Reverse Attack: Black-box Attacks on Collaborative Recommendation

Yihe Zhang
yihe.zhang1@louisiana.edu
University of Louisiana at Lafayette
Lafayette, LA, USA

Xu Yuan*
xu.yuan@louisiana.edu
University of Louisiana at Lafayette
Lafayette, LA, USA

Jin Li
lijin@gzhu.edu.cn
Guangzhou University
Guangzhou, Guangdong, China

Jiadong Lou
jiadong.lou1@louisiana.edu
University of Louisiana at Lafayette
Lafayette, LA, USA

Li Chen
li.chen@louisiana.edu
University of Louisiana at Lafayette
Lafayette, LA, USA

Nian-Feng Tzeng
tzeng@louisiana.edu
University of Louisiana at Lafayette
Lafayette, LA, USA

ABSTRACT

Collaborative filtering (CF) recommender systems have been extensively developed and widely deployed in various social websites, promoting products or services to the users of interest. Meanwhile, work has been attempted at poisoning attacks to CF recommender systems for distorting the recommend results to reap commercial or personal gains stealthily. While existing poisoning attacks have demonstrated their effectiveness with the offline social datasets, they are impractical when applied to the real setting on online social websites. This paper develops a novel and practical poisoning attack solution toward the CF recommender systems without knowing involved specific algorithms nor historical social data information a priori. Instead of directly attacking the unknown recommender systems, our solution performs certain operations on the social websites to collect a set of sampling data for use in constructing a surrogate model for deeply learning the inherent recommendation patterns. This surrogate model can estimate the item proximities, learned by the recommender systems. By attacking the surrogate model, the corresponding solutions (for availability and target attacks) can be directly migrated to attack the original recommender systems. Extensive experiments validate the generated surrogate model's reproductive capability and demonstrate the effectiveness of our attack upon various CF recommender algorithms.

CCS CONCEPTS

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

KEYWORDS

Recommender System; Poisoning Attack

*Corresponding author

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484805

ACM Reference Format:
Yihe Zhang, Xu Yuan, Jin Li, Jiadong Lou, Li Chen, and Nian-Feng Tzeng. 2021. Reverse Attack: Black-box Attacks on Collaborative Recommendation. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3460120.3484805

1 INTRODUCTION

The web service providers, driven by profits, have devoted efforts to promoting their products and enriching user experience, through employing the recommender systems. The mainstream of recommender systems is based on the collaborative filtering (CF) methods, with a series of algorithms developed and some of them widely deployed in the E-commerces [30, 42, 71], social networks [56, 73], online websites [29, 67], and mobile applications [55, 85]. They aim to help the service providers promote products or services that can increase the visibility of their items (*i.e.*, products and services) to the users of interest. Most social websites reportedly are using the CF recommender algorithms, for example, Amazon [33], Airbnb [43], NetEase Music [74], and eBay [88], although the specific algorithms are not revealed. The essence of a CF recommender system is to mine the intrinsic correlations of user behaviors, item-item relationships, and user-item interactions, to locate users/items in the preference of other users. In literature, the CF recommender algorithms can be categorized as being item-based [8, 17, 54, 68], matrix-factorization-based [44, 77, 87], neural network-based [6, 16, 37, 78, 86, 90], and graph structure-based [5, 26]. Plentiful algorithms have been proposed, with some of them already deployed into online websites for production use.

However, existing works have demonstrated that the CF recommender systems are vulnerable to an attacker with the purpose of distorting the recommender results for some specific profits. Data poisoning attack is the most popular and effective technique that has been applied for recommendation result falsification via injecting fake users and operations to intentionally change the items or users' recommendation relationships so that the original recommendation list is distorted. Various poisoning attack strategies have been proposed to target item-based [9, 11, 19, 31, 32, 48, 58, 63, 70, 82], matrix factorization-based [14, 23, 39, 51], neural network-based [12, 20, 36, 53, 75, 80], graph-based recommender systems [24, 84], respectively. However, all the aforementioned data poisoning attack solutions fall into the white-box attack, where the specific recommender algorithm and historical training data information

are all assumed to be exposed to the attacker. This is impractical in real-world social websites, since the respective algorithm and information typically are kept secret by service providers.

Although some approaches have been proposed in [15, 22, 72, 83] without knowing the specific recommender algorithms, they still require certain historical data knowledge, *e.g.*, the training set and the frequency of items that were recommended within a certain period, which are not always available in practice. In this paper, we aim to develop a practical black-box attack strategy for effectively distorting recommender systems embedded in social websites, without the prior knowledge of either recommender algorithms or historical data information. An attacker only needs to have public knowledge obtained simply by performing the operations as normal users.

Our strategy is to construct a surrogate model that estimates item proximities for the embedded recommender systems. Instead of performing attack straight, we craft our attack strategy in the surrogate model to optimize the attack profits, and then directly migrate such a strategy to the target social website for distorting its recommendations. Specifically, we first intentionally perform some regular operations (*e.g.*, viewing, clicking, rating) as normal users on the social websites through user's interface and gather the recommended results as the sampling data. Two alternative methods, *i.e.*, random walk collection and random injection collection, are proposed, to suit different social websites. Since the sampling data include affluent and implicit information of the embedded recommender algorithms (no matter which specific algorithm is using), we construct a surrogate model by deeply learning the item proximities. Focusing on attacking the surrogate model, we craft our solutions for both the availability attack and the target attack. We define the objective functions of maximizing attack profits, *i.e.*, demoting and promoting recommendation results by the largest degree, under the availability attack and target attack, respectively. By solving the formulated optimization problems, the number of operations can be obtained, representing our attack solutions, which can be applied to target social websites for an effective attack.

We conduct extensive experiments to verify the capability of our constructed surrogate model in estimating item proximities on the sampling data from several online websites and some real-world datasets. Results demonstrate that our models efficiently calculate the items' distributional representations, highly correlated with those obtained from the original recommender systems. To show our attack performance, we conduct experiments solely on real-world datasets due to ethical considerations. Our attack strategies generated from the surrogate model are directly migrated to the target recommender systems for attacking. Through experimenting on 9 datasets, we obtain consistent results, exhibiting that both availability attack and target attack work effectively on all 9 examined CF algorithms. In addition, our solutions outperform the compared counterparts which conduct white-box attacks with both the recommender algorithms and training sets known a priori.

2 BACKGROUND KNOWLEDGE

2.1 CF Recommender Systems

CF is a widely adopted method in many categories of recommender systems and has been implemented in various social medias. It makes the personalized recommendation of items or users to a certain user by mining the latent intricate relationships of users and items in the historical data for modeling the similarity of users' interests or behaviors. Collectively, CF recommender systems can be grouped into the following four categories.

1) Item-based CF calculates the similarity among users or items using *Pearson correlation*, *Vector cosine*, *Euclidean distance*, and others across the entire user-item relationship matrix, denoted by $\boldsymbol{M}_{uv} \in \mathbb{R}^{m \times n}$, where the rows represent m users and the columns represent n items, to identify the K most similar users or items for recommendation. Each entry is a numerical value representing the relationship between a user and an item. For example, in MovieLens, each entry denotes a rating score (0.5 to 5) or no-relationship (Ω) from a user to a movie. In Amazon, each entry represents the visiting behavior (denoted as 1 or Ω) of a user to an item.

2) Matrix factorization-based CF further decomposes the user-item interaction matrix $\boldsymbol{M}_{uv}$ into the product of two lower dimensional matrices, where the first one has a row for each user while the second one has a column for each item. Such two lower dimensional matrices can be considered as the distributional representation of users or items, representing the latent factors of users or items similarity. The product of two lower dimensional matrices will then result in a full ranked matrix, which models all the users and items relationships, assisting item recommendations.

3) Graph-based CF models the relationships of users or items as the bipartite graph $\boldsymbol{G}_b = \{\mathcal{U}, \mathcal{V}, \mathcal{E}\}$ or co-visitation graph $\boldsymbol{G}_c = \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{U}$ represents users, $\mathcal{V}$ denotes items and $\mathcal{E}$ denotes relationships of two vertices. Graph-based recommender systems recommend products to users based on the geometry relationships of users and items in the user-item bipartite graph. Different graph-based techniques can be utilized to find users' or items' similarities and relationships, which will be utilized to make recommendations.

4) Neural network-based CF encodes each user or item into a latent vector space. At each hidden layer, new features can be extracted from the previous vector space to a new vector space. At the output layer, the similarity among the latent user vectors or item vectors will be calculated. Such a method can make an in-depth calculation of the inherent and sophisticated relationships of users to users, items to items, or users to items so as to find the similarity patterns.

2.2 Related Works

Regarding item-based CF recommender systems, the early attack solutions proposed in [11, 19] aimed to help the developer to build a robust CF method when suffering from unfair ratings in online trading communities. Later, [63] showed that by injecting users with biased ratings, an attacker can promote or demote the target items. [48] demonstrated that some naive methods like injecting RandomBot and AverageBot could also promote or demote the target item by assigning the maximum or minimum ratings to the randomly selected items, respectively. Furthermore, some advanced methods, *i.e.*, Bandwagon attack[9], Probe attack[57], Consistency attack[10] and Segment attack [31], were proposed. These attack methods, although effective, are not efficient in poisoning recommender systems. To broaden the impact of poisoning attack, the authors in [70, 82] proposed the power user/item attack model based on in-degree centrality, via using power users to lunch an attack or solely

attack the power items. However, this line of attacks requires the recommender algorithms or users' historical knowledge.

Work on attacking the matrix factorization-based CF abounds. Specifically, [51] leveraged the first-order KKT conditions to derive the fake ratings for the attacking strategy. [39] developed a bi-level program to perform target attacks. [14] proposed a generative adversarial network (GAN)-based attack model to generate ratings for fake user profiles. [23] proposed to select the influential users for attacking. However, each of these attacks targets only one specific algorithm while requiring full or partial historical user-item interaction knowledge for designing the attack strategies.

In the line of graph-based CF system attacks, [84] proposed to inject the fake co-visitations into a co-visitation graph. The optimized attack strategy was derived by solving the constrained linear optimization problems given a bounded number of fake visitations, with full graph knowledge available. They then proposed a low knowledge attack method, which is only feasible to the simplified recommender algorithm with the linear model. [24] proposed an attack solution that calculates the fake user rating scores by solving an optimization problem. However, it still requires full graph topology knowledge, often unavailable in the real-world systems.

Recently, the successful applications of neural network techniques to recommender systems attract adversary attacks. Such attacks target the personality ranking [36], E-commerce websites [20], visual-based recommender systems [75], and others. However, all these attacks rely on the gradient-based methods, such as Fast Gradient Sign Method (FGSM) [28] and Projected Gradient Descent (PGD) [46], requiring to generate the adversarial perturbations from the entire datasets, apparently inapplicable to real-world social platforms, since the whole datasets are unlikely to be available. Similarly, GAN-based [27] attack models [12, 15, 80] also suffer from the same shortcomings. Moreover, [53] proposed to generate the applicable fake user profiles based on GAN, but it needs to use the existing real user profile as a "template", which is very hard to acquire from recommender systems and also raises the privacy issue. [41] and [89] proposed the poisoning attacks on neural-network based recommender systems. However, they requires the whole knowledge of the recommender system structure and of the dataset. Again, the specific neural network algorithms in aforementioned works must be known prior to performing all these attacks.

All the aforementioned attack solutions belong to the white-box attack, since the specific recommender algorithms are known a priori. Meanwhile, some black-box attacks without knowing the recommender algorithms, have been pursued. Specifically, [83] discovered the vulnerability of the YouTube recommender system and conducted a real-world pollution attack. However, this attack solution only distorted the partial functionality of the recommender systems. In [22], reinforcement learning was introduced to develop black-box attack on a source domain, with the attack solution then transferred to the target systems. Such a solution has the strict requirement that the source domain and the target system should have overlapped user profiles, but how to guarantee overlapped user profiles in real-world attack remains questionable. [72] also employed reinforcement learning to develop a black-box attack solution, leveraging the binary tree structure to generate fake user profiles. It requires knowledge on the recommended frequency of some items within a certain period, which rarely holds in practice.

3 PROBLEM STATEMENT

This paper aims to design effective strategies to perform black-box poisoning attacks on CF recommender systems that are embedded in social websites, with the goal of twofold distortions: 1) *demoting* recommended results and 2) *promoting* the target items, referring to as *availability attack* and as *target attack*, respectively.

3.1 Threat Model

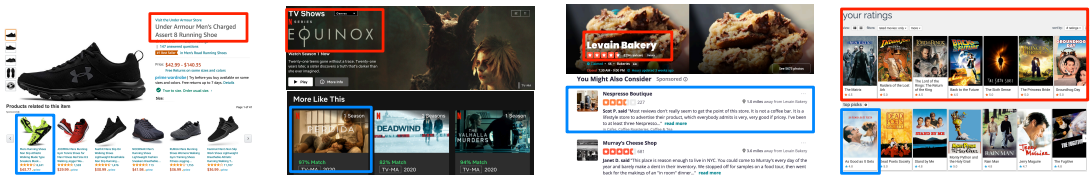
To perform the attacks, the amounts of adversarial behaviors (*i.e.*, fake users and their operations) are constrained due to the resource limitation and detection avoidance. An attacker will craft an effective attack strategy, subject to these constrained resources, for optimizing the attack profits in target social websites. By injecting the well-crafted fake users and operations, an attacker can achieve its goal of maximally distorting the recommendation results. In practice, social websites do not expose the specific recommender algorithms currently being used for security reasons. Public information only indicates that Amazon [33], Airbnb [43], NetEase Music [74], Spotify [74], eBay [88], and others currently employ the CF methods to produce high quality recommendations [54]. Thus, when performing practical attacks on a social website, an attacker does not have the knowledge of its specific recommender algorithm being used, other than that it falls in the category of CF. Also, the historical data of the users and items relationships are unknown to the attacker either. All knowledge obtainable by an attacker is from the attacker's normal interactions with the social websites. In our attack, we treat both the recommender system and historical data in a target social website as a black box. Nonetheless, an attacker can access the social website through its user interface to view or visit users and items, via regular operations provided to users. The attacker then collects the public data and uses them to design its attack strategy. Following the attack strategy, the attacker injects fake users and operations allowed by the systems, such as rating, clicking, or viewing actions, to perform effective attacks.

