Adversarial Attacks on an Oblivious Recommender

Konstantina Christakopoulou*
Google Inc., Mountain View, California
chri3275@umn.edu

Arindam Banerjee
University of Minnesota, Twin Cities
banerjee@cs.umn.edu

ABSTRACT

Can machine learning models be easily fooled? Despite the recent surge of interest in *learned adversarial attacks* in other domains, in the context of recommendation systems this question has mainly been answered using *hand-engineered* fake user profiles. This paper attempts to reduce this gap. We provide a formulation for learning to attack a recommender as a repeated general-sum game between two players, i.e., an adversary and a recommender oblivious to the adversary's existence. We consider the challenging case of poisoning attacks, which focus on the training phase of the recommender model. We generate adversarial user profiles targeting subsets of users or items, or generally the top-K recommendation quality. Moreover, we ensure that the adversarial user profiles remain unnoticeable by preserving proximity of the real user rating/interaction distribution to the adversarial fake user distribution. To cope with the challenge of the adversary not having access to the gradient of the recommender's objective with respect to the fake user profiles, we provide a non-trivial algorithm building upon zero-order optimization techniques. We offer a wide range of experiments, instantiating the proposed method for the case of the classic popular approach of a low-rank recommender, and illustrating the extent of the recommender's vulnerability to a variety of adversarial intents. These results can serve as a motivating point for more research into recommender defense strategies against *machine learned* attacks.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

Learned Adversarial Attacks; Recommender Systems

ACM Reference Format:
Konstantina Christakopoulou and Arindam Banerjee. 2019. Adversarial Attacks on an Oblivious Recommender. In *Thirteenth ACM Conference on Recommender Systems (RecSys '19), September 16–20, 2019, Copenhagen, Denmark*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3298689.3347031

1 INTRODUCTION

Machine learning models for recommendation, which help us make our daily decisions (e.g. from which news articles to read, to which products to purchase), are exposed to threats from adversarial parties. Recently, we have seen a plethora of real-life examples where adversaries desire to influence users' beliefs and decisions for their own malicious purposes: fake social media accounts being created to promote news articles about a political ideology; false online product reviews being posted to bias users' opinions favorably or against certain products; and so on. Thus, studying the degree to which machine learning models for recommendation can be manipulated is important. This is a problem well-aligned with the goals of studying machine learned models' robustness to adversarial examples, to ultimately build safer artificial intelligence [23].

To be concrete, an adversarial attack against a recommendation system has the form of injecting a small set of adversarial user profiles, i.e., users who rate/interact with items with some intent, such as promote an item, or reduce the recommendation quality for some users. A significant amount of research has happened studying the problem of robustness of recommender models against adversarial user profiles. However, they have only focused on hand-engineering such adversarial examples — e.g. users rating the target item with a small/large score, and the rest of the items with e.g. normal distributed scores to mimic the true rating distribution [22].

Instead, inspired by pioneering works on adversarial attacks for other domains such as classification [11], our goal is to revisit the question of adversarial attacks on a recommendation system from a *machine learned, optimization perspective.* In attacks for classification settings, the goal is to find the minimal perturbation vector to add to the feature vector of an example so that an oblivious classifier misclassifies the perturbed example. In our setting, we need to find a matrix of fake users×items, so that the distance between the rating/interaction distributions of real and fake users is small, and the adversary's intent is accomplished.

Despite the similarities of the considered setting with adversarial examples in classification, there are important differences/challenges to consider. (1) Recommender models typically rely on the collaborative filtering principle, i.e., similar users tend to rate/interact with items in a similar fashion. On the one hand, this interdependency among users and items might improve robustness, as predictions are not based on individual instances but on various instances jointly. On the other hand, the information propagation among these instances might lead to cascading effects, where attacks on a single instance can influence many others. This coupling of users and items makes attacks on recommendation systems very different from attacks in the classification setting: the latter focus on manipulating an individual instance to enforce its wrong prediction, while in the recommendation setting the adversary has the power to manipulate other users/items at the same time. (2) In the image classification domain, for the adversarial perturbations to be unnoticeable, one enforces a maximum deviation per pixel value. How can we capture the notion of undetectable attack in a recommender system? (3) The recommendation model parameters are learned

*Work performed while at the University of Minnesota, Twin Cities.

RecSys '19, September 16–20, 2019, Copenhagen, Denmark
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6243-6/19/09…$15.00
https://doi.org/10.1145/3298689.3347031

in an iterative fashion over users and items; the model has to be retrained on the manipulated data. Thus, attacks on recommenders are inherently related to the challenging poisoning attacks, where the assumption that the parameters are static as in the classification model is not realistic. (4) How can we overcome that the adversary lacks access to the recommender's gradient with respect to the manipulated data, aka. the fake user profiles? This differs from most attacks on classifiers using the model's gradient for the attack.

Given these challenges, we propose a principled approach for adversarial attacks on recommendation systems. Our method is applicable for any recommendation model — but we only showcase the method's potential for the popular low-rank recommendation models. We assume that the recommender is *oblivious*, i.e., is unaware of the adversary's existence. Moreover, we assume that the adversary has knowledge of the recommender's model and algorithm (so that it can fit similar substitute models on new/fake data), and can only inject a few fake users in the actual recommender.

Overall, our contributions are: *(1) General approach:* We propose a framework for adversarial attacks on oblivious recommenders. Our attacks can inject fake users during the recommender's training, and thus manipulate certain parts of the recommender structure, while ensuring unnoticeable changes by preserving important data characteristics. We introduce new types of attacks where we explicitly target the top-K recommendations; our machine learning approach allows us to optimize general intents, beyond those targeted by the hand-engineered user profiles in the so far considered shilling attacks. *(2) Algorithm:* We devise an algorithm for computing these attacks based on zero-order optimization techniques, to overcome the challenge that the adversary does not have access to the recommender's gradient. This is non-trivial as there is an iterative procedure involved; in order for the adversary to know whether their generated fake users help optimize further their adversarial intent, the adversary needs to evaluate and optimize the recommender's optimization objective with respect to the just generated fake users. *(3) Experiments:* As a striking example of a malicious attack, we illustrate that in order to ruin the predicted scores of a specific item for users who would have loved or hated that item, it suffices to minimize the predicted score of the user with the highest predicted score before the attack, i.e., to target just this single (user, item) entry. Overall, this and other considered attacks highlight the need to handle machine learned attacks on recommenders.

