finally, at first glance computing (11) may require $O(m'n^2k)$ time complexity, however, by exploiting the fact that each row of $\nabla_{\mathbf{M}'} \mathbf{V}^* \in \mathbb{R}^{(m' \times n) \times (n \times k)}$ (where we view this as a $\mathbb{R}^{m'n \times nk}$ matrix) contains only k non-zero values, the actual time complexity can be expressed as $O(m'nk)$. To summarize, the time complexity of our algorithm is $O((m + m')nk)$. Note that this is the same time complexity as the poisoning algorithm of Li et al. [32].

Finally, we provide a simple way to boost efficiency of our proposed algorithm. We use this alternative algorithm in the experiment in Section 4. As we explained in Section 3.5.2, to compute $\nabla_{\mathbf{M}'} \mathbf{V}^*$ we randomly sampled a user profile $\mathbf{U} \in \mathbb{R}^{m \times k}$ to simulate the rating matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$. This was because the item profile $\mathbf{V}$ looses all information on the users, and to reconstruct $\mathbf{M}$, we had to simulate the user profile $\mathbf{U}$. However, put differently, since $\mathbf{V}$ does not include any information on the users, there is in fact no reason for the algorithm to choose a user profile of size $m \times k$. Specifically, since we assume that the latent factors of the users and items have a low-dimensional representation, our algorithm works equivalently well with a random user profile $\widetilde{\mathbf{U}} \in \mathbb{R}^{k \times k}$ which is a more succinct representation of the user property. Recall here that k was a small number representing the dimension of the latent factors. Put differently, since we only had knowledge of the item profile to begin with, our algorithm is agnostic to the number of users in the recommender system. Therefore, substituting this simple modification into our proposed algorithm, we can lower the computational time of $\nabla_{\mathbf{M}'} \mathbf{V}^*$ down to $O((nk + \Omega')k)$. In total, the time complexity of the algorithm is now $O((k + m')nk)$. Since $k \ll m$, this is much more efficient than the original algorithm.

4 Experimental Evaluation

In this section, we provide experimental results to validate the effectiveness of our proposed inference attack. We then discuss possible defenses against our attack.

4.1 Datasets and System Settings

In our experiments, we considered two types of recommender systems using factorization-based collaborative filtering: a *point-of-interest (POI) recommender system* [33], and a *movie recommender system* [29]. The POI recommender system presents unvisited locations that a user would be concerned with from the locations the user visited in the past. The movie recommender system proposes unseen movies that a target user would prefer based on the past ratings on her watched movies. Below we explain the datasets and system settings in detail.

Datasets. We used the *Foursquare global-scale check-ins* dataset [67] for the POI recommender system, and the *MovieLens 20M* dataset [19] for the movie recommender system. Below we explain these datasets.

- *Foursquare global-scale check-ins [67]:* The Foursquare dataset contains user's check-in data at various venues throughout the world. In our experiments, we used check-in data at venues located in Manhattan (latitude: $[40.7, 40.8]$, longitude: $[-74.04, -73.92]$). We extracted users with at least 10 check-ins, and venues with at least 10 check-ins in the target region. The check-in data for each user was converted to binary rating data; i.e., 1 means that the user has visited the venue one or more times, and 0 means that the user has not visited the venue. We randomly selected 80% of ratings per each user for training, and used the remaining 20% ratings for testing (we did not use the rating time in selecting training data, since the factorization-based CF does use the time information). We created the rating matrix $\mathbf{M}$ from the training data. The numbers of users and items in $\mathbf{M}$ were $m = 1722$ and $n = 1983$, respectively.

- *MovieLens 20M [19]:* The MovieLens dataset includes user's ratings on movies. Each rating is made on a value between 0.5 and 5.0 with 0.5 increments. We extracted the most recent 100K ratings based on time stamps, and then selected users with at least 20 ratings and movies with at least 20 ratings. We randomly selected 80% of ratings per each user for training (i.e. for creating the rating matrix $\mathbf{M}$), and used the remaining 20% ratings for testing. The number of users and items in $\mathbf{M}$ were $m = 814$ and $n = 1124$, respectively.

Parameter settings of the recommender system. To set the parameters of the target recommender systems, we used three performance metrics: the root mean square error (RMSE), and precision and recall in the top-l recommendation (RSPre@l and RSRecl@l). Their definitions are given in Appendix B.

We evaluated the RMSE to determine the number k of columns (or rows) in $\mathbf{U}$ (or $\mathbf{V}$) and the regularization parameter λ. For each dataset, we determined the values of k and λ that achieve the lowest RMSE. We evaluated RSPre@l and RSRecl@l to determine the number l of recommended items. We determined l so that RSPre@l and RSRecl@l took almost the same value.

For each dataset, we performed matrix factorization 20 times with different initial values of $\mathbf{U}$ and $\mathbf{V}$. As a result, for the Foursquare dataset, the parameters k, λ, and l were set to $k = 30$ and $\lambda = 0.01$, and $l = 5$. The average values of RMSE, RSPre@l, and RSRec@l were 0.0306, 0.0215, and 0.0340, respectively. For the MovieLens dataset, the parameters k, λ, and l were set to $k = 30$ and $\lambda = 0.02$, and $l = 10$. The average values of RMSE, RSPre@l, and RSRec@l were 0.8525, 0.0421, and 0.0466, respectively.

4.2 Attack Performance

Experimental Set-up. We first evaluated the performance of our MI attack when an attack detector is not used in the recommender system. To this end, we performed our attack without consideration for attack detection; i.e., we set $\mu = 0$ in (5). Below we describe how we performed our attack for each of the two systems.