3.2 Sketch of Our Attack Strategy

Before presenting design details in the next two sections (Sections 4 and 5), we first give a sketch of our black-box attack strategy. Since an attacker neither knows recommendation algorithms nor has the prior knowledge of users or items relationships, the first step is to interact with target social websites via the normal operations and collect a set of recommended results to serve as the sampling data for learning. To this end, a surrogate model is developed for deeply learning the recommendation relationships from the sampled data, aiming to estimate the item proximities and the implicit patterns. It will assist in developing solutions offline for evaluation. After that, a solution that is effective on the surrogate model, is directly transferred to the target social websites to achieve the similar attack goals. Notably, the recommended objects can be either users or items corresponding to different social websites. For ease of expression, we uniformly call them items in the following sections.

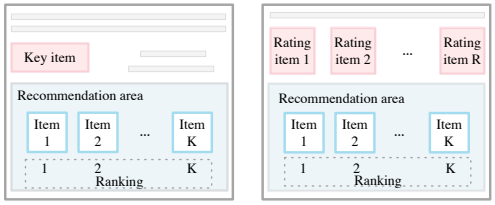
4 CONSTRUCTING SURROGATE MODEL

To construct the surrogate model, we first collect the sampling data in online social websites for learning use. Two alternative approaches, *i.e.,* Random Walk Collection and Random Injection



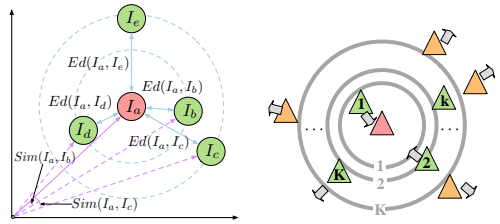
(a) Amazon (b) Netflix (c) Yelp (d) MoiveLens

Figure 1: Layout on different websites. Red and blue boxes denote the key item and the related recommendation, respectively.



(a) Type-I layout (b) Type-II layout

Figure 2: Two common recommendation web page layouts.



(a) (b)

Figure 3: (a) Euclidean distance and Cosine similarity. (b) An illustration of the surrogate model. Red, green, and yellow triangles denote the target item, the recommended ones, and other items, respectively. Gray arrows denote the gradient direction. The numbers denote the recommendation rank.

Collection, are proposed for this purpose. Next, we design a training strategy for the surrogate model to efficiently learn from the collected dataset for exploring item proximities.

4.1 Sampling Data Collection

Random Walk Collection. This approach is suitable for those websites that recommend items based on items' similarities, *e.g.*, Amazon, Netflix, and Yelp, exampled as in Figures 1(a), 1(b), and 1(c). The recommendation web page layout suitable for applying this collection approach is illustrated in Figure 2(a), called Type-I layout. There is one key item I and a recommendation area which lists K similar items for recommendation, denoted as $F(I) = \{I^{(1)}, I^{(2)}, \ldots, I^{(K)}\}$. Notably, rankings of these items are usually implicit to users but their proximities are disclosed. Our collection procedure takes into account such relative ranking information. The indicator of recommendation area varies on different social websites. For example, in Amazon, this section is named as *"Customers who viewed this item also viewed"*. To start the Random Walk Collection, we randomly select an item I_1 (called the key item) for the first sampling trail $C_w(1)$. Assuming there are K similar recommender items on the Type-I recommender area, we shall record all of them as $C_w(1) = \{(I_1, I_1^{(1)}, 1), (I_1, I_1^{(2)}, 2), \ldots, (I_1, I_1^{(K)}, K)\}$, where 1 to K are the listed rankings for them. Corresponding to each recommended item, a sampling score $c_k = e^{-\lambda k}$ is assigned based on its ranking, where k is item's ranking, and λ is a parameter to adjust item ranking importance, ranging from 0.001 to 0.5. Hence, each item can be sampled with the probability of $p(I_1^{(k)}) = \frac{c_k}{\sum_{i=1}^{K} c_i}$. The larger the λ, the higher the probability to sample a higher ranking item. We iteratively execute operations above to select the next nodes until the maximum walk length $\mathcal{Z}$ is reached. With further trails sampled following the similar way, we eventually obtain a set of sampled data, denoted as $\mathbb{C}_w = \{C_w(1), C_w(2), \ldots, C_w(\mathcal{Z})\}$.

Random Injection Collection. This approach is suitable for those websites that recommend items based on users' historical preferences, *e.g.*, MovieLens as shown in Figure 1(d). We call this type of website a Type-II layout, as sketched in Figure 2(b), where the items recommended to users are not targeted to one particular item, but depending on users' historical behaviors, *e.g.*, rating or viewing on a group of items. In this approach, an attacker can create a set of user accounts on social media websites and randomly perform some operations. The social media gives the top recommendations to each user, which will be collected and stored in the form of a list. Here, each operated item can be considered as the target item. In particular, the fake users rate or view $\mathcal{A}$ items in the websites, with the fake ratings from a fake user u denoted as $R_i(u) = \{(I_u^{(1)}, r_u^{(1)}), (I_u^{(2)}, r_u^{(2)}), \ldots, (I_u^{(\mathcal{A})}, r_u^{(\mathcal{A})})\}$, where $I_u^{(k)}$ is the k-th item rated by u and $r_u^{(k)}$ is the corresponding rating. The recommendation area for Type-II layout is similar to that for Type-I layout. We record all the recommended K items with their ranking numbers, denoted as $C_i(1) = \{(I_{u1}^{(1)}, r_{u1}^{(1)}, I_1^{(1)}, 1), (I_{u1}^{(1)}, r_{u1}^{(1)}, I_1^{(2)}, 2), \ldots, (I_{u1}^{(1)}, r_{u1}^{(1)}, I_1^{(K)}, K), \ldots, (I_{u1}^{(\mathcal{A})}, r_{u1}^{(\mathcal{A})}, I_1^{(K)}, K)\}$. Such a procedure repeatedly executes to create a set of user accounts for collecting a sufficient amount of sampling data. For $\mathcal{Z}$ fake users, the sampled data are denoted as $\mathbb{C}_i = \{C_i(1), C_i(2), \ldots, C_i(\mathcal{Z})\}$.

We explore popular websites across E-commerce, social networks, Entertainment, Tourism, and Review categories, that an attacker can explore the two methods for sampling data, summarized in Table 9 of Appendix A.1. It exhibits that an attacker could use at least one method to collect data from these popular websites.

4.2 Generating Surrogate Model

We next construct a surrogate model to learn recommendation patterns from the sampling data for producing item proximities. Several challenges exist in designing such a surrogate model. First, the model ought to thoroughly learn the item proximities by leveraging either ranking or rating information. But, how to design the model for effectively capturing such item proximity remains challenging. Second, due to the limit of an attacker's available resources

and the intent to avoid triggering websites' detection mechanisms, the sample data cannot be collected in any arbitrarily large size. Hence, the constrained sample data size further raises the challenge for developing a surrogate model to learn item proximities. Third, the surrogate model is desired as simple as possible while realizing items' potential relationships without the knowledge of recommender algorithm. How to simplify the surrogate model design while still achieving our attack purpose remains open.

Notably, the design of surrogate model aims to learn the item proximities under a small subset of the data. For example, when a recommender system recommends new products based on historical user operations, a surrogate model is expected to effectively learn the relationships from those recommended products. Poisoning the proximities of items therein may migrate to distort the proximities over an entire dataset. Naturally, training a surrogate model can be considered as a metric learning problem [18].

Euclidean distance metric is used to gauge the distance between two items. For items I_a and I_b, as shown in Figure 3(a), the corresponding vectors are denoted by $\boldsymbol{v}_a$ and $\boldsymbol{v}_b$. The Euclidean distance $Ed(I_a, I_b)$ between I_a and I_b is calculated as $Ed(I_a, I_b) = \|\boldsymbol{v}_a - \boldsymbol{v}_b\| = \sqrt{\sum_{i=1}^{d}(\boldsymbol{v}_a(i) - \boldsymbol{v}_b(i))^2}$, where d is the vector size. For example, if item I_b is closer to item I_a than item I_c, the Euclidean distance $Ed(I_a, I_b)$ is smaller than the distance $Ed(I_a, I_c)$. Then, a recommender system using the nearest neighborhood strategy (like YouTube [16]) would have a higher probability to recommend video I_b than video I_c, if a user is watching video I_a. As such, items with closer proximities have smaller Euclidean distances. Thus, we can minimize the Euclidean distance among similar items to derive item distributional representation. Similar to [38], the Euclidean distance metric aims to minimize its metric loss:

$$\mathcal{L}_d = \sum_{(I_i, I_j)\in\mathbb{C}} \sum_{(I_i, I_k)\notin\mathbb{C}} [m + Ed(I_i, I_j)^2 - Ed(I_i, I_k)^2]_+, \quad (1)$$

where $\mathbb{C}$ is either the sampling dataset $\mathbb{C}_w$ or $\mathbb{C}_i$, I_i is the key item, I_j is any recommended item, and I_k is any item not recommended. $[x]_+ = max(x, 0)$ is hinge loss, and m is the safety margin size. By minimizing $\mathcal{L}_d$, we can capture the distance features among items.

Cosine similarity metric is also widely applied in collaborative filtering. Considering two items I_a and I_b with their vectors being $\boldsymbol{v}_a$ and $\boldsymbol{v}_b$, the cosine similarity measures the cosine of the angle between the two item vectors, calculated by: $Sim(I_a, I_b) = \boldsymbol{v}_a \cdot \boldsymbol{v}_b / (\|\boldsymbol{v}_a\| \times \|\boldsymbol{v}_b\|)$, as shown in Figure 3(a). If users often purchase items I_a and I_b together, the distributional expression of the two items, say $\boldsymbol{v}_a$ and $\boldsymbol{v}_b$, would have closer vector angle, with $Sim(I_a, I_b) \approx 1$. If a user bought item I_a, he/she would receive item I_b as a recommendation. Thus, a similar method to [65] is proposed to minimize the Cosine similarity metric loss:

$$\mathcal{L}_c = \sum_{(I_i, I_j)\in\mathbb{C}} \sum_{(I_i, I_k)\notin\mathbb{C}} (-log\sigma(Sim(I_i, I_j) - Sim(I_i, I_k))), \quad (2)$$

where σ denotes the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$.

4.2.1 ***Our Construction.*** We aim to design a uniform surrogate model, applicable for effectively learning item proximities from all categories of CF recommender systems. However, neither Euclidean distance nor Cosine similarity metric loss can fully leverage affluent information (*e.g.*, rankings and ratings) present in the sampling data ($\mathbb{C}_w$ or $\mathbb{C}_i$). As such, we take advantage of both metric losses and include necessary information for use in constructing the surrogate model. We define six rules in training the surrogate model:

R1. *For each key item I_{key}, the recommended item I_{pos} is closer to I_{key}, than any other item I_{neg} not in the recommended area,* i.e., $Ed(I_{key}, I_{pos}) < Ed(I_{key}, I_{neg}) \wedge Sim(I_{key}, I_{pos}) > Sim(I_{key}, I_{neg})$.

R2. *For each key item I_{key}, the recommended item I_{hrk} with higher ranking is closer to I_{key}, than any lower-ranked item I_{lrk},* i.e., $Ed(I_{key}, I_{hrk}) < Ed(I_{key}, I_{lrk})$.

R3. *For each key item I_{key}, the recommended item I_{hfq} appearing more frequently is closer to I_{key}, than any lower frequency item I_{lfq},* i.e., $Sim(I_{key}, I_{hfq}) > Sim(I_{key}, I_{lfq})$.

R4. *For a recommended item I_{rec}, the higher-rated item I_{hrt} is closer to I_{rec} than any other item I_{neg} not in the recommended area,* i.e., $Sim(I_{hrt}, I_{rec}) > Sim(I_{neg}, I_{rec})$.

R5. *For a recommended item I_{rec}, the lower-rated item I_{lrt} is farther to I_{rec} than any other item I_{neg} not in the recommended area,* i.e., $Sim(I_{lrt}, I_{rec}) < Sim(I_{neg}, I_{rec})$.

R6. *The surrogate model is universal across both Type-I and Type-II websites.*

Notably, Euclidean distance metric loss and Cosine similarity metric loss can only satisfy the rule **R1**, which is not sufficient for use. Hence, according to these rules, the following loss function is defined for our surrogate model, yielding

$$\begin{aligned}\mathcal{L}_s = \sum_{(I_i, I_j)\in\mathbb{C}} \sum_{(I_i, I_k)\notin\mathbb{C}} \hat{r}_i(-log\sigma(Sim(I_i, I_j) - Sim(I_i, I_k)) \\ + w_{i,j}[m + log(\frac{|\mathcal{M}|}{k_{I_i, I_j}}) + Ed(I_i, I_j)^2 - Ed(I_i, I_k)^2]_+),\end{aligned} \quad (3)$$

where $\hat{r}_i$ indicates a Standard Mapping Rating, which is used to normalize the rating for fake user u on item I_i at different websites, to capture the properties of rules **R4** and **R5**. It can be expressed by

$$\hat{r}_i = r_i - (r_{min} + \frac{r_{max} - r_{min}}{2}), \quad (4)$$

where r_i is user's rating on item I_i, r_{min} and r_{max} represent the minimum and maximum ratings, respectively, on the website. For example, in MovieLens, $r_{min} = 0.5$ and $r_{max} = 5$. Notably, there is no ranking information in the sampling $\mathbb{C}_w$ collected from websites like Amazon, Airbnb, Yelp, *etc.*, so we just simply set $r_i = 1$ as it is implicit feedback information. $w_{i,j}$ represents the Approximated Ranking Weight (ARW) for the ranking of item I_j to item I_i, which has been widely used in [38, 81], calculated by:

$$w_{i,j} = log(rank_s(I_i, I_j) + 1), \quad (5)$$

where $rank_s(I_i, I_j)$ is the ranking of I_j to I_i within the surrogate model. Given I_i and I_j, the direct item ranking calculation of them is extremely expensive, so we approximate the ranking by:

$$rank_s(I_i, I_j) = \lfloor \frac{|\mathcal{M}| \times J}{|\mathbb{S}|} \rfloor, \quad (6)$$

where $\mathbb{S}$ is a list of sampled items from the surrogate model, J is the count of all items $I_k \in \mathbb{S}$ satisfying $[Ed(I_i, I_j)^2 - Ed(I_i, I_k)^2]_+ > 0$, $|\mathcal{M}|$ is the total number of unique items in the collected dataset, and k_{I_i, I_j} is the rank of item I_j to item I_i based on the targeted recommender system and collected in the sampling dataset $\mathbb{C}_w$ or $\mathbb{C}_i$. Notably, the use of ARW in Eqn. (5) can achieve the surrogate model rules **R2** and **R3**. Since both Type-I and Type-II layout cases are

taken into accounts in Eqn. (3), the rating information is considered in Eqn. (4), and the ranking information is included in Eqn. (5), our design naturally meets rule **R6**. Figure 3(b) illustrates the relative locations of items of different rankings from our surrogate model.

