Towards Poisoning the Neural Collaborative Filtering-Based Recommender Systems

Yihe Zhang[1], Jiadong Lou[1], Li Chen[1], Xu Yuan[1,*], Jin Li[2], Tom Johnsten[3], and Nian-Feng Tzeng[1]

[1] University of Louisiana at Lafayette, Lafayette, LA 70503, USA
{yihe.zhang1, jiadong.lou1, li.chen, xu.yuan, nianfeng.tzeng}@louisiana.edu
[2] Guangzhou University, Guangzhou, Guangdong 510006, China
lijin@gzhu.edu.cn
[3] University of South Alabama, Mobile, AL 36688, USA
tjohnsten@southalabama.edu

Abstract. In this paper, we conduct a systematic study for the very first time on the poisoning attack to neural collaborative filtering-based recommender systems, exploring both availability and target attacks with their respective goals of distorting recommended results and promoting specific targets. The key challenge arises on how to perform effective poisoning attacks by an attacker with limited manipulations to reduce expense, while achieving the maximum attack objectives. With an extensive study for exploring the characteristics of neural collaborative filterings, we develop a rigorous model for specifying the constraints of attacks, and then define different objective functions to capture the essential goals for availability attack and target attack. Formulated into optimization problems which are in the complex forms of non-convex programming, these attack models are effectively solved by our delicately designed algorithms. Our proposed poisoning attack solutions are evaluated on datasets from different web platforms, *e.g.,* Amazon, Twitter, and MovieLens. Experimental results have demonstrated that both of them are effective, soundly outperforming the baseline methods.

1 Introduction

The recommender systems become prevalent in various E-commerce systems, social networks, and others, for promoting products or services to users of interest. The objective of a recommender system is to mine the intrinsic correlations of users' behavioral data so as to predict the relevant objects that may attract users' interest for promotion. Many traditional solutions leveraging the matrix factorization [16], association rule [8, 22, 6], and graph structure [10] techniques have been proposed by exploring such correlations to implement the recommender systems. Recently, the rapid advances in neural network (NN) techniques and

*Corresponding author.

their successful applications to diverse fields have inspired service providers to leverage such emerging solutions to deeply learn the intrinsic correlations of historical data for far more effective prediction, *i.e.*, recommendation, to improve users' experiences in using their web services [27, 5, 7, 14, 23]. A survey in [25] has summarized a series of NN-based recommender systems for objects recommendation. The neural collaborative filtering-based recommender system is among these emerging systems that have been proposed or envisioned for use in Youtube [7], Netflix [20], MovieLen [4], Airbnb [12], Amazon [18], and others.

However, existing studies have demonstrated that the traditional recommender systems are vulnerable to the poisoning attack [19, 9]. That is, an attacker can inject some fake users and operations on the network objects to disrupt the correlation relationships among objects. Such disrupted correlations can lead the recommender system to make wrong recommendation or promote attacker's specified objects, instead of original recommendation results that achieve its goal. For example, the poisoning attack solutions proposed in [19], [24], and [9] have been demonstrated with high efficiency in the matrix factorization-based, association rule-based, and graph-based recommender systems, respectively. To date, the poisoning attacks on the NN-based recommender systems have yet to be explored. It is still unsettled if this attack is effective to NN-based recommender systems.

Typically, such a recommender system takes all users' M latest operations to form a historical training dataset, which is used to train the weight matrices existing between two adjacent layers in neural networks. With the well trained weight matrices, the NN can recommend a set of objects that have the high correlations (similarity probabilities) within historical data. The goal of this paper is to perform poisoning attacks in neural collaborative filtering-based recommender systems and verify its effectiveness. The general idea of our poisoning attack approach is to inject a set of specified fake users and data. Once the recommender system takes the latest historical data for training, the injected data can be selected and act effectively to change the trained weight matrices, thus leading to wrong recommendation results. Specifically, two categories of attacks are studied, *i.e.*, availability attacks and target attacks. The first one aims to demote the recommended results by injecting poisoned data into the NN so as to change its output, *i.e.*, each object's recommendation probability. This category of attack refers to the scenario that some malicious users or service providers target to destroy the performance of other web applications' recommender systems. The second one aims to not only distort the recommended results but also promote a target set of objects to users. This category of attack connotes the application scenario that some users or business operators aim at promoting their target products, against the web servers' original recommendation.

In both attacks, we assume an attacker will inject as few operations as possible to minimize the expense of an attack while yielding the maximum attack outcomes. This objective is due to the fact that an attack's available resource may be limited and its detection should be avoided as best as possible. By defining the effective objective functions and modeling the resource constraints for each

attack, we formulate each corresponding scenario into an optimization problem. Even though the formulated problems are in the complex form of non-linear and non-convex programming, we design effective algorithms based on the gradient descent technique to solve them efficiently. To validate the effectiveness of our proposed two attack mechanisms, we take the real-world datasets from Amazon, Twitter, and MovieLens as the application inputs for evaluation. Experimental results show that our availability and target attacks both substantially outperform their baseline counterparts. Specifically, our availability attack mechanism lowers the accuracy by at least 60%, 17.7%, and 58%, in Amazon, Twitter, and MovieLens, respectively, given the malicious actions of only 5% users. Main contributions of this work are summarized as follows:

- We are the first to propose poisoning attack frameworks on the NN-based recommender systems. Through experiments, we have successfully demonstrated that the neural collaborative filtering-based recommender systems are also vulnerable to the poisoning attacks. This calls for the service providers to take into account the potential impact of poisoning attacks in the design of their NN-based recommender systems.
- We present two types of poisoning attacks, *i.e.*, availability attack and target attack, with the aim to demote recommendation results and promote the target objects, respectively. Through rigorous modeling and objective function creation, both types of attacks are formulated as the optimization problems with the goal of maximizing an attacker's profits while minimizing its involved operational cost (*i.e.*, the amount of injected data). Two effective algorithms are designed respectively to solve the two optimization problems.
- We implement our attack mechanisms in various real-world scenarios. Experimental results demonstrate that our poisoning attack mechanisms are effective in demoting recommended results and promoting target objects in the neural collaborative filtering-based recommender system.

2 Problem Statement

In this paper, we aim to design effective strategies for poisoning attacks on a general category of recommender systems based on neural collaborative filtering. Specifically, our goal is to achieve twofold distortions — *demoting* the recommended results and *promoting* the target objects — through *availability attack* and *target attack*, respectively.

2.1 Problem Setting

Given N users in the set $\mathcal{N}$ and D objects in the set $\mathcal{D}$, a recommender system is designed to recommend a small subset of objects to each user based on an estimate of the user's interest. In a neural collaborative-based recommender system, an NN is trained via historical data from each user to learn the probabilities of recommending objects to a given user, based on which the top-K objects will be selected for recommendation.

