Influence Function based Data Poisoning Attacks to Top-N Recommender Systems

Minghong Fang
Iowa State University
myfang@iastate.edu

Neil Zhenqiang Gong
Duke University
neil.gong@duke.edu

Jia Liu
Iowa State University
jialiu@iastate.edu

ABSTRACT

Recommender system is an essential component of web services to engage users. Popular recommender systems model user preferences and item properties using a large amount of crowdsourced user-item interaction data, e.g., rating scores; then top-N items that match the best with a user's preference are recommended to the user. In this work, we show that an attacker can launch a *data poisoning attack* to a recommender system to make recommendations as the attacker desires via injecting fake users with carefully crafted user-item interaction data. Specifically, an attacker can trick a recommender system to recommend a target item to as many normal users as possible. We focus on matrix factorization based recommender systems because they have been widely deployed in industry. Given the number of fake users the attacker can inject, we formulate the crafting of rating scores for the fake users as an optimization problem. However, this optimization problem is challenging to solve as it is a non-convex integer programming problem. To address the challenge, we develop several techniques to approximately solve the optimization problem. For instance, we leverage *influence function* to select a subset of normal users who are influential to the recommendations and solve our formulated optimization problem based on these influential users. Our results show that our attacks are effective and outperform existing methods.

CCS CONCEPTS

• **Security and privacy** → **Web application security**.

KEYWORDS

Adversarial recommender systems, data poisoning attacks, adversarial machine learning.

1 INTRODUCTION

Recommender system is a key component of many web services to help users locate items they are interested in. Many recommender systems are based on collaborative filtering. For instance, given a large amount of user-item interaction data (we consider rating scores in this work) provided by users, a recommender system learns to model latent users' preferences and items' features, and then the system recommends top-N items to each user, where the features of the top-N items best match with the user's preference.

As a recommender system is driven by user-item interaction data, an attacker can manipulate a recommender system via injecting fake users with fake user-item interaction data to the system. Such attacks are known as *data poisoning attacks* [9, 10, 17, 19, 23, 35, 39]. Several recent studies designed recommender-system-specific data poisoning attacks to association-rule-based [39], graph-based [10] and matrix-factorization-based recommender systems [19]. However, how to design customized attacks to matrix-factorization-based top-N recommender systems remains an open question even though such recommender systems have been widely deployed in the industry. In this work, we aim to bridge the gap. In particular, we aim to design an optimized data poisoning attack to *matrix-factorization-based top-N recommender systems*. Suppose that an attacker can inject m fake users into the recommender system and each fake user can rate at most n items, which we call *filler items*. Then, the key question is: *how to select the filler items and assign rating scores to them such that an attacker-chosen target item is recommended to as many normal users as possible?* To answer this question, we formulate an optimization problem for selecting filler items and assigning rating scores for the fake users, with an objective to maximize the number of normal users to whom the target item is recommended.

However, it is challenging to solve this optimization problem because it is a non-convex integer programming problem. To address the challenge, we propose a series of techniques to approximately solve the optimization problem. First, we propose to use a loss function to approximate the number of normal users to whom the target item is recommended. We relax the integer rating scores to continuous variables and convert them back to integer rating scores after solving the reformulated optimization problem. Second, to enhance the effectiveness of our attack, we leverage the *influence function approach* inspired by the interpretable machine learning literature [14, 15, 34] to account for the reality that the top-N recommendations may be only affected by a subset $\mathcal{S}$ of influential users. For convenience, throughout the rest of this paper, we refer to our attack as $\mathcal{S}$-attack. We show that the influential user selection subproblem enjoys the *submodular* property, which guarantees a $(1 - 1/e)$ approximation ratio with a simple greedy selection algorithm. Lastly, given $\mathcal{S}$, we develop a gradient-based optimization algorithm to determine rating scores for the fake users.

We evaluate our $\mathcal{S}$-attack and compare it with multiple baseline attacks on two benchmark datasets, including Yelp and Amazon Digital Music (Music). Our results show that our attacks can effectively promote a target item. For instance, on the Yelp dataset,

when injecting only 0.5% of fake users, our attack can make a randomly selected target item appear in the top-N recommendation lists of 150 times more normal users. Our $\mathcal{S}$-attack outperforms the baseline attacks and continues to be effective even if the attacker does not know the parameters of the target recommender system. We also investigate the effects of our attacks on recommender systems that are equipped with fake users detection capabilities. For this purpose, we train a binary classifier to distinguish between fake users and normal ones. Our results show that this classifier is effective against traditional attack schemes, e.g., PGA attack [19], etc. Remarkably, we find that our influence-function-based attack continues to be effective. The reason is that our proposed attack is designed with stealth in mind, and the detection method can detect some fake users but miss a large fraction of them.

Finally, we show that our influence function based approach can also be used to enhance data poisoning attacks to graph-based top-N recommender systems. Moreover, we show that instead of using influence function to select a subset of influential users, using influence function to weight each normal user can further improve the effectiveness of data poisoning attacks, though such approach sacrifices computational efficiency.

In summary, our contributions are as follows:

- We propose the first data poisoning attack to matrix-factorization-based Top-N recommender systems, which we formulate as a non-convex integer optimization problem.
- We propose a series of techniques to approximately solve the optimization problem with provable performance guarantee.
- We evaluate our $\mathcal{S}$-attack and compare it with state-of-the-art using two benchmark datasets. Our results show that our attack is effective and outperforms existing ones.

2 RELATED WORK

Data poisoning attacks to recommender systems: The security and privacy issues in machine learning models have been studied in many scenarios [24, 30–32, 40, 42, 43]. The importance of data poisoning attacks has also been recognized in recommender systems [7, 21–23, 29, 38]. Earlier work on poisoning attacks against recommender systems are mostly agnostic to recommender systems and do not achieve satisfactory attack performance, e.g., random attack [17] and average attack [17]. Recently, there is a line of work focusing on attacking specific types of recommender systems [10, 19, 39]. For example, Fang et al. [10] proposed efficient poisoning attacks to graph-based recommender systems. They injected fake users with carefully crafted rating scores to the recommender systems in order to promote a target item. They modeled the attack as an optimization problem to decide the rating scores for the fake users. Li et al. [19] proposed poisoning attacks to matrix-factorization-based recommender systems. Instead of attacking the top-N recommendation lists, their goal was to manipulate the predictions for all missing entries of the rating matrix. As a result, the effectiveness of their attacks is unsatisfactory in matrix-factorization-based top-N recommender systems.

Data poisoning attacks to other systems: Data poisoning attacks generally refer to attacks that manipulate the training data of a machine learning or data mining system such that the learnt

model makes predictions as an attacker desires. Other than recommender systems, data poisoning attacks were also studied for other systems. For instance, existing studies have demonstrated effective data poisoning attacks can be launched to anomaly detectors [28], spam filters [25], SVMs [4, 37], regression methods [12, 36], graph-based methods [33, 44], neural networks [5, 11, 20], and federated learning [9], which significantly affect their performance.