2 RELATED WORK

Adversarial Attacks on Recommender Systems. Attacking a recommender system by injecting fake users (i.e., "shilling attacks") has a long history — e.g., [2, 4, 18, 21, 22]. However, they have focused on hand-engineering user profiles, while we offer an approach to learn *end-to-end* such attacks for a large range of intents.

The closest line of work to ours is the one of *poisoning attacks*, such as [16], [7] optimized for factorized-based collaborative filtering or graph-based recommenders respectively. [27] considers attacks on graphs for node classification, and considers similar challenges as in poisoning attacks, interdependencies among nodes, and the need for unnoticeability. [16] has a close setting to ours, but provides an attack method only specific to the low-rank or nuclear-norm collaborative filtering recommender, using first-order KKT conditions for the gradient computation and a bayesian formulation for detecting fake users, and focuses on a subset of our attacks.

Generating Adversarial Perturbations. While most works have focused on generating adversarial perturbations for evasion attacks, poisoning attacks are far less studied, as they require a bi-level optimization problem that considers learning the model. Almost all works exploit the gradient or other moments of a given differentiable surrogate loss function to guide the search over unnoticeable perturbations [9, 11, 19, 20, 25]. For the recommendation setting, such methods are not applicable, as the adversary does not have access to the gradient. Also, the interdependency among users and items leads to attacks of considerably bigger size than e.g. one-pixel attacks [25], or small-norm perturbation attacks.

The work of [14] is perhaps the most related, as it considers learned adversarial perturbations for recommendation ranking models, but has a distinct difference: it considers perturbations in the learned embeddings (of the popular bayesian personalized ranking model), demonstrates its vulnerability to such perturbations, and offers adversarial learning techniques similar to the classification-based ones to improve robustness. In sharp contrast, we operate on the level of learning entire fake user profiles, which is closer to the real-life setting of adversaries attacking a recommender, and we consider the challenging poisonous attack case, where the model needs to be retrained on both real and fake data.

Generative Adversarial Networks (GANs) in recommendation. Although other works have considered GANs [10] in the context of recommender systems [5, 15, 26], they only use them to augment the data so that the recommendation quality is improved. Our setting is fundamentally different: we simply rely on GANs for the first round of the "game" between the adversary and the recommender, so that the real and fake user distributions are close.

Providing defense strategies against attacks ([8, 14, 23]) and developing robust recommender systems [4, 18] is out of the scope of this paper, so we do not discuss these approaches here.

3 PROBLEM FORMULATION

Preliminaries. We start by introducing the notation, and by describing the setting under consideration. Let Adv denote the model performing the attack on the recommender, and Rec the oblivious recommender under attack. The goal of the recommendation system Rec is to build a model with parameters θ_R to minimize a suitable loss function between true and predicted ratings over all users and items. Let $\mathcal{I}$ be the set of items, and $\mathcal{U}$ the set of real users, present in the recommendation system. We will denote with $m = |\mathcal{I}|$ the number of items and with $n = |\mathcal{U}|$ the number of real users, where $|\cdot|$ is the cardinality of a set. Let $X \in \mathbf{R}^{n \times m}$ denote the matrix of ratings/interactions of real users on the items. Let the adversary Adv have a certain budget of k fake user profiles, where $k \ll n$. Each fake user profile is represented as a m-dimensional vector, i.e., how the user has rated/interacted with the different items in $\mathcal{I}$, with zero values denoting missing ratings/absence of interaction. Adv outputs a matrix $Z \in \mathbf{R}^{k \times m}$, where each row $\mathbf{z}_{i'}$ for $\{i'\}_1^k$ is a fake user profile. The total of real and fake users is denoted by n', where $n' = n + k$.

The goal of the adversary Adv is to generate fake users such that two goals are achieved: **(G1)** the fake users are **indistinguishable** from the real users based on reasonable metrics, (e.g., fake rating/ interaction distributions are similar to real users, eigen-spectrum of the fake user ratings is similar to that of real user ratings, or an anomaly detection method cannot tell real from fake users apart, etc.), and **(G2)** an attack with a suitable **adversarial intent**, (e.g., Rec's predicted scores for a target subset of real users and/or items are pushed down, a target item is removed from the target user's top-K recommendation list, etc.), is successful when the fake users Z generated from Adv are added to the dataset. We will refer to (G1) as the unnoticeability goal, and (G2) as the adversarial intent.

Attack Model Formulation as a Min-Max Game. The attack model can be formulated as a *general-sum game* between two players: the *row* player, i.e., the recommender Rec, and the *column* player, i.e., the adversary Adv. Both players consider a loss function (the negative of a payoff function) they wish to minimize. Let f_R denote the loss function of Rec, and f_A be the loss function of Adv. The recommender Rec, parameterized by θ_R, maps tuples of user, item, rating (u, j, y) to the predicted rating of user u on item j. The actions of Rec include all θ_R (e.g., for low-rank recommenders, each θ_R corresponds to a pair of U, V latent factor matrices). The actions of Adv include all fake user profiles Z.

The way the parameters θ_R, Z are updated is the following. In a repeated game setting, let (θ_R^t, Z^t) be the current parameterizations of the two players. In the next step, the goal of each player is to find optimal parameters θ_R^{t+1} and Z^{t+1} respectively such that their corresponding expected loss is minimized:

$$\theta_R^{t+1} = \underset{\theta_R}{\text{argmin}} f_R(\theta_R, Z^t),\ Z^{t+1} = \underset{Z}{\text{argmin}} f_A(\theta_R^t, Z) \qquad (1)$$

Next, we instantiante the learning procedures for the two players.

4 OBLIVIOUS RECOMMENDER

We assume that the recommender Rec is *oblivious* to the existence of the adversary; hence, it optimizes its loss over the parameters θ_R using *all* given data. Before the attack, the recommender model is learned over only the C_{real} original training user-item-rating tuples $\{u_c, j_c, y_c\}_{c=1}^{C_{\text{real}}}$. During/after the attack, the recommender's model is trained on both the real user-item-score data $\{u_c, j_c, y_c\}_{c=1}^{C_{\text{real}}}$ (represented as a sparse matrix $X \in \mathbf{R}^{n \times m}$) and the C_{fake} non-zero ratings of the k fake user profiles (succinctly represented as a sparse matrix $Z \in \mathbf{R}^{k \times m}$ produced by Adv). This results in an augmented training set of $\{u_c, j_c, y_c\}_{c=1}^{C_{\text{all}}}$, with $C_{\text{all}} = C_{\text{real}} + C_{\text{fake}}$, and corresponding augmented ratings matrix $[X; Z]$ where $[;]$ denotes concatenation of two matrices over the rows. Rec learns a function parameterized by θ_R mapping input tuples $\{u_c, j_c, y_c\}_{c=1}^{C_{\text{all}}}$ to estimated scores $\{\hat{y}_c\}_{c=1}^{C_{\text{all}}}$, so that the loss f_R is minimized over C_{all}:

$$\min_{\theta_R} f_R(\theta_R, Z) = \min_{\theta_R} \frac{1}{C^{\text{all}}} \sum_{c=1}^{C_{\text{all}}} \ell(y_c, \hat{y}_c(u_c, j_c; \theta_R, Z)), \qquad (2)$$

where $\ell(\cdot)$ is a suitable loss between the real and predicted scores.