The historical operation behaviors (explicit [3] or implicit [15]) of a user can be learnt by a neural collaborative-based recommender system, such as marking, reviewing, clicking, watching, or purchasing. In practice, it typically truncates the M latest objects operated from each user n in the historical data, denoted as $\mathbf{u}_n = (u_n^1, u_n^2, ..., u_n^M)$, for learning. Let $\mathcal{U}$ denotes the entire dataset from all users, *i.e.*, $\mathcal{U} = \{\mathbf{u}_n | n \in \mathcal{N}\}$. The neural collaborative filtering model can be modeled as a function of $C^K(F(\mathcal{N}, \mathcal{D})) = \mathcal{P}^K$, which takes $\mathcal{N}$ and $\mathcal{D}$ as the input, and outputs the recommendation probability of each object j, i.e., $\mathbf{p}_n = \{p_n(j) | j \in \mathcal{D}\}$, for each user n. Here, F represents the neural collaborative filtering model and $p_n(j)$ satisfies $0 \leq p_n(j) \leq 1$ and $p_n(1) + p_n(2) + ... + p_n(D) = 1$ for $n \in \mathcal{N}$. $C^K(\cdot)$ function indicates the top-K objects selected according to the recommendation probabilities. For each user n, top-K recommended objects are denoted as $\mathbf{r}_n^K = (r_n^1, r_n^2, ..., r_n^K)$, and their corresponding scores (*i.e.*, probabilities) are represented as $\mathbf{p}_n^K$. The recommendation results of all users are denoted by $\mathcal{R} = \{\mathbf{r}_n^K | n \in \mathcal{N}\}$.

We perform poisoning attacks by injecting fake users into the network and enabling them to perform certain operations to distort the recommendation results for normal users. Assume there are $\hat{N}$ injected fake users with each performing at most M operations. Denote $\hat{\mathbf{u}}_n = (\hat{u}_n^1, \hat{u}_n^2, ..., \hat{u}_n^M)$ as the objects operated by a fake user n and $\hat{\mathcal{U}} = \{\hat{\mathbf{u}}_n | n \in \hat{\mathcal{N}}\}$ as the complete set of objects of $\hat{N}$ fake users. As the fake users disguise themselves over the normal users, the recommender system takes both normal users and fake users operations as the historical data for training, denoted by $\tilde{\mathcal{U}} = \mathcal{U} \cup \hat{\mathcal{U}}$, which naturally impacts the recommendation results. We denote $\tilde{\mathbf{r}}_n^K = (\tilde{r}_n^1, \tilde{r}_n^2, ..., \tilde{r}_n^K)$ as the top-K recommended objects for a normal user n after poisoning attacks and denote $\tilde{\mathcal{R}} = \{\tilde{\mathbf{r}}_n^K | n \in \mathcal{N}\}$ as the recommended results for all users. The attacker will control the amount of poisoned data operations (*i.e.*, fake users and their operations) to achieve its goal of demoting originally recommended objects or promoting target objects.

2.2 Attacks as Optimization Problems

We consider two types of poisoning attacks, *i.e.*, availability attack and target attack, with different profit requirements. In what follows, we will formulate them as optimization problems with the objective of optimizing their respective profits constrained by the available resources.

Availability Attack. The goal of availability attack is to demote the original recommendation results by poisoning data into the training dataset to change the NN output for the objects having top K highest probabilities. The profit of an attacker depends on the degree of distortion of the recommendation results. For its profit maximization, the attacker tries to achieve the largest discrepancy between recommendation results before and after the attack. We formulate an optimization problem in the following general form:

$$\begin{aligned} \text{OPT-A:} \ \ & \min S(\mathcal{R}, \tilde{\mathcal{R}}) \\ & s.t. \ \|\hat{\mathcal{U}}\|_0 \leq B, \ \hat{u}_n^m \in \{c_1, ..., c_d\} \ , \end{aligned} \tag{1}$$

where $S(\mathcal{R}, \tilde{\mathcal{R}})$ represents the *accuracy* metric of the recommendation results ($\tilde{\mathcal{R}}$) after the attack, as compared with those before the attack ($\mathcal{R}$). B represents the maximum limit of the poisoned data. With the $L0$-norm [21], the resource constraints address both limited fake users and their operations. The set of $\{c_1, ..., c_d\}$ represents a fake user's available operations. For example, in Ebay, this set is $\{0, 1\}$, representing whether a user clicks one product or not; In MovieLens, this set is $\{0, 1, 2, 3, 4, 5\}$, representing the rating score of one movie.

Target Attack. The goal of target attack is to not only distort the originally recommended results but also promote the target objects to users. Let $T = \{t^1, t^2, ..., t^K\}$ denote the K target objects that an attacker wishes to promote. We define a metric named the *successful score*, expressed as $H_T(\cdot)$, to measure the fraction of normal users whose top-K recommendation results include the target objects after the attack. To maximize the *successful score*, an attacker not only promotes target objects to as many users as possible but also promote as many target objects as possible for each user. This two-faceted consideration will be incorporated in the mathematical expression of $H_T(\cdot)$, to be discussed later. Such a problem is formulated in the following general form:

$$\begin{aligned} \text{OPT-T:} \ \ & \max H_T(\tilde{\mathcal{R}}) \\ & \quad s.t. \ \|\hat{\mathcal{U}}\|_0 \leq B, \ \hat{u}_n^m \in \{c_1, ..., c_d\} \ . \end{aligned} \tag{2}$$

The difference between the problem formulations of the availability attack and the target attack (*i.e.*, OPT-A and OPT-T) lies in their objective functions. We hope to design a general attack strategy, which can flexibly achieve different goals by switching the objective function in a lightweight manner.

3 Availability Attack in Recommender System

In this section, we present our strategy for performing the availability attack in neural collaborative filtering-based recommender systems. Assume that an attacker can create or operate a set of fake users and inject bogus data into the recommender system and distort the recommended results. The structure of our attack model is illustrated in Fig. 1, where the flow with solid arrow lines illustrates the general framework of a neural collaborative filtering model and the flow with the dashed arrow lines represents our attack framework. The NN model takes a user vector $\mathbf{x}_n$ and an object vector $\bar{\mathbf{x}}_j$ as inputs to calculate the score $p_n(j)$, denoting the probability of recommending object $j \in \mathcal{D}$ to user $n \in \mathcal{U}$. To perform the attack, an attacker collects a set of recommended results and train an attack table A, used to serve as the guideline for the attacker to determine the number of fake users and their operations. Details of the attack process are elaborated in the sequence text.