- *Attack for the POI recommender system:* For the POI recommender system, we assumed a scenario where an adversary attempts to disclose a *hospital* the user has visited in the past. The Foursquare dataset has a category label such as "Restaurant", "Office", and "Hospital" for each venue. We used this information to determine sensitive and decoy items. Specifically, we regarded a "Hospital" venue visited by the most users as a sensitive item (18 users out of 1722 users visited this venue). We then randomly selected three decoy items from venues not included in the "Hospital" category. The categories of the decoy items were "Restaurant", "Music Venue", and "Bus Line".

We optimized each rating in the malicious rating matrix $\mathbf{M}'$ in the range $[0, 1]$ using **Algorithm 2** in Section 3.5. Here, since the average number of items rated by each (normal) user was 15, we set the maximum number of items a malicious user could rate to $b_{\max} = 15$. We set the number of malicious users m' so that the ratio α between m' and m $(= 1722)$ was $\alpha = 0.01, 0.02, 0.03, 0.04$, or 0.05; i.e., $m' = \alpha m = 17, 34, 52, 69$, or 86. After computing $\mathbf{M}'$ using **Algorithm 2**, we rounded each rating in $\mathbf{M}'$ to 0/1.

Then we performed our MI attack using **Algorithm 1** in Section 3.2. We set a threshold l' to determine whether a user rated a sensitive item to $l' = 1, 2$, or 3.

- *Attack for the movie recommender system.* For the movie recommender system, we assumed a scenario where an adversary attempts to disclose an *immoral* movie the user has given the highest rating (5-star rating). The MovieLens dataset includes a category label such as "sci-fi", "teen", and "adultery" for each movie. These category labels were tagged by users. We selected sensitive and non-sensitive items based on this information. Specifically, we regarded an "adultery" movie highly rated by the most users as a sensitive item (20 users out of 814 users gave 5-star ratings to this movie). We then randomly selected five decoy items from movies not in the "adultery" category. The categories of the decoy items were "thriller", "action", "mockumentary", "teen", and "satire".

We optimized each rating in the malicious rating matrix $\mathbf{M}'$ in the range $[0, 5]$ using **Algorithm 2**. Since the average number of items rated by each user (normal) was 62, we set the maximum number of items that a malicious user could rate to $b_{\max} = 62$. We set the number of malicious users m' so that $\alpha = 0.01, 0.02, 0.03, 0.04,$ or 0.05; i.e., $m' = \alpha m = 8, 16, 24, 33,$ or 41. After computing $\mathbf{M}'$ using **Algorithm 2**, we rounded each rating to a value between 0.5 and 5.0 with 0.5 increments.

Then we performed our MI attack using **Algorithm 1**. We set a threshold l' to $l' = 1, 2, 3, 4,$ or 5.

As the performance metrics for MI attacks, we used the following precision (MIPre) and recall (MIRec):

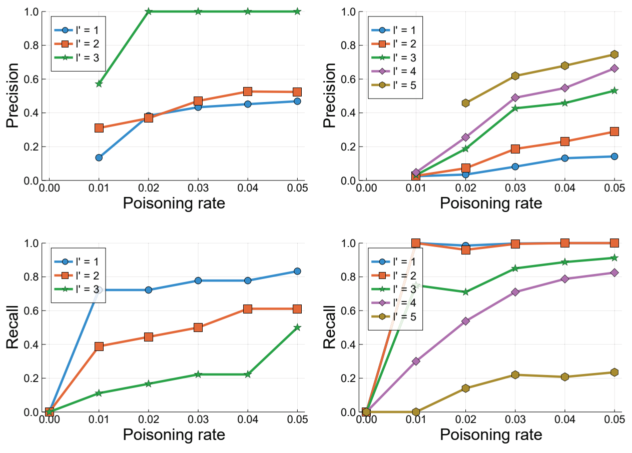
$$MIPre = \frac{|\mathcal{U}_{\mathcal{X}} \bigcap \mathcal{U}_{\mathcal{Y},l'}|}{|\mathcal{U}_{\mathcal{Y},l'}|}, \; MIRec = \frac{|\mathcal{U}_{\mathcal{X}} \bigcap \mathcal{U}_{\mathcal{Y},l'}|}{|\mathcal{U}_{\mathcal{X}}|}, \quad (16)$$

where $\mathcal{U}_{\mathcal{X}}$ is the set of users who have highly rated a sensitive item, and $\mathcal{U}_{\mathcal{Y},l'}$ is the set of users to whom l' or more decoy items are recommended.

We performed our MI attack 20 times for each dataset with different initial values of $\mathbf{U}$. As described in Section 3.5.2, we generated $\mathbf{U}$ from a uniform distribution over the interval $[0, 1]$. We then evaluated the averages of MIPre and MIRec to stabilize performance.

We also considered a *random guess* as a baseline attack for comparison. In this attack, the adversary randomly selects a user and predicts that the user highly rated a target sensitive item in the past. The success probability of this attack (i.e., the probability that the adversary's prediction is correct) is given by $|\mathcal{U}_{\mathcal{X}}|/m$, where $|\mathcal{U}_{\mathcal{X}}|$ is the number of users who have highly rated a sensitive item. We call this success probability the *baseline precision*. For the Foursquare and MovieLens datasets, the baseline precision was 1.05% (= 18/1722) and 2.46% (= 20/814), respectively.

Evaluation Results. Fig. 5 shows the experimental results. The horizontal axis shows the poisoning rate ($\alpha = m'/m$), whereas the vertical axis shows precision (MIPre) or recall (MIRec) of our MI attack. Note that when no decoy items are recommended (i.e., when the denominator of (16) is zero), we cannot compute the precision. We do not plot the precision for such cases.