5 GENERATING ADVERSARIAL USERS

Given an oblivious recommender with parameters θ_R and objective function f_R, our goal is to learn a set of $k \ll n$ fake users, aka. the matrix Z, so that if these users are injected to the recommender, the recommender's performance, measured in some way, drops.

Let the index h specify the target user/item/set: $u_h \in \mathcal{U}$ be the target user, i_h the target item, or $\mathcal{U}_h/\mathcal{I}_h$ the target set of users/items respectively. While in the image/graph node classification setting the goal is to change the target example's predictions, in the recommendation setting, the goal is to hurt the recommendation performance or remove a certain item from the top-K list of recommendations.

Importantly, given the interdependence among users and items (e.g., coupled via the latent factors U, V), the attacker can target a single item/user h, but can influence a whole subset of users-items.

To ensure that the attacker can not modify the learned user-item structure completely, we limit the number of allowed fake users by a budget k, where $k \ll n$.

For the augmented data with fake user profiles, the optimal parameters θ_R^* are learned, matching the poisonous attack setting; thus, we have a bi-level optimization problem.

5.1 Unnoticeable Users

In an adversarial attack scenario, the attackers try to modify the input data such that the changes are *unnoticeable.* Unlike to image data, where this can be easily verified visually and by using simple constraints, in the recommendation setting this is much harder for the following reason: sufficiently large user-item rating/interaction matrices are not suitable for visual inspection, and the structure of users-items is discrete preventing from the use of infinitely small changes. Also, instead of minimally perturbing (e.g., by one-pixel), in the recommendation setting multiple user accounts/profiles are typically injected into the data to achieve the adversarial intent.

How can we ensure unnoticeable attacks in our setting? Only limiting the budget of fake users to $k \ll n$ might not be enough. We want a realistically looking user-item rating/interaction matrix after the injection of these profiles. Thus, we generate fake users preserving specific inherent properties of the real user distribution.

Distribution-preserving adversarial users. Concretely, for every item $j \in \mathcal{I}$, we need the rating/interaction distribution over all fake users $u' \in \mathcal{U}^{\text{fake}}$ denoted as Q^j to be close to the rating/interaction distribution over all real users $u \in \mathcal{U}$, denoted as P^j. Particularly, we want the average distribution distance among these two distributions for all items $j \in \mathcal{I}$ to be minimized, as measured by the Jensen-Shannon divergence:

$$\frac{1}{|\mathcal{I}|} \sum_{j=1}^{|\mathcal{I}|} \frac{1}{2}(D(P^j || \frac{1}{2}(P^j + Q^j) + D(Q^j || \frac{1}{2}(P^j + Q^j)), \qquad (3)$$

where D denotes the Kullback-Leibler divergence.

To find a matrix $Z \in \mathbf{R}^{k \times m}$ whose resulting Q^j distributions are close to the P^j distributions resulting from the $X \in \mathbf{R}^{n \times m}$ real rating/interaction matrix, the framework of generative adversarial networks (GANs) [10] is a great fit. The GANs framework consists of jointly training a pair of networks — a *Discriminator network D* learning to discriminate fake from real samples, and a *Generator network G* learning to generate samples with the goal of fooling the discriminator D to not being able to distinguish them from real. In our case, samples are m-dimensional sparse rating/interaction vectors (user profiles).

In fact, the global minimum of the training criterion of the *Generator* network of GANs is achieved if and only if the Jensen-Shannon divergence between the real and the fake data distributions is minimized; this is exactly Equation (3) which captures our unnoticeability constraint. At convergence, the conditional distribution of the generator G should be able to give fake user samples which a fake user detection module D cannot discriminate from real ones.

This forms the *first stage* of the attacker strategy — train GANs on the real rating/interaction data until convergence; and obtain an initial sample of fake users $Z_1 = Z_{\text{GAN}} \in \mathbf{R}^{k \times m}$, by sampling from the conditional posterior of the generator. This initial sample Z^1 is then used as input to the *second stage*, i.e., that of optimizing the adversarial intent, described in what follows.

We will see in experiments, (Figure 1), that augmenting the data X with Z_{GAN} results in preserving the eigen-spectrum properties. Also, based on visualizations (omitted) we observed that the sparsity structure is preserved (e.g. a fake user cannot rate all items), and the samples comprising Z_{GAN} do not result in a single user being replicated multiple times (which would be detectable).

These observations together suggest that *we can produce fake users which will not be detectable by fraud detection modules* typically relying on distribution properties shifts [4].

5.2 Optimizing the Adversarial Intent

The attacker wants to use the fake user profiles (matrix Z) as a way to achieve its adversarial intent as encoded by f_A. Concretely, the problem of interest to the adversary is:

$$\min_{Z} f_A(Z) \text{ s.t. } Q^Z \sim P^{\text{real}}, \tag{4}$$

where the argument θ_R^t is dropped from the list of f_A's arguments for brevity, Q^Z denotes the distribution constructed by the fake users and P^{real} the distribution constructed by the real users.

f_A is general: It can vary from minimizing the predicted score $\hat{y}(u,h)$ on a target user-item pair (Section 6.3), or the mean predicted score of a target item $\sum_{u \in \mathcal{U}, \notin \text{Ra}(h)} \hat{y}(u,h)$ (6.4), to targeting the user with the maximum predicted score for an item — $\max_{u \in \mathcal{U}, \notin \text{Ra}(h)} \hat{y}(u,h)$ (6.5). It can also encode targeting the modeling of an item group (6.6), or a user group's experience (6.7).

To approximate solving (4), we use an iterative procedure. First, we initialize Z by setting them to the sample of fake users $Z_1 = Z_{\text{GAN}} \in \mathbf{R}^{k \times m}$ we obtained by sampling from the conditional posterior of the converged GAN's generator network G. Then, for iterations $\{t\}_1^T$ we update Z_{t+1} so to optimize f_A:

$$\tilde{Z}_{t+1} = Z_t - \eta \nabla_{Z_t} f_A(Z), \tag{5}$$

where η is the learning rate, and $\nabla_{Z_t} f_A$ is the gradient of the adversarial loss w.r.t the fake users at iteration t Z_t.