3.1 Attack Model Design

In Fig. 1, $\mathbf{x}_n$ and $\bar{\mathbf{x}}_j$ indicates one-hot encoding vectors that keep 1 in their corresponding categorical entries and leave all other entries as 0. $\mathbf{W}_1$ and $\mathbf{W}_2$ are

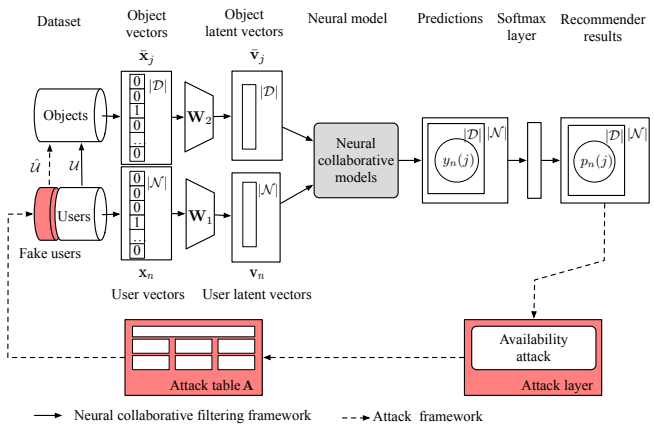


Fig. 1. Our attack framework.

two weight matrices that encode user vectors and object vectors into respective dense vectors, as follows:

$$\mathbf{v}_n = \mathbf{W}_1 \mathbf{x}_n \ , \ \bar{\mathbf{v}}_j = \mathbf{W}_2 \bar{\mathbf{x}}_j \ , \tag{3}$$

where $\mathbf{v}_n$ and $\bar{\mathbf{v}}_j$ indicate the user n's encoding vector and the object j's encoding vector, respectively. Here, both user and object encoding vectors are set to the same size. We assume the attacker knows the neural collaborative filtering-based recommender system used, but its detailed design is in black-box. According to characteristic of this system, we calculate the similarity of two encoding vectors from a user n and an object j by taking their *dot product* as follows: $y_n(j) = \bar{\mathbf{v}}_j \cdot \mathbf{v}_n^T$. Then, a Softmax layer is leveraged to calculate the probability of the object j by considering all the objects in the dataset:

$$p_n(j) = \frac{e^{\bar{\mathbf{v}}_j \mathbf{v}_n^T}}{\sum_{i=1}^{|\mathcal{D}|} e^{\bar{\mathbf{v}}_i \mathbf{v}_n^T}} \ . \tag{4}$$

To train this model to get W_1 and W_2, we employ the binary log likelihood [26] as follows:

$$G(\mathbf{v}_n, \bar{\mathbf{v}}_j) = L_{jn} \log(\sigma(\bar{\mathbf{v}}_j \mathbf{v}_n^T)) + (1 - L_{jn}) \log(\sigma(-\bar{\mathbf{v}}_j \mathbf{v}_n^T)) \ , \tag{5}$$

where $L_{jn} = 1$ when a user n has operated on an object j in the training dataset (positive sample) and $L_{jn} = 0$, otherwise (negative sample) [11]. Sigmoid function σ [13] is leveraged to calculate the similarity of two encoded vectors and Log likelihood is employed here for easy gradient calculation.

Given the historical data $\mathcal{U}$, we can express the total likelihood function as:

$$G(\mathcal{U}) = \sum_{n \in \mathcal{N}} \sum_{\bar{\mathbf{v}}_j \in \bar{\mathbf{V}}_n} \left[L_{jn} \log(\sigma(\bar{\mathbf{v}}_j \mathbf{v}_n^T)) + (1 - L_{jn}) \log \sigma(-\bar{\mathbf{v}}_j \mathbf{v}_n^T)) \right] \ , \tag{6}$$

where $\bar{\mathbf{V}}_n$ indicates the union of the positive and negative samples of a user n, which is sampled for efficiency consideration. To maximize $G(\mathcal{U})$, the stochastic gradient decent method is leveraged to solve it iteratively.

Based on our notations in Section 2, the number of normal users is N before the attack and the historical data of user operations is $\mathcal{U} = \{\mathbf{u}_n | 1 \leq n \leq N\}$. Assume the attacker injects a total of $\hat{N}$ users, with the whole set of operations being $\hat{U} = \{\mathbf{u}_n | 1 \leq n \leq \hat{N}\}$. With both historical and newly injected datasets, a recommender system will maximize the following objective in the training phase:

$$\begin{aligned} \tilde{G}(\tilde{U}) = &\sum_{n \in \mathcal{N}} \sum_{\bar{\mathbf{v}}_j \in \bar{\mathbf{V}}_n} \left[L_{jn} \log(\sigma(\bar{\mathbf{v}}_j \mathbf{v}_n^T)) + (1 - L_{jn}) \log \sigma(-\bar{\mathbf{v}}_j \mathbf{v}_n^T)) \right] \\ &+ \sum_{n \in \hat{\mathcal{N}}} \sum_{\bar{\mathbf{v}}_j \in \bar{\mathbf{V}}_n} \left[L_{jn} \log(\sigma(\bar{\mathbf{v}}_j \mathbf{v}_n^T)) + (1 - L_{jn}) \log \sigma(-\bar{\mathbf{v}}_j \mathbf{v}_n^T)) \right] \ , \end{aligned} \tag{7}$$

where the first summation term deals with the historical data of normal users while the second summation term is for the injected bogus data.

To distort the recommended results through polluting NN training in Eqn. (7), we first expand on the objective function of an availability attack presented in Section 2.2 and then model the constraints of injected fake users and their operations. With mathematical expressions of the goal and constraints, our availability attack is formulated and solved as an optimization problem, elaborated next.

3.2 Attack Objective Function

To distort the recommender system, the objective function of availability attack should be able to measure the discrepancy of recommended results before and after attack, i.e., $S(\mathcal{R}, \tilde{\mathcal{R}})$. As the recommender system typically recommends the top-K objects to a user, it is sufficient to consider only the top-K recommendations. Given the probabilities of all objects generated by the neural network, the recommended results $\mathcal{R}$ and $\tilde{\mathcal{R}}$ will store the objects that have the top K highest probabilities for each user before and after the attack, respectively. Before the attack, each object in the top-K recommendation list has a higher probability than any object after the K-th one. After the attack, if successful, at least one of the originally recommended objects, assuming object $r_n^k \in \mathcal{R}$ for a user n, will have a lower probability than the K-th object $\tilde{r}_n^K \in \tilde{\mathcal{R}}$ in the new ranking list, so that the object r_n^k will have a low chance to be recommended to the user. Thus, for each user n, we can define the following function to model the discrepancy of recommended results before and after attack:

$$\operatorname{sgn}[(p_n(r_n^k) - p_n(r_n^K)) \cdot (\tilde{p}_n(\tilde{r}_n^K) - \tilde{p}_n(r_n^k))] = \begin{cases} 1, & \tilde{p}_n(r_n^k) < \tilde{p}_n(\tilde{r}_n^K); \\ 0, & \tilde{p}_n(r_n^k) = \tilde{p}_n(\tilde{r}_n^K); \\ -1, & \tilde{p}_n(r_n^k) > \tilde{p}_n(\tilde{r}_n^K). \end{cases}$$