3 PROBLEM FORMULATION

3.1 Matrix-Factorization-Based Recommender Systems: A Primer

A matrix-factorization-based recommender system [16] maps users and items into latent factor vectors. Let $\mathcal{U}$, $\mathcal{I}$ and $\mathcal{E}$ denote the user, item and rating sets, respectively. We also let $|\mathcal{U}|$, $|\mathcal{I}|$ and $|\mathcal{E}|$ denote the numbers of users, items and ratings, respectively. Let $R \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{I}|}$ represent the user-item rating matrix, where each entry r_{ui} denotes the score that user u rates the item i. Let $\boldsymbol{x}_u \in \mathbb{R}^d$ and $\boldsymbol{y}_i \in \mathbb{R}^d$ denote the latent factor vector for user u and item i, respectively, where d is the dimension of latent factor vector. For convenience, we use matrices $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{|\mathcal{U}|}]$ and $Y = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{|\mathcal{I}|}]$ to group all $\boldsymbol{x}$- and $\boldsymbol{y}$-vectors. In matrix-factorization-based recommender systems, we aim to learn X and Y via solving the following optimization problem:

$$\underset{X, Y}{\arg\min} \sum_{(u,i) \in \mathcal{E}} \left(r_{ui} - \boldsymbol{x}_u^\top \boldsymbol{y}_i\right)^2 + \lambda \left(\sum_u \|\boldsymbol{x}_u\|_2^2 + \sum_i \|\boldsymbol{y}_i\|_2^2\right), \quad (1)$$

where $\|\cdot\|_2$ is the ℓ_2 norm and λ is the regularization parameter. Then, the rating score that a user u gives to an unseen item i is predicted as $\hat{r}_{ui} = \boldsymbol{x}_u^\top \boldsymbol{y}_i$, where $\boldsymbol{x}_u^\top$ denotes the transpose of vector $\boldsymbol{x}_u$. Lastly, the N unseen items with the highest predicted rating scores are recommended to each user.

3.2 Threat Model

Given a target item t, the goal of the attacker is to promote item t to as many normal users as possible and maximize the hit ratio $h(t)$, which is defined as the fraction of normal users whose top-N recommendation lists include the target item t. We assume that the attacker is able to inject some fake users into the recommender system, each fake user will rate the target item t with high rating score and give carefully crafted rating scores to other well-selected items. The attacker may have full knowledge of the target recommender system (e.g., all the rating data, the recommendation algorithm). The attacker may also only have partial knowledge of the target recommender system, e.g., the attacker only has access to some ratings. We will show that our attacks are still effective when the attacker has partial knowledge of the target recommender system.

3.3 Attack Strategy

We assume that the rating scores of the target recommender system are integer-valued and can only be selected from the set $\{0, 1, \cdots, r_{max}\}$, where r_{max} is the maximum rating score. We assume that the attacker can inject m fake users into the recommender system. We denote by $\mathcal{M}$ the set of m fake users. Each fake user will rate the target item t and at most n other carefully selected items (called *filler items*). We consider each fake user rates at most n filler items

to avoid being easily detected. We let r_v and Ω_v denote the rating score vector of fake user v and the set of items rated by v, respectively, where $v \in \mathcal{M}$ and $|\Omega_v| \le n + 1$. Then, r_{vi} is the score that user v rates the item i, $i \in \Omega_v$. Clearly, Ω_v satisfies $|\Omega_v| = \|r_v\|_0$, where $\|\cdot\|_0$ is the ℓ_0 norm (i.e., the number of non-zero entries in a vector). The attacker's goal is to find an optimal rating score vector for each fake user v to maximize the hit ratio $h(t)$. We formulate this hit ratio maximization problem (HRM) as follows:

$$\text{HRM: max } h(t) \tag{2}$$

$$\text{s.t. } |\Omega_v| \le n + 1, \qquad \forall v \in \mathcal{M}, \tag{3}$$

$$r_{vi} \in \{0, 1, \cdots, r_{max}\}, \quad \forall v \in \mathcal{M}, \forall i \in \Omega_v. \tag{4}$$

Problem HRM is an integer programming problem and is NP-hard in general. Thus, finding an optimal solution is challenging. In the next section, we will propose techniques to approximately solve the problem.

4 OUR SOLUTION

We optimize the rating scores for fake users one by one instead of optimizing for all the m fake users simultaneously. In particular, we repeatedly optimize the rating scores of one fake user and add the fake user to the recommender system until we have m fake users. However, it is still challenging to solve the HRM problem even if we consider only one fake user. To address the challenge, we design several techniques to approximately solve the HRM problem for one fake user. First, we relax the discrete ratings to continuous data and convert them back to discrete ratings after solving the problem. Second, we use a differentiable loss function to approximate the hit ratio. Third, instead of using all normal users, we use a selected subset of influential users to solve the HRM problem, which makes our attack more effective. Fourth, we develop a gradient-based method to solve the HRM problem to determine the rating scores for the fake user.

4.1 Relaxing Rating Scores

We let vector $w_v = [w_{vi}, i \in \Omega_v]^\top$ be the relaxed continuous rating score vector of fake user v, where w_{vi} is the rating score that user v gives to the item i. Since $r_{vi} \in \{0, 1, \cdots, r_{max}\}$ is discrete, which makes it difficult to solve the optimization problem defined in (2), we relax the discrete rating score r_{vi} to continuous variables w_{vi} that satisfy $w_{vi} \in [0, r_{max}]$. Then, we can use gradient-based methods to compute w_v. After we solve the optimization problem, we convert each w_{vi} back to a discrete integer value in the set $\{0, 1, \cdots, r_{max}\}$.

4.2 Approximating the Hit Ratio

We let Γ_u be the set of top-N recommended items for a user u, i.e., Γ_u consists of the N items that u has not rated before and have the largest predicted rating scores. To approximate the optimization problem defined in (2), we define a loss function that is subject to the following rules: 1) for each item $i \in \Gamma_u$, if $\hat{r}_{ui} < \hat{r}_{ut}$, then the loss is small, where $\hat{r}_{ui}$ and $\hat{r}_{ut}$ are the predicted rating scores that user u gives to item i and target item t, respectively; 2) the higher target item t ranks in Γ_u, the smaller the loss. Based on these rules,

we reformulate the HRM problem as the following problem:

$$\min_{w_v} \mathcal{L}_{\mathcal{U}}(w_v) = \sum_{u \in \mathcal{U}} \sum_{i \in \Gamma_u} g(\hat{r}_{ui} - \hat{r}_{ut}) + \eta \|w_v\|_1$$
$$\text{s.t. } w_{vi} \in [0, r_{max}], \tag{5}$$

where $g(x) = \frac{1}{1 + \exp(-x/b)}$ is the Wilcoxon-Mann-Whitney loss function [2], b is the *width* parameter, η is the regularization parameter, and $\|\cdot\|_1$ is the ℓ_1 norm. Note that $g(\cdot)$ guarantees that $\mathcal{L}_{\mathcal{U}}(w_v) \ge 0$ and is differentiable. The ℓ_1 regularizer $\|w_v\|_1$ aims to model the constraint that each fake user rates at most n filler items. In particular, the ℓ_1 regularizer makes a fake user's ratings small to many items and we can select the n items with the largest ratings as the filler items.

4.3 Determining the Set of Influential Users

It has been observed in [18, 34] that different training samples have different contributions to the solution quality of an optimization problem, and the performance of the model training could be improved if we drop some training samples with low contributions. Motivated by this observation, instead of optimizing the ratings of a fake user over all normal users, we solve the problem in (5) using a subset of *influential users*, who are the most responsible for the prediction of the target item before attack. We let $\mathcal{S} \in \mathcal{U}$ represent the set of influential users for the target item t. For convenience, in what follows, we refer to our attack as $\mathcal{S}$-attack. Under the $\mathcal{S}$-attack, we further reformulate (5) as the following problem:

$$\min_{w_v} \mathcal{L}_{\mathcal{S}}(w_v) = \sum_{u \in \mathcal{S}} \sum_{i \in \Gamma_u} g(\hat{r}_{ui} - \hat{r}_{ut}) + \eta \|w_v\|_1$$
$$\text{s.t. } w_{vi} \in [0, r_{max}]. \tag{6}$$

Next, we propose an influence function approach to determine $\mathcal{S}$ and then solve the optimization problem defined in (6). We let $F(\mathcal{S}, t)$ denote the influence of removing all users in the set $\mathcal{S}$ on the prediction at the target item t, where influence here is defined as the change of the predicted rating score. We want to find a set of influential users that have the largest influence on the target item t. Formally, the influence maximization problem can be defined as:

$$\max \ F(\mathcal{S}, t), \quad \text{s.t. } |\mathcal{S}| = \Delta, \tag{7}$$

where Δ is the desired set size (i.e., the number of users in set $\mathcal{S}$). However, it can be shown that the problem is NP-hard [13]. In order to solve the above influence maximization problem of (7), we first show how to measure the influence of one user, then we show how to approximately find a set of Δ users with the maximum influence.