In the training phase, the negative items I_k in item-pairs $(I_i, I_k) \notin \mathbb{C}$ are randomly sampled from the entire item space. The number of sampled negative items is set to 4 for each key item. The training process terminates after the item distributional expressions become stable. The trained model will represent our constructed surrogate model, which can estimate item proximities in a way similar to the recommender system, enabling us to develop an attack strategy on this model and then apply it directly to the original recommender systems to achieve a similar goal by poisoning the item proximities.

4.2.2 ***Relationship between Our Surrogate Model and the Original Recommender System.*** We further explore the relationships between our surrogate model and the original recommender system. A recommender system explores the user-item relationship from the historical user-item interaction dataset. Each item is encoded into a dense vector called the item embedding in most recommender systems. Usually, item embedding does not hold a specific meaning but captures the relationship to other items. The main difference between recommender algorithms and learning metrics lies in that the former focuses on recommendation accuracy while the latter aims to capture item proximities.

To test the capability of the surrogate model in capturing such relationships, we trained five recommender systems on *ml*-1*m* [60] dataset, including item-based CF (IBCF [54]), matrix factorization-based CF (MF [45], CML [38], and BPR [65]), and graph-based CF (GCF [13]). Next, we use our constructed surrogate model (with learning metric $\mathcal{L}_s$) to learn item relationships from the data collected from each of the recommender system. For comparison, we also take into account another two learning metrics $\mathcal{L}_d$ and $\mathcal{L}_c$, corresponding to Eqns. (1) and (2), respectively. The Pearson correlation values between the item-item similarities from recommender systems and item-item similarities learned by the surrogate model under three metrics are listed in Table 1.

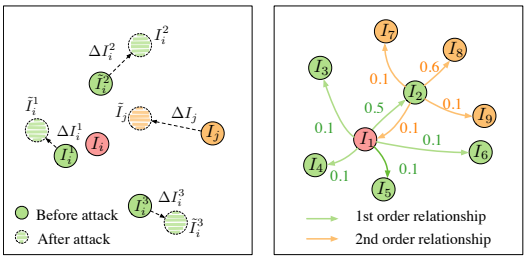
From Table 1, we can observe Pearson correlation values of items similarity between our surrogate model (with $\mathcal{L}_s$) and all five examined recommender system are always more than 0.5. As evidenced in [69], these values (greater than 0.5) can clearly claim that the two systems are highly correlated. In contrast, for the surrogate model with $\mathcal{L}_d$ and $\mathcal{L}_c$, their correlations to five recommender systems are much lower, with only one or two correlation values can reach to 0.5. For example, the surrogate models with $\mathcal{L}_d$ metric and with $\mathcal{L}_c$ metric mostly correlates to *CML* and *BPR*, respectively, both having the correlation values of more than 0.5. The reason is that *CML* encodes item into Euclidean space while *BPR* uses cosine similarity to represent item relationships. But corresponding to other recommender systems, their values are much inferior, as low as 0.19 and 0.20, respectively. These results demonstrate the effectiveness of our constructed surrogate model with $\mathcal{L}_s$ in terms of learning the item proximities.

5 CRAFTING ATTACKING STRATEGY

This section presents both availability attack and target attack on our surrogate model, with the resulting attack strategies directly applicable to original recommender system for similar attack goals. An attacker aims to maximize the profit, leading to (1) the minimum recommendation accuracy for availability attack or (2) maximally promoting target items to the normal users for target attack.

Table 1: Pearson correlation values

	Type-I			Type-II		
	$\mathcal{L}_d$	$\mathcal{L}_c$	$\mathcal{L}_s$	$\mathcal{L}_d$	$\mathcal{L}_c$	$\mathcal{L}_s$
IBCF [54]	0.41	0.46	**0.56**	0.45	0.48	**0.57**
MF[45]	0.32	0.38	**0.48**	0.27	0.35	**0.50**
CML[38]	0.51	0.25	**0.65**	0.53	0.36	**0.68**
BPR[65]	0.19	0.55	**0.56**	0.20	0.57	**0.61**
GCF[13]	0.44	0.46	**0.57**	0.51	0.42	**0.62**



(a) (b)

Figure 4: (a) Attacking item proximities by altering the item distributional representations. (b) Sampling items with first-order relationship or second-order relationship.

5.1 Attack Objective Functions

Availability attack objective function. Given that the goal of availability attack is to demote the original recommendations, our design should be able to measure the discrepancy of recommended results before and after the attack. The profit of an attacker depends on the degree of distortion on recommendations. Instead of directly attacking the original recommender system, we perform our attack on the surrogate model to find the attack strategy that can be for later use. We define $\mathcal{Y}_i = \{I_i^1, \ldots, I_i^K\}$ and $\tilde{\mathcal{Y}}_i = \{\tilde{I}_i^1, \ldots, \tilde{I}_i^K\}$, respectively, as the K most similar items to a targeted item I_i before and after the attack, respectively, on the surrogate model. I_i^k and $\tilde{I}_i^k$ represent the k-th similar item for the item I_i before and after the attack, respectively. The metric of accuracy, *i.e.*, $S(\mathcal{Y}, \tilde{\mathcal{Y}})$, is introduced to measure the discrepancy before and after the attack. Notably, we enable the surrogate model to estimate the K most similar items to an item, so that it is then sufficient to consider only the top-K recommendations.

Denote $Sim(I_i, I_j)$ and $Sim_\Delta(I_i, I_j)$ as the cosine similarity between the items I_i and I_j from the surrogate model before and after the attack, respectively. If the cosine similarity $Sim_\Delta(I_i, I_j)$ between I_i and any item I_j (I_j is not in the I_i's original most K similar items) is larger than $Sim_\Delta(I_i, I_i^k)$ between I_i and its k-th ($k \le K$) similar item I_i^k, *i.e.*, $Sim_\Delta(I_i, I_i^k) - Sim_\Delta(I_i, I_j) < 0$, we say this attack is successful. To poison the item space, we define $S(\mathcal{Y}, \tilde{\mathcal{Y}})$ as:

$$S(\mathcal{Y}, \tilde{\mathcal{Y}}) = \sum_{i=1}^{|\mathcal{M}|} \sum_{j=1}^{|\mathcal{M}|} \sum_{k=1}^{K} \sigma(\Delta_{i,j,ik}(Sim_\Delta(I_i, I_i^k) - Sim_\Delta(I_i, I_j))) ,$$

where $\sigma(\cdot)$ is the sigmoid function and $\Delta_{i,j,ik}$ is a constant which denotes similarity difference before the attack: $\Delta_{i,j,ik} = Sim(I_i, I_i^k) -$

$Sim(I_i, I_j)$. By minimizing $S(\mathcal{Y}, \tilde{\mathcal{Y}})$, we can minimize the recommendation accuracy comparing to that before the attack. We can consider that minimizing $S(\mathcal{Y}, \tilde{\mathcal{Y}})$ is equivalent to deriving new item distributional representations in the surrogate model item space. Assuming the original item distributional representation set is $\boldsymbol{W}$, and the poisoned item distributional representation set is $\tilde{\boldsymbol{W}}$, we have $\tilde{\boldsymbol{v}}_i = \boldsymbol{v}_i + \Delta\boldsymbol{v}_i, \boldsymbol{v}_i \in \boldsymbol{W}, \tilde{\boldsymbol{v}}_i \in \tilde{\boldsymbol{W}}$, as shown in Figure 4(a). Since we do not aim to alter too many original item distributional representations during the attack, as otherwise it may require to inject too many fake operations, we add $\|\tilde{\boldsymbol{W}} - \boldsymbol{W}\|^2$ as a constrained term. We can adopt the Lagrange multiplier method [66] to formulate our attack as follows:

$$\text{OPT-A:} \quad \min_{\tilde{\boldsymbol{W}}} S(\mathcal{Y}, \tilde{\mathcal{Y}}) + \kappa \|\tilde{\boldsymbol{W}} - \boldsymbol{W}\|^2 \,, \tag{7}$$

where κ is the penalty coefficient, setting to 0.01 in our experiments.

Target attack objective function. Denote $\mathcal{T} = \{I_t^1, I_t^2, \ldots, I_t^K\}$ as the K target items that an attacker wishes to promote. We define a metric *successful score*, expressed as $H_{\mathcal{T}}(\cdot)$, to measure the attacker's profit on the degree of success in promoting target items. Specifically, $H_{\mathcal{T}}(\cdot)$ denotes the fraction of normal users whose top-K recommendations include the target items after the attack on the surrogate model. Denote $Sim(I_i, I_t^k)$ as the cosine similarity between any item I_i and its k-th target item before the attack. A successful attack lifts the t-th target item before any other item I_j in I_i's recommendation, say $Sim_\Delta(I_i, I_t^k) > Sim_\Delta(I_i, I_j)$. An attacker aims to promote target items to as many users as possible and promote as many target items as possible for each user. The target attack is modeled as:

$$H_T(\tilde{\mathcal{Y}}) = \sum_{i=1}^{|\mathcal{M}|} \sum_{j=1}^{|\mathcal{M}|} \sum_{k=1}^{K} \sigma(\Delta_{i,j,tk}(Sim_\Delta(I_i, I_t^k) - Sim_\Delta(I_i, I_j))) \,,$$

where I_j can be any item not in the target items set $\mathcal{T}$, and $\Delta_{i,j,tk} = Sim(I_i, I_t^k) - Sim(I_i, I_j)$. By maximizing $H_T(\tilde{\mathcal{Y}})$, we maximize the successful score of the target attack. To constrain the range of item space shifting, the target attack problem OPT-T is formulated as:

$$\text{OPT-T:} \quad \max_{\tilde{\boldsymbol{W}}} H_T(\tilde{\mathcal{Y}}) + \kappa \|\tilde{\boldsymbol{W}} - \boldsymbol{W}\|^2 \,. \tag{8}$$

After solving Eqn. (7) (for OPT-A) or Eqn. (8) (for OPT-T), we get the poisoned item distributional representations $\tilde{\boldsymbol{W}}$, which include a set of reference item vectors, *i.e.*, $\tilde{\boldsymbol{v}}_i \in \tilde{\boldsymbol{W}}$. An attacker needs to operate the items to achieve the attack objective, which lets each distributional vector move towards the reference item vector, *e.g.*, $\boldsymbol{v}_i \rightarrow \tilde{\boldsymbol{v}}_i$. Here, the challenge is how to "move" the item distributional vector toward the reference item vector. We next construct a reference matrix which can help the attacker to achieve this goal.

5.2 Reference Matrix

Reference matrix is an $|\mathcal{M}| \times |\mathcal{M}|$ matrix, with each entry $\mathcal{R}_{ij}$ representing the relationship between item I_i and item I_j.

Generating First-Order Reference Matrix $\boldsymbol{\mathcal{R}}_1$. First-Order Reference Matrix $\boldsymbol{\mathcal{R}}_1$ captures the first-order proximity features among items from the surrogate model. It illustrates how close two items are related to each other, *e.g.* item I_i and item I_j are bought at the same time, or they both get high ratings from the same user. In our surrogate model, the first-order proximities of item I_1 are shown in Figure 4(b) with green arrows. To generate $\boldsymbol{\mathcal{R}}_1$, we start from an arbitrary item I_i as the key item. Then we sample the next key item I_j with probability $p(I_j|I_i) = Sim(I_i, I_j)/\sum_{k=1}^{K} Sim(I_i, I_k)$. Note that the key item is sampled from I_i's K most similar items for accelerating the sampling speed. For each sampled item pair (I_i, I_j), we add 1 to the corresponding entry $\mathcal{R}_{ij}$ in $\boldsymbol{\mathcal{R}}_1$. Next, we use the item I_j as the new key item to continue reference sampling, which stops upon reaching a relatively large number, *e.g.*, $1,000,000$. Each entry $\mathcal{R}_{ij}$ in $\boldsymbol{\mathcal{R}}_1$ can be considered as the number of co-occurring times for items I_i and I_j under first-order proximity.

Generating Second-Order Reference Matrix $\boldsymbol{\mathcal{R}}_2$. This Reference Matrix $\boldsymbol{\mathcal{R}}_2$ captures the second-order proximity feature among items from the surrogate model, *e.g.*, item I_i and item I_k are usually bought together with item I_j, respectively. As shown in Figure 4(b), items I_7, I_8, and I_9 have the second-order proximity with item I_1. To generate $\boldsymbol{\mathcal{R}}_2$, we start from an arbitrary item I_i as the key item. Then, we sample the next key item I_k with probability $p(I_k|I_i, I_j) = Sim(I_i, I_j) \cdot Sim(I_j, I_k)/\sum_{j=1}^{K}\sum_{k=1}^{K}(Sim(I_i, I_j) \cdot Sim(I_j, I_k))$. For each sampled item pair (I_i, I_k), we add 1 to the corresponding entry $\mathcal{R}_{ij}$ in $\boldsymbol{\mathcal{R}}_2$. The steps above repeat until exhausting all item pairs. The entry $\mathcal{R}_{ij}$ in $\boldsymbol{\mathcal{R}}_2$ represents the number of co-occurring times for items I_i and I_j under the second-order proximity.