where $p_n(r_n^k)$ and $p_n(r_n^K)$ represent the probabilities of object r_n^k and the K-th object, respectively, in the recommendation list $\mathcal{R}$, while $\tilde{p}_n(r_n^k)$ and $\tilde{p}_n(\tilde{r}_n^K)$ stand for the probabilities of an object r_n^k and the Kth object in the recommendation results $\tilde{\mathcal{R}}$ after the attack. Note that $p_n(r_n^k)$ and $p_n(r_n^K)$ are known (*i.e.*,

constant) while $\tilde{p}_n(r_n^k)$ and $\tilde{p}_n(\tilde{r}_n^K)$ are variables, resulted from the poisoning attack. The intuition of this equation is explained as follows. After the attack, if an object r_n^k that is previously in the recommendation list $\mathcal{R}$ has a lower probability than the K-th object $\tilde{r}_n^K \in \tilde{\mathcal{R}}$, r_n^k will not appear in the top-K recommendation list, and thus the function $\text{sgn}(\cdot)$ returns 1, indicating a successful attack. Otherwise, $\text{sgn}(\cdot)$ returns 0 or -1, meaning an unsuccessful attack.

Furthermore, considering all the users, the discrepancy of recommended results, i.e., $S(\mathcal{R}, \tilde{\mathcal{R}})$, is expressed as follows:

$$S(\mathcal{R}, \tilde{\mathcal{R}}) = \sum_{n=1}^{N} \sum_{k=1}^{K} \frac{1}{2} \{1 - \text{sgn}[(p_n(r_n^k) - p_n(r_n^K)) \cdot (\tilde{p}_n(\tilde{r}_n^K) - \tilde{p}_n(r_n^k))]\}, \quad (8)$$

where $\frac{1}{2}(1 - \text{sgn}(\cdot))$ is used to map the inner part over a value between 0 and 1. Minimizing $S(\mathcal{R}, \tilde{\mathcal{R}})$ implies minimizing the recommendation accuracy after poisoning attack, and it is equivalent to maximizing the discrepancy of recommended results before and after the attack. However, Eqn. (8) is not continuous, unable to be solved directly. Based on the characteristics of $\frac{1}{2}(1-\text{sgn}(x))$, we use $1 - \frac{1}{exp(-\theta x)}$ to approximate it by setting θ to an approximate value. After reformulating Eqn. (8), we obtain the following objective function for $\min S(\mathcal{R}, \tilde{\mathcal{R}})$:

$$\min \sum_{n=1}^{N} \sum_{k=1}^{K} \{1 - \frac{1}{1 + \exp[-\theta_1 (p_n(r_n^k) - p_n(r_n^K)) \cdot (\tilde{p}_n(\tilde{r}_n^K) - \tilde{p}_n(r_n^k))]}\}, \quad (9)$$

where θ_1 is a positive parameter with its value adjusted into a suitable range.

3.3 Attack Constraints

We present the resource constraints for the injected fake users and operations.
Budget Constraints. To meet the resource limitation, the amount of injected fake users and operations is constrained by a budget B, expressed as follows:

$$\|\hat{\mathcal{U}}\|_0 \leq B \ , \quad (10)$$

where $\hat{\mathcal{U}}$ denotes the set of fake users and their operations while $\|\hat{\mathcal{U}}\|_0$ represents the L_0-norm of their operations.
Operation Constraints. The operations of fake users are reflected in the attack table $\mathbf{A}$ as shown in Fig. 1, which is used by an attacker to determine the operations of each fake user on each object. Attack table A is defined with the dimensions of $\hat{N} \times D$, with rows and columns representing fake users and objects. Each entry δ_{nj} is a weight value in the range of $[0, 1]$, indicating the likelihood that a fake user n interacts with object j. By adjusting the weight values, the attacker can change the strategies of operating its fake users, thus bending the recommendation results. Each entry can be approximated as follows:

$$\delta_{nj} = \frac{1}{2}(\tanh(a_{nj}) + 1) \ , \quad (11)$$

where $\tanh(\cdot)$ represents the *hyperbolic tangent function*.

We let each fake user n operate on at most M objects, *i.e.*, $\hat{\mathbf{u}}_n = (\hat{u}_n^1, \hat{u}_n^2, ..., \hat{u}_n^M)$. For a given fake user's dense vector $\mathbf{v}_n$ and an object's vector $\bar{\mathbf{v}}_j$, we can define our expected output of object j for a fake user n as a reference function $\tilde{y}_n(j)$:

$$\tilde{y}_n(j) = \delta_{nj} \cdot \sigma(\theta_p \bar{\mathbf{v}}_j \mathbf{v}_n^T) + (1 - \delta_{nj}) \cdot \sigma(-\theta_n \bar{\mathbf{v}}_j \mathbf{v}_n^T), \tag{12}$$

where θ_p and θ_n are shape parameters which adjust the sensitivity of user-object relationship, with the value of 0.5 as a threshold. If δ_{nj} is larger than this threshold, the attacker activates the operation. Since no attacker can manipulate normal users, δ_{nj} is a variable only related to fake users. For normal users, δ_{nj} is set to 1 if user i has relationship with object j, or set to 0, otherwise.

3.4 Solving the Optimization Problem

Based on our discussion, the availability attack can be reformulated as follows:

$$\begin{aligned} &\min\ S(\mathcal{R}, \tilde{\mathcal{R}}) \\ &s.t. \{\mathbf{W}_1, \mathbf{W}_2\} \leftarrow \operatorname{argmax} \tilde{G}(\tilde{\mathcal{U}}) \\ &\quad \|\hat{\mathcal{U}}\|_0 \leq B,\ \hat{u}_n^m \in \{c_1, ..., c_d\}, \hat{u}_n^m \in \hat{\mathcal{U}}. \end{aligned} \tag{13}$$

We first solve the outer objective of $\min S(\mathcal{R}, \tilde{\mathcal{R}})$ and then solve the inner objective of $\operatorname{argmax} \tilde{G}(\tilde{\mathcal{U}})$.

We use $E_a(\mathbf{v}_n, \bar{\mathbf{v}}_k)$ to represent the inner part of Eqn. (9), i.e.,

$$E_a(\mathbf{v}_n, \bar{\mathbf{v}}_k) = 1 - \frac{1}{1 + \exp[-\theta_1(p_n(r_n^k) - p_n(r_n^K)) \cdot (\tilde{p}_n(\tilde{r}_n^K) - \tilde{p}_n(r_n^k))]}\ . \tag{14}$$

Here, $p_n(r_n^k)$ and $p_n(r_n^K)$ are constants and can be obtained from Eqn. (12) by setting δ_{nj} to 0 or 1. $\tilde{p}_n(r_n^k)$ and $\tilde{p}_n(\tilde{r}_n^K)$ are variables, derived via Eqn. (12) by updating matrix $\mathbf{A}$. To reduce the computation cost, $\tilde{p}_n(\tilde{r}_n^K)$ will be updated only after we have finished one round of calculation on all objects and users. If our attack is successful, E will be close to 0.