However, if we just do that, the gradient descent updates bear the danger of the $Q^{\tilde{Z}_{t+1}}$ fake user samples distribution to move further away from the real one P^{real}. To prevent this, we instead employ a projected gradient descent variant, where $\forall t$, we also do:

$$Z_{t+1} = \Pi_{\text{allowed range}}(\tilde{Z}_{t+1}), \tag{6}$$

where the projection Π is to ensure that the marginals of real and fake users remain close after the descent (e.g., for a star-based recommender, allowed range = [min. # stars, max. # stars].

Challenges in Gradient Computation. Now the question is, how can we compute the gradient $\nabla_{Z_t} f_A$ in (5)? One main challenge which makes the learning of Z for the adversary non-trivial is that there is a two-step process going on: (1) Given some fake ratings, learn θ_R (say, a low-rank model) based on the recommendation system objective f_R, typically using non-convex optimization. (2) Use the learned model parameters of the recommender to evaluate the adversarial objective f_A. As a result, the adversary typically cannot compute the gradient w.r.t. Z.

For illustration purposes, let us consider as adversarial intent: minimize the predicted score of item h over all real users who have not rated/interacted with the item. For the case of a low-rank recommender, this can be encoded as: $\min_{Z_t} \frac{1}{|\{u \notin \text{Ra}(h)\}|} \sum_{u \notin \text{Ra}(h)} \mathbf{u}_u^T \mathbf{v}_h$, where $\mathbf{u}_u$ denotes the u-th row of latent factor U, $\mathbf{v}_h$ denotes the h-th row of latent factor V, and $u \notin \text{Ra}(h)$ denotes the set of users who have not rated item h. At first glance, we can see that the loss is a function of the recommender's parameters, and not Z. But, the parameters of the recommender *are* in fact a function of Z—more generally, f_A is a function of (Z, θ_R^t). After playing fake matrix Z, the adversary player gets to observe the loss *only for this single Z played*, and not the loss it would have incurred had it played other actions/matrices. Thus, the adversary gets limited information, or else *bandit feedback*. Put differently, the gradient of the loss is not directly given for the optimization of f_A over Z.

Approximate Solution. To obtain an *approximation of the gradient*, we build upon zero-order optimization works in bandit optimization [1, 6] and related methods in stochastic and evolutionary optimization [3, 12]. The idea is that if we can only perform query evaluations, to obtain the gradient of $f_A(Z)$, we need to query $f_A(Z)$ at two nearby points: Z_t and $Z_t + \alpha Z_0$, for a small α and a suitable fixed matrix Z_0. Then we can compute the gradient as the directional derivative along the direction Z_0:

$$\nabla f_A(Z_t) = (f_A(Z_t + \alpha Z_0) - f_A(Z_t)) Z_0 / \alpha. \tag{7}$$

Instead of a two-point evaluation, aka. on Z_t and Z_0, we can use K directions and compute the gradient using all K directions. For this, we use a refinement of [1]. For the K directions, we use the K top left and right singular vectors of the fake user matrix at round t Z_t, obtained from a Singular Value Decomposition (SVD) on Z_t: $Z_t = \tilde{B} \Sigma \tilde{C}^T$. Assuming each direction is indexed by h, where $\{h\}_1^K$, let $Z^{(h)}$ be the rank-1 matrices built from each left singular vector $\tilde{b}_h$ and right singular vector $\tilde{c}_h$ of Z_t:

$$Z^{(h)} = \tilde{b}_h \tilde{c}_h^T, \; h = 1, 2, \dots K, \tag{8}$$

where K is the rank of Z_t. Then, using these rank-1 matrices $Z^{(h)}$ as K possible directions, we can compute the matrix gradient:

$$\nabla f_A(Z_t) = \frac{1}{\alpha} \sum_{h=1}^{K} (f_A(Z_t + \alpha Z^{(h)}) - f_A(Z_t)) Z^{(h)} \tag{9}$$

Notice that the computation of the approximate gradient, as defined in (9), involves $K + 1$ evaluations of the function $f_A(Z)$, with K the total number of directions used, or else the rank of Z_t. To make things faster, we use warm-start: we first evaluate $f_A(Z_t)$, then, for any $f_A(Z_t + \alpha Z^{(h)})$, we use the final parameters learned for $f_A(Z_t)$ to warm start the iterates.

Algorithm 2 gives an overview of our overall proposed learning approach for learning to attack an oblivious recommender.

```
Algorithm 1 Constructing the gradient for $\nabla_{Z_t} f_A$
Require: $Z_t$, Objective $f_A$, rank $K$ of $Z_t$
1: Construct the $K$ directions $Z^{(h)}$ by doing singular value decomposition on $Z_t$ (Eq. (8)), and get the top-$K$ rank-1 singular matrices.
2: for each of the $K$ directions $Z^{(h)}$, and the $Z_t$ do
3:     Evaluate $f_A$ on the direction $Z^{(h)}$ or $Z_t$. This entails injecting the fake users $Z^{(h)}$ or $Z_t$ during the recommender's training, and getting a new estimate of the recommender's parameters.
4: end for
5: Output the gradient approximation $\nabla_{Z_t} f_A$ based on the evaluation of $f_A$ on the above $K + 1$ directions using (9).
```

```
Algorithm 2 Learning Algorithm for Adversary's Strategy
1: Get an estimate of the recommender parameters $\theta_R$, optimizing $\min_{\theta_R} f_R(\theta_R)$ using only real data $X$ (this is before the attack).
2: Train a Generator $G$-Discriminator $D$ GANs network using samples of real users from $X$.
3: At GANs' convergence, get $k$ samples from the posterior of the Generator $G$, to form the initial fake user sample $Z^1 \in \mathbf{R}^{k \times m}$. By construction, these samples are hard to discriminate from the real ones, thus satisfying unnoticeability goal(Eq. (3)).
4: for $t = 1, \ldots, T$ do
5:     Get the gradient approximation of $\nabla_{Z_t} f_A$ (Algorithm 1).
6:     Obtain the new fake user matrix $Z^{t+1}$ by projected gradient descent optimizing $f_A$ (thus satisfying adversarial intent (G2)), following equations (5) and (6).
7: end for
8: Output the final fake user matrix $Z_T$ to achieve the intended attack against the oblivious recommender.
```

6 EXPERIMENTS

We design our experiments to study the effectiveness of our proposed Algorithm 2 in adversarial attacks that are (G1) unnoticeable and (G2) optimize adversarial intents.