(a) Foursquare dataset. (b) MovieLens dataset.

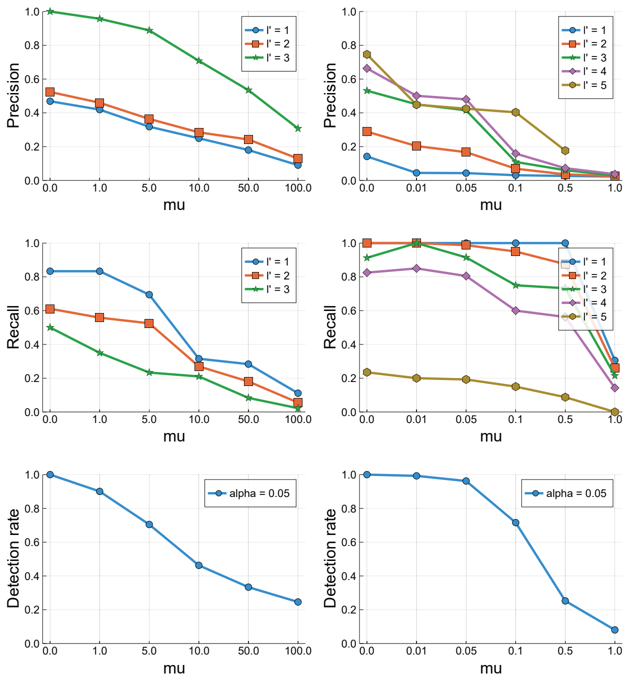
Fig. 5. Precision (MIPrec) and recall (MIRec) of our MI attack with $\mu = 0$.

It can be seen from Fig. 5 that our inference attack is a realistic and possible threat for actual recommender systems. For the Foursquare dataset, our attack achieved 100% precision and 50% recall when $\alpha = 0.05$ and $l' = 3$. This means that the adversary successfully revealed a half of the users who had visited a hospital with 100% accuracy. For the MovieLens dataset, the precision and recall were 74.6% and 23.5%, respectively, when $\alpha = 0.05$ and $l' = 5$. As explained above, the baseline precision for the Forsquare dataset (resp. the MovieLens dataset) was 1.05% (resp. 2.46%). Thus, our attack has much higher precision than a baseline attack.

On the other hand, at first glance, one may feel that recall is much lower compared to precision. However, as described in Section 3.2, precision is considered to be more desirable than recall for the adversary in practical scenarios. For our experiment of MovieLens dataset, around one-fourth of the users who rated highly an immoral movie is exposed by a fairly high confidence (74.6% precision). For some users this may be higher exposure rate than they are comfortable with.

It can also be seen from Fig. 5 that both precision and recall of our attack are increased with an increase in the number of malicious users. In addition, a trade-off between the precision and recall can be controlled by changing the threshold l', as described in Section 3.2. The adversary can increase her precision at the cost of recall by increasing l'.

Furthermore, it is important to note that the poisoning rate of our attack is much lower than that of the data poisoning attack by Li et al. [32]. The attack of Li et al. requires 10% or more poisoning rate to achieve high performance. On the other hand, our attack re-



(a) Foursquare dataset. **(b)** MovieLens dataset.

Fig. 6. Precision (MIPrec) and recall (MIRec), and detection rate of malicious users.

quires at most 5% poisoning rate. We consider this is because our attack only changes the columns for sensitive items and decoy items in $\mathbf{V}$ (see the first and second terms in (5)). More specifically, the attack of Li et al. aims at degrading the accuracy of a recommender system. In this case, the adversary needs to drastically change many elements in user profile $\mathbf{U}$ and item profile $\mathbf{V}$. Therefore, the adversary needs to inject quite a lot of malicious user accounts. On the other hand, our attack only changes a small portion of $\mathbf{V}$, and therefore requires much smaller malicious data.

4.3 Resistance to Attack Detection

Experimental Set-up. If the behavior of malicious users is totally different from that of normal users, it would be easy to detect them using anomaly detection techniques [10]. However, our MI attack can avoid such detection by increasing the value of a hyper parameter μ in (5). In this subsection, we evaluate the resilience of our attack to anomaly detection by changing μ.

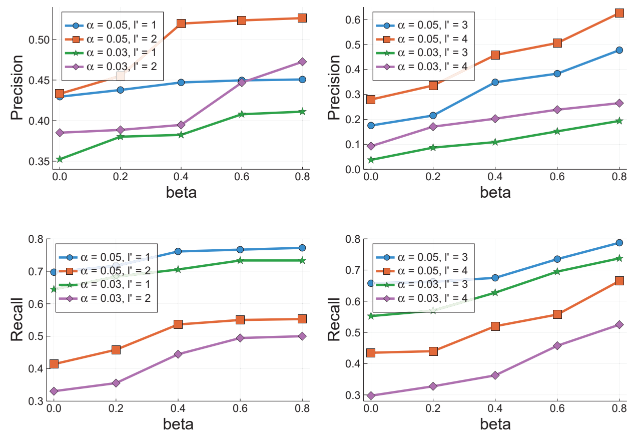
We considered an anomaly detector using the one-class SVM (OCSVM) [9, 54]. It is well-known that the OCSVM has strong detection power for various use cases of outlier detection. In our experiments, the OCSVM was leveraged in supervised learning settings. We trained a model for detection from the training data in the rating matrix $\mathbf{M}$ (described in Section 4.1). Here, we regarded each user's ratings as a n-dimensional feature vector. Since the dimension n of the feature vector was very high, we introduced the principal components analysis (PCA) for dimensionality reduction, and reduced the dimension from n to the number k $(= 30)$ of columns in the item profile $\mathbf{V}$. Given the ratings of a malicious user, the detector checks whether or not the ratings are malicious using the learned detection model.