To model the goal of taking as few operations as possible, the L_1 norm is added to the inner part of objective function Eqn. (14), resulting in a new function, denoted as $loss_a(\mathbf{v}_n, \bar{\mathbf{v}}_k)$:

$$loss_a(\mathbf{v}_n, \bar{\mathbf{v}}_k) = E_a(\mathbf{v}_n, \bar{\mathbf{v}}_k) + c|\delta_{nk}|, \tag{15}$$

where c is a constant that adjusts the importance of two objective terms. Now, we use two phases to solve Eqn. (15).

Phase I: This phase aims to update δ values in $\mathbf{A}$, by minimizing the loss function $loss_a$, with a two-step iterative procedure stated as follows:

Step 1: We initialize the random value for each a in Eqn. (11) and employ the gradient descent method to update a iteratively, as follows:

$$a_{ik} := a_{ik} - \eta_a \nabla_a loss_a(\mathbf{v}_n, \mathbf{v}_k)\ , \tag{16}$$

where η_a is the step size of updating a.

Step 2: We fix the attack table $\mathbf{A}$ and update the ranking of reference recommendation scores of all objects using Eqn. (12). According to the new ranking, we can find the object $\tilde{r}_n^{K*}$.

The two steps repeat until the convergence criterion is satisfied, *i.e.*, the gradient difference between two consecutive iterations is less than a threshold.

Phase II: This phase is to add the fake users and update the vector representation of injected objects. Again, a two-step iterative procedure is used:

Step 3: Assume the attacker will control one fake user n and select S objects to perform the fake operations, we have: $S = \min\{\frac{B}{N'}, M\}$, where N' is the total number of fake users.

Step 4: After adding a fake user and its operations to the training dataset, we train new $\mathbf{W}_1$ and $\mathbf{W}_2$ by solving the inner optimization problem. Let $\mathbf{v}_n$ and $\bar{\mathbf{v}}_j$ denote the user vector and object vector, respectively. For an easy expression, we denote the terms inside the summation in Eqn. (7) as $\mathbb{L}(\mathbf{v}_n, \bar{\mathbf{v}}_j)$ and then calculate the derivative of $\mathbb{L}(\mathbf{v}_n, \bar{\mathbf{v}}_j)$ with respect to $\mathbf{v}_n$ and $\bar{\mathbf{v}}_j$, i.e.,

$$\nabla_j \mathbb{L}(\mathbf{v}_n, \bar{\mathbf{v}}_j) = \frac{\partial \mathbb{L}(\mathbf{v}_n, \bar{\mathbf{v}}_j)}{\partial \bar{\mathbf{v}}_j} = (L_{jn} - \sigma(\bar{\mathbf{v}}_j \mathbf{v}_n^T))\mathbf{v}_n^T. \tag{17}$$

Then, we update $\mathbf{v}_j$ as follows: $\bar{\mathbf{v}}_j := \bar{\mathbf{v}}_j + \eta_j \cdot \nabla_j \mathbb{L}(\mathbf{v}_n, \bar{\mathbf{v}}_j)$, where η_j is the update step. With respect to $\mathbf{v}_n$, we have: $\nabla_n \mathbb{L}(\mathbf{v}_n, \bar{\mathbf{v}}_j) = \frac{\partial \mathbb{L}(\mathbf{v}_n, \bar{\mathbf{v}}_j)}{\partial \bar{\mathbf{v}}_j} = (L_{jn} - \sigma(\bar{\mathbf{v}}_j \mathbf{v}_n^T))\bar{\mathbf{v}}_j$. Then, for each $\mathbf{v}_i$ employed to aggregate $\mathbf{v}_n$, we have:

$$\mathbf{v}_n^T := \mathbf{v}_n^T + \eta_n \cdot \sum_j \nabla_n \mathbb{L}(\mathbf{v}_n, \bar{\mathbf{v}}_j)/(M-1)\ , \tag{18}$$

where η_n is the step size to update $\mathbf{v}_n$. Step 4 repeats until the convergence criterion is satisfied, i.e., the object vector expression result changes negligibly.

Phase 1 and **Phase 2** will repeat until the budget constraints are met or the attack purpose is achieved.

4 Target Attack

The goal of target attack is to promote specific objects to the normal users. The key challenge lies in designing an expression to measure the *successful score* of a targeted object. Denote $\mathcal{T} = \{t^1, t^2, ..., t^K\}$ as the target objects that an attacker wishes to promote to normal users. An attacker aims to promote the target objects in $\mathcal{T}$ to the top-K recommendation list by making them ranked higher than the highest recommended item r_n^1, formulated as follows:

$$\text{sgn}[(p_n(r_n^t) - p_n(r_n^1)) \cdot (\tilde{p}_n(\hat{r}_n^t) - \tilde{p}_n(r_n^1))] = \begin{cases} -1, \tilde{p}_n(r_n^t) > \tilde{p}_n(r_n^1); \\ 0, \quad \tilde{p}_n(r_n^t) = \tilde{p}_n(r_n^1); \\ 1, \quad \tilde{p}_n(r_n^t) < \tilde{p}_n(r_n^1). \end{cases} \tag{19}$$

In this function, r_n^t and r_n^1 are constant, which are known for a recommender system before attack. After performing poisoning attack, a success results in $\tilde{r}_n^t$

ranked higher than $\tilde{r}_n^1$ and $\text{sgn}(x)$ returning -1; otherwise, the $\text{sgn}(x)$ returns 1 or 0. Its corresponding *successful score* can be expressed by:

$$H_T(\tilde{\mathcal{R}}) = \sum_{n=1}^{N}\sum_{t=1}^{K}\frac{1}{2}\{1 - \text{sgn}[(p_n(r_n^t) - p_n(r_n^1)) \cdot (\tilde{p}_n(\hat{r}_n^t) - \tilde{p}_n(r_n^1))]\}. \tag{20}$$

To make this problem solvable, the *successful score* is approximated as follows:

$$\max \sum_{n=1}^{N}\sum_{t=1}^{K}\{1 - \frac{1}{1 + exp[-\theta_1(p_n(r_n^t) - p_n(r_n^1)) \cdot (\tilde{p}_n(\hat{r}_n^t) - \tilde{p}_n(r_n^1))]}\}. \tag{21}$$

The attack constraints illustrated in Section 3.3, i.e., budget and operation constraints, can also apply to the target attack. Thus, target attack can be formulated as follows:

$$\begin{aligned} &\max\ H_T(\tilde{\mathcal{R}}) \\ &s.t.\ \{\mathbf{W}_1, \mathbf{W}_2\} = \text{argmax}\,\tilde{G}(\tilde{U}) \\ &\quad \|\hat{U}\|_0 \leq B,\ \hat{u}_n^m \in \{c_1, ..., c_d\}, \hat{u}_n^m \in \hat{U}. \end{aligned} \tag{22}$$

Like Eqn. (14), we have:

$$loss_t(\mathbf{v}_n, \bar{\mathbf{v}}_t) = E_t(\mathbf{v}_n, \bar{\mathbf{v}}_t) + c|\delta_{nt}|, \tag{23}$$

where $E_t(\mathbf{v}_n, \bar{\mathbf{v}}_t)$ is the inner part of Eqn. (21).