To demonstrate our results, we will consider the recommender model to be a low-rank one, and the algorithm to optimize for square loss between the predicted and true scores plus a regularization term, with regularization parameter λ [17]. Briefly, let $U \in \mathbf{R}^{n' \times d}$ be the latent factor capturing the latent preferences of users, $V \in \mathbf{R}^{m \times d}$ the item latent factors, and the optimal parameters U^*, V^* are learned by minimizing the loss using alternating minimization:

$$(U^*, V^*) = \arg\min_{U,V} \|[X; Z] - UV^T\|_2^2 + \lambda(\|U\|_2^2 + \|V\|_2^2) . \quad (10)$$

6.1 Can We Learn Realistic User Profiles?

First, we want to evaluate whether the choice of using GANs to generate the initial fake user sample is suitable for achieving the unnoticeability goal before starting to optimize for the adversarial intent; this corresponds to Lines 2 and 3 of Algorithm 2.

Datasets. We used two popular movie recommendation datasets, MovieLens 100K, MovieLens 1M [13] (Table 1). They contain the ratings of users on different movies in the scale $\{0, 1, 2, 3, 4, 5\}$ with 5 the highest, and 0 a lack of rating.

DCGAN Architecture and Parameters. For implementing GANs, we used the popular DCGAN architecture, thanks to its good empirical behavior [24]; in principle though, other GANs-inspired architecture could have been used. Next, we specify the details of the used architecture for reproducibility.

The Discriminator D takes an image of size $H \times W$ (fake or real user sample) and outputs either a 0 or a 1 (is it fake or real?). It consists of four 2D convolutional `CONV` units, with `leaky ReLUs` and batch normalization (`BN`), whose depths are respectively [64, 128, 256, 512], followed by a single-output fully connected (`FC`) unit with sigmoid activation. The Generator G takes as input noise $z \sim \mathcal{N}(0, 100)$ and outputs an $H \times W$ image (the fake user sample). It consists of a `FC` unit of dimension 2×4 (or 7) $\times 512$ with `ReLU` and `BN`, reshaped to a 2, 4 or 7, 512 image, followed by four `transposed CONV` units of depths [256, 128, 64, 1] respectively, each with `ReLU` and `BN`, except for the final with a tanh. We set for the (transposed) `CONV` units the stride to 2, and the kernel size to 5×5.

We set batch size to 64, and run DCGAN for 100 epochs (each epoch does a cyclic pass).

We transformed the datasets' real ratings from $[0, 5]$ to $[-1, 1]$ using $r' = (r - 2.5)/2.5$ so to have them in the same range as the ones produced by the Generator network G. Each user profile is an $|\mathcal{I}|$-d sparse vector, which needs to be transformed to a $H \times W$ 2D array to go through the DCGAN 2D (de-)convolutional units. We set as H the smallest factor of $|\mathcal{I}|$ and as $W = |\mathcal{I}|/H$. This way, each user is viewed as a 2D $H \times W$ image with pixel values the ratings of the user on the different items.

Dataset	# of Items	2D Shape	# of Users
MovieLens 100K	1682	29×58	943
MovieLens 1M	3706	34×109	6040

Table 1: Dataset Statistics.

Note that the goal of this experiment is not to argue that this specific DCGAN architecture, or the certain transformation of user's sparse rating vectors to a 2D array, is better compared to other architectures/ transformations; in fact, we expect that other GAN-based architectures which are not based on convolutions might be a better fit, given that transforming a user 1-d vector to a 2-d array does not necessarily have the neighborhood structure that convolutional layers thrive on. Our goal is rather to provide a proof-of-concept experiment that using some version of GANs can at convergence be useful for generating realistic user samples.

Results. We validate *quantitatively* that the fake user distribution at DCGAN's convergence is close to the real one. We sample 700 fake users from the last epoch's G so that the sizes of the real and fake user distribution are comparable, and we report results averaged over 5 DCGAN runs. We compute the correlation matrix over items of the fake data $Z^T Z$ and of the real data $X^T X$, and compare their **top-10 eigenvalues** (Figure 1, *left*). We also compute **distance metrics** between the real and fake user distributions: For each item j, the real users form a distribution P^j over the rating values $[-1.0, 1.0]$, and the fake users $G(z)$ form a Q^j distribution over $[-1.0, 1.0]$. To measure the distance among these two distributions, we discretize the values to the 6 bins $[-1.0, -0.6, -0.2, 0.2, 0.6, 1.0]$ (corresponding to $[0, 1, 2, 3, 4, 5]$). For each bin we compute the fraction of (real or fake) users who have rated j in this bin out of all (real or fake) users. We report the average over the distance metrics of all items, i.e., mean Jensen-Shannon Divergence (Eq. (3)) in Figure 1, *right*. These results for MovieLens 1M, and for MovieLens 100K

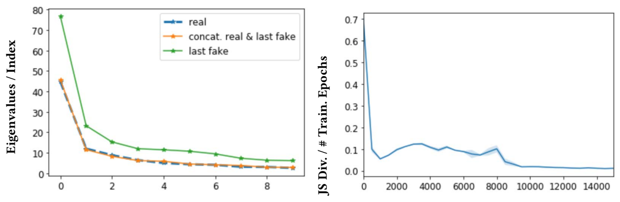


Figure 1: Real-Fake Eigen-spectrum (*left*) & JS Div (*right*). The fake-real rating distributions are close.

(omitted), indicate:

GANs can produce fake user samples whose distribution is close to the real user distribution.

This experiment supports the promise of GANs that the Generator network can learn to generate fake user samples that cannot be discriminated from real ones at convergence; thus, making the choice of GANs a good fit for learning the initial fake user matrix Z^1 which satisfies the unnoticeability goal (G1). Following, in all attack experiments we report Jensen-Shannon divergence as a measure of the unnoticeability goal while the adversarial intent is optimized.

6.2 Experimental Design

We now focus on illustrating the validity of the method for optimizing the adversarial objective f_A while continuing to satisfy the unnoticeability goal. Particularly, after training DCGAN, we perform the Z gradient descent updates (later referred to as *Z-SGD updates*), initialized by a sample from the converged G (Z_{GAN}), transformed from the $[-1, 1]$ to the expected by the recommender range of $[0, 5]$. We set the sample size of Z_{GAN} to 64; these 64 fake users, iteratively optimized during the Z-SGD updates, are only 0.063 fraction of all system users (real and fake) for MovieLens 100K, and 0.01 for MovieLens 1M. The reader can return to Lines 3-10 of Algorithm 2 for recalling the procedure of iterative updates on the fake user matrix Z, initialized by Z_{GAN}.