There are two parameters γ and ν in the OCSVM; γ is the complexity of the model, and ν is an upper bound on the fraction of outliers in training data. We optimized γ so that the variance of the gram matrix was maximized (see [28] for more detail). We determined ν by assuming the Gaussian distribution for feature vectors and applying the 3σ principle. For the Foursquare (resp. MovieLens) dataset, we set the parameters to $\gamma = 0.1$ and $\nu = 0.003$ (resp. $\gamma = 0.02$ and $\nu = 0.003$).

We performed our MI attack with five different values of μ for each dataset. For the Foursquare (resp. MovieLens) dataset, we set μ to $\mu = 1, 5, 10, 50,$ or 100 (resp. $\mu = 0.01, 0.05, 0.1, 0.5,$ or 1). We fixed the poisoning rate to $\alpha = 0.05$. As a metric for the resistance to attack detection, we used the *detection rate*, which is the ratio of the number of detected malicious user accounts divided by the number of all malicious user accounts. As metrics for the attack performance, we used the precision (MIPre) and the recall (MIRec). As with Section 4.2, we performed our MI attack 20 times with different initial values of $\mathbf{U}$, and then averaged the result values for each of the three metrics.

Evaluation Results. Fig. 6 shows the experimental results. For each dataset, the top two graphs show the attack performance (precision and recall), whereas the bottom graph shows the detection rate.

It can be seen that the resistance to attack detection is improved by increasing the value of μ. When $\mu = 0$, all of the malicious user accounts were detected. However, the detection rate was significantly decreased with an increase in μ. For instance, the detection rate decreased by 57.5% at $\mu = 10.0$ in the Foursquare dataset. The detection rate decreased by 28.4% at $\mu = 0.1$ in the MovieLens dataset. It can also be seen that the attack performance (i.e., MIPre and MIRec) decreased with an increase in μ. This means that there is a trade-off between the attack performance and the resistance to attack detection, as described in Section 3.3.



(a) Foursquare dataset. **(b)** MovieLens dataset.

Fig. 7. Precision (MIPrec) and recall (MIRec) of our MI attack using approximate $\mathbf{V}$.

Nevertheless, our MI attack was still effective for the above values of μ; i.e., our MI attack achieved 70.8% precision and 23.3% recall at $\mu = 10.0$ in the Foursquare dataset, and 42.5% precision and 15% recall at $\mu = 0.1$ in the MovieLens dataset. Our attack keeps much higher precision than the baseline attack, even with an increase in μ. Thus we conclude that the adversary can disclose the user's past private behavior, such as the fact that she visited a hospital and the fact that she highly rated an immoral movie, while avoiding attack detection.

4.4 MI Attack Using Approximate V

Experimental Set-up. Here we consider an attack scenario in which the adversary does not have full knowledge of $\mathbf{V}$. As described in Section 3.1, in some real-world services (e.g., Amazon, Google Maps), we can observe all of the public ratings for some users by accessing the users' profile pages. In this case, the adversary can access some rows of $\mathbf{M}$, and approximate $\mathbf{V}$ by factorizing the partial $\mathbf{M}$. We show that our MI attack is effective even when we use such an approximate $\mathbf{V}$.

In our experiments, we assumed that the adversary can observe all the ratings from a part of users. We denote by β a ratio of users whose ratings are available to the adversary. For both datasets, we set $\beta = 0, 0.2, 0.4, 0.6,$ or 0.8, and approximated $\mathbf{V}$ using the partial $\mathbf{M}$. We set $\mu = 0$, the poisoning rate to $\alpha = 0.02$ or 0.05, and a threshold l' to $l' = 1$ or 2 (resp. $l' = 3$ or 4) for the Foursquare (resp. MovieLens) dataset. As with Sections 4.2 and 4.3, we performed our MI attack 20 times, and averaged MIPre and MIRec.

Evaluation Results. Fig. 7 shows the experimental results. It can be seen that both MIPre and MIRec increase with an increase in β. In Algorithm 2, we initialize $\mathbf{M}'$ by setting the ratings of sensitive and decoy items to the highest rating. MIPre and MIRec thereby have some moderate values even if any information on $\mathbf{V}$ does not leak (i.e., $\beta = 0$). However, when a part of ratings leaks, the attack performance is further improved by optimization of $\mathbf{M}'$ with an approximate $\mathbf{V}$. Moreover, our MI attack can also avoid anomaly detection by increasing μ, as shown in Section 4.3. Thus we conclude that our MI attack is effective even when the adversary cannot obtain the complete $\mathbf{V}$.

4.5 Discussions on Defenses

In Section 4.3, we showed that our MI attack can avoid the anomaly detector using OCSVM [9, 54] while keeping high precision. We finally discuss possible defenses against our MI attack other than anomaly detection.

For example, differentially private matrix factorization (DPMF) [34] might be a promising defense against our MI attack. Specifically, Wang *et al.* [62] showed that posterior sampling-based Bayesian learning algorithms, which train parameters by sampling from a posterior distribution, provide differential privacy (DP) [11] for free (without additional noise). Subsequently, Liu *et al.* [34] proposed DPMF that trains $\mathbf{U}$ and $\mathbf{V}$ via posterior sampling, and proved that DPMF provides ϵ-DP. ϵ-DP guarantees that the adversary who obtains $\mathbf{U}$ and $\mathbf{V}$ cannot infer ratings of a particular user (or even whether the user is included in $\mathbf{M}$) with a certain degree of confidence. In particular, when a privacy budget ϵ (≥ 0) is small, the inference of the ratings (i.e., past private behavior) can be strongly protected.