We can follow the similar procedure as in **Phases 1** and **2** of Section 3.4 to solve this problem iteratively.

5 Experiments

We implement the proposed attack frameworks and evaluate them by using three real-world datasets from Amazon, Twitter and MoiveLens, as outlined next.

5.1 Datasets

Amazon [2]. This dataset contains the item-to-item relationships, *e.g.*, a customer who bought one product X also purchased another product Y. The historical purchasing records are used by a recommender system to recommend product to a user. In our experiments, we take the data in the *Beauty* category as our dataset, which includes 1000 users and 5100 items.

Twitter [17]. The social closeness of users, represented by friend or follower relationships, is provided in this dataset. Specifically, we use 1000 users and 3457 friendships in our evaluation.

MoiveLens [1]. This dataset is from a non-commercial and personalized movie recommender system collected by GroupLens, consisting of 1000 users' ratings on 1700 movies. The rating scores range from 0 to 5, indicating the preference of users for movies.

5.2 Comparison

The state-of-the-art poisoning attacks target specific categories of recommender systems: [19], [24], [9] aiming to the matrix factorization based, association rule based, and graph based recommender systems, respectively. There is no straightforward method to apply them to the neural network-based recommender systems, making it infeasible to compare our solution with them. As far as we know, there is no existing poisoning attack solution for neural collaborative filtering-based recommender systems. Therefore, we propose to compare our solutions with the baseline methods, as described next, to show their performance gains.

Baseline Availability Attack. An attacker randomly selects M objects, with the number of injected fake users ranging from 1% to 30% of the users in the historical dataset. Each fake user operates on these M objects to poison the historical dataset. To be specific, in Amazon, each fake user purchases M selected products; in Twitter, each fake user follows M other users; in MovieLens, each user chooses M movies and rates them with random scores.

Baseline Target Attack. An attacker has a set of target objects $\mathcal{T}$. Each fake user randomly selects a target object from $\mathcal{T}$ and then selects another $M-1$ popular objects in the network to operate on. The purpose of such a selection is to build close correlation of the target object and the popular objects so as to have the chance of being recommended. With respect to the selection of target objects in this attack, we consider two strategies: 1) selecting objects randomly and 2) selecting unpopular objects.

5.3 Performance of Availability Attack

We implement the neural collaborative filtering-based recommender systems for the Amazon, Twitter and MovieLens datasets and then perform both our poisoning attack solutions and the baseline availability attack to distort the recommended results. We set $K = 30$, i.e., recommending the top 30 objects.

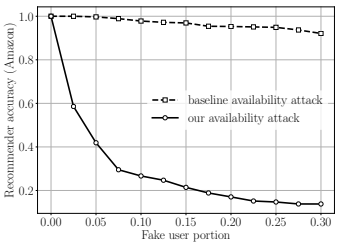


Fig. 2. Comparison of recommendation accuracy of our availability attack and the baseline method in Amazon with fake users portions varying from 0 to 30%.

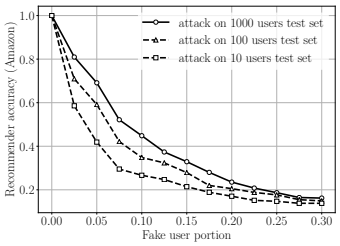


Fig. 3. The recommendation accuracy of our availability attack for the Amazon dataset under different amounts of users in test dataset.

Amazon. There are 1000 users selected, each with the latest $M = 8$ purchased products as the historical dataset $\mathcal{U}$. The neural collaborative filtering-based recommender system takes the historical dataset for training and then makes recommendations. We first randomly select 10 users for testing. Fig. 2 shows the recommendation accuracy after performing our availability attack and baseline attack with the portion (the percentage of inject fake users over the total normal users in the historical data) of injected fake users increasing from 1% to 30%, *i.e.*, from 10 to 100 users. From this figure, it is obvious that our availability attack significantly outperforms the baseline method in terms of reducing the recommendation accuracy. Especially, the accuracy of recommendation drops by $60\%, 80\%$ and 90% when the percentages of fake users are $5\%, 15\%$ and 30%, respectively, in our availability attack. However, the recommendation accuracy drops only by $0.1\%, 2\%$ and 9%, respectively, in the baseline attack.

We now exam our solutions on various amounts of testing users, including 10, 100 and 1000. Fig. 3 illustrates the recommendation accuracy outcomes, given different amounts of testing users when injecting the fake user count ranging from 1% to 30% in our attack. Intuitively, to achieve the same attack goal (in terms of the accuracy drop rate), more fake users need to be injected to change the correlation relationships of more testing users. However, our result is seen to be highly effective, i.e., recommendation accuracy drops less when the amount of testing users rises. Specifically, to achieve a 60% accuracy decrease, our solution only needs to inject 8% and 10% fake users respectively under 100 and 1000 users in the test set. This demonstrates that an attacker can achieve an excellent attack goal by resorting to only a small number of fake users.

Twitter. We select 1000 users and their corresponding relationships with other users sampled from [17] as the training dataset, while another 1000 users and their relationships are selected for testing. As it is hard to identify the latest friends in Twitter, we let each user randomly select 8 friends. 10 users are randomly selected from the test dataset as the targets to perform our proposed attack and the baseline availability attack. Fig. 4 demonstrates the recommendation accuracy after injecting the fake user count in the range of 1% to 30% into the Twitter dataset. From this figure, the recommendation accuracy from our attack is found to drop by $17.7\%, 45.2\%$ and 61.8%, when the portions of fake users are $5\%, 15\%$ and 30%, respectively. Compared to the accuracy results attained by the baseline availability attack, which drops by only $0.07\%, 1.5\%$ and 5.7%, respectively, we conclude that our attack is starkly more effective.

To show the effectiveness of our attack on different types of users, we select 100 popular users that are followed by most users and 100 unpopular users that do not have any follower. Besides, we randomly select another 100 users as the third group of target users. Fig. 5 shows the results after our attack on the three groups of users: the recommendation accuracy drops slower from the popular dataset than from both random and unpopular datasets, with the accuracy from the unpopular dataset dropping fastest. The reason is that the popular users have more knitted relationships with other users, thus requiring an attacker to invoke more operations to change such correlated friendships.