We use two types of experimental setups:

(E1) `Adv` targets unrated user-item entries (thus entries which are candidates for recommendation).
(E2) `Adv` targets a small subset from the recommender's true (user, item, rating) tuples, held out from `Rec`'s training.

Parameters. The low-rank recommender `Rec` under attack is trained on explicit ratings in the scale $\{0, 1, 2, 3, 4, 5\}$. Unless otherwise specified, we set the latent factor dimension d to 40, λ to 0.001, and train `Rec` before the attack for 10 alternating minimization (*alt-min*) iterations. For the adversary `Adv`, we set the SVD approximation rank K to 30, and α to 0.0001. During a single Z-SGD iteration for each of the $K + 1$ f_A evaluations, 5 alt-min iterations of `Rec` are performed. We use warm-start, i.e., for the $t + 1$ Z-SGD iteration, `Rec`'s parameters are initialized from the ones obtained at the end of the alt-min `Rec` iterations from the previous t Z-SGD step. Every time f_A is evaluated (e.g. during the gradient/ loss computation), Zs are rounded to the closest integers, and get clipped to $[0, 5]$. Also, the projection step of (5) is realized by a box-projection: $Z_{t+1} = \text{clip}(\tilde{Z}_{t+1}, 0, 5)$.

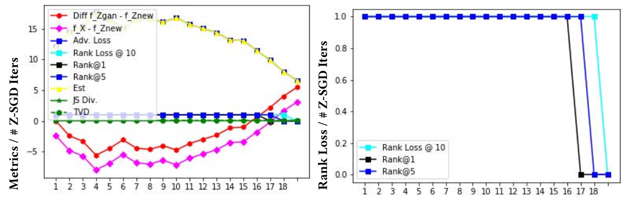


Figure 2: `Adv` successfully targets top item-1062 for user-0.

Metrics. To evaluate the adversary's success, we mainly use the metric of **Attack Difference** denoted by $\Delta(Z)$:

$$\Delta(Z) = f_{\text{before}}(X) - f_A(X; Z) \tag{11}$$

where $f_{\text{before}}(X)$ is the adversary's loss before the attack. $\Delta(Z)$ measures the adversarial loss decrease, and larger values are better.

Each section below introduces a separate attack type, as specified by target user(s), item(s) and the attacker's intent.

6.3 Targeting a User-Item Pair

We start with the adversarial intent: can `Adv` learn realistic users that reduce the predicted score for an *unrated user-item entry*? We adopt the (E1) setup. We set η to 100, α to 50, K to 5, and total of T Z-SGD iterations to 21 or fewer if an early-stopping criterion is satisfied. The adversarial loss $f_A^{(u,h)}$ is the predicted score $\hat{y}(u, h)$ for a target user u and a target item h, for a $(u, h) \notin$ the train set.

We found that for (i) **early-stopping criterion** $\Delta(Z) \geq 1$, when randomly sampling 70 target items from MovieLens 100K, and sampling per item a user who has not rated it, only for 2 out of 70 pairs the attack was not successful, i.e., $\Delta(Z) \leq 0$ (success rate 97.14%). For (ii) early stopping criterion that the **target item does not exist in the top of the recommendation list anymore**—a criterion better aligned with the actual user experience in recommendation—, we randomly sampled target users, and for each user u, we considered as target item h the unrated one with the highest predicted score for u before the attack, i.e., **the top item h of user u**. If we set top=10, out of the 55 sampled users, only for 2 the attack was unsuccessful (item h remained in the top-10)—for the rest, notably, the adversary managed to remove the target item from the target user's top-10 list, while looking realistic (success rate 96.36%).

As an example, Figure 2 shows how various metrics for the movie "A Little Princess" which appeared before the attack in the top-1 of user ID-0, vary as Z-SGD iterations progress. Beyond the attack difference (11), and the distance metrics for measuring unnoticeability (Jensen-Shannon Divergence (3) and Total Variation Distance *TVD*), we report **Rank Loss @ top**(h) = $\mathbb{1}$[item h @ top], for top=$\{1, 5, 10\}$. As Z-SGD updates progress, `Adv` successfully optimizes f_A=$\hat{y}(u, h)$ (yellow, *left*), thus increasing $\Delta(Z)$ (magenta, *left*), and $f_A(X; Z_{\text{GAN}}) - f_A(X; Z)$ (red, *left*). The rank losses @ top-1/5/10 (black/blue/cyan lines, *right*) reach 0 after 17/18/19 Z-SGD iterations. We also see that the real-fake distribution distance metrics ('JS Div', 'TVD', green, *left*) remain close to 0. We find:

`Adv` *successfully targets the top predicted item of a user.*

6.4 Targeting Item's Mean Predicted Score

Here, we examine: can `Adv` target (push down) the mean predicted score of a target item h over *all* real users who have not rated h in the training dataset ($f_A^h = \sum_{u \in \mathcal{U}, \notin \mathrm{Ra}(h)} \hat{y}(u,h)$)?—again, we adopt the (E1) setup. The reason for choosing this target user set is that these are the users for which h can be candidate for recommendation. We keep the same setting as before, except for setting η to 1000 and choosing α in $\{500, 1000\}$, as we found that for this experiment larger values led to larger Δ. The stopping criterion is $\Delta(Z) \geq 1$. We found that for 6 out of 29 random target items from MovieLens 100K, the attack was unsuccessful; $\Delta \leq 0$, i.e., the average score after the attack remained the same or increased—also, out of the 23 successfully targeted items, only for 6 $\Delta(Z) \geq 1$. We conclude that:

Targeting the mean predicted score of an item is hard.

To understand why this is hard, we examine how the distribution of Δ over users $u \notin \mathrm{Ra}(h)$ evolves over the Z-SGD iterations. From Figure 3 we see for the target movie "Mille bolle blu (1993)" (similar behavior is noticed in others too), that although the average difference reached 0.2 (magenta, *left*), every user's Δ follows its own trend (*right*); with mainly the users with the largest or smallest Δ affecting the average Δ. This shows that the fake users cannot move *all* users' scores on h *simultaneously* to the same direction.

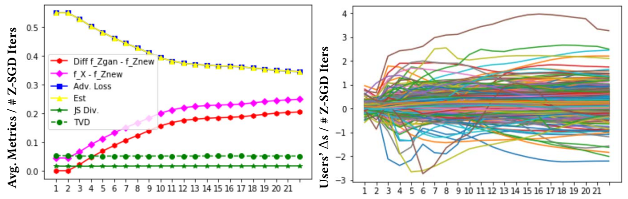


Figure 3: Targeting item ID-1348 avg. predicted score is a hard task—each user's Δ follows its own trend (*right*).