Unfortunately, ϵ can be very large in DPMF, as ϵ increases with increase in the number of ratings per user. For example, [34] reported that ϵ needs to be larger than 250 (which provides almost no privacy guarantee in DP) to achieve high utility. However, DP protects privacy of any user, including a malicious spammer that provides ratings completely opposite from what the model would predict, as discussed in [34]. For normal users whose ratings are not far away from the average ratings, the privacy guarantee might be much more stronger. Thus as future work, we would like to consider a more suitable privacy notion (e.g., plausible deniability [5, 43]) for DPMF, and evaluate the relationship between the privacy notion and the accuracy of our MI attack.

We finally note that another well-known privacy-preserving matrix factorization [46], which is based on garbled circuits, cannot defend against our attack. Specifically, the method in [46] assumes that the recommender system sends $\mathbf{V}$ to a user (or release it publicly). This is a typical scenario where our MI attack works well, as shown in our experiments. Similarly, the privacy-preserving matrix factorization in [40] shares $\mathbf{V}$ to a user, and hence cannot defend against our attack.

5 Conclusion

In this paper, we proposed the first inference attack on factorization-based collaborative filtering systems that can reveal *past* private user behaviors. Our inference attack is based on model inversion techniques, and leverages data poisoning in a novel way. Our experimental results using real-world datasets showed that it is indeed possible to reveal user's past private behaviors in CF-based recommender systems using our attack.

Acknowledgement: The third and fifth authors were supported by JST CREST Grant Number JPMJCR19F6.

References

[1] Chaabane Abdelberi, Gergely Ács, and Mohamed Ali Kâafar. You are what you like! Information leakage through users' interests. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium (NDSS)*, 2012.

[2] Scott Alfeld, Xiaojin Zhu, and Paul Barford. Data poisoning attacks against autoregressive models. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1452–1458, 2016.

[3] Battista Biggio, Giorgio Fumera, and Fabio Roli. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):984–996, 2014.

[4] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012.

[5] Vincent Bindschaedler, Reza Shokri, and Carl A. Gunter. Plausible deniability for privacy-preserving data synthesis. *Proceedings of the VLDB Endowment*, 10(5):481–492, 2017.

[6] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[7] Jian-Feng Cai, Emmanuel J. Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[8] Joseph A Calandrino, Ann Kilzer, Arvind Narayanan, Edward W Felten, and Vitaly Shmatikov. "You Might Also Like:" Privacy risks of collaborative filtering. In *Proceedings of the 32nd IEEE Symposium on Security and Privacy*, pages 231–246, 2011.

[9] Colin Campbell and Kristin P. Bennett. A linear programming approach to novelty detection. In *Proceedings of the 13th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 395–401, 2000.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15:1–15:58, 2009.

[11] Cynthia Dwork and Aaron Roth. *The Algorithmic Foundations of Differential Privacy*. Now Publishers, 2014.

[12] Julian Eggert and Edgar Kömer. Sparse coding and NMF. In *Proceedings of the 2004 IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 2529–2533, 2004.

[13] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. Poisoning attacks to graph-based recommender systems. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC)*, 2018.

[14] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1322–1333, 2015.

[15] Matthew Fredrikson, Eric Lantz, and Somesh Jha. Model inversion attacks that exploit confidence information and basic countermeasures. In *the Proceedings of the 23rd USENIX Security Symposium*, pages 17–32, 2014.

[16] Arik Friedman, Bart P Knijnenburg, Kris Vanhecke, Luc Martens, and Shlomo Berkovsky. Privacy aspects of recommender systems. In *Recommender Systems Handbook*, pages 649–688. Springer, 2015.

[17] Neil-Zhenqiang Gong and Bin Liu. You are who you know and how you behave: Attribute inference attacks via users' social friends and behaviors. In *Proceedings of the 25th USENIX Security Symposium*, pages 979–995, 2016.

[18] Neil Zhenqiang Gong and Bin Liu. Attribute inference attacks in online social networks. *ACM Transactions on Privacy and Security (TOPS)*, 21(1):3, 2018.

[19] GroupLens Research. MovieLens 20M Dataset. http://grouplens.org/datasets/movielens/, 2016.

[20] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Spinger, 2nd edition, 2009.

[21] Seira Hidano, Takao Murakami, Shuichi Katsumata, Shinsaku Kiyomoto, and Goichiro Hanaoka. Model inversion attacks for online prediction systems: Without knowledge of non-sensitive attributes. In *Proceedings of the 15th International Conference on Privacy, Security, and Trust (PST)*, pages 1–10, 2017.

[22] T Ryan Hoens, Marina Blanton, and Nitesh V Chawla. A private and reliable recommendation system for social networks. In *Proceedings of the second IEEE International Conference on Social Computing (SocialCom)*, pages 816–825, 2010.

[23] Kejun Huang, Nicholas D. Sidiropoulos, and Ananthram Swami. Non-negative matrix factorization revisited: Uniqueness and algorithm for symmetric decomposition. *IEEE*

Transactions on Signal Processing, 62(1):211–224, 2014.

[24] Danesh Irani, Steve Webb, Kang Li, and Calton Pu. Large online social footprints–an emerging threat. In *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering (CSE)*, pages 271–276, 2009.

[25] Arjan Jeckmans, Qiang Tang, and Pieter Hartel. Privacy-preserving collaborative filtering based on horizontally partitioned dataset. In *Proceedings of the 2012 International Conference on Collaboration Technologies and Systems (CTS)*, pages 439–446, 2012.