Our Attack Reference Matrix $\boldsymbol{\mathcal{R}}_{1,2}$. Since different recommender systems have different capabilities to learn the first-order proximity and the second-order proximity, we define the reference matrix $\boldsymbol{\mathcal{R}}_{1,2}$ considering both proximities: $\boldsymbol{\mathcal{R}}_{1,2} \triangleq 1/2\boldsymbol{\mathcal{R}}_1 + 1/2\boldsymbol{\mathcal{R}}_2$.

Relationship Between Surrogate Model and Reference Matrix. Considering the surrogate model expressed by Eqn. (3), its first part $log\sigma(Sim(I_i, I_j) - Sim(I_i, I_k)$ is the well-known pointwise mutual information (PMI) of item I_i and item I_j [50]. It captures the feature of co-occurring times $C_{i,j}$ for items I_i and I_j in the sampling dataset. Reference Matrix estimates the co-occurring times with respect to the item vectors, which is learned by the surrogate model. The second part can be considered as a regularization term that constrains item vectors into a specification space. Thus, for Type-I recommendation, the reference matrix's values signify the co-occurring count $C_{i,j}$ of item pairs. For Type-II recommendation, the values in the reference matrix reflect the weighted co-occurring count $\hat{r}_i C_{i,j}$ of item pairs, with $\hat{r}_i = r_i - (r_{min} + (r_{max} - r_{min})/2)$.

5.3 Complete Attack Solutions

Now, we can summarize our attack solution to poison the item distributional representations by using the reference matrix.

Phase I: *Reproducing items' proximities.* We use the sampling data to train the surrogate model as shown in Section 4.2 and apply this trained model to calculate the similarities $Sim(I_i, I_j)$ between any two items I_i and I_j.

Phase II: *Optimizing OPT-A or OPT-T.* For availability attack, we find the original top-K similar items for each item I_i before performing the attack. Notably, the original top-K similar items for each item I_i do not change during algorithm execution. Since $Sim(I_i, I_i^k)$ for $k = 1, \ldots, K$ are constant, we use the gradient method to minimize Eqn. (7). Then we derive the reference item vectors $\tilde{\boldsymbol{W}}$ by using Stochastic Gradient Descent (SGD)[7] method. Specifically, in each iteration of the SGD algorithm, three items I_i, I_i^k, and I_j are selected, where I_i^k is item I_i's k-th similar item. The optimization

stops when $\tilde{\boldsymbol{W}}$ becomes stable. For target attack, we first randomly select an item I_t^k from the target item set $\mathcal{T}$. Then items I_i and I_j are randomly selected. Since $Sim(I_i, I_j)$ is a constant derived from surrogate model, Stochastic Gradient Ascent (SGA) method is used to maximize Eqn. (8) to derive the reference item vectors.

Phase III: *Generating reference matrix.* We generate the reference matrix $\boldsymbol{\mathcal{R}}_{1,2}$ before attack with respect to $\boldsymbol{W}$. Then we generate reference matrix $\tilde{\boldsymbol{\mathcal{R}}}_{1,2}$ after optimizing OPT-A or OPT-T by using the item reference vector $\tilde{\boldsymbol{W}}$. Since reference matrix estimates the item co-occurrence count, we derive the count differences as $\Delta\boldsymbol{\mathcal{R}}^I_{1,2} = max\{\tilde{\boldsymbol{\mathcal{R}}}_{1,2} - \boldsymbol{\mathcal{R}}_{1,2}, 0\}$ for attacking Type-I websites and $\Delta\boldsymbol{\mathcal{R}}^{II}_{1,2} = \tilde{\boldsymbol{\mathcal{R}}}_{1,2} - \boldsymbol{\mathcal{R}}_{1,2}$ for attacking Type-II websites.

Phase IV: *Crafting fake user strategy.* We craft fake users' behaviors according to $\Delta\boldsymbol{\mathcal{R}}^I_{1,2}$ or $\Delta\boldsymbol{\mathcal{R}}^{II}_{1,2}$. Fake user crafting strategies are different on Type-I and Type-II websites. For Type-I social websites, fake users can only click and purchase items. The strategy for crafting fake users are to find item pairs matching the count change matrix $\Delta\boldsymbol{\mathcal{R}}_{1,2}$. However, finding a perfect match is an NP-hard problem. We use a greedy method to generate fake users. Specifically, we first sample the item pairs from the count changing matrix with the likelihood according to the values. Assuming item pair (I_i, I_j) is sampled, we subtract 1 from the matrix and add the item pair to the attack area of fake user $\hat{u}_1$'s operation list, *i.e.*,

$$\tilde{u} : \{\underbrace{(\text{item item})}_{\text{attack area}} + \underbrace{(\text{item item item})}_{\text{filler area}} + \cdots + \underbrace{(\text{item item})}_{\text{attack area}}\}$$

Then, we randomly sample several popular items and fill them in the filler area until meeting the maximum requirement. In our attack, we set the total length of each user operation list to 50, which is divided into ten segments. Each segment contains one sampled item pair and 3 filler items. Thus, there are a total of 10 item pairs and 30 filler items in each fake user's operation list.

For Type-II websites, fake users can rate items. To simplify the attack strategy, the first step is to sample the item pair (I_i, I_j) according to the absolute value as the likelihood in $\Delta\boldsymbol{\mathcal{R}}^{II}_{1,2}$. We set a default rating value for item I_j as the maximum rating r_{max}. The rating of I_i depends on the sign of $\Delta\boldsymbol{\mathcal{R}}^{II}_{1,2}(I_i, I_j)$. The maximum rating r_{max} is assigned to I_i if $\Delta\boldsymbol{\mathcal{R}}^{II}_{1,2}(I_i, I_j) > 0$; otherwise, the minimum rating r_{min} is assigned to I_i. Next, we subtract $\hat{r}_i$ from the corresponding entry in $\Delta\boldsymbol{\mathcal{R}}^{II}_{1,2}$. For item I_k sampled for a filler area, its rating is assigned as the average rating of I_k on the website. After calculating one fake user's operation list, we continue to calculate other fake users' operation lists until reaching the attack budget, $\sum_{i=1}^{\mathcal{U}} |\tilde{u}_i| \leq \Delta_{max}$.

Phase V: *Injecting fake users.* Finally, we inject fake users on the websites and operate on items according to the strategy derived in Phase IV.

6 EXPERIMENT

We implement our proposed black-box attack and conduct extensive experiments for performance evaluation. Our main goal is twofold. First, we evaluate the capability of our constructed surrogate model in reproducing original recommendations. Second, we perform our crafted availability attack and target attack solutions on a set of CF-based recommender systems to quantify our attack performance.

6.1 Experimental Setup

6.1.1 Dataset. Our experiments are conducted on two types of data, one is collected by us from online websites and the other is from existing real-world datasets, described sequentially as follows.

1) Sampling Data Collection from Online Websites. We employ the Random Walk Collection method (in Section 4.1) to collect sampling trails from three real-world websites, *i.e.*, Airbnb, Amazon, and NetEase Music, and apply the Random Injection Collection method (in Section 4.1) to gather the data from MovieLens. The underlying recommender algorithms of the three websites are all unknown to us. For each sampling trail, the key item is randomly selected. Specifically, in Amazon, we collect 50, 000 unique items from the entire website ($Amazon_R$) and another 50, 000 unique items from *Books* category ($Amazon_B$). In NetEase Music, 10, 000 unique songs are collected. For Airbnb, we collect data from three locations in different scales, *i.e.*, Manhattan ($Airbnb_{MA}$), New York City($Airbnb_{NY}$), and United States ($Airbnb_{US}$), collecting 5, 000, 10, 000, and 50, 000 unique items, respectively. In MovieLens, 2, 000 and 5, 000 unique movies are collected, denoted respectively as $MoiveLens_{2k}$ and $MoiveLens_{5k}$. In this procedure, we strictly follow the rules (regulated by Airbnb [1], Amazon [3], and MovieLens[61]) to gather the data. Since NetEase Music does not provide robot rules for data crawling, we follow similar rules to select songs in each website and record those recommended songs from the "similar songs" section. We set a threshold of 4 requests/min, far below servers' limits in all four platforms. For collecting data in MovieLens, we repeatedly rate different items and collect the results returned from the recommending area. For data privacy consideration, each gathered item is directly hashed to a unique string as its index, without exposing its name or other information.

2) Real-world Datasets. We consider several real-world datasets widely adopted by the literature, as stated below.

- *MovieLens [34] (ml-100k, ml-1m, and ml-20m).* MovieLens dataset collected users' ratings from MovieLens website [59], containing user ID, item ID, ratings and timestamp. Each user has rated at least 20 movies. We randomly sample three different dataset sizes: 100K ratings (*ml-100k*), 1M ratings (*ml-1m*), and 20M ratings (*ml-20m*), for experiments.
- *Netflix [62] (nf).* This is a Netflix Prize Open Competition dataset which contains movies and their rating information from users. The dataset *nf* is sampled from the users, each of which has rated at least 20 movies.
- *Amazon [35] (am-b and am-d).* Amazon dataset contains reviews from Amazon [2] spanning May 1996 - July 2014, with each record containing user ID, item ID, ratings and timestamp. The dataset *am-b* and *am-d* are sampled from the categories of Books and Digital Music, respectively, with each user rating at least 5 items and each item having at least 5 reviews.
- *Twitter [47] (tr).* This dataset includes the entire follower-following topology of Twitter [76] network, collected in 2009. We sample the dataset *tr* by selecting users with more than 20 friends or followers.
- *Google+ [49] (g+).* *g+* is sampled over the users on Google+ with more than 20 friends or followers.
- *AMiner Citation Network [4] (ac).* This is a citation dataset extracted from DBLP, ACM, MAG (Microsoft Academic Graph),

and other sources by AMiner. *ac* is sampled over articles with at least 5 citations.

The numbers of users and items included in each dataset are listed in Table 10 in Appendix A.2. For each of these real-world datasets, we collect 100,000 sampling trails for training our surrogate model.

6.1.2 CF Recommender Algorithms. Each of four categories of CF recommender algorithms takes 1 to 4 algorithms listed below for experiments:

- *(1) Item-based CF: IBCF* [54];
- *(2) Matrix Factorization-based CF:* Singular vector decomposition (*SVD*)[64], Alternating least squares (*ALS*)[40], and Bayesian personalized ranking (*BPR*)[65];
- *(3) Neural Network-based CF:* Neural collaborative filtering (*NCF*)[37], Collaborative metric learning (*CML*)[38], Deep collaborative filtering (*DCF*)[52], and EmbeddingDotBias model in the fast.ai library (*FAST*)[25];
- *(4) Graph-based CF:* Knowledge graph convolution networks (*KGCN*)[79].

6.1.3 Counterpart Methods. For availability attack, since there is no existing work for comparison, we consider a baseline attack counterpart, i.e., $Random_A$: an attacker randomly selects the items to perform operations. If ratings are needed, the attacker generates random ratings centering around the averaged rating in dataset.

For target attack, we consider the following five attack counterparts from the literature. *(1)* $Random_T$ *[48]*: An attacker randomly selects some filler items along with the targeted items to perform operations. If ratings are needed, the attacker generates the highest ratings for target items and random ratings for filler items. *(2) Average [48]*: An attacker performs operations on the target items and a set of filler items sampled based on the item degrees. *(3) Bandwagon [10]*: An attacker operates on both the targeted items and a set of popular items. *(4)* PGA_T *[51]*: An attacker carefully selects a set of items to operate upon for boosting a subset of items. *(5)* $SGLD_T$ *[51]*: An attacker carefully selects a set of items and mimics normal user behaviors to perform attacks. All compared counterparts belong to the white-box attacks with full or partial knowledge. Those requiring the least knowledge for attacking will be presented in Section 6.3. In addition, it should be noted that the counterparts of PGA_T and $SGLD_T$ are designed specifically for attacking MF-based CF with the full knowledge of underlying algorithms, dataset, and item relationship information. In contrast, our attack belongs to the black-box attack, without knowledge about prior information.

6.1.4 Metrics and Settings. We define a precision metric $PRE_S@K$ to measure the performance of our surrogate model in reproducing original recommender systems' outcomes. The metric is defined as the fraction of the top-K recommended items falling into the recommended ones of the original recommender system. That is, $PRE_S@K = \sum_i \frac{|F^K(i) \cap \mathcal{L}_s^K(i)|}{K} / |\mathcal{M}|$, where $|\mathcal{M}|$ is the total number of items, and $F^K(i)$ and $\mathcal{L}_s^K(i)$ indicate the top-K recommended items for item i from the target recommender system and from the surrogate model, respectively. On the other hand, we define two other metrics, precision $PRE@K$ and hit ratio $HR@K$, to measure the performance of our availability attack and target attack, respectively, when attacking the original recommender systems. Specifically, $PRE@K$ represents the fraction of items demoted from users' top-K recommender list, *i.e.*, $PRE@K = \sum_i \frac{|F^K(i) - \tilde{F}^K(i)|}{K} / |\mathcal{M}|$, where $\tilde{F}^K(i)$ denotes the top-K recommended items after the attack. $HR@K$ indicates the fraction of target items promoted to user's top-K recommender list, i.e., $HR@K = \sum_i \frac{|\tilde{F}^K(i) \cap \mathcal{T}|}{K} / |\mathcal{M}|$.