We define $\pi(k, t)$ as the influence of removing user k on the prediction at the target item t:

$$\pi(k, t) \stackrel{\text{def}}{=} \sum\nolimits_{j \in \Omega_k} \varphi((k, j), t), \tag{8}$$

where $\varphi((k, j), t)$ is the influence of removing edge (k, j) in the user-item bipartite on the prediction at the target item t, Ω_k is the set of items rated by user k. Then, the influence of removing user set $\mathcal{S}$ on the prediction at the target item t can be defined as:

$$F(\mathcal{S}, t) \stackrel{\text{def}}{=} \sum\nolimits_{k \in \mathcal{S}} \pi(k, t). \tag{9}$$

Since the influence of user and user set can be computed based on the edge influence $\varphi((k, j), t)$, the key challenge boils down to

how to evaluate $\varphi((k, j), t)$ efficiently. Next, we will propose an appropriate influence function to efficiently compute $\varphi((k, j), t)$.

4.3.1 Influence Function for Matrix-factorization-based Recommender Systems.
For a given matrix-factorization-based recommender system, we can rewrite (1) as follows:

$$\theta^* = \arg\min_{\theta} \frac{1}{|\mathcal{E}|} \sum_{(u,i) \in \mathcal{E}} \ell((u, i), \theta), \qquad (10)$$

where $\theta \triangleq (X, Y)$. We let $\hat{r}_{ui}(\theta)$ denote the predicted rating score user u gives to item i under parameter θ, and $\hat{r}_{ui}(\theta) \triangleq \mathbf{x}_u^\top(\theta)\mathbf{y}_i(\theta)$.

If we increase the weight of the edge $(k, j) \in \mathcal{E}$ by some ζ, then the perturbed optimal parameter $\theta^*_{\zeta,(k,j)}$ can be written as:

$$\theta^*_{\zeta,(k,j)} = \arg\min_{\theta} \frac{1}{|\mathcal{E}|} \sum_{(u,i) \in \mathcal{E}} \ell((u, i), \theta) + \zeta \ell((k, j), \theta). \qquad (11)$$

Since removing the edge (k, j) is equivalent to increasing its weight by $\zeta = -\frac{1}{|\mathcal{E}|}$, the influence of removing edge (k, j) on the prediction at edge (o, t) can be approximated as follows [8, 15]:

$$\Phi((k, j), (o, t)) \stackrel{\text{def}}{=} \hat{r}_{ot}\left(\theta^*_{\varepsilon\setminus(k,j)}\right) - \hat{r}_{ot}(\theta^*) \approx -\frac{1}{|\mathcal{E}|} \cdot \left.\frac{\partial \hat{r}_{ot}\left(\theta^*_{\zeta,(k,j)}\right)}{\partial \zeta}\right|_{\zeta=0}$$

$$= \frac{1}{|\mathcal{E}|} \nabla_\theta \hat{r}_{ot}^\top(\theta^*) H_{\theta^*}^{-1} \nabla_\theta \ell((k, j), \theta^*), \qquad (12)$$

where $\theta^*_{\varepsilon\setminus(k,j)}$ is the optimal model parameter after removing edge (k, j) and H_{θ^*} represents the Hessian matrix of the objective function defined in (10). Therefore, the influence of removing edge (k, j) on the prediction at the target item t can be computed as:

$$\varphi((k, j), t) \stackrel{\text{def}}{=} \sum_{o \in \mathcal{U}} |\Phi((k, j), (o, t))|, \qquad (13)$$

where $|\cdot|$ is the absolute value.

4.3.2 Approximation Algorithm for Determining $\mathcal{S}$.
Due to the combinatorial complexity, solving the optimization problem defined in (7) remains an NP-hard problem. Fortunately, based on the observation that the influence of set $\mathcal{S}$ (e.g., $F(\mathcal{S}, t)$) exhibits a diminishing returns property, we propose a greedy selection algorithm to find a solution to (7) with an approximation ratio guarantee. The approximation algorithm is a direct consequence of the following result, which says that the influence $F(\mathcal{S}, t)$ is monotone and submodular.

Theorem 1 (Submodularity). *The influence $F(\mathcal{S}, t)$ is normalized, monotonically non-decreasing and submodular.*

Proof. Define three sets $\mathcal{A}$, $\mathcal{B}$ and C, where $\mathcal{A} \subseteq \mathcal{B}$ and $C = \mathcal{B} \setminus \mathcal{A}$. To simplify the notation, we use $F(\mathcal{A})$ to denote $F(\mathcal{A}, t)$. It is clear that the influence function is normalized since $F(\emptyset) = 0$. When there is no ambiguity, we let $F(u)$ denote $F(\{u\})$ for $u \in \mathcal{U}$. Since $F(\mathcal{B}) - F(\mathcal{A}) = \sum_{u \in \mathcal{B}} F(u) - \sum_{u \in \mathcal{A}} F(u) = \sum_{u \in \mathcal{B} \setminus \mathcal{A}} F(u) = F(C) \geq 0$, which implies that the influence $F(\mathcal{S}, t)$ is monotonically non-decreasing. To show the submodular property, we let $\overline{\mathcal{S}}$ denote the complement of a set $\mathcal{S}$. Now, consider an arbitrary set $\mathcal{D}$, for which we have: $F(\mathcal{B} \cup \mathcal{D}) - F(\mathcal{A} \cup \mathcal{D}) = F((\mathcal{B} \cup \mathcal{D}) \setminus (\mathcal{A} \cup \mathcal{D})) \stackrel{(a)}{=} F(C \setminus (C \cap \mathcal{D})) \leq F(C) = F(\mathcal{B}) - F(\mathcal{A})$, where (a) follows from $(\mathcal{B} \cup \mathcal{D}) \setminus (\mathcal{A} \cup \mathcal{D}) = (\mathcal{B} \cup \mathcal{D}) \cap \overline{(\mathcal{A} \cup \mathcal{D})} = (\mathcal{B} \cup \mathcal{D}) \cap (\overline{\mathcal{A}} \cap \overline{\mathcal{D}}) =$

Algorithm 1 *Greedy Influential User Selection.*

Input: Rating matrix R, budget Δ.
Output: Influential user set $\mathcal{S}$.
1: Initialize $\mathcal{S} = \emptyset$.
2: **while** $|\mathcal{S}| < \Delta$ **do**
3:    Select $u = \arg\max_{k \in \mathcal{U} \setminus \mathcal{S}} \pi(k, t)$.
4:    $\mathcal{S} \leftarrow \mathcal{S} \cup \{u\}$.
5: **end while**
6: **return** $\mathcal{S}$.