However, for an unpopular dataset, the friendships relationship is much lighter, thus making it far easier for our attack to change its correlation with others.

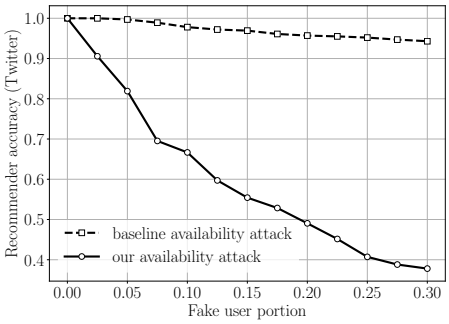


Fig. 4. Comparison of recommendation accuracy of our availability attack and the baseline method in Twitter under a range of fake users portions.

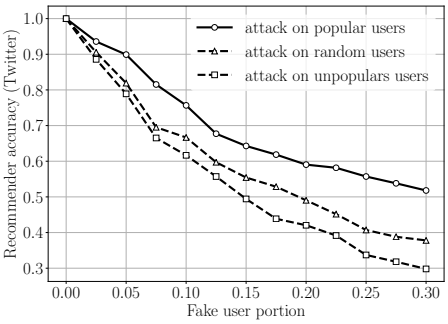


Fig. 5. Recommendation accuracy of availability attack in Twitter with respect to the popular, random and unpopular user datasets.

MovieLens. We select 1000 users and their rating scores in a total of 1700 movies sampled from [1] as the historical data and use other 10 users as the testing data. The portion of fake users increases from 0% to 5%, 10% and 30% when conducting our attack while K value increases from 1 to 30. Figure 6 depicts the recommendation accuracy of our attack with an increase in the K value under various portions of fake users. In this figure, we can see the recommendation accuracy drops less when K increases. The reason is that with a higher K value (more recommendation results), the fake users need to change more correlations among movies, thus lowering the successful rate. But with the portion of fake users rising, the recommendation accuracy can drop more as the attacker invokes more efforts to change the correlations among movies. Specifically, with 5% injected fake users, the recommendation accuracy drops by more than 50%. This demonstrates the superior effectiveness of our attack on the MovieLens.

Figure 7 shows the recommendation accuracy of baseline availability attack. It is seen the recommendation accuracy drops less than that from our attack (Figure 6). Even when the portion of fake users rises to 30%, it drops less than 14%, which is still worse than our attack with only 1% fake users injected. Thus, the baseline solution is clearly outperformed by our attack.

5.4 Performance of Target attack

We define a metric *hit ratio* to indicate the fraction of normal users whose top K recommendations contain the target items, after the attack.

Amazon. We use the same historical data in Amazon as in Section 5.3 and select 1000 other users for testing. 30 products outside the original top 30 recommendation list are randomly selected as our targets. Figure 8 compares hit

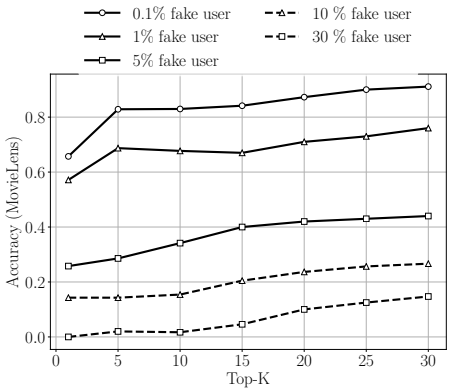


Fig. 6. Recommendation accuracy of availability attack in MovieLens with different portions of fake users and various K values.

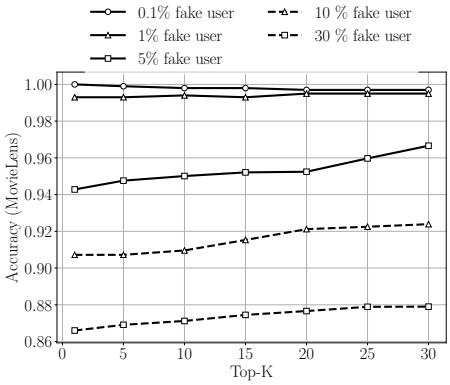


Fig. 7. Recommendation accuracy of baseline availability attack in MovieLens with different portions of fake users and various K values.

ratios of our target attack and those of the baseline method for various amounts of injected fake users, ranging from 1% to 30%. It is seen that our target attack significantly outperforms the baseline approach. Specifically, our attack can achieve the hit ratios of $4.83\%, 18.00\%$ and 40.20%, respectively, under the injected fake user count of $5\%, 15\%$ and 30%. However, the baseline method can only attain the hit ratios of $0.00\%, 0.7\%$ and 5.36%, respectively.

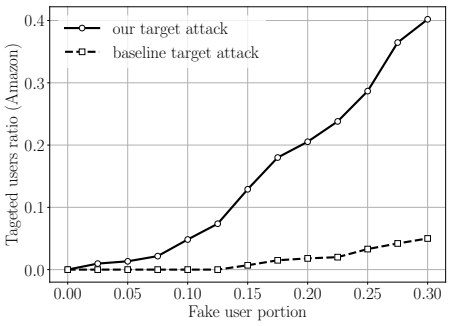


Fig. 8. Comparison of our target attack and the baseline method in Amazon under a range of fake users portions in the dataset.

We then fix the amount of fake users to 5% of all normal users in historical data (*i.e.*, 50 users) and change the amounts of both desired recommended results and target products. We randomly select the amount of target products from 1, 5, 10, to 30, while varying K from 1 to 5, 10 and 30. Table 1 lists the hit ratios

from our attack and the baseline method. The hit ratios of both our attack and the baseline method are found to increase with a larger K or more target products. However, our attack always achieves a much higher hit ratio than the baseline counterpart. For example, when the amount of target products is 10, the hit ratios increase from 0.06% to 1.85% in our attack and from 0 to 0.09% in the baseline method, respectively, when K varies from 1 to 30.