Table 2: Performance of the surrogate model on online data

Type	$PRE_S@K(\%)$	K				
		3	5	10	15	20
Type-I	$Airbnb_{US}$	85.12	83.34	81.25	78.61	65.76
	$Airbnb_{NY}$	86.60	92.76	93.54	93.20	88.46
	$Airbnb_{MA}$	95.21	97.33	97.42	97.25	96.14
	$Amazon_R$	65.30	75.46	73.81	66.53	50.21
	$Amazon_B$	72.15	78.02	80.11	77.26	75.43
	NetEase Music	83.26	81.30	-	-	-
Type-II	$MoiveLens_{2k}$	70.08	71.13	74.56	77.67	79.34
	$MoiveLens_{5k}$	71.26	73.42	78.96	80.01	82.22

In our surrogate model, the vector size for item representation is set to 128. The hyper-parameters for all the targeted recommender systems are taken from original papers or in default settings from the library. All experiments are implemented by one lab computer, with AMD Ryzen 5 5600X CPU and 64 GB DRAM, equipped with one Nvidia GeForce RTX 3080 GPU.

6.2 Performance of Surrogate Model

6.2.1 Recommendation on Online Social Data. We train our surrogate model based on data collected from different online websites. To test its reproductive capability, we randomly select 5,000 unique items in each dataset collected from Airbnb, Amazon, and MoiveLens, and 2,000 unique songs from NetEase Music. For each selected item, we use the trained surrogate model to find its top-K similar items, with K being 3, 5, 10, 15, and 20, respectively. Notably, NetEase Music only supports the top-5 recommendation, so we consider its K to be 3 and 5. We compare the surrogate model's top-K similar items to the top-K recommendations from the original online websites. Table 2 shows the results of $PRE_S@K$.

From this table, we can see our surrogate model performs best on the $Airbnb_{MA}$ with $PRE_S@K$ values always higher than 95% under various K values. Specifically, when $K = 5$ and $K = 10$, our surrogate model achieves the $PRE_S@K$ of 97.33% and 97.42%. We notice that, the surrogate model performs worse on $Airbnb_{US}$ than on $Airbnb_{NY}$ and $Airbnb_{MA}$. The reason is that the portion of collected items over the entire data in $Airbnb_{US}$ is only 7.58%, smaller than those in $Airbnb_{NY}$ and $Airbnb_{MA}$, which are 20% and 25%, respectively. Similarly, our model performs the worst on $Amazon_R$ with the highest and lowest $PRE_S@K$ values of 75.46% and 50.21%, respectively, for $K = 5$ and $K = 20$. The reason is that Amazon includes over 350 million items, but the number of items included in our sampling trails takes only 0.014%. Nonetheless, we see better performance if the surrogate model is learned on just one category of items, $Amazon_B$, with 82.22% of $PRE_S@K$ for $K = 20$. In *NetEase Music*, our model still achieves 83.26% and 81.30% for $K = 3$ and $K = 5$, even though the portion of collected items only equals 0.1%. This is because only popular songs in NetEase Music support the similarity recommendation,

meaning that collected songs are popular and making our model learn the item relationships easier. For sampling data collected from MovieLens, the $PRE_S@K$ are all higher than 70%. Obviously, the surrogate model is seen to perform better on $MoiveLens_{5k}$ than on $MoiveLens_{2k}$. These results show that our surrogate model is effective on learning the item relationship of original recommender systems, without the prior knowledge of both the recommender algorithms and data information. The model enables us to perform effective attacks. It also gives us the insight that the attacker can focus more on each small category of items in websites, to be more practical and effective.

6.2.2 Recommendation on Real-world Datasets. We implement different recommender algorithms, *i.e.*, *IBCF*, *SVD*, *ALS*, *BPR*, *NCF*, *CML*, *DCF*, *KGCN*, and *FAST* discussed in Section 6.1.1 on each real-world dataset, to make item recommendations. The Random Injection Collection method proposed in Section 4.1 is employed to collect the sampling trails from these datasets when running different algorithms. Specifically, in each dataset under one algorithm, we sample 100, 000 trails, each including the top-20 recommended items (or users). Those sampled trails are used to train our surrogate model for recommendation. We evaluate the performance of the surrogate model by considering the top-5 and top-10 recommendations. Table 3 shows the $PRE_S@K$ values of our surrogate model on each dataset when reproducing the results of different recommender algorithms. We observe that the $PRE_S@K$ values of our surrogate model are always more than 70% in all test scenarios, meaning it reproduces at least 70% of original recommendations. In different datasets, the surrogate model has disparate performance in mimicking different recommender algorithms. Specifically, for dataset *ml*-100*k* and *g+*, the surrogate model can best reproduce *BPR* in the top-10 recommendation with the $PRE_S@10$ of 95.12% and 88.21%, respectively. In *ml*-1*m*, *ml*-20*m*, nf, tr, and *ac*, it best mimics *CML*, with $PRE_S@10$ values of 93.17%, 89.26%, 87.05%, 87.65% and 82.46%, respectively. In *am-b* and *am-d*, it best reproduces *IBCF* with $PRE_S@10$ values of 92.75% and 91.84%, respectively. The results demonstrate that our surrogate model is effective in learning the item proximities from the black-box CF recommendations.

When comparing different datasets, the surrogate model performs best on *ml*-100*k* for both top-5 and top-10 recommendations with the averaged $PRE_S@5$ of 90.39% and $PRE_S@10$ of 91.59%, respectively. It performs worse in the *ac* dataset, able to recover 75.79% top-5 recommender results and 76.98% top-10 recommendations in average. The performance discrepancy is caused by the number of items in the dataset. For example, there are 1682 items in *ml*-100*k* while 4, 107, 340 nodes are in *ac*. Our collected sample trails from *ml*-100*k* include almost all items, but from *ac*, they include only a small portion of items. When examining four movie datasets (*i.e.*, *ml*-100*k*, *ml*-1*m*, *ml*-20*m*, and nf), the averaged $PRE_S@10$ are 91.59%, 90.29%, 85.14%, and 81.34%, respectively. We also explore the impacts of various parameters (i.e., the numbers of items and sampling trails), which are deferred to Appendix A.3.

6.3 Attack Performance

We evaluate our attack performance corresponding to different CF recommender algorithms. For ethical consideration, we use merely the real-world datasets to perform the attack under those algorithms. When generating the surrogate model and performing attack, our solutions are in the black-box setting. However, all compared methods perform white-box attacks with full or partial knowledge of recommender algorithms and dataset information. Details about the knowledge requirement of each attack are given in Table 4. Three knowledge requirements are shown: 1) *Item*, which includes item information and item statistics (*e.g.*, average rating and popularity), 2) *Relationship*, which denotes the item-item relationships in the historical dataset, and 3) *Algorithm* which is the specific algorithm used by the targeted recommender system. Notably, $Random_T$, $Average$, and *Bandwagon* require to have item information while PGA_T and $SGLD_T$ require full attack knowledge. In contrast, both our proposed availability and target attacks require none of such knowledge, applicable to various categories of CF algorithms.

For each category of CF algorithms, both the results of availability and target attacks are examined. We define the attack ratio (fr) as the fraction of the injected fake user count over the total number of users in dataset.

Attack Performance on Item-based CF (IBCF). Table 5 shows the averaged *PRE*@10 values of our availability attack and baseline attack $Random_A$ on *IBCF* with 4 examined datasets (*ml*-100*k*, *am-b*, *tr*, and *ac*), when the attack ratio fr increases from 0.1% to 5%. We observe the *PRE*@10 values of our availability attack are always much higher than those from $Random_A$ with the same fr in the same dataset. When fr = 0.1%, the *PRE*@10 results of $Random_A$ are only in the range of 1.07% and 1.76% across all datasets, but our attack achieves results in the range of 5.05% to 9.34%. When fr increases from 0.1% to 5%, the *PRE*@10 values of our availability attack rise much faster than those of $Random_A$. Specifically, *PRE*@10 values of our attack in *ac* dataset increase from 5.05% to 80.16%. Among the four datasets, $Random_A$ performs the best for *am-b*, under which its *PRE*@10 values rise only from 1.23% to 15.33%. Our attack performs the worst for tr dataset, but under which its *PRE*@10 values grow from 6.28% to 50.71%. Our attack always well outperforms its $Random_A$ counterpart under any given dataset. Note that, for fr = 5% on *ml*-100*k*, only 48 fake users are required for *PRE*@10 value to reach 44.28%. The running times of our attack are 43.1s, 1253.6s, 1074.3s, and 652.4s, respectively, for attacking *ml*-100*k*, *am-b*, *tr*, and *ac* datasets.

Table 6 shows the *HR*@10 results of our target attack ($Reverse_T$) and $Random_T$, on *IBCF*, under *ml*-20*m*, *am-b*, and *g+* datasets. For fr = 0.1%, our target attack achieves the *HR*@10 of 4.38%, 6.22%, and 5.33% on three datasets, respectively, compared favorably to those of all other attacks which always yield no more than 2%. The *HR*@10 values of our target attack increase much faster than those of all other attackers, when fr increases from 0.1% to 5%. Specifically, for fr = 5%, the *HR*@10 values of our target attack on three datasets increase to 38.53%, 39.81%, and 46.04%, respectively. However, for $Random_T$, its *HR*@10 values under three datasets increase only to 8.77%, 3.51%, and 3.14%, respectively.

Attack Performance on Matrix Factorization-based CF (SVD, ALS, and BPR). Due to the space limitation, we show our results for only two datasets, *i.e.*, *ml*-100*k* and *am-d*. Figures 5(a) and 5(b)

Table 3: Performance ($PRE_S@K$) of the surrogate model on real-world datasets

$PRE_S@K$	*IBCF*		*SVD*		*ALS*		*BPR*		*NCF*		*CML*		*DCF*		*KGCN*		*FAST*	
K	5	10	5	10	5	10	5	10	5	10	5	10	5	10	5	10	5	10
ml-100k	94.31	94.67	92.56	93.55	92.11	92.70	93.63	**95.12**	91.07	90.62	90.25	93.76	82.19	85.76	88.14	88.40	89.26	89.77
ml-1m	91.07	92.56	90.42	91.57	90.29	93.08	90.35	91.77	90.55	91.24	90.03	**93.17**	81.37	82.18	86.72	87.91	88.42	89.15
ml-20m	84.25	87.64	85.20	86.04	85.46	87.28	82.46	83.31	85.29	85.59	87.45	**89.26**	82.38	82.67	80.04	80.27	83.79	84.21
nf	82.06	85.70	75.97	74.20	78.22	81.60	83.20	80.72	82.01	81.34	85.60	**87.05**	71.52	71.08	81.35	84.65	84.26	85.63
am-b	90.04	**92.75**	85.30	86.06	86.01	83.39	85.55	87.64	83.71	85.54	88.32	89.49	74.32	75.79	80.01	82.35	81.19	82.20
am-d	91.22	**91.84**	80.14	85.06	86.37	86.88	86.78	88.42	86.28	88.36	86.25	87.56	81.25	83.34	78.45	79.23	81.27	83.15
tr	82.69	85.31	80.25	87.34	82.07	83.18	83.05	84.29	82.65	82.99	85.34	**87.65**	76.25	79.57	77.65	78.27	77.12	81.21
g+	87.02	85.47	82.05	84.26	78.45	78.61	85.34	**88.21**	87.12	88.65	87.73	88.43	73.93	74.66	82.35	83.33	82.22	85.39
ac	78.15	79.50	73.66	77.99	75.19	74.82	78.20	78.41	75.13	77.49	81.78	**82.46**	70.27	70.35	74.61	75.82	75.16	76.02

Table 4: Comparisons of knowledge requirements among different attack methods

Attack methods		Knowledge requirements		
		Item	*Relationship*	*Algorithm*
Availability attack	$Random_A$	√	×	×
	$Reverse_A$	×	×	×
Target attack	$Random_T$	√	×	×
	Average	√	×	×
	Bandwagon	√	×	×
	PGA_T	√	√	√
	$SGLD_T$	√	√	√
	$Reverse_T$	×	×	×

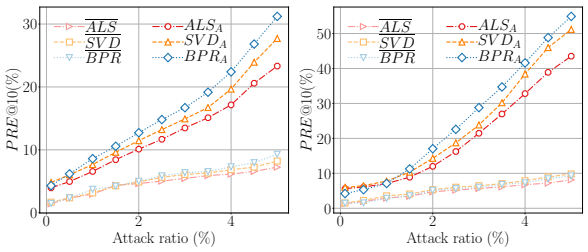
Table 5: *PRE*@10 results of our availability attack $Reverse_A$ and its counterpart $Random_A$ on the Item-based CF (*i.e.*, *IBCF*)

PRE@10(%)		Attack Ratio (%)					
		0.1	0.3	0.5	1.0	3.0	5.0
$Random_A$	*ml-100k*	1.07	2.21	3.56	7.25	9.52	11.07
	am-b	1.23	3.56	6.79	11.80	12.31	15.33
	tr	1.76	2.39	2.54	3.28	4.21	8.64
	ac	1.58	3.96	5.35	6.74	8.56	13.14
$Reverse_A$	*ml-100k*	5.83	7.74	9.31	13.75	22.16	44.28
	am-b	**9.34**	**14.33**	**20.01**	25.65	36.81	63.04
	tr	6.28	7.52	9.49	15.35	24.06	50.71
	ac	5.05	8.94	12.25	**25.80**	**51.36**	**80.16**

Table 6: *HR*@10 results of our target attack and baseline attacks on *IBCF* over three datasets

HR@10(%)		Attack Ratio (%)					
		0.1	0.3	0.5	1.0	3.0	5.0
$Random_T$	*ml-20m*	1.26	1.87	1.95	2.33	6.45	8.77
	am-b	1.55	1.76	1.92	2.05	2.44	3.51
	g+	1.58	1.96	2.35	2.74	2.56	3.14
$Reverse_T$	*ml-20m*	4.38	5.89	7.03	11.72	21.07	**38.52**
	am-b	6.22	8.72	9.21	12.08	16.35	**39.81**
	g+	5.35	6.55	7.87	14.25	27.26	**46.04**

show the *PRE*@10 of our availability attack and $Random_A$ for top-10 recommendations on *ml*-100*k* and *am-d*, where $\overline{SVD}$, $\overline{ALS}$, and $\overline{BPR}$ indicate the results of $Random_A$ attack while SVD_A, ALS_A, and BPR_A indicate our availability attack results respectively under the *SVD*, *ALS*, and *BPR* algorithms. The numerical values corresponding to the figures are also listed in Appendix A.4. We observe that our availability attacks on three algorithms greatly outperform $Random_A$ under both datasets. Specifically, under *ml*-100*k* (or *am-d*), the *PRE*@10 values of our availability attacks on *SVD*, *ALS*, and *BPR* reach 24.34%, 29.33%, 33.26% (or 45.82%, 53.14%, 57.74%), respectively, when fr increases to 5%. In addition, our availability attack performs better even with only 2% fake users injected than $Random_A$ attack with 5% fakes users injected.