[26] Arjan JP Jeckmans, Michael Beye, Zekeriya Erkin, Pieter Hartel, Reginald L Lagendijk, and Qiang Tang. Privacy in recommender systems. In *Social Media Retrieval*, pages 263–281. Springer, 2013.

[27] Jinyuan Jia, Binghui Wang, Le Zhang, and Neil Zhenqiang Gong. Attriinfer: Inferring user attributes in online social networks using markov random fields. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*, 2017.

[28] Hiromasa Kaneko and Kimito Funatsu. Fast optimization of hyperparameters for support vector regression models with highly predictive ability. *Chemometrics and Intelligent Laboratory Systems*, 142:64–69, 2015.

[29] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer and Information Science*, 42(8):30–37, 2009.

[30] Michal Kosinski, David Stillwell, and Thore Graepel. Private traits and attributes are predictable from digital records of human behavior. *Proceedings of the National Academy of Sciences*, 110(15):5802–5805, 2013.

[31] Shyong K Lam and John Riedl. Shilling recommender systems for fun and profit. In *Proceedings of the 13th International Conference on World Wide Web (WWW)*, pages 393–402, 2004.

[32] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. In *Proceedings of the 30th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1885–1893, 2016.

[33] Yiding Liu, T. Nguyen Pham, Gao Cong, and Quan Yuan. An experimental evaluation of point-of-interest recommendation in location-based social networksk. *Proceedings of the VLDB Endowment*, 10(10):1010–1021, 2017.

[34] Ziqi Liu, Yu-Xiang Wang, and Alexander J. Smola. Fast differentially private matrix factorization. In *Proceedings of the 9th ACM Conference on Recommender Systems (RecSys)*, pages 171–178, 2015.

[35] Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. Sorec: Social recommendation using probabilistic matrix factorization. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM)*, pages 931–940, 2008.

[36] Julien Mairal, Francis Bach, and Jean Ponce. *Sparse Modeling for Image and Vision Processing*. Now Publishers, 2014.

[37] Jonathan R Mayer and John C Mitchell. Third-party web tracking: Policy and technology. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, pages 413–427, 2012.

[38] Frank McSherry and Ilya Mironov. Differentially private recommender systems: Building privacy into the Netflix prize contenders. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Dining (KDD)*, pages 627–636, 2009.

[39] Shike Mei and Xiaojin Zhu. The security of latent dirichlet allocation. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 681–689, 2015.

[40] Xuying Meng, Suhang Wang, Kai Shu, Jundong Li, Bo Chen, Huan Liu, and Yujun Zhang. Personalized privacy-preserving social recommendation. In *Proceedings of 32nd AAAI Conference on Artificial Intelligence (AAAI)*, pages 1–8, 2018.

[41] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology (TOIT)*, 7(4):23, 2007.

[42] Luis Muñoz González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec)*, pages 27–38, 2017.

[43] Takao Murakami, Koki Hamada, Yusuke Kawamoto, and Takuma Hatano. Privacy-preserving multiple tensor factorization for synthesizing large-scale location traces. *CoRR*, abs/1911.04226, 2019.

[44] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Proceedings of the 29th IEEE Symposium on Security and Privacy*, pages 111–125, 2008.

[45] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, pages 173–187, 2009.

[46] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. Privacy-preserving matrix factorization. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS)*, pages 801–812, 2013.

[47] Michael O'Mahony, Neil Hurley, Nicholas Kushmerick, and Guénolé Silvestre. Collaborative recommendation: A robustness analysis. *ACM Transactions on Internet Technology (TOIT)*, 4(4):344–377, 2004.

[48] Jahna Otterbacher. Inferring gender of movie reviewers: Exploiting writing style, content and metadata. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 369–378, 2010.

[49] Huseyin Polat and Wenliang Du. Privacy-preserving collaborative filtering using randomized perturbation techniques. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM)*, pages 625–628, 2003.

[50] Huseyin Polat and Wenliang Du. Achieving private recommendations using randomized response techniques. In *Proceedings of the 10th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD)*, pages 637–646, 2006.

[51] Naren Ramakrishnan, Benjamin J Keller, Batul J Mirza, Ananth Y Grama, and George Karypis. Privacy risks in recommender systems. *IEEE Internet Computing*, 5(6):54, 2001.

[52] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[53] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web (WWW)*, pages 285–295, 2001.

[54] Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. In *Proceedings of the 12th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 582–588, 1999.

[55] Guy Shani, Max Chickering, and Christopher Meek. Mining recommendations from the web. In *Proceedings of the 2008 ACM Conference on Recommender Systems (RecSys)*, pages 35–42, 2008.

[56] Kai Shu, Suhang Wang, Jiliang Tang, Yilin Wang, and Huan Liu. Crossfire: Cross media joint friend and item recommendations. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining (WSDM)*, 2018.

[57] Nathan Srebro, Jason Rennie, and Tommi S. Jaakkola. Maximum-margin matrix factorization. In *Proceedings of the 18th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1329–1336, 2004.

[58] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009(4):1–19, 2009.

[59] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Investigation of various matrix factorization methods for large recommender systems. In *Proceedings of the 17th IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 553–562, 2008.

[60] Jiliang Tang, Xia Hu, and Huan Liu. Social recommendation: A review. *Social Network Analysis and Mining*, 3(4):1113–1133, 2013.

[61] Qiang Tang and Jun Wang. Privacy-preserving friendship-based recommender systems. *IEEE Transactions on Dependable and Secure Computing*, 15:784–796, 2016.

[62] Yu-Xiang Wang, Stephen E. Fienberg, and Alexander J. Smola. Privacy for free: Posterior sampling and stochastic gradient monte carlo. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML)*, pages 2493–2502, 2015.