6.5 Targeting the Top User of an Item

In reality, to attack a target item, the adversary does not need to solve the more difficult problem of pushing down *all unrated users*'s score. Instead, they only need to push the score of users who would be good candidates for getting this item in their recommendations—in other words, those with the higher predicted scores from Rec before the attack; the other users would not get h in their recommendations either way. Thus, here we explore the adversary's intent to target the *top user* of an item, i.e., the user from $u \notin \mathrm{Ra}(h)$ with the largest predicted score from Rec before the attack, under the (E1) setup. In Figure 4 as an example we show the results for target item "The Joy Luck Club". We see how the mean $\Delta(Z)$ (y-axis) when considering only the top/bottom predicted users for item h (*left panel*), or the top/ bottom predicted items for user u (*right panel*), changes as we vary the size of top/bottom (x-axis). The results explained in the caption show our **main result**:

When Adv *targets the score of the top predicted user u for item h, then all top-K predicted users for h, and all top-K predicted items for u are targeted as well.*

Same results hold for all other sampled target items.

6.6 Targeting a Group of Items

Next, we focus on attacks targeting an entire *group* of items, in contrast to the so-far presented attacks targeting a single-item. We examine two goals: **(A1)** minimize the mean predicted score over all items in a group, and **(A2)** maximize the prediction error, as measured by mean absolute error, over a group. We adopt the (E2) setup of using a "target set"—Adv, besides making queries to access Rec's predictions, has the added power of targeting some held-out tuples of (user, item, score) invisible to Rec during training. This is in contrast to the (E1) setup where Adv targeted one or a set of *unrated* user-item entries. We used the setup of 80-10-10 split of ratings per user. For Rec, we set d to 100, and λ to 0.1, and we train it for 100 alt-min iterations before the attack. For the adversary Adv, we set η to 1000, K to 5, α to 50, and $T = 30$. We report for the Z-SGD iteration with the best value of f_A in the target set the metric:

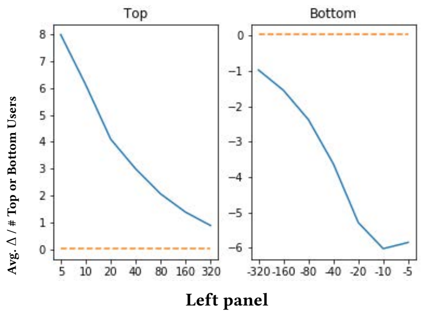
$$\textbf{\% Target Improved} = \frac{\Delta * 100}{f_A^{\text{before}}}, \tag{12}$$

where Δ is given by (11). For **goal (A1)**, defining target item groups based on deciles by predicted scores by Rec before the attack, we see from Figure 5 (*left*) that: Adv is capable of larger % of decrease in the predicted score for the groups with larger original predicted scores (up to 10.9% for the [4.47, 6.53) bin). This is interesting, as these are the entries which would be more likely to appear on users' lists, if the attack did not happen. For **goal (A2)**, we see from Figure 5 (*right*) that for groups based on deciles defined by the target prediction error of Rec before the attack, Adv can do up to 59.3% increase in target prediction error for the well-modeled buckets, i.e., those with [0.02, 0.49) error before the attack.

6.7 Targeting Improved Modeling for a Group

Last, we explore: can Adv achieve targeted *improvement* in the modeling of groups of users or items in a target set? We examine three goals: **(I1)** improve the average recommendation quality of a user group, measured by *Hit Rate@10* (on average per user, is the user's held-out entry included in the top-10?) **(I2)** improve the mean absolute predicted error over an item group, and **(I3)** ensure fair treatment in the modeling of two user groups, i.e., the gap (absolute difference) between their prediction errors is reduced. The setting is the same as above, and the setup is again the (E2) 80-10-10 setup, except that for (I1) a leave-one-out (E2) setup is used. We report the % target improved metric for (I1), (I2), and the target gap for (I3).

Figure 6a focuses on **goal (I1)**—improving the Hit Rates (HRs) for user groups based on deciles by user age, i.e., [7, 20), [20, 23) up to [51, 73). We see that a targeted improvement (red bars) is possible, with the largest % observed for the youngest group. But, these targeted improvements do not transfer to an unseen test set from the same age group (yellow bars). We argue that this happens as the before-the-attack trends of HRs over the age groups in the target and test set differ (Figure 6a, *right*). Figure 6b focuses on **goal (I2)**, and shows the % decrease in prediction error results of item groups defined by # of training ratings per item. We find that groups with the smallest target prediction errors before the attack (annotated on top of the bars), are better targeted. We can see from the yellow bars the % improvements in the test set—Adv results in targeted improvements in an unseen test set, albeit of typically smaller size compared to those in the target set. Although not shown, the original metrics of the target and test set hold similar trends across groups, which might be one reason why the attack generalizes here (further future analysis is needed). Figure



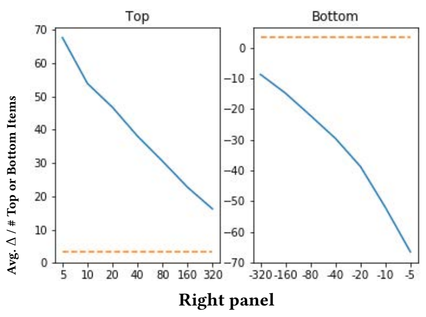
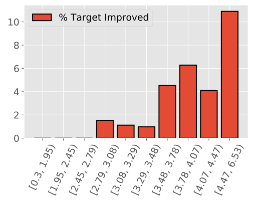


Figure 4: The adversary Adv targets the *top user* u of item h ID-1417 "The Joy Luck Club" (1993). We say *top/bottom* for what was predicted with the highest/lowest score from Rec before the attack. Left: x-axis: varying # users considered for $\Delta(Z)$, from 5 to 320 top users, or from 320 to 5 bottom users, for item h. The mean $\Delta(Z)$ over *all* users $\notin$ Ra(h) is .48 (orange dotted line, y-axis). For the top-5 users $\Delta(Z) = 8$, for the bottom-5 users $\Delta(Z) = -6$ (blue line). Right: x-axis: varying # items from $\notin$ RatedBy(u) for top/bottom items for user u, y-axis: Δ is computed over top/bottom items for u. Gist: Targeting the top user of an item attacks also the top-K users of this item, and the top-K items of this user. Also, the bottom users/items are attacked, as their pred. score is increased ($\Delta(Z) < 0$), which is the opposite from what they would want.