Table 1. *Hit ratios of our target attack and the baseline method in the Amazon dataset under a range of K values for a range of target products*

hit ratio (%)		K			
		1	5	10	30
# of target products (Our attack)	1	0.01	0.03	0.08	0.31
	5	0.03	0.10	0.54	1.02
	10	0.06	0.15	1.17	1.85
	30	0.08	0.27	1.32	2.17
# of target products (Baseline)	1	0.00	0.00	0.01	0.03
	5	0.00	0.01	0.02	0.04
	10	0.00	0.02	0.04	0.09
	30	0.00	0.03	0.05	0.11

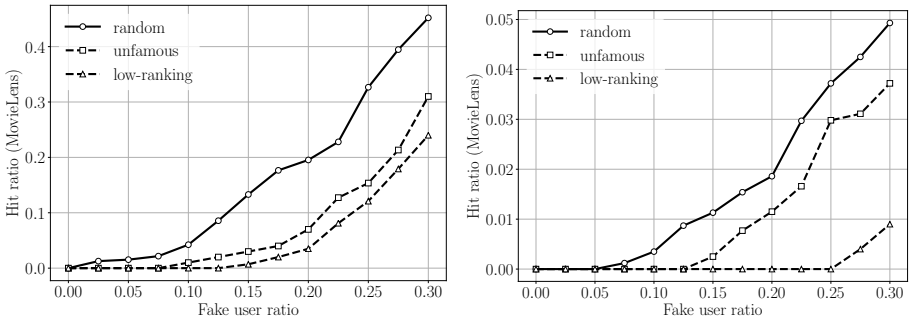
Twitter. We use the same sampled training and test datasets from Section 5.3 to perform the target attack. The portion of fake users is fixed to 10%, *i.e.*, 100 users, and K ranges from 1 to 5, 10 and 30. The target users are randomly selected, with the user number varying from 1 to 5, 10 and 30. Table 2 provides the hit ratios of our attack and the baseline method. It is seen that the hit ratios increase with a larger K or more target users in both our attack and the baseline counterpart. But our attack significantly outperforms the baseline method. Especially, when $K = 30$, our attack can achieve the hit ratios of $0.39\%, 1.02\%, 2.21\%$ and 4.62% with the number of target users varying from 1 to $5, 10$ and 30, respectively, while the baseline method reaches the respective hit ratios of $0.05\%, 0.08\%, 0.09$ and 0.18%.

MovieLens. We use the same training dataset from Section 5.3 and sample three categories of test dataset: random, unpopular, and low-ranking movies. For each category, we randomly select 10 movies as the target set. For recommendation, we consider the top 30 ones. Fig. 9(a) shows the hit ratios of our attack under the three categories of target movie set. From this figure, we can see the hit ratios of the unpopular and low ranking movie sets are lower than those from the random set. The reason is that the unpopular and low ranking targets have fewer correlations with others, thus making it much harder to promote them for recommendations when compared with the random target set. Still, our attack achieves 23% and 31% hit ratios by injecting 30% fake users. This demonstrates the advantages of our proposed attack on promoting products.

Table 2. *Hit ratios of our target attack and the baseline method in Twitter dataset under a range of K values for a range of target products*

hit ratio (%)		K			
		1	5	10	30
# of target users (Our attack)	1	0.02	0.03	0.011	0.39
	5	0.04	0.12	0.66	1.02
	10	0.07	0.20	1.72	2.21
	30	0.11	0.49	2.01	4.62
# of target users (Baseline)	1	0.01	0.01	0.03	0.05
	5	0.01	0.01	0.04	0.08
	10	0.01	0.02	0.05	0.09
	30	0.02	0.04	0.07	0.18

Fig. 9(b) depicts the hit ratios of baseline target attack. Compared to Fig. 9(a), the hit ratios under baseline attack are much lower than those from our attack. Even in the random target movie set, the baseline has its hit ratio of only 0.048% with the injection of 30% fake users.



(a) Our Target Attack. (b) Baseline Target Attack.

Fig. 9. Comparisons of hit ratios of baseline and our target attack in MovieLens on the random, unpopular, low-ranking target sets for a range of fake users portions.

6 Conclusion

This paper proposes the first poisoning attack framework targeting the neural collaborative filtering-based recommender system. We have studied two types of poisoning attacks: the availability attack and the target attack, with the goals of demoting recommendation results and promoting specific target objects, respectively. By developing the mathematical models based on the resource constraints and establishing objective functions according to attacker's goals, we

formulated these two types of attacks into optimization problems. Through algorithm designs, we solved the proposed optimization problems effectively. We have implemented the proposed solutions and conducted our attacks on the real-world datasets from Amazon, Twitter and MovieLens. Experimental results have demonstrated that the proposed availability attack and target attack are highly effective in demoting recommendation results and promoting specific targets, respectively, in the neural collaborative filtering-based recommender systems.

Acknowledgement

This work was supported in part by NSF under Grants 1948374, 1763620, and 1652107, and in part by Louisiana Board of Regents under Contract Numbers LEQSF(2018-21)-RD-A-24 and LEQSF(2019-22)-RD-A-21. Any opinion and findings expressed in the paper are those of the authors and do not necessarily reflect the view of funding agency.

References

1. grouplens.org. `https://grouplens.org/datasets/movielens/100k/`, 2019.
2. Mcauley,julian. `http://jmcauley.ucsd.edu/data/amazon/`, 2019.
3. Xavier Amatriain, Josep M Pujol, Nava Tintarev, and Nuria Oliver. Rate it again: increasing recommendation accuracy by user re-rating. In *Proceedings of the third ACM conference on Recommender Systems*, pages 173–180, 2009.
4. Ting Bai, Ji-Rong Wen, Jun Zhang, and Wayne Xin Zhao. A neural collaborative filtering model with interaction-based neighborhood. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1979–1982, 2017.
5. Oren Barkan and Noam Koenigstein. Item2vec: neural item embedding for collaborative filtering. In *26th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, 2016.
6. John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998.
7. Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 191–198, 2016.
8. James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM Conference on Recommender Systems*, pages 293–296, 2010.
9. 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)*, pages 381–392, 2018.
10. Francois Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):355–369, 2007.

11. Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. 2014.
12. Mihajlo Grbovic and Haibin Cheng. Real-time personalization using embeddings for search ranking at airbnb. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 311–320, 2018.
13. Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pages 195–201. Springer, 1995.
14. Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*, pages 173–182, 2017.
15. Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. Ieee, 2008.
16. Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, (8):30–37, 2009.
17. Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pages 591–600. AcM, 2010.
18. Hardesty Larry. The history of amazon's recommendation algorithm. `https://www.amazon.science/the-history-of-amazons-recommendation-algorithm`, 2020.
19. Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1885–1893, 2016.
20. Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, pages 689–698, 2018.
21. Luis Mancera and Javier Portilla. L0-norm-based sparse representation through alternate projections. In *2006 IEEE International Conference on Image Processing*, pages 2089–2092, 2006.
22. 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.
23. Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1235–1244, 2015.
24. Guolei Yang, Neil Zhenqiang Gong, and Ying Cai. Fake co-visitation injection attacks to recommender systems. *NDSS*, 2017.
25. Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):5, 2019.
26. Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in Neural Information Processing Systems (NIPS)*, pages 8778–8788, 2018.
27. Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiusi Chen, and Jun Gao. Atrank: An attention-based user behavior modeling framework for recommendation. In *31th AAAI Conference on Artificial Intelligence*, 2018.