(a) *ml*-100*k* **(b)** *am-d*

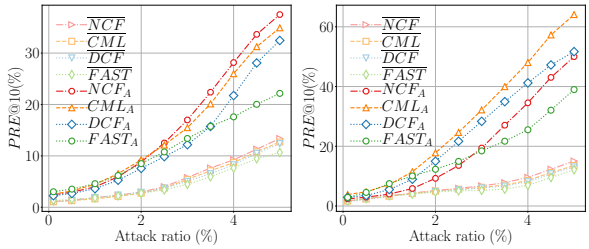
Figure 5: Availability attack on MF-based recommender system with *ml*-100*k* and *am-d* datasets.

We next examine our target attack on *SVD*, *ALS*, and *BPR*, comparing its results with those of $Random_T$, *Average*, *Bandwagon*, PGA_T, and $SGLD_T$ attack methods. Our experiments are still conducted under the *ml*-100*k* and *am-d* datasets for evaluation. Table 7 lists the *HR*@10 results of our target attack and of the other five attack methods under *SVD*, *ALS*, and *BPR* algorithms, when fr increases from 0.1% to 5%. According to those table columns, our attack is seen to perform far better than $Random_T$, *Average*, and *Bandwagon*, for both datasets. It performs a little bit inferior to PGA_T and $SGLD_T$ for both datasets, but notably PGA_T and $SGLD_T$ are of white-box attacks, requiring full knowledge on underlying recommendation algorithms, dataset, and item relationships information as exhibited in Table 4, deemed impractical for real-world attacks. Also, PGA_T and $SGLD_T$ are limited to only attacking the Matrix Factorization-based CF algorithm. In contrast, our attack belongs to the black-box ones, applicable to various categories of CF algorithms.

***Attack Performance on NN-based CF* (i.e., *NCF*, *CML*, *DCF*, *and* *FAST*).** Figures 6(a) and 6(b) depict the results of our availability attack and $Random_A$ for *NCF*, *CML*, *DCF*, and *FAST* under *ml*-1*m* and *am-b*, where $\overline{NCF}$, $\overline{CML}$, $\overline{DCF}$, and $\overline{FAST}$ indicate the results of $Random_A$ whereas NCF_T, CML_T, DCF_T, and $FAST_T$ represent the results of our availability attack. The numerical values corresponding to the two figures are also listed in Appendix A.5. We can see that our availability attacks always beat $Random_A$ under both datasets. Specifically, in *ml*-1*m*, our method achieves the best result (*i.e.*, NCF_A) when attacking the *NCF* algorithm, with the maximum *PRE*@10 value of 39.01%. Here, the required fake users amount is only 302, giving rise to the total number of users to be 6040 as shown in Table 10 in Appendix A.2. In *am-d*, the best result is achieved by CML_A when attacking the *CML* algorithm, with the

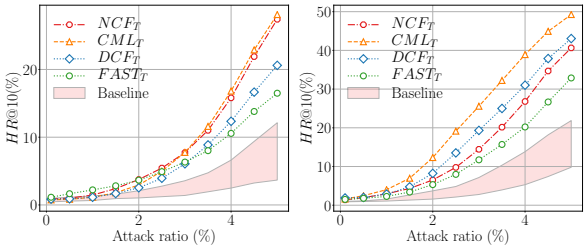
Table 7: Comparison of our target attack $Reverse_T$ and other target attack methods on MF-based CF Algorithms

HR@10(%)		*ALS* Attack Ratio (%)					*SVD* Attack Ratio (%)					*BPR* Attack Ratio (%)				
		0.1	0.5	1.0	3.0	5.0	0.1	0.5	1.0	3.0	5.0	0.1	0.5	1.0	3.0	5.0
ml-100*k*	$Random_T$	0.85	0.91	0.94	1.61	3.85	0.91	0.92	0.96	1.64	3.88	0.93	0.94	0.97	1.70	3.94
	Average	0.94	1.11	1.33	2.21	5.30	0.93	1.08	1.28	2.08	4.87	0.94	1.17	1.41	2.13	5.40
	Bandwagon	0.96	1.27	1.52	2.67	12.42	0.99	1.22	1.38	2.52	11.04	1.08	1.32	1.48	2.76	13.83
	PGA_T	1.51	3.57	4.82	10.62	30.58	1.10	2.76	3.35	9.11	**27.55**	1.22	3.89	6.01	17.03	**36.54**
	$SGLD_T$	1.55	3.76	5.03	10.85	**31.22**	1.78	2.86	4.01	10.53	26.67	1.15	3.71	5.86	15.39	35.21
	$Reverse_T$	1.24	2.01	2.62	8.77	25.42	1.32	2.64	3.40	9.04	25.80	1.82	2.94	4.53	13.65	29.76
am-d	$Random_T$	0.55	0.79	0.94	2.04	8.65	0.53	0.79	0.88	1.98	8.63	0.57	0.76	0.94	2.10	8.67
	Average	0.58	0.84	1.01	2.15	11.35	0.49	0.77	0.89	2.03	11.26	0.64	0.85	1.13	2.24	11.53
	Bandwagon	0.83	0.92	1.25	5.66	20.01	0.79	0.87	1.11	5.32	19.94	0.81	1.00	1.30	5.97	20.29
	PGA_T	1.98	3.71	7.61	21.29	46.35	2.55	5.51	9.06	25.80	46.62	1.99	4.33	8.41	15.05	45.20
	$SGLD_T$	2.13	3.54	7.74	22.36	**47.83**	3.08	5.78	10.08	26.01	**49.62**	1.87	5.22	8.67	16.37	**47.28**
	$Reverse_T$	1.65	2.93	4.30	16.87	35.85	2.04	3.89	5.17	21.65	43.28	1.75	2.70	3.97	13.62	44.60



(a) *ml*-1*m* **(b)** *am-b*

Figure 6: Comparison between our availability attack and $Random_A$ under NN-based CF recommender algorithms with *ml*-1*m* and *am-b* datasets.



(a) *ml*-1*m* **(b)** *am-b*

Figure 7: Target attack on neural network-based recommender system under *ml*-1*m* and *am-b* datasets.

maximum *PRE*@10 value of 66.72%. Our method performs worse on *FAST* for both datasets, but its *PRE*@10 values still reach 23.04% and 42.18%, respectively, for the attack ratio $fr = 5\%$. On the other hand, $Random_A$ achieves the best attack performance (*i.e.*, $\overline{NCF}$) under both datasets when attacking the *NCF* algorithm among all. But its *PRE*@10 values are still far less than those of our attack on all four algorithms.

Figures 7(a) and 7(b) show the results of target attack under *NCF*, *CML*, *DCF*, and *FAST* in *ml*-1*m* and *am-b*, respectively. The respective numerical values can be found in Appendix A.6. All *HR*@10 values of $Random_T$, *Average*, and *Bandwagon* on attacking *NCF*, *CML*, *DCF*, and *FAST* algorithms fall into the red shaded area. Clearly, our target attack beats all three other attacks under both datasets. Our method achieves the best results (*i.e.*, CML_T) when attacking the *CML* algorithm, with the maximum *HR*@10 values of 31.26% and 53.08%, respectively, for the attack ratio $fr = 5\%$. Our target attack performs the worst on *FAST* under both datasets, but it still has the *HR*@10 values of 18.64% and 36.62%, respectively. Other baseline target attacks yeild the best *HR*@10 values under two datasets equal to only 10.54% and 11.37%, respectively, even upon injecting 5% fake users.

Attack Performance on Graph-based CF (i.e., KGCN). We next perform our availability attack and target attack under the *KGCN* with the *ml*-100*k*, *am-d*, and *tr* datasets. We compare our availability attack to $Random_A$ and compare our target attack to $Random_T$, over three datasets. Table 8 lists the attack results with various attack ratios, ranging from 0.1% to 5%. Our availability and target attacks both are seen to significantly outperform other methods, across

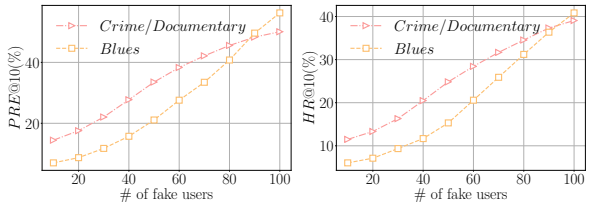
Table 8: The results of our availability attack, target attack, $Random_A$, and $Random_T$ under graph-based CF, *i.e.*, *KGCN*

PRE@10(%)		Attack Ratio (%) 0.1	0.3	0.5	1.0	3.0	5.0
$Random_A$	*ml*-100*k*	1.57	2.30	2.51	3.74	5.05	7.64
	am-d	1.67	2.80	4.55	7.25	11.36	15.27
	tr	1.18	1.94	2.07	3.82	4.72	9.26
$Reverse_A$	*ml*-100*k*	3.40	5.89	7.69	12.54	18.53	25.87
	am-d	5.30	7.26	10.33	17.56	42.08	56.36
	tr	5.36	7.10	10.25	18.50	36.11	49.04
HT@10(%)		Attack Ratio (%) 0.1	0.3	0.5	1.0	3.0	5.0
$Random_T$	*ml*-100*k*	0.66	0.74	1.25	1.34	2.08	5.77
	am-d	0.82	1.01	1.89	2.42	5.30	8.05
	tr	0.93	1.00	1.54	1.97	3.22	5.49
$Reverse_T$	*ml*-100*k*	1.04	3.97	6.72	11.74	17.62	22.78
	am-d	1.27	5.36	10.02	16.30	30.11	38.42
	tr	1.10	3.29	7.82	11.23	15.39	30.60

all datasets. Specifically, with *am-d* under availability attack, our method achieves *PRE*@10 of 5.30%, 7.26%, 10.33%, 17.56%, 42.08%, and 56.36% when r increases from 0.1% to 0.3%, 0.5%, 1%, 3% and 5%, while $Random_A$ only has the *PRE*@10 of 1.67%, 2.80%, 4.55%, 7.25%, 11.36% and 15.27%. For target attack, our attack yields the best *HT*@10 of 38.42% while $Random_T$ only has *HT*@10 of 8.05%.

6.4 Practicality of Our Attacks

The experimental results in Section 6.3 have exhibited the advantage of both proposed availability and target attacks over the compared counterparts on various dataset sizes. In the real-world scenario, it is impractical and rare to attack the entire websites or a large category with many users; instead, an attacker may more likely be interested in a specific subcategory. Hence, we next conduct experiments to show our attack performance on some subcategories,



(a) Availability attack (b) Target attack

Figure 8: Availability attack and target attack on NCF for 'Crime/Documentary' and 'Blues' subcategories.

for validating the practicality of our solutions. We extract two subcategories 'Crime/Documentary' and 'Blues' from the *ml-20m* and *am-d* dataset, respectively. The former subcategory includes 40 items and 858 users whereas the latter one has 3205 items and 6772 users. The recommender system NCF is employed as the underlying algorithm to train the original dataset, *i.e.*, *ml-20m*, and *am-d*, for recommendation. For each subcategory, we take 100, 000 sampling trails, and use the collected data for training our surrogate model and for generating our availability and target attack strategies. The crafted fake users and their operations from our solution will be injected into the original dataset for distorting the recommendation results of the subcategory. Figure 8(a) shows the *PRE*@10 values of our availability attack under the two subcategories. From this figure, we observe that attack performance rises with an increase in the fake user amounts. Specifically, when adding 100 fake users, our attack can achieve the *PRE*@10 values of 50.75% and 58.87%, respectively, on 'Crime/Documentary' and 'Blues' subcategories. Figure 8(b) shows the *HR*@10 of our target attack on the two subcategories. It is seen that when adding 100 fake users, our target attack achieves the *HR*@10 values of 39.65%, and 42.89%, respectively. We also observe that although the 'Blues' subcategory includes more users and items, our attack performance is better under it than under the 'Crime/Documentary' subcategory. The reason is that, the items in *am-d* have less ratings (*i.e.*, 3.14 rating in average) than those in *ml-20m* (*i.e.*, 748 ratings in average), thereby making it easier for the item relationships to be distorted.

To further validate the practicality, we explore 21, 967 subcategories defined by eBay[21] and then browse each subcategory name from two largest E-commerce websites (eBay and Amazon), to see the size of each category. We found that, in eBay, 19.39% of the subcategories contain less than 1000 items and 41.98% of the subcategories contain less than 10, 000 items. In Amazon, 26.91% of subcategories contains less than 1000 items and 76.51% of the subcategories contain less than 10, 000 items. The detailed distribution of all subcategory sizes are shown in Figure 11 of Appendix A.8. Such results evidence subcategories commonly have limited numbers of items in real-world websites.

We further evaluate our attack performance upon the complicated recommender systemsthe and the dynamic recommender systems, with results deferred to Appendix A.9 and Appendix A.10, respectively, due to the page limit.

7 DISCUSSION

Our work presents a novel black-box attack solution to the social websites that employ the CF-based recommender systems, with higher practicality and effectiveness. This section gives some discussions on our work below.

First, the social websites possess large numbers users and items, so it is unrealistic and uncommon to target the entire websites or a large category with many users, for achieving the attack purpose. Instead, an attacker may more likely be interested in the specific subcategories to attack. For performing an effective real-world attack, one can let each attack target a subcategory/group of items (with fewer items as what pervasively exists in social websites detailed in Section 6.4) and then launch multiple parallel attacks to different subcategories simultaneously. This way of attack is more practical and effective.