$(\mathcal{B} \cap \overline{\mathcal{A}} \cap \overline{\mathcal{D}}) \cup (\mathcal{D} \cap \overline{\mathcal{A}} \cap \overline{\mathcal{D}}) = C \cap \overline{\mathcal{D}} = C \setminus (C \cap \mathcal{D})$. Hence, the influence $F(\mathcal{S}, t)$ is submodular and the proof is completed. $\square$

Based on the submodular property of $F(\mathcal{S}, t)$, we propose Algorithm 1, a greedy-based selection method to select an influential user set $\mathcal{S}$ with Δ users. More specifically, we first compute the influence of each user, and add the user with the largest influence to the candidate set $\mathcal{S}$ (breaking ties randomly). Then, we recompute the influence of the remaining users in the set $\mathcal{U} \setminus \mathcal{S}$, and find the user with the largest influence within the remaining users, so on and so forth. We repeat this process until we find Δ users. Clearly, the running time of Algorithm 1 is linear. The following result states that Algorithm 1 achieves a $(1 - 1/e)$ approximation ratio, and its proof follows immediately from standard results in submodular optimization [26].

Theorem 2. *Let $\mathcal{S}$ be the influential user set returned by Algorithm 1 and let $\mathcal{S}^*$ be the optimal influential user set, respectively. It then holds that $F(\mathcal{S}, t) \geq \left(1 - \frac{1}{e}\right) F(\mathcal{S}^*, t)$.*

Proof. Define function $F_{\mathcal{S}}(u) = F(\mathcal{S} \cup \{u\}) - F(\mathcal{S})$. We let $\mathcal{S}^* = \{s_1, s_2, ..., s_\Delta\}$ be the optimal influential user set, and $\mathcal{S}_i$ be the user set after the i-th iteration of Algorithm 1. Then we have:

$$F(\mathcal{S}^*) \stackrel{(a)}{\leq} F(\mathcal{S}_i \cup \mathcal{S}^*)$$
$$= F(\mathcal{S}_i) + F(\mathcal{S}_i \cup \{s_1\}) - F(\mathcal{S}_i) +$$
$$\quad F(\mathcal{S}_i \cup \{s_1, s_2\}) - F(\mathcal{S}_i \cup \{s_1\}) + \cdots$$
$$\stackrel{(b)}{\leq} F(\mathcal{S}_i) + F_{\mathcal{S}_i}(s_1) + F_{\mathcal{S}_i}(s_2) + \cdots + F_{\mathcal{S}_i}(s_\Delta)$$
$$\stackrel{(c)}{\leq} F(\mathcal{S}_i) + \Delta(F(\mathcal{S}_{i+1}) - F(\mathcal{S}_i)),$$

where (a) follows from the monotonically non-decreasing property of influence $F(\mathcal{S}, t)$; (b) uses the submodular property of $F(\mathcal{S}, t)$. By Theorem 1 we know that $F(\mathcal{B} \cup \{u\}) - F(\mathcal{A} \cup \{u\}) \leq F(\mathcal{B}) - F(\mathcal{A})$, then we have $F_{\mathcal{B}}(u) \leq F_{\mathcal{A}}(u)$, for two sets $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{A} \subseteq \mathcal{B}$; (c) is because $|\mathcal{S}_i| \leq \Delta$. Arranging the terms, we have:

$$F(\mathcal{S}^*) - F(\mathcal{S}_{i+1}) \leq \left(1 - \frac{1}{\Delta}\right)(F(\mathcal{S}^*) - F(\mathcal{S}_i)).$$

Recursively applying the inequality, we have:

$$F(\mathcal{S}^*) - F(\mathcal{S}_\Delta) \leq \left(1 - \frac{1}{\Delta}\right)^\Delta F(\mathcal{S}^*) \leq \frac{1}{e} F(\mathcal{S}^*).$$

Thus, we have $\left(1 - \frac{1}{e}\right) F(\mathcal{S}^*) \leq F(\mathcal{S}_\Delta)$, which completes the proof. $\square$

4.4 Solving Rating Scores for a Fake User

Given $\mathcal{S}$, we design a gradient-based method to solve the problem in (6). Recall that we let $\boldsymbol{w}_v = [w_{vi}, i \in \Omega_v]^\top$ be the rating vector for the current injected fake user v. We first determine his/her latent factors by solving Eq. (1), which can be restated as:

$$\underset{X,Y,z}{\arg\min} \sum_{(u,i)\in\mathcal{E}'} \left(r_{ui} - \boldsymbol{x}_u^\top \boldsymbol{y}_i\right)^2 + \sum_{i\in\mathcal{I}} \left(w_{vi} - z^\top \boldsymbol{y}_i\right)^2$$
$$+ \lambda \left(\sum_u \|\boldsymbol{x}_u\|_2^2 + \sum_i \|\boldsymbol{y}_i\|_2^2 + \|z\|_2^2\right), \quad (14)$$

where $z \in \mathbb{R}^d$ is the latent factor vector for fake user v, and $\mathcal{E}'$ is the current rating set (rating set $\mathcal{E}$ without attack plus injected ratings of fake users added before user v).

Toward this end, note that a subgradient of loss $\mathcal{L}_\mathcal{S}(\boldsymbol{w}_v)$ in (6) can be computed as:

$$G(\boldsymbol{w}_v) = \sum_{u\in\mathcal{S}} \sum_{i\in\Gamma_u} \nabla_{\boldsymbol{w}_v} g(\hat{r}_{ui} - \hat{r}_{ut}) + \eta \partial \|\boldsymbol{w}_v\|_1$$
$$= \sum_{u\in\mathcal{S}} \sum_{i\in\Gamma_u} \frac{\partial g\left(\delta_{u,it}\right)}{\partial \delta_{u,it}} \left(\nabla_{\boldsymbol{w}_v} \hat{r}_{ui} - \nabla_{\boldsymbol{w}_v} \hat{r}_{ut}\right) + \eta \partial \|\boldsymbol{w}_v\|_1, \quad (15)$$

where $\delta_{u,it} = \hat{r}_{ui} - \hat{r}_{ut}$ and $\frac{\partial g(\delta_{u,it})}{\partial \delta_{u,it}} = \frac{g(\delta_{u,it})(1-g(\delta_{u,it}))}{b}$. The subgradient $\partial \|\boldsymbol{w}_v\|_1$ can be computed as $\frac{\partial}{\partial w_{vi}} \|\boldsymbol{w}_v\|_1 = \frac{w_{vi}}{|w_{vi}|}$. To compute $\nabla_{\boldsymbol{w}_v} \hat{r}_{ui}$, noting that $\hat{r}_{ui} = \boldsymbol{x}_u^\top \boldsymbol{y}_i$, then the gradient $\frac{\partial \hat{r}_{ui}}{\partial \boldsymbol{w}_v}$ can be computed as:

$$\frac{\partial \hat{r}_{ui}}{\partial \boldsymbol{w}_v} = J_{\boldsymbol{w}_v}(\boldsymbol{x}_u)^\top \boldsymbol{y}_i + J_{\boldsymbol{w}_v}(\boldsymbol{y}_i)^\top \boldsymbol{x}_u, \quad (16)$$

where $J_{\boldsymbol{w}_v}(\boldsymbol{x}_u)$ and $J_{\boldsymbol{w}_v}(\boldsymbol{y}_i)$ are the Jacobian matrices of $\boldsymbol{x}_u$ and $\boldsymbol{y}_i$ taken with respect to $\boldsymbol{w}_v$, respectively. Next, we leverage first-order stationary condition to approximately compute $J_{\boldsymbol{w}_v}(\boldsymbol{x}_u)$ and $J_{\boldsymbol{w}_v}(\boldsymbol{y}_i)$. Note that the optimal solution of problem in (14) satisfies the following first-order stationary condition:

$$\lambda \boldsymbol{x}_u = \sum_{i\in\Omega_u} (r_{ui} - \boldsymbol{x}_u^\top \boldsymbol{y}_i)\boldsymbol{y}_i, \quad (17)$$
$$\lambda \boldsymbol{y}_i = \sum_{u\in\Omega^i} (r_{ui} - \boldsymbol{x}_u^\top \boldsymbol{y}_i)\boldsymbol{x}_u + (w_{vi} - z^\top \boldsymbol{y}_i)z, \quad (18)$$
$$\lambda z = \sum_{i\in\mathcal{I}} (w_{vi} - z^\top \boldsymbol{y}_i)\boldsymbol{y}_i, \quad (19)$$

where Ω_u is the set of items rated by user u and Ω^i is the set of users who rate the item i. Inspired by [19, 36], we assume that the optimality conditions given by (17)–(19) remain valid under an infinitesimal change of $\boldsymbol{w}_v$. Thus, setting the derivatives of (17)–(19) with respect to $\boldsymbol{w}_v$ to zero and with some algebraic computations, we can derive that:

$$\frac{\partial \boldsymbol{x}_u}{\partial w_{vi}} = \boldsymbol{0}, \quad (20)$$
$$\frac{\partial \boldsymbol{y}_i}{\partial w_{vi}} = \left(\lambda I + \sum_{u\in\Omega^i} \boldsymbol{x}_u \boldsymbol{x}_u^\top + zz^\top\right)^{-1} z, \quad (21)$$

where I is the identity matrix and (21) follows from $(\boldsymbol{x}_u^\top \boldsymbol{y}_i)\boldsymbol{x}_u = (\boldsymbol{x}_u \boldsymbol{x}_u^\top)\boldsymbol{y}_i$. Lastly, computing (20) and (21) for all $i \in \Gamma_u$ yields $J_{\boldsymbol{w}_v}(\boldsymbol{x}_u)$ and $J_{\boldsymbol{w}_v}(\boldsymbol{y}_i)$. Note that $\nabla_{\boldsymbol{w}_v} \hat{r}_{ut}$ can be computed in exactly the same procedure. Finally, after obtaining $G(\boldsymbol{w}_v)$, we can use the projected subgradient method [3] to solve $\boldsymbol{w}_v$ for fake user v. With $\boldsymbol{w}_v$, we select the top n items with largest values of w_{vi} as the filler items. However, the values of $\boldsymbol{w}_v$ obtained from solving (6) may not

---

Algorithm 2 *Our $\mathcal{S}$-Attack.*

Input: Rating matrix R, target item t, parameters $m, n, d, \eta, \lambda, \Delta, b$.
Output: Fake user set $\mathcal{M}$.
1: Find influential user set $\mathcal{S}$ according to Algorithm 1 for item t.
2: Let $\mathcal{M} = \emptyset$.
3: **for** $v = 1, \cdots, m$ **do**
4:     Solve the optimization problem defined in Eq. (6) to get $\boldsymbol{w}_v$.
5:     Select n items with the largest values of w_{vi} as filler items.
6:     Set $r_{vt} = r_{max}$.
7:     Let μ_i and σ_i^2 be item i's mean and variance of the scores rated by all normal users. Let $r_{vi} \sim \mathcal{N}(\mu_i, \sigma_i^2)$ be the random rating for each filler item i given by fake user v.
8:     Let $R \leftarrow R \cup \{r_v\}$ and $\mathcal{M} \leftarrow \mathcal{M} \cup \{v\}$.
9: **end for**
10: **return** $\{r_v\}_{v=1}^m$ and $\mathcal{M}$.

---

mimic the rating behaviors of normal users. To make our $\mathcal{S}$-attack more "stealthy," we will show how to generate rating scores to disguise fake user v. We first set $r_{vt} = r_{max}$ to promote the target item t. Then, we generate rating scores for the filler items by rating each filler item with a normal distribution around the mean rating for this item by legitimate users, where $\mathcal{N}(\mu_i, \sigma_i^2)$ is the normal distribution with mean μ_i and variance σ_i^2 of item i. Our $\mathcal{S}$-attack algorithm is summarized in Algorithm 2.

5 EXPERIMENTS

5.1 Experimental Setup

5.1.1 Datasets. We evaluate our attack on two real-world datasets. The first dataset is **Amazon Digital Music (Music)** [1]. This dataset consists of 88,639 ratings on 15,442 music by 8,844 users. The second dataset is **Yelp** [41], which contains 504,713 ratings of 11,534 users on 25,229 items.

5.1.2 $\mathcal{S}$-Attack Variants. With different ways of choosing the influential user set $\mathcal{S}$, we compare three variants of our $\mathcal{S}$-attack.

$\mathcal{U}$-Top-N attack ($\mathcal{U}$-TNA): This variant uses all normal users as the influential user set $\mathcal{S}$, i.e., $\mathcal{S} = \mathcal{U}$, then solve Problem (6).

$\mathcal{S}$-Top-N attack+Random ($\mathcal{S}$-TNA-Rand): This variant randomly selects Δ users as the influential user set $\mathcal{S}$, then solve Problem (6).

$\mathcal{S}$-Top-N attack+Influence ($\mathcal{S}$-TNA-Inf): This variant finds the influential user set $\mathcal{S}$ by Algorithm 1, then solve Problem (6).

5.1.3 Baseline Attacks. We compare our $\mathcal{S}$-attack variants with the following baseline attacks.

Projected gradient ascent attack (PGA) [19]: PGA attack aims to assign high rating scores to the target items and generates filler items randomly for the fake users to rate.

Stochastic gradient Langevin dynamics attack (SGLD) [19]: This attack also aims to assign high rating scores to the target items, but it mimics the rating behavior of normal users. Each fake user will select n items with the largest absolute ratings as filler items.

5.1.4 Parameter Setting. Unless otherwise stated, we use the following default parameter setting: $d = 64$, $\Delta = 400$, $\eta = 0.01$, $b = 0.01$, and $N = 10$. Moreover, we set the attack size to be 3% (i.e., the number of fake users is 3% of the number of normal users) and

Table 1: HR@10 for different attacks.

Dataset	Attack	Attack size				
		0.3%	0.5%	1%	3%	5%
Music	None	0.0017	0.0017	0.0017	0.0017	0.0017
	PGA [19]	0.0107	0.0945	0.1803	0.3681	0.5702
	SGLD [19]	0.0138	0.1021	0.1985	0.3587	0.5731
	$\mathcal{U}$-TNA	0.0498	0.1355	0.2492	0.4015	0.5832
	$\mathcal{S}$-TNA-Rand	0.0141	0.0942	0.2054	0.3511	0.5653
	$\mathcal{S}$-TNA-Inf	**0.0543**	**0.1521**	**0.2567**	**0.4172**	**0.6021**
Yelp	None	0.0015	0.0015	0.0015	0.0015	0.0015
	PGA [19]	0.0224	0.1623	0.4162	0.4924	0.6442
	SGLD [19]	0.0261	0.1757	0.4101	0.5131	0.6431
	$\mathcal{U}$-TNA	0.0619	**0.2304**	0.4323	0.5316	0.6806
	$\mathcal{S}$-TNA-Rand	0.0258	0.1647	0.4173	0.4923	0.6532
	$\mathcal{S}$-TNA-Inf	**0.0643**	0.2262	**0.4415**	**0.5429**	**0.6813**

the number of filler items is set to $n = 20$. We randomly select 10 items as our target items and the hit ratio (HR@N) is averaged over the 10 target items, where HR@N of a target item is the fraction of normal users whose top-N recommendation lists contain the target item. Note that our $\mathcal{S}$-attack is $\mathcal{S}$-TNA-Inf attack.