[63] Udi Weinsberg, Smriti Bhagat, Stratis Ioannidis, and Nina Taft. Blurme: Inferring and obfuscating user gender based on ratings. In *Proceedings of the 6th ACM conference on Recommender systems (RecSys)*, pages 195–202, 2012.

[64] Xi Wu, Matthew Fredrikson Somesh Jha, and Jeffrey F. Naughton. A methodology for formalizing model-inversion attacks. In *Proceedings of the 29th IEEE Computer Security Foundations Symposium (CSF)*, pages 355–370, 2016.

[65] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. Is feature selection secure against training data poisoning? In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 1689–1698, 2015.

[66] Xinyu Xing, Wei Meng, Dan Doozan, Alex C Snoeren, Nick Feamster, and Wenke Lee. Take this personally: Pollution attacks on personalized services. In *Proceedings of the 22nd USENIX Security Symposium*, pages 671–686, 2013.

[67] Dingqi Yang, Daqing Zhang, and Bingqing Qu. Participatory cultural mapping based on collective behavior data in location based social networks. *ACM Transction on Intelligent Systems and Technology (TIST)*, 7(30):1–23, 2016.

[68] Guolei Yang, Neil Zhenqiang Gong, and Ying Cai. Fake co-visitation injection attacks to recommender systems. In *Proceedings of 24th Network and Distributed System Security Symposium (NDSS)*, 2017.

[69] William Zeller and Edward W Felten. Cross-site request forgeries: Exploitation and prevention. *Technical Report, Princeton University*, 2008.

[70] Elena Zheleva and Lise Getoor. To join or not to join: The illusion of privacy in social networks with mixed public and private user profiles. In *Proceedings of the 18th International Conference on World Wide Web (WWW)*, pages 531–540, 2009.

A Notations

Table 1 shows the notations used throughout this paper. The notations are separated into three groups by horizontal lines. The notations in the second and third groups are related to factorization-based CF and data poisoning attacks, respectively, which we introduce in the following Sections 2.1 and 2.4.

B Performance Metrics for Recommender Systems

In our experiments, we considered three performance metrics: the root mean square error (RMSE), precision and recall in the top-l recommendation (RSPre@l and RSRecl@l).

The RMSE is defined as follows:

$$RMSE = \frac{1}{|\Omega^c|}\sqrt{\sum_{(i,j)\in\Omega^c}(\widehat{\mathbf{M}}_{i,j} - \mathbf{M}^c_{i,j})^2}, \tag{17}$$

where $\mathbf{M}^c_{i,j}$ is (i,j)-th element in the rating matrix $\mathbf{M}^c$ created from the testing data, and Ω^c is the index set of observed elements in $\mathbf{M}^c$.

RSPre@l and RSRecl@l are defined as follows:

$$RSPre@l = \frac{1}{m}\sum_i \frac{|\mathcal{S}_i \bigcap \mathcal{T}_i|}{l} \tag{18}$$

$$RSRec@l = \frac{1}{m}\sum_i \frac{|\mathcal{S}_i \bigcap \mathcal{T}_i|}{|\mathcal{T}_i|}, \tag{19}$$

where $\mathcal{S}_i$ is the set of items recommended to the i-th user, and $\mathcal{T}_i$ is the set of items the i-th user has rated

Table 1. Notations used throughout this paper. The notations in the first, second, and third groups are introduced in "Preliminaries" and "Prediction Algorithm" in Section 2.1 and 2.4, respectively.

Symbol	Description
$\mathbb{N}$	**Set of natural numbers.**
$\mathbb{R}$	**Set of real numbers.**
m	**Number of users.**
n	**Number of items.**
$\mathbf{M}$	**Rating matrix.**
$\mathbf{\Omega}$	**$= \{(i,j) : \mathrm{M}_{i,j}$ is observed$\}$.**
$\mathcal{R}_{\Omega}$	**Masking function using Ω (see (1)).**
$\mathbf{U}$	**User profile.**
$\mathbf{V}$	**Item profile.**
k	**Number of columns/rows in the user/item profile.**
$\mathbf{\Theta}$	**Model ($= \{\mathrm{U}, \mathrm{V}\}$).**
$\widehat{\mathbf{M}}$	**Prediction of M including unobserved elements.**
m'	**Number of malicious users.**
$\mathbf{M}'$	**Rating matrix for m' malicious users.**
$\mathbf{\Omega}'$	**$= \{(i,j) : \mathrm{M}'_{i,j}$ is observed$\}$.**
$\mathbf{U}^*$	**Poisoned user profile for m users.**
$\mathbf{U}'$	**Poisoned user profile for m' malicious users.**
$\mathbf{V}^*$	**Poisoned item profile.**
$\mathbf{\Theta}^*$	**Poisoned model ($= \{\mathrm{U}^*, \mathrm{U}', \mathrm{V}^*\}$).**
$\widehat{\mathbf{M}}^*$	**Prediction of M after data poisoning.**
$\widehat{\mathbf{M}}'$	**Prediction of M' after data poisoning.**

in the testing data (in the Foursquare dataset, $\mathcal{T}_i$ is the set of venues the user has visited one or more times in the testing data).

C More Details on Exploitation and Privacy Risks for Recommender Systems

In this section, we provide more details on related works that were omitted from Section 2.2 regarding exploitation and privacy risks for recommender systems.

C.1 Exploitation

We review attacks which maliciously modify the recommender system in a way so that an adversary can control how items will be recommended to users.