Second, our data sampling collection strictly follows the social websites' rate limitation, never triggering their incorporated abnormal behavior detection mechanisms. The 'robots.txt' file stipulates the crawler's behaviors, which can only acquire resources that are publicly available from the social websites. Our experiments in Section 6 never trigger any blocking by the social websites. On the other hand, our real attacks are only performed on the local dataset for testing, rather than the actual websites themselves, avoiding from causing spam/malicious behaviors to the real-world. In practice, an attacker my leverage clusters to boost the number of requests to a specific website.

Third, this work aims to demonstrates the plausibility of our black-box attack on a collection of CF-based recommender systems. Definitely, some recommender systems may incorporate certain features extracted from users/items to enhance their recommendation outcomes. Given that the nature of our solution is to first learn the item proximities via a surrogate model, able to capture the complicated patterns from the recommender systems, our proposed attack shall still work on those systems. The strong reproductive capability of our surrogate model on the online social data presented in Section 6.2 has validated that our surrogate model is powerful enough to learn the item proximity regardless of the underlying algorithms. Our experimental results in Section A.9 on attacking the recommender system with features incorporated have validated this point. More exploration is left in our future work.

8 CONCLUSION

This paper has developed a novel black-box poisoning attack to the CF recommender systems embedded in social websites. Without prior knowledge about the recommender algorithm or historical data information, we collected data from social websites and learned their implicit patterns for training a surrogate model, which can reproduce the recommendation functionality of the original recommender system. We then crafted solutions for surrogate model attack before applying them to attack the original recommender systems for similar goals. Extensive experimental results have demonstrated that our proposed solutions are more effective in all four categories of CF recommender algorithms than their counterparts.

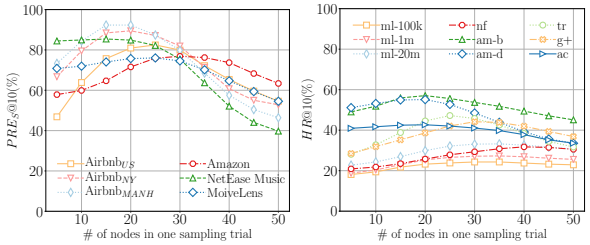
ACKNOWLEDGMENTS

This work was supported in part by NSF under Grants 1763620, 1948374, and 2019511. 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] Airbnb. 2021. robots.txt. https://www.airbnb.com/robots.txt.
[2] Amazon. 2020. Amazon.com. https://www.amazon.com/.
[3] Amazon. 2021. robots.txt. https://www.amazon.com/robots.txt.
[4] Aminer. 2020. Aminer.org. https://aminer.org/citation/.
[5] Frederick Ayala-Gómez, Bálint Daróczy, András Benczúr, Michael Mathioudakis, and Aristides Gionis. 2018. Global citation recommendation using knowledge graphs. *Journal of Intelligent & Fuzzy Systems* 34, 5 (2018), 3089–3100.
[6] Oren Barkan and Noam Koenigstein. 2016. Item2vec: neural item embedding for collaborative filtering. In *Proceedings of the 26th International Workshop on Machine Learning for Signal Processing (MLSP)*. 1–6.
[7] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of the International Conference on Computational Statistics (COMPSTAT)*. 177–186.
[8] John S Breese, David Heckerman, and Carl Kadie. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*. 43–52.
[9] Robin Burke, Bamshad Mobasher, and Runa Bhaumik. 2005. Limited knowledge shilling attacks in collaborative filtering systems. In *Proceedings of 3rd International Workshop on Intelligent Techniques for Web Personalization (ITWP), 19th International Joint Conference on Artificial Intelligence (IJCAI)*. 17–24.
[10] Robin Burke, Bamshad Mobasher, Roman Zabicki, and Runa Bhaumik. 2005. Identifying attack models for secure recommendation. *Beyond Personalization* (2005).
[11] John Canny. 2002. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 238–245.
[12] Dong-Kyu Chae, Jin-Soo Kang, Sang-Wook Kim, and Jung-Tae Lee. 2018. Cfgan: A generic collaborative filtering framework based on generative adversarial networks. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 137–146.
[13] Hsinchun Chen, Xin Li, and Zan Huang. 2005. Link prediction approach to collaborative filtering. In *Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'05)*. 141–142.
[14] Konstantina Christakopoulou and Arindam Banerjee. 2018. Adversarial recommendation: Attack of the learned fake users. (2018).
[15] Konstantina Christakopoulou and Arindam Banerjee. 2019. Adversarial attacks on an oblivious recommender. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 322–330.
[16] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. 191–198.
[17] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. 2010. The YouTube video recommendation system. In *Proceedings of the 4th ACM Conference on Recommender Systems*. 293–296.
[18] Jason V Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S Dhillon. 2007. Information-theoretic metric learning. In *Proceedings of the 24th International Conference on Machine Learning*. 209–216.
[19] Chrysanthos Dellarocas. 2000. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*. 150–157.
[20] Tommaso Di Noia, Daniele Malitesta, and Felice Antonio Merra. 2020. TAaMR: Targeted adversarial attack against multimedia recommender systems. In *Proceedings of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-DSML'20)*.
[21] eBay. 2021. US new category structure. https://ir.ebaystatic.com/pictures/aw/pics/catchanges/2021/May/NTF-update/US_New_Structure_(May2021)_NFT-update.csv.
[22] Wenqi Fan, Tyler Derr, Xiangyu Zhao, Yao Ma, Hui Liu, Jianping Wang, Jiliang Tang, and Qing Li. 2020. Attacking Black-box Recommendations via Copying Cross-domain User Profiles. (2020).
[23] Minghong Fang, Neil Zhenqiang Gong, and Jia Liu. 2020. Influence function based data poisoning attacks to top-n recommender systems. In *Proceedings of The Web Conference*. 3019–3025.
[24] 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)*. 381–392.
[25] fast.ai. 2020. collab. https://docs.fast.ai/collab.html.
[26] Francois Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering* 19, 3 (2007), 355–369.
[27] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Proceedings of the Advances in Neural Information Processing Systems*. 2672–2680.
[28] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. (2014).
[29] Mihajlo Grbovic and Haibin Cheng. 2018. Real-time personalization using embeddings for search ranking at airbnb. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 311–320.
[30] Asnat Greenstein-Messica and Lior Rokach. 2018. Personal price aware multi-seller recommender system: Evidence from eBay. *Knowledge-Based Systems* 150 (2018), 14–26.
[31] Ihsan Gunes, Alper Bilge, and Huseyin Polat. 2013. Shilling attacks against memory-Based privacy-preserving recommendation algorithms. *KSII Transactions on Internet & Information Systems* 7, 5 (2013).
[32] Ihsan Gunes, Cihan Kaleli, Alper Bilge, and Huseyin Polat. 2014. Shilling attacks against recommender systems: A comprehensive survey. *Artificial Intelligence Review* 42, 4 (2014), 767–799.
[33] Larry Hardesty. 2020. The history of Amazon's recommendation algorithm. https://www.amazon.science/the-history-of-amazons\-recommendation-algorithm.
[34] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TIIS)* 5, 4 (2016), 19.
[35] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th International Conference on World Wide Web (WWW)*. 507–517.
[36] Xiangnan He, Zhankui He, Xiaoyu Du, and Tat-Seng Chua. 2018. Adversarial personalized ranking for recommendation. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 355–364.
[37] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*. 173–182.
[38] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*. 193–201.
[39] Rui Hu, Yuanxiong Guo, Miao Pan, and Yanmin Gong. 2019. Targeted poisoning attacks on social recommender systems. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*. 1–6.
[40] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 8th IEEE International Conference on Data Mining*. 263–272.
[41] Hai Huang, Jiaming Mu, Neil Zhenqiang Gong, Qi Li, Bin Liu, and Mingwei Xu. 2021. Data poisoning attacks to deep learning based recommender systems. In *Proceedings of the Annual Network & Distributed System Security Symposium (NDSS)*.
[42] Tao Huang, Huan Zhou, and Kai Zhou. 2017. Food recommender system on Amazon. *San Diego: University of California San Diego* (2017).
[43] Bar Ifrach. 2020. How Airbnb uses machine learning to detect host preferences. https://medium.com/airbnb-engineering/how-airbnb-uses\-machine-learning-to-detect-host-\preferences-18ce07150fa3.
[44] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *IEEE Computer* 8 (2009), 30–37.
[45] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.
[46] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. (2016).
[47] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a social network or a news media?. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*. 591–600.
[48] 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)*. 393–402.
[49] Jure Leskovec and Julian J Mcauley. 2012. Learning to discover social circles in ego networks. In *Proceedings of the Annual Conference on Advances in Neural Information Processing Systems (NIPS)*. 539–547.
[50] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. *Proceedings of the Annual Conference on Advances in Neural Information Processing Systems* 27 (2014), 2177–2185.
[51] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. 2016. Data poisoning attacks on factorization-based collaborative filtering. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*. 1885–1893.
[52] Sheng Li, Jaya Kawale, and Yun Fu. 2015. Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM)*. 811–820.
[53] Chen Lin, Si Chen, Hui Li, Yanghua Xiao, Lianyun Li, and Qian Yang. 2020. Attacking Recommender Systems with Augmented User Profiles. (2020).
[54] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing* 1 (2003), 76–80.
[55] Dong Liu and Wenjun Jiang. 2018. Personalized app recommendation based on hierarchical embedding. In *Proceedings of the IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing &*

Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). 1323–1328.
[56] Tinghuai Ma, Jinjuan Zhou, Meili Tang, Yuan Tian, Abdullah Al-Dhelaan, Mznah Al-Rodhaan, and Sungyoung Lee. 2015. Social network and tag sources based augmenting collaborative recommender system. *IEICE Transactions on Information and Systems* 98, 4 (2015), 902–910.
[57] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Jeff J Sandvig. 2007. Attacks and remedies in collaborative recommendation. *IEEE Intelligent Systems* 22, 3 (2007), 56–63.
[58] 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*. 13–23.
[59] Movielens. 2019. Movielens. https://movielens.org/.
[60] Movielens. 2019. Movielens datasets. https://grouplens.org/datasets/movielens/.
[61] MovieLens. 2021. robots.txt. https://movielens.org/robots.txt.
[62] Netflix.com. 2019. Netflix. https://www.kaggle.com/netflix-inc/netflix-prize-data/.
[63] Michael O'Mahony, Neil Hurley, Nicholas Kushmerick, and Guénolé Silvestre. 2004. Collaborative recommendation: A robustness analysis. *ACM Transactions on Internet Technology (TOIT)* 4, 4 (2004), 344–377.
[64] Arkadiusz Paterek. 2007. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup and Workshop*, Vol. 2007. 5–8.
[65] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).
[66] R Tyrrell Rockafellar. 1993. Lagrange multipliers and optimality. *SIAM Rev.* 35, 2 (1993), 183–238.
[67] Jorge Rodas-Silva, José A Galindo, Jorge García-Gutiérrez, and David Benavides. 2019. RESDEC: online management tool for implementation components selection in software product lines using recommender systems. In *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume B*. 63.
[68] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web (WWW)*. 285–295.
[69] Roei Schuster, Tal Schuster, Yoav Meri, and Vitaly Shmatikov. 2020. Humpty Dumpty: Controlling word meanings via corpus poisoning. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 1295–1313.
[70] 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*. 57–64.
[71] Brent Smith and Greg Linden. 2017. Two decades of recommender systems at Amazon.com. *IEEE Internet Computing* 21, 3 (2017), 12–18.
[72] Junshuai Song, Zhao Li, Zehong Hu, Yucheng Wu, Zhenpeng Li, Jian Li, and Jun Gao. 2020. PoisonRec: An Adaptive Data Poisoning Framework for Attacking Black-box Recommender Systems. In *Proceedings of the IEEE 36th International Conference on Data Engineering (ICDE)*. 157–168.
[73] Zhoubao Sun, Lixin Han, Wenliang Huang, Xueting Wang, Xiaoqin Zeng, Min Wang, and Hong Yan. 2015. Recommender systems based on social networks. *Journal of Systems and Software* 99 (2015), 109–119.
[74] Synced. 2020. China's AI-powered NetEase is music to your ears. https://syncedreview.com/2018/02/14/chinas-netease-music-uses-ai-to-win-hearts/
[75] Jinhui Tang, Xiaoyu Du, Xiangnan He, Fajie Yuan, Qi Tian, and Tat-Seng Chua. 2019. Adversarial training towards robust multimedia recommender system. *IEEE Transactions on Knowledge and Data Engineering* (2019).
[76] Twitter. 2020. Twitter.com. https://twitter.com/.
[77] Chao Wang, Qi Liu, Runze Wu, Enhong Chen, Chuanren Liu, Xunpeng Huang, and Zhenya Huang. 2018. Confidence-aware matrix factorization for recommender systems. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.
[78] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1235–1244.
[79] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. 2019. Knowledge graph convolutional networks for recommender systems. In *Proceedings of the 28th International Conference on World Wide Web (WWW)*. 3307–3313.
[80] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 515–524.
[81] Jason Weston, Samy Bengio, and Nicolas Usunier. 2010. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine Learning* 81, 1 (2010), 21–35.
[82] 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*. 427–430.



(a) Impact on PRE_S@10. **(b) Impact on HR@10.**

Figure 9: Impact of the number of nodes in one sampling trail.