5.2 Full-Knowledge Attack

In this section, we consider the worst-case attack scenario, where the attacker has full knowledge of the recommender system, e.g., the type of the target recommender system (matrix-factorization-based), all rating data, and the parameters of the recommender system (e.g., the dimension d and the tradeoff parameter λ in use).

Table 1 summaries the results of different attacks. "None" means the hit ratios without any attacks. First, we observe that the variants of our $\mathcal{S}$-attack can effectively promote the target items using only a small number of fake users. For instance, in the Yelp dataset, when injecting only 0.5% of fake users, $\mathcal{S}$-TNA-Inf attack improves the hit ratio by 150 times for a random target item compared to that of the non-attack setting. Second, the variants of our $\mathcal{S}$-attack outperform the baseline attacks in most cases. This is because the baseline attacks aim to manipulate all the missing entries of the rating matrix, while our attack aims to manipulate the top-N recommendation lists. Third, it is somewhat surprising to see that the $\mathcal{S}$-TNA-Inf attack outperforms the $\mathcal{U}$-TNA attack. Our observation shows that by dropping the users that are not influential to the recommendation of the target items when optimizing the rating scores for the fake users, we can improve the effectiveness of our attack.

5.3 Partial-Knowledge Attack

In this section, we consider partial-knowledge attack. In particular, we consider the case where the attacker knows the type of the target recommender system (matrix-factorization-based), but the attacker has access to a subset of the ratings for the normal users and does not know the dimension d. In particular, we view the user-item rating matrix as a bipartite graph. Given a size of observed data, we construct the subset of ratings by selecting nodes (users and items) with increasing distance from the target item (e.g., one-hop distance to the target item, then two-hop distance and so on) on the bipartite graph until we reach the size of observed data.

Figure 1 shows the attack results when the attacker observes different amounts of normal users ratings and our attack uses different d, where the target recommender system uses $d = 64$. The
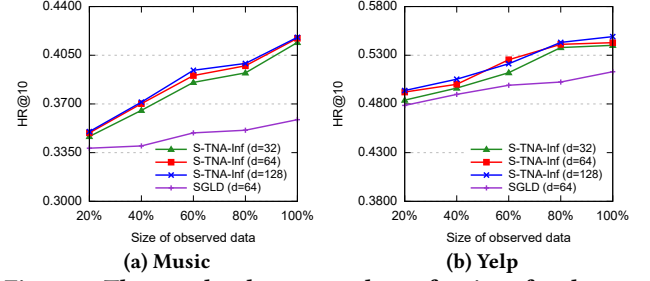

(a) Music **(b) Yelp**

Figure 1: The attacker knows a subset of ratings for the normal users and does not know d.
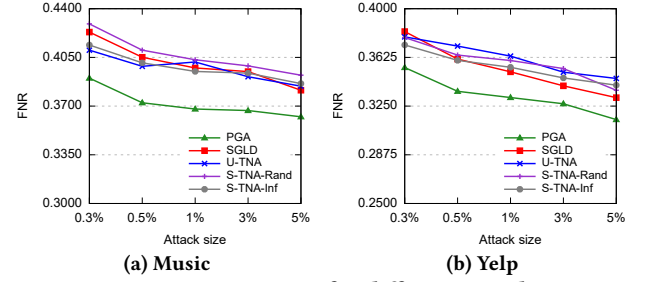

(a) Music **(b) Yelp**

Figure 2: FNR scores for different attacks.

attack size is set to be 3%. Note that in the partial-knowledge attack, the attacker selects the influential user set and generates fake users based only on the observed data. Naturally, we observe that as the attacker has access to more ratings of the normal users, the attack performance improves. We find that our attack also outperforms SGLD attack (which performs better than PGA attack) in the partial-knowledge setting. Moreover, our attack is still effective even if the attacker does not know d. In particular, the curves corresponding to different d are close to each other for our attack in Figure 1.

6 DETECTING FAKE USERS

To minimize the impact of potential attacks on recommender systems, a service provider may arm the recommender systems with certain fake-user detection capability. In this section, we investigate whether our attack is still effective in attacking the fake-user-aware recommender systems. Specifically, we extract six features–namely, RDMA [6], WDMA [23], WDA [23], TMF [23], FMTD [23], and MeanVar [23]–for each user from its ratings. Then, for each attack, we construct a training dataset consisting of 800 fake users generated by the attack and 800 randomly sampled normal users. We use the training dataset to learn a SVM classifier. Note that the classifier may be different for different attacks.

Fake-user detection results: We deploy the trained SVM classifiers to detect the fake users under different attacks settings. Figure 2 reports the fake users detection results of different attacks, where False Negative Rate (FNR) represents the fraction of fake users that are predicted to be normal. From Figure 2, we find that PGA attack is most likely to be detected. The reason is that the fake users generated by PGA attack do not rate the filler items according to normal users' behavior, thus the generated fake users are easily detected. We also observe that a large fraction of fake users are not detected.

Attacking fake-user-aware recommender systems: We now test the performance of attacks on fake-user-aware recommender

Table 2: HR@10 for different attacks when attacking the fake-user-aware recommender systems.

Dataset	Attack	Attack size				
		0.3%	0.5%	1%	3%	5%
Music	None	0.0011	0.0011	0.0011	0.0011	0.0011
	PGA [19]	0.0028	0.0043	0.0311	0.2282	0.3243
	SGLD [19]	0.0064	0.0145	0.0916	0.2631	0.3516
	$\mathcal{U}$-TNA	0.0127	0.0298	**0.1282**	0.2846	0.3652
	$\mathcal{S}$-TNA-Rand	0.0068	0.0139	0.0934	0.2679	0.3531
	$\mathcal{S}$-TNA-Inf	**0.0199**	**0.0342**	0.1215	**0.2994**	**0.3704**
Yelp	None	0.0010	0.0010	0.0010	0.0010	0.0010
	PGA [19]	0.0018	0.0062	0.1143	0.3301	0.4081
	SGLD [19]	0.0097	0.0278	0.1585	0.3674	0.4223
	$\mathcal{U}$-TNA	0.0231	0.0431	0.1774	0.3951	0.4486
	$\mathcal{S}$-TNA-Rand	0.0093	0.0265	0.1612	0.3665	0.4269
	$\mathcal{S}$-TNA-Inf	**0.0242**	**0.0474**	**0.1831**	**0.3968**	**0.4501**

systems. Suppose that the service provider removes the predicted fake users from the system detected by the trained SVM classifiers. We recompute the hit ratio after the service provider excludes the predicted fake users from the systems. Note that a large portion of fake users and a small number of normal users will be deleted. The results are shown in Table 2. We observe that PGA attack achieves the worst attack performance when the service provider removes the predicted fake users from the systems. The reason is that the PGA attack is most likely to be detected. Comparing Table 1 and Table 2, we can see that when the target recommender system is equipped with fake-user detectors, our attacks remain effective in promoting the target items and outperform the baseline attacks. This is because the detectors miss a large portion of the fake users.

7 DISCUSSION

We show that our influence function based approach can be extended to enhance data poisoning attacks to graph-based top-N recommender systems. In particular, we select a subset of normal users based on influence function and optimize data poisoning attacks using them. Moreover, we show that an attacker can also use influence function to weight each normal user instead of selecting the most influential ones, which sacrifices computational efficiency but achieves even better attack effectiveness.