The most well-studied exploitation is *poisoning* attacks (also known as *shilling* attacks) [13, 31, 32, 41, 47, 68]. The aim of poisoning attacks is to inject malicious users into the recommender system in such a way that an adversary can control the output behavior of the system. Concretely, an adversary creates new accounts for malicious users and strategically rates items so that when the recommender system once updates its model Θ (See Section 2.1) the recommended item output by the system would behave according to the adversary's intent. Here, an adversary's intent may be to maximize the error of the collaborative filtering system or to boost/reduce the popularity of particular items. In the early days of poisoning attacks, the attacks were heuristic and were not optimized to a particular type of recommender system [31, 41, 47]. For example, the original poisoning attack of Lam et al. [31] is agnostic to the type of recommender system and the two proposed attacks (*RandomBot* and *AverageBot*) are handcrafted based on intuition. However, recently, more sophisticated poisoning attacks for specific types of recommender systems have emerged [13, 32, 68]. In these attacks, the behavior of maliciously injected users are decided via solving a (non-convex) optimization problem. Thus far, poisoning attacks on factorization-based [32], association-rule-based [68], and graph-based [13] recommender systems have been proposed.

Another type of exploitation known is the *profile pollution* attack proposed by Xing et al. [66]. Their idea is to inject fake information to the user's profiles such as web search history via cross-site request forgery attacks (CSRF) [69]. Informally, if an adversary was able to pollute the user profile, she can make the system recommend arbitrary item to the user. However, one of the limitations of profile pollution attacks is that they rely on CSRF, which is typically a difficult attack to conduct over a large scaled-system.

C.2 Privacy Risks

We review the two types of inference attacks: attacks against *user attributes* and *user behaviors*.

The former inference attack on user attributes tries to infer a user's sensitive attributes such as gender, political view, and sexual orientation, based on its rating behavior. Since rating behaviors are often times statistically correlated with user's attributes, such inference attack has shown to be feasible in numerous prior works, e.g., [1, 17, 18, 27, 30, 48, 63, 70]. A prominent example of a real-world attribute inference attack is the recent Facebook-Cambridge Analytica data scandal; Cambridge Analytica harvested Facebook user's rating behavior such as page likes to infer their attributes and used it for political purposes.

The latter inference attack on user behaviors, which is the main target of this paper, tries to infer a user's past private behavior over the recommender system such as purchase or rating histories of sensitive items, e.g., medicinal items, sexual movies, and books expressing political views. So far, there have been two works [8, 51] studying such attacks on recommender systems. The key observation of the initial work of Ramakrishnan et al. [51] was that the model Θ used in a recommender system is generated by *all* the user's information in the system. For example, if a user who rated item A got recommended an item B, then the user would know that there exists another user who have rated both items A and B. Building on this observation, they provided a theoretical attack and provided experimental results on a small scaled recommender system. Calandrino *et al.* [8] showed a much stronger and practical attack on item-to-item recommender systems based on collaborative filtering by further exploiting the knowledge of the item profile and user's public behavior. Here, public behavior points to the non-sensitive items that were rated by the users or recommended to the users, which the users do not intend to keep private. Specifically, in their proposed attack, an adversary keeps track of the covariance of items known to be associated with the target user and infers that the target item was rated by the user once all the tracked covariance increased, which can be done by keeping track of the item profile. In other words, an adversary collects a moderate amount of a user's public behavior before some time T and infers the user's private behavior *after* time T. Since their attack works by tracking the change in the item profile after time T (the time on which the adversary decides to attack the system), it cannot be used to infer user's private behavior before time T.

D More Details on MI Attacks

In this section, we provide more details on MI attacks. As we have mentioned in the main body, MI attacks were proposed by Fredrikson et al. [15]. By now, there are several models for MI attacks. We say it is *black-box* if the adversary only gets oracle access to the model f, *grey-box* if the structure of the model f and some auxiliary information is further known, e.g., the adversary knows the fact that f is a linear regression model and some (not all) regression coefficients, and *white-box* if it gets the full description of f, e.g., the adversary knows all of the regression coefficients. Then, the goal of the adversary is to predict the sensitive portion (or all) of the features $\mathbf{x}$ of the target user. As one can see, this is closely related to inference attack on user behaviors on recommender systems by viewing the feature $\mathbf{x}$ as the user's rating history. For example, the nominal work of Fredrikson et al. [15] proposed an MI attack for a linear regression model that predicted the amount of dosage for the drug Warfarin, where the feature vector consisted of the patient's demographic information, medical history, and genetic markers. In their work, genetic markers were considered to be the sensitive information. Soon after, Fredrikson et al. [14] extended the MI attack to other prediction models such as decision trees and specific types of neural networks. They exploited the fact that MLaaS systems such BigML and Microsoft Azure Learning provides confidence values of their predictions. In their attack, the extra information of confidence values are modeled as part of the auxiliary information stated above. Wu et al. [64] gave a game-based definition of MI attacks to formally treat privacy, analogous to the methodologies used in cryptography. All of the above works mainly focus on non-dynamic prediction models and the attack algorithms are based on statistical tools (i.e., maximum a posterior estimators).

Recently, Hidano et al. [21] proposed a new type of MI attacks which targets *dynamic* prediction systems. They observed that in a dynamic system, similarly to poisoning attacks [2–4, 32, 39, 42, 65], an adversary can modify the system's model Θ by injecting malicious data. By leveraging this extra dynamic feature, [21] improved the MI attack on linear regression models proposed by [15]. Namely, in [15], the adversary was assumed to have knowledge of all the non-sensitive portion of the target user's features $\mathbf{x}$ as part of the auxiliary information, however, [21] circumvent this knowledge assumption by providing the adversary capability of modifying the model Θ.