[83] Xingyu Xing, Wei Meng, Dan Doozan, Alex C Snoeren, Nick Feamster, and Wenke Lee. 2013. Take this personally: Pollution attacks on personalized services. In *Proceedings of the 22nd USENIX Security Symposium (USENIX)*. 671–686.
[84] Guolei Yang, Neil Zhenqiang Gong, and Ying Cai. 2017. Fake co-visitation injection attacks to recommender systems. *Proceedings of the Annual Network & Distributed System Security Symposium (NDSS)* (2017).
[85] Hongzhi Yin, Weiqing Wang, Liang Chen, Xingzhong Du, Quoc Viet Hung Nguyen, and Zi Huang. 2018. Mobi-SAGE-RS: A sparse additive generative model-based mobile application recommender system. *Knowledge-Based Systems* 157 (2018), 68–80.
[86] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.
[87] Yonghong Yu, Yang Gao, Hao Wang, and Ruili Wang. 2018. Joint user knowledge and matrix factorization for recommender systems. *World Wide Web* 21, 4 (2018), 1141–1163.
[88] M. Brovman Yuri. 2020. Complementary item recommendations at eBay scale. https://tech.ebayinc.com/engineering/complementary-item\-recommendations-at-ebay-scale/.
[89] Yihe Zhang, Jiadong Lou, Li Chen, Xu Yuan, Jin Li, Tom Johnsten, and Nian-Feng Tzeng. 2020. Towards Poisoning the Neural Collaborative Filtering-Based Recommender Systems. In *European Symposium on Research in Computer Security*. 461–479.
[90] Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiusi Chen, and Jun Gao. 2018. Atrank: An attention-based user behavior modeling framework for recommendation. In *Proceedings of the 31th AAAI Conference on Artificial Intelligence*.

A APPENDIX

A.1 The Indicators of Different Social Websites

Table 9 summarizes the popular websites of different categories where an attacker can apply the two methods for sampling data. It exhibits an attacker could use at least one method to collect sampling data from these popular websites.

A.2 Table 10: Statistics of each dataset

A.3 Impacts of Various Parameters

We further conduct experiments to evaluate the impacts of various parameters involved in the sampling trails collection phase.

Impact of node counts on each sampling trail. We target to the *CML* recommender algorithm and vary the number of nodes on each sampling trail from 5 to 50 in the sampling collection phases. Figure 9(a) shows the performance (*i.e.*, PRE_S@10) of our surrogate model for reproducing *CML* with varying node counts in the sampling trail. The results exhibit PRE_S@10 values increase first and then drop. This indicates that both too many and too few nodes will degrade the performance of surrogate model. The reason

Table 9: The sampling data collection methods that can be adopted by the popular social websites.

	Website	Random Walk Collection	Type-I Recommendation Indicator	Random Injection Collection	Type-II Recommendation Indicator
E-commerce	Amazon	Y	Products related to this item Customers who bought this item also bought Customers who viewed this item also viewed	Y	Gift ideas inspired by your shopping history Inspired by your browsing history
	eBay	Y	Similar sponsored items People who viewed this item also viewed	Y	Sponsored items based on your recent views
	Taobao	Y	Find similar items		
Tourism	Airbnb	Y	More places to stay		
Social network	Twitter	Y	You might like	Y	Who to follow
	Facebook			Y	People You May Know
Entertainment	MovieLens	Y	Similar movies	Y	Top picks
	NetEase Music	Y	Similar songs		
	Netflix	Y	More like this	Y	Main view
	YouTube	Y	Up next	Y	YouTube's homepage
Review	Yelp	Y	You might also consider		

Table 10: Statistics of the dataset

Dataset	*ml*-100*k*	*ml*-1*m*	*ml*-20*m*	*nf*	*am-b*	*am-d*	*tr*	*g+*	*ac*
# users	943	6,040	138,494	480,189	8,026,324	478,235	8,262,545	106,476	4,107,340
# items	1,682	3,706	26,745	17,770	2,330,066	266,414	8,262,545	106,476	4,107,340

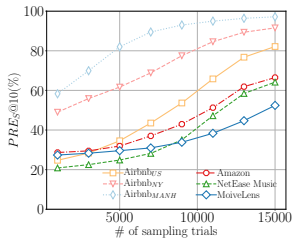


Figure 10: Impact of the number of sampling trails.

is that too few nodes will limit the total number of collected items, which thus cannot help surrogate model to capture the sufficient intrinsic patterns for training. On the other hand, too many nodes will capture more noises and redundant information, which will harm the surrogate model training. Our empirical study shows if the dataset size is unknown, 20 nodes in each trail can derive a satisfactory surrogate model.

We continue to examine the impact of the node counts in each trail on the attack performance. Figure 9(b) shows the results (*i.e.*, *HR*@10) of our target attack on the *CML* with various node counts in the sampling trail. The trending is similar as in Figure 9(a), where the *HR*@10 values first increase and then drop. This set of experiments also confirms that 20 is a suitable choice for the node count in the sampling trail.

Impact of the Number of Sampling Trails. We target to the *IBCF* recommender algorithm and vary the number of sampling trails from 1,000 to 15,000. Figure 10 shows the performance (*i.e.*, $PRE_S@10$) of the surrogate model when varying the number of sampling trails. We can see the performance of surrogate model improves with the increasing of sampling trails amount until it reaches to the stable status. For $Airbnb_{MANH}$, which is the smallest dataset, it reaches to stable status with fewer sampling trails than other datasets. For the two largest datasets, Amazon and NetEase Music, they reach to the stable status slower than others. This implies that for a larger dataset, we should collect more sampling trails to train our surrogate model. That is, the *PRE*@10 and *HT*@10 values fluctuate respectively from 49.6% to 52.2% and from 40.5% to 44.2% only. This experiment demonstrates that our attack is effective to dynamic recommender systems as well.

A.4 Table 11: Availability Attack on Matrix Factorization-based CF Algorithms

A.5 Table 12: Availability Attack on Neural Network-based CF Algorithms

A.6 Table 13: Target Attack on Neural Network-based CF Algorithms

A.7 Table 14: Availability attack and target attack on NCF with 'Crime/Documentary' and 'Blues' subcategories

A.8 Figure 11: Subcategory Size Distributions on eBay and Amazon

A.9 Attack Complicated Recommender Systems

Some CF-based recommender algorithms may incorporate certain features to enhance the recommender systems' recommendation capability, so we conduct experiments to validate our attack performance on such a category of systems. We consider *ml*-1*m* and *am-b* datasets and extract the movie and book genres, respectively, as item features. *DCF* is employed as the underlying recommender algorithm and the item features are incorporated to train the two datasets for recommendations. The results of our availability attack and target attack are presented in Table 15. We observe that when

Table 11: Comparison of our availability attack $Reverse_A$ and other availability attack method $Random_A$ on matrix-factorization based CF Algorithms

PRE@10(%)		*ALS* Attack Ratio (%)						*SVD* Attack Ratio (%)						*BPR* Attack Ratio (%)					
		0.1	0.3	0.5	1.0	3.0	5.0	0.1	0.3	0.5	1.0	3.0	5.0	0.1	0.3	0.5	1.0	3.0	5.0
ml-100*k*	$Random_A$	1.44	1.89	2.33	2.95	5.45	7.24	1.67	1.96	2.38	3.14	6.08	8.22	1.48	1.85	2.37	3.72	6.38	9.29
	$Reverse_A$	3.57	3.69	4.82	6.41	11.42	24.34	4.27	4.88	7.38	3.40	15.08	29.33	3.57	4.01	5.98	9.07	16.34	33.26
am-d	$Random_A$	1.32	0.62	1.95	2.95	5.67	8.15	1.53	0.65	2.24	3.55	6.45	9.82	1.44	0.63	1.55	2.84	6.02	9.28
	$Reverse_A$	5.53	2.08	5.82	7.01	21.54	45.82	5.87	3.25	6.12	7.52	23.87	53.14	3.66	2.35	5.65	5.89	29.94	57.74

Table 12: Comparison of our availability attack $Reverse_A$ and other availability attack method $Random_A$ on neural network based CF Algorithms

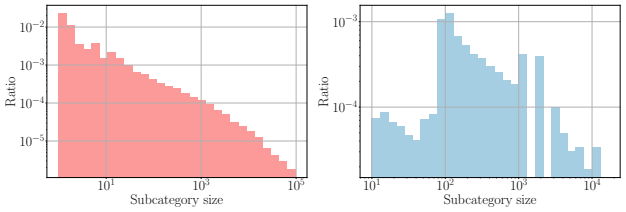
PRE@10(%)		$Random_A$ *NCF*		$Random_A$ *CML*		$Random_A$ *DCF*		$Random_A$ *FAST*		$Reverse_A$ *NCF*		$Reverse_A$ *CML*		$Reverse_A$ *DCF*		$Reverse_A$ *FAST*	
		ml-1*m*	*am-b*	*ml*-1*m*	*am-b*	*ml*-1*m*	*am-b*	*ml*-1*m*	*am-b*	*ml*-1*m*	*am-b*	*ml*-1*m*	*am-b*	*ml*-1*m*	*am-b*	*ml*-1*m*	*am-b*
Attack ratio (%)	0.1	1.00	1.51	1.02	1.42	0.94	0.99	1.23	1.22	2.26	2.19	2.38	3.67	2.12	2.23	2.89	2.37
	0.3	1.18	1.96	1.15	6.41	1.10	2.05	1.32	1.93	2.59	2.65	2.66	3.98	2.27	2.40	4.27	3.12
	0.5	1.34	2.33	1.24	2.27	1.18	2.18	1.55	2.58	2.87	2.82	2.94	4.12	2.52	2.65	3.25	3.83
	1.0	1.87	3.59	1.62	3.33	2.05	3.45	1.75	3.68	5.67	3.89	4.26	6.76	3.42	5.66	4.64	8.02
	3.0	5.82	6.72	5.58	6.34	5.12	6.28	4.34	5.23	16.54	18.09	15.21	32.15	12.11	28.37	13.51	18.89
	5.0	14.32	16.33	13.52	14.52	13.20	14.02	11.10	13.33	39.01	52.87	36.44	66.72	34.18	53.50	23.04	42.18

Table 13: Comparison of our target attack $Reverse_T$ and other target attack method $Random_T$ on neural network based CF Algorithms

PRE@10(%)		$Random_T$ *NCF*		$Random_T$ *CML*		$Random_T$ *DCF*		$Random_T$ *FAST*		$Reverse_T$ *NCF*		$Reverse_T$ *CML*		$Reverse_T$ *DCF*		$Reverse_T$ *FAST*	
		ml-1*m*	*am-b*	*ml*-1*m*	*am-b*	*ml*-1*m*	*am-b*	*ml*-1*m*	*am-b*	*ml*-1*m*	*am-b*	*ml*-1*m*	*am-b*	*ml*-1*m*	*am-b*	*ml*-1*m*	*am-b*
Attack ratio (%)	0.1	0.45	0.87	0.63	1.42	0.77	0.92	0.97	0.99	0.85	1.19	0.72	1.29	0.84	1.43	0.89	1.35
	0.3	0.48	0.91	0.71	6.41	0.80	0.95	0.98	1.01	0.90	1.75	0.79	2.12	0.86	1.60	1.29	1.45
	0.5	0.57	0.96	0.77	2.27	0.82	0.98	1.00	1.05	0.92	1.82	0.81	2.35	0.87	1.65	1.75	1.83
	1.0	0.63	1.05	0.83	3.33	1.01	1.12	1.23	1.22	1.24	2.89	0.95	3.64	0.97	2.66	2.18	3.05
	3.0	1.35	6.72	1.98	2.70	2.11	3.52	2.76	6.84	7.52	14.09	7.46	25.16	5.88	19.37	6.34	11.55
	5.0	3.72	11.37	4.75	10.10	4.98	10.06	10.54	11.26	30.01	42.98	30.40	51.04	22.38	45.07	18.64	36.62

Table 14: Availability attack and target attack on *NCF* with *Crime/Documentary* and *Blues* subcategories

NCF	Fake users									
	10	30	50	70	100	10	30	50	70	100
	PRE@10(%)					*HR*@10(%)				
Crime/Documentary	13.16	21.35	33.95	42.08	50.75	10.75	15.67	25.39	32.07	39.65
Blues	6.43	11.52	20.08	32.45	58.87	5.82	9.87	14.39	25.52	42.89



(a) eBay. **(b) Amazon.**

Figure 11: Distributions of subcategory size on eBay and Amazon.

Table 15: Attack on complicated recommender system

PRE@10(%)	Attack Ratio (%) 0.1	0.3	0.5	1.0	3.0	5.0
ml-1*m*	2.07	2.22	2.36	2.98	9.76	26.35
am-b	2.22	2.27	2.45	4.80	27.33	50.26
HT@10(%)	Attack Ratio (%) 0.1	0.3	0.5	1.0	3.0	5.0
ml-1*m*	1.45	1.86	1.90	2.35	6.74	20.30
am-b	1.37	1.55	1.58	1.76	19.25	43.27

injecting 5% fake users, our availability attack achieves the *PRE*@10 values of 26.35%, and 50.26%, and our target attack achieves the *HR*@10 values of 20.30% and 43.27%, on *ml*-1*m* and *am-b*, respectively. This experiment demonstrates that our attack solutions can retain effectiveness on attacking much complicate recommender

Table 16: Attack on dynamic recommender system

$PRE@10(\%)$	test dataset order									
	1	2	3	4	5	6	7	8	9	10
$Reverse_A$	51.3	50.5	49.6	50.2	50.3	50.7	51.1	50.3	52.2	50.6
$HT@10(\%)$	test dataset order									
	1	2	3	4	5	6	7	8	9	10
$Reverse_T$	41.7	42.6	40.5	44.2	43.3	42.5	40.8	41.7	41.2	40.5

systems, due to strong reproductive capability of our developed surrogate model.

A.10 Attack on Dynamic Recommender Systems

We further evaluate our attack performance upon the dynamic recommender systems in terms of the *PRE*@10 and the *HT*@10 measures. *am-b* dataset is taken for experiments, where the first 426, 006 ratings according to the chronological timestamp order are used as the original training dataset and the next 200, 000 ratings are used as the test dataset. The test dataset is further divided into 10 groups according to the chronological timestamp order. We simulate a dynamic system resulting from adding those 10 test data groups sequentially. *NCF* is employed to continuously train the recommender system with the newly added data and make recommendation for the next data group. We deploy availability attack and target attack at each training phase, with the attack ration set to 5%, to examine our attack performance. Table 16 shows our attack results of the next data group. From the table, we can see that the performance of our availability attack and target attack fluctuate just slightly. That is, the *PRE*@10 and *HT*@10 values fluctuate respectively from 49.6% to 52.2% and from 40.5% to 44.2% only. This experiment demonstrates that our attack is effective to dynamic recommender systems as well.