7.1 Influence Function for Graph-based Recommender Systems

We investigate whether we can extend our influence function based method to optimize data poisoning attacks to graph-based recommender systems [10]. Specifically, we aim to find a subset of users who have the largest impact on the target items in graph-based recommender systems. It turns out that, when optimizing the attack over these subset of users, we obtain better attack effectiveness. Toward this end, we will first show how to find a subset of influential users for the target items in graph-based recommender systems. Then, we optimize the attack proposed by [10] over the subset of influential users.

We consider a graph-based recommender system using random walks [27]. Specifically, the recommender system models the user-item ratings as a bipartite graph, where a node is a user or an item, an edge between a user and an item means that the user rated the

item, and an edge weight is the corresponding rating score. We let p_u represent the stationary distribution of a random walk with restart that starts from the user u in the bipartite graph. Then, p_u can be computed by solving the following equation:

$$p_u = (1 - \alpha) \cdot Q \cdot p_u + \alpha \cdot e_u, \tag{22}$$

where e_u is a basis vector whose u-th entry is 1 and all other entries are 0, Q is the *transition matrix*, and α is the restart probability. We let q_{ui} denote the value at (u, i)-th entry of matrix Q. Then q_{ui} can be computed as:

$$q_{ui} = \begin{cases} \frac{r_{ui}}{\sum_j r_{uj}}, & \text{if } (u, i) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{23}$$

The N items that were not rated by user u and that have the largest probabilities in the stationary distribution p_u are recommended to u. We define the influence of removing edge $(k, j) \in \mathcal{E}$ in the user-item bipartite graph on the target item t when performing a random walk from user u as the change of prediction at p_{ut} upon removing edge (k, j):

$$\delta((k, j), p_{ut}) \overset{\text{def}}{=} \frac{\partial p_{ut}}{\partial q_{kj}}, \tag{24}$$

where q_{kj} is the transition matrix entry as defined in (23). According to (22), $\frac{\partial p_u}{\partial q_{kj}}$ can be computed as:

$$\frac{\partial p_u}{\partial q_{kj}} = (1 - \alpha)\frac{\partial Q}{\partial q_{kj}}p_u + (1 - \alpha)Q\frac{\partial p_u}{\partial q_{kj}}. \tag{25}$$

After rearranging terms in (25), we have:

$$\frac{\partial p_u}{\partial q_{kj}} = (1 - \alpha)(I - (1 - \alpha)Q)^{-1}\frac{\partial Q}{\partial q_{kj}}p_u, \tag{26}$$

where I is the identity matrix, $\frac{\partial Q}{\partial q_{kj}}$ is a single-nonzero-entry matrix with its (k, j)-th entry being 1 and 0 elsewhere. By letting $M \triangleq (I - (1 - \alpha)Q)^{-1}$, we have the following:

$$\frac{\partial p_u}{\partial q_{kj}} = (1 - \alpha)p_{uj}M(:, k), \tag{27}$$

where $M(:, k)$ is the k-th column of matrix M. Then, the influence of removing edge (k, j) on the prediction at the target item t when performing a random walk from user u can be calculated as:

$$\delta((k, j), p_{ut}) = \frac{\partial p_{ut}}{\partial q_{kj}} = (1 - \alpha)p_{uj}M(t, k). \tag{28}$$

Therefore, the influence of removing edge (k, j) on the prediction at the target item t can be computed as:

$$\varphi((k, j), t) \overset{\text{def}}{=} \sum_{u \in \mathcal{U}} \delta((k, j), p_{ut}). \tag{29}$$

We could approximate matrix M by using Taylor expansion $M = (I - (1 - \alpha)Q)^{-1} \approx I + \sum_{i=1}^{T}(1 - \alpha)^i Q^i$. For example, we can choose $T = 1$ if we use first order Taylor approximation.

After obtaining $\varphi((k, j), t)$, we can compute the influence of user k at the target item t, namely $\pi(k, t)$, based on (8). Then, we apply Algorithm 1 to approximately find an influential user set $\mathcal{S}$. With the influential user set $\mathcal{S}$, we can optimize the attack proposed by [10] over the most influential user set and compare with the attack

Table 3: HR@10 for attacks to graph-based recommender systems.

Dataset	Attack	Attack size				
		0.3%	0.5%	1%	3%	5%
Music	None	0.0021	0.0021	0.0021	0.0021	0.0021
	Fang et. al [10]	0.0252	0.1021	0.2067	0.2949	0.5224
	$\mathcal{S}$-Graph	0.0245	**0.1046**	**0.2125**	**0.3067**	**0.5368**
Yelp	None	0.0023	0.0023	0.0023	0.0023	0.0023
	Fang et. al [10]	0.0256	0.1359	0.2663	**0.4024**	0.5704
	$\mathcal{S}$-Graph	**0.0342**	**0.1514**	**0.2701**	0.4011	**0.5723**

Table 4: HR@10 for weighting-based attacks to matrix-factorization-based recommender systems.

Dataset	Attack	Attack size				
		0.3%	0.5%	1%	3%	5%
Music	Weighting	**0.0652**	**0.1543**	0.2436	**0.4285**	**0.6087**
Yelp	Weighting	**0.0698**	**0.2312**	**0.4498**	**0.5501**	**0.6924**

proposed by [10], which uses all normal users. The poisoning attack results of graph-based recommender systems are shown in Table 3, where the experimental settings are the same as those in [10]. Here, "None" in Table 3 means the hit ratios without attacks computed in graph-based recommender systems; and "$\mathcal{S}$-Graph" means optimizing the attack proposed by [10] over the most influential users in $\mathcal{S}$, where we select 400 influential users. From Table 3, we observe that the optimized attacks based on influence function outperform existing ones [10].

7.2 Weighting Normal Users

In this section, we show that our approach can be extended to a more general framework: we can weight the normal users instead of dropping some of them using the influence function. More specifically, we optimize our attack over all normal users, and different normal users are assigned different weights in the objective function based on their importance with respect to the target item. Intuitively, the important users should receive more penalty if the target item does not appear in those users' recommendation lists. Toward this end, we let $\mathcal{H} = [\mathcal{H}_u, u \in \mathcal{U}]^\top$ be the weight vector for all normal users, then we can modify the loss function defined in (6) as:

$$\min_{\boldsymbol{w}_v, \forall v} \mathcal{L}_{\mathcal{U}}(\boldsymbol{w}_v) = \sum_{u \in \mathcal{U}} \sum_{i \in \Gamma_u} g(\hat{r}_{ui} - \hat{r}_{ut}) \cdot \mathcal{H}_u + \eta \|\boldsymbol{w}_v\|_1$$
$$\text{s.t.} \sum_{u \in \mathcal{U}} \mathcal{H}_u = 1, \qquad (30)$$
$$w_{vi} \in [0, r_{max}],$$

where $\mathcal{H}_u$ is the weight for normal user u and satisfies $\mathcal{H}_u \geq 0$. We can again leverage the influence function technique to compute the weight vector $\mathcal{H}$. For a normal user k, the weight can be computed in a normalized fashion as follows:

$$\mathcal{H}_k = \frac{\pi(k, t)}{\sum_{u \in \mathcal{U}} \pi(u, t)}, \qquad (31)$$

where $\pi(k, t)$ is the influence of user k at the target item t, and can be computed according to (8). Note that here we compute $\pi(k, t)$ for each user k at one time.

After obtaining the weight vector $\mathcal{H}$, we can compute the derivative of function defined in (30) in a similar way. Table 4 illustrates the attack results on matrix-factorization-based recommender systems when we weight normal users, where the experimental settings are the same as those in Table 1. Here, "Weighting" means that we weight each normal user and optimize the attack of (30) over the weighted normal users, and the weight of each normal user is computed based on (31). Comparing Tables 1 and 4, we can see

that the performance is improved when we consider the weights of different normal users with respect to the target items. Our results show that, when an attacker has enough computational resource, the attacker can further improve attack effectiveness using influence function to weight normal users instead of dropping some of them.