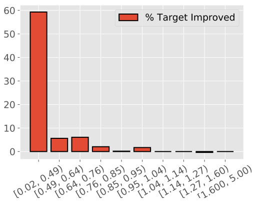
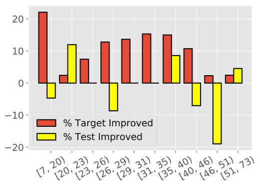
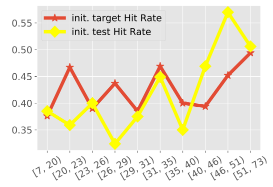
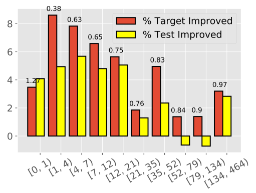


Figure 5: Item Group attack for (A1) (*left, predicted score-based groups*), (A2) (*right, prediction error-based groups*).

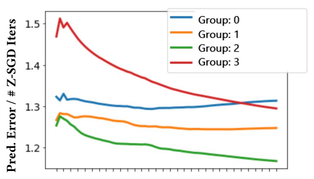




(a) Goal (I1), Age groups



(b) (I2), # ratings item groups



(c) (I3), # ratings user groups

Figure 6: Adv can successfully target groups, and (a) improve users' Hit Rates, (b) decrease items' prediction error, (c) reduce user groups 0 and 3 modeling gap.

6c focuses on **goal (I3)**, targeting the gap between groups 0 and 3, for 25-percentile groups 0, 1, 2, and 3 of users based on # of training ratings per user. We see that the target gap becomes indeed 0 (the point where lines for 0 and 3 cross); similar omitted results hold for groups based on gender, age. Also, we see how targeting 0-3 gap affects the prediction errors of the other groups as well as Z-SGD updates progress—this observation holds for all group-based attacks: targeting a group has an effect of improvement/decrease in the other groups, too. Together the results of this and the previous section serve as proof of concept that:

> *The fake users of* Adv *can affect how* Rec *models user or item groups in a target set.*[1]

7 CONCLUSIONS

We proposed machine-learned adversarial attacks to oblivious recommender systems for various complex adversarial intents. Our attacks target sets of users and/or items, aiming to manipulate the recommendation performance for them. To ensure unnoticeability, we proposed using as a first step a sampled set of fake users generated from the GANs framework, so that the distributional properties of the real users' interactions are preserved, and then using projected gradient descent updates to preserve this. To overcome the challenge of the adversary having no access to the recommender's gradient with respect to the adversarial user profiles, we proposed a non-trivial zero-based optimization method for gradient approximation. Thanks to this approximation, we can optimize a variety of adversarial intents which have not been considered before in the shilling attacks literature. Our experiments highlight the vulnerability of a low-rank recommender to these learned attacks, serving as further motivation for building recommendation models robust to such learned attacks. Future work includes relaxing the adversary's knowledge level, and studying adversary-aware recommenders.

Acknowledgements. The research was supported by NSF grants IIS-1563950, IIS-1447566, IIS-1447574, IIS-1422557, CCF-1451986, CNS-1314560, IIS-0953274, IIS-1029711, NASA grant NNX12AQ39A, and gifts from Adobe, IBM, and Yahoo.

[1]Although omitted, for the last two experiments, the unnoticeability goal measured by (3) is still satisfied.

REFERENCES

[1] Alekh Agarwal, Ofer Dekel, and Lin Xiao. 2010. Optimal Algorithms for Online Convex Optimization with Multi-Point Bandit Feedback. In *COLT*. Citeseer, 28–40.

[2] Charu C Aggarwal. 2016. Attack-resistant recommender systems. In *Recommender Systems*. Springer, 385–410.

[3] Shalabh Bhatnagar, HL Prasad, and LA Prashanth. 2012. *Stochastic recursive algorithms for optimization: simultaneous perturbation methods*. Vol. 434. Springer.

[4] Robin Burke, Michael P Oï£¡Mahony, and Neil J Hurley. 2015. Robust collaborative recommendation. In *Recommender systems handbook*. Springer, 961–995.

[5] 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 *CIKM*. ACM, 137–146.

[6] John C Duchi, Michael I Jordan, Martin J Wainwright, and Andre Wibisono. 2015. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory* 61, 5 (2015), 2788–2806.

[7] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. 2018. Poisoning Attacks to Graph-Based Recommender Systems. In *ACSAC*. ACM, 381–392.

[8] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. 2018. Making machine learning robust against adversarial inputs. *Commun. ACM* 61, 7 (2018), 56–66.

[9] Ian Goodfellow, Nicolas Papernot, Patrick McDaniel, R Feinman, F Faghri, A Matyasko, K Hambardzumyan, YL Juang, A Kurakin, R Sheatsley, et al. 2016. cleverhans v0. 1: an adversarial machine learning library. *arXiv preprint* (2016).

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *NIPS*. 2672–2680.

[11] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[12] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation* 9, 2 (2001), 159–195.

[13] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *TIIS* 5, 4 (2016), 19.

[14] Xiangnan He, Zhankui He, Xiaoyu Du, and Tat-Seng Chua. 2018. Adversarial personalized ranking for recommendation. In *SIGIR*. ACM, 355–364.

[15] Wang-Cheng Kang, Chen Fang, Zhaowen Wang, and Julian McAuley. 2017. Visually-aware fashion recommendation and design with generative image models. In *ICDM*. IEEE, 207–216.

[16] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. 2016. Data poisoning attacks on factorization-based collaborative filtering. In *NIPS*. 1885–1893.

[17] Andriy Mnih and Ruslan R Salakhutdinov. 2008. Probabilistic matrix factorization. In *NIPS*. 1257–1264.

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

[19] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2016. Universal adversarial perturbations. *arXiv preprint arXiv:1610.08401* (2016).

[20] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *CVPR*. 2574–2582.

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

[22] Michael P OâĂŹMahony, Neil J Hurley, and Guenole CM Silvestre. 2002. Promoting recommendations: An attack on collaborative filtering. In *DEXA*. Springer, 494–503.

[23] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. 2016. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814* (2016).

[24] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).

[25] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. 2017. One pixel attack for fooling deep neural networks. *arXiv preprint arXiv:1710.08864* (2017).

[26] 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 *SIGIR*. ACM, 515–524.

[27] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *KDD*. ACM, 2847–2856.