8 CONCLUSION

In this paper, we proposed the first data poisoning attack to matrix-factorization-based top-N recommender systems. Our key idea is that, instead of optimizing the ratings of a fake user using all normal users, we use a subset of influential users. Moreover, we proposed an efficient influence function based method to determine the influential user set for a specific target item. We also performed extensive experimental studies to demonstrate the efficacy of our proposed attacks. Our results showed that our proposed attacks outperform existing ones.

REFERENCES

[1] Amazon Digital Music Dataset. 2018. http://jmcauley.ucsd.edu/data/amazon/
[2] Lars Backstrom and Jure Leskovec. 2011. Supervised Random Walks: Predicting and Recommending Links in Social Networks. In *Proceedings of the Fourth ACM International Conference On Web Search and Data Mining (WSDM)*. ACM, 635–644.
[3] Mokhtar S. Bazaraa, John J. Jarvis, and Hanif D. Sherali. 2010. *Linear Programming and Network Flows* (4 ed.). John Wiley & Sons Inc., New York.
[4] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines. In *ICML*.
[5] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. In *arxiv*.
[6] Paul-Alexandru Chirita, Wolfgang Nejdl, and Cristian Zamfir. 2005. Preventing Shilling Attacks in Online Recommender Systems. In *Proceedings of the 7th Annual ACM International Workshop on Web Information and Data Management (WIDM)*. ACM, 67–74.
[7] Konstantina Christakopoulou and Arindam Banerjee. 2019. Adversarial Ataks on an Oblivious Recommender. In *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys)*. 322–330.
[8] R Dennis Cook and Sanford Weisberg. 1980. Characterizations of an Empirical Influence Function for Detecting Influential Cases in Regression. *Technometrics* 22, 4 (1980), 495–508.
[9] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2020. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. In *Usenix Security Symposium*.
[10] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. 2018. Poisoning Attacks to Graph-Based Recommender Systems. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC)*. ACM, 381–392.
[11] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. In *Machine Learning and Computer Security Workshop*.
[12] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. 2018. Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. In *IEEE S & P*.
[13] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the Spread of Influence Through a Social Network. In *Proceedings of the ninth ACM SIGKDD*

International Conference on Knowledge Discovery and Data Mining (KDD). ACM, 137–146.

[14] Pang Wei Koh, Kai-Siang Ang, Hubert HK Teo, and Percy Liang. 2019. On the Accuracy of Influence Functions for Measuring Group Effects. In *Advances in Neural Information Processing Systems (NeurIPS).*

[15] Pang Wei Koh and Percy Liang. 2017. Understanding Black-box Predictions via Influence Functions. In *International Conference on Machine Learning (ICML).* 1885–1894.

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

[17] Shyong K Lam and John Riedl. 2004. Shilling Recommender Systems for Fun and Profit. In *Proceedings of the 13th International Conference on World Wide Web (WWW).* ACM, 393–402.

[18] Agata Lapedriza, Hamed Pirsiavash, Zoya Bylinskii, and Antonio Torralba. 2013. Are All Training Examples Equally Valuable? *arXiv preprint arXiv:1311.6510* (2013).

[19] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. 2016. Data Poisoning Attacks on Factorization-Based Collaborative Filtering. In *Advances in Neural Information Processing Systems (NeurIPS).* 1885–1893.

[20] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning Attack on Neural Networks. In *NDSS.*

[21] Bhaskar Mehta and Wolfgang Nejdl. 2008. Attack Resistant Collaborative Filtering. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval.* ACM, 75–82.

[22] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. 2005. Effective Attack Models for Shilling Item-Based Collaborative Filtering Systems. In *Proceedings of the WebKDD Workshop.* Citeseer, 13–23.

[23] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. 2007. Toward Trustworthy Recommender Systems: An Analysis of Attack Models and Algorithm Robustness. In *ACM Transactions on Internet Technology (TOIT)*, Vol. 7. ACM, 23.

[24] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive Privacy Analysis of Deep Learning: Stand-alone and Federated Learning under Passive and Active White-box Inference Attacks. In *2019 IEEE Symposium on Security and Privacy (SP).* IEEE.

[25] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia. 2008. Exploiting machine learning to subvert your spam filter. In *LEET.*

[26] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An Analysis Of Approximations For Maximizing Submodular Set Functions—I. *Mathematical programming* 14, 1 (1978), 265–294.

[27] Alain Pirotte, Jean-Michel Renders, Marco Saerens, et al. 2007. Random-Walk Computation of Similarities Between Nodes of a Graph with Application to Collaborative Recommendation. In *IEEE Transactions on Knowledge and Data Engineering.* IEEE, 355–369.

[28] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shinghon Lau, Satish Rao, Nina Taft, and JD Tygar. 2009. Antidote: understanding and defending against poisoning of anomaly detectors. In *ACM IMC.*

[29] Carlos E Seminario and David C Wilson. 2014. Attacking Item-Based Recommender Systems with Power Items. In *Proceedings of the 8th ACM Conference on Recommender systems (RecSys).* ACM, 57–64.

[30] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS).* 6103–6113.

[31] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP).* IEEE, 3–18.

[32] Binghui Wang and Neil Zhenqiang Gong. 2018. Stealing Hyperparameters in Machine Learning. In *2018 IEEE Symposium on Security and Privacy (SP).* IEEE, 36–52.

[33] Binghui Wang and Neil Zhenqiang Gong. 2019. Attacking Graph-based Classification via Manipulating the Graph Structure. In *CCS.*

[34] Tianyang Wang, Jun Huan, and Bo Li. 2018. Data Dropout: Optimizing Training Data for Convolutional Neural Networks. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI).* IEEE, 39–46.

[35] David C Wilson and Carlos E Seminario. 2013. When Power Users Attack: Assessing Impacts in Collaborative Recommender Systems. In *Proceedings of the 7th ACM conference on Recommender Systems (RecSys).* ACM, 427–430.

[36] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. 2015. Is Feature Selection Secure against Training Data Poisoning?. In *International Conference on Machine Learning (ICML).* 1689–1698.

[37] Han Xiao, Huang Xiao, and Claudia Eckert. 2012. Adversarial Label Flips Attack on Support Vector Machines. In *ECAI.*

[38] Xingyu Xing, Wei Meng, Dan Doozan, Alex C Snoeren, Nick Feamster, and Wenke Lee. 2013. Take This Personally: Pollution Attacks on Personalized Services. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13).* 671–686.

[39] Guolei Yang, Neil Zhenqiang Gong, and Ying Cai. 2017. Fake Co-visitation Injection Attacks to Recommender Systems. In *NDSS.*

[40] Haibo Yang, Xin Zhang, Minghong Fang, and Jia Liu. 2019. Byzantine-Resilient Stochastic Gradient Descent for Distributed Learning: A Lipschitz-Inspired Coordinate-wise Median Approach. In *Conference on Decision and Control (CDC).*

[41] Yelp Challenge Dataset. 2018. https://www.yelp.com/dataset/challenge

[42] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. 2018. Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates. In *International Conference on Machine Learning (ICML).*

[43] Xin Zhang, Minghong Fang, Jia Liu, and Zhengyuan Zhu. 2020. Private and Communication-Efficient Edge Learning: A Sparse Differential Gaussian-Masking Distributed SGD Approach. *arXiv preprint arXiv:2001.03836* (2020).

[44] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial Attacks on Neural Networks for Graph Data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD).* ACM, 2847–2856.