IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

On the feasibility of crawling-based attacks against recommender systems

This is the author's manuscript

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/1874872 since 2022-09-26T10:03:40Z

Published version:

DOI:10.3233/JCS-210041

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

16 August 2024

On the Feasibility of Crawling-based Attacks against Recommender Systems

Fabio Aiolli [a] Mauro Conti [a] Stjepan Picek [b] and Mirko Polato [a,*],
[a] *Department of Mathematics, University of Padova, Padova, Italy*
E-mails: aiolli@math.unipd.it, conti@math.unipd.it, mpolato@math.unipd.it
[b] *Department of Intelligent Systems, Delft University of Technology, Delft, The Netherlands*
E-mail: s.picek@tudelft.nl

Abstract.
Nowadays, online services, like e-commerce or streaming services, provide a personalized user experience through recommender systems. Recommender systems are built upon a vast amount of data about users/items acquired by the services. Such knowledge represents an invaluable resource. However, commonly, part of this knowledge is public and can be easily accessed via the Internet. Unfortunately, that same knowledge can be leveraged by competitors or malicious users. The literature offers a large number of works concerning attacks on recommender systems, but most of them assume that the attacker can easily access the full rating matrix. In practice, this is never the case. The only way to access the rating matrix is by gathering the ratings (e.g., reviews) by crawling the service's website. Crawling a website has a cost in terms of time and resources. What is more, the targeted website can employ defensive measures to detect automatic scraping.

In this paper, we assess the impact of a series of attacks on recommender systems. Our analysis aims to set up the most realistic scenarios considering both the possibilities and the potential attacker's limitations. In particular, we assess the impact of different crawling approaches when attacking a recommendation service. From the collected information, we mount various profile injection attacks. We measure the value of the collected knowledge through the identification of the most similar user/item. Our empirical results show that while crawling can indeed bring knowledge to the attacker (up to 65% of neighborhood reconstruction on a mid-size dataset and up to 90% on a small-size dataset), this will not be enough to mount a successful shilling attack in practice.

Keywords: Recommender Systems, Security, Crawling, Shilling Attack, Collaborative Filtering

1. Introduction

The world's most valuable resource is no longer oil but data. Nowadays, many companies base most of their business on the data they own about users. The companies usually leverage this knowledge to build a user profile which is then used to provide a personalized experience. Advertising, for example, is one of the many applications in which designing good user profiles is crucial [1]. Another example is recommender systems (RSs), i.e., services that help users in finding what they want/like [2–5]. For instance, e-commerce sites like `amazon.com` invest significant resources in building accurate RSs to increase sales [1].

State-of-the-art recommendation algorithms are based on the concept that similar users tend to be interested in similar products, for some notion of similarity. This similarity is often computed based on

*Corresponding author. E-mail: mpolato@math.unipd.it.
[1] https://www.amazon.science/the-history-of-amazons-recommendation-algorithm

0926-227X/$35.00 © 0 – IOS Press and the authors. All rights reserved

the history of the purchases/rates (called interactions in general) of the users with items. This approach is known as collaborative filtering (CF). The system must consider as much historical knowledge about the user as possible to get reliable similarities. Famous and successful companies (e.g., Amazon [5], Pinterest [3], or Netflix [2]) base their recommendation on information about the users-items interaction collected through the years among millions of users.

Although valuable for the companies, some of this information about users is disclosed through the web. Usually, this is done to improve the user experience. For instance, `amazon.com` publishes the users' ratings and comments to allow users to get a better idea about the products. Additionally, other forms of aggregated information may be publicly available, such as the total number of reviews or an item's average rating.

On the other hand, being public, this knowledge can be accessed by anyone who has an Internet connection. Thus, competitors can leverage it to improve their services. Potentially, a competitor can design a mechanism to collect as much public information as possible at almost zero cost and then use such "stolen" knowledge to its advantage. In an ideal case, this scenario could represent a substantial competitive advantage.

In this paper, we investigate whether collecting public information can be considered a real threat. Specifically, we design a straightforward and almost cost-free attack pipeline analyzing in what conditions it can be potentially successful and to which extent. To measure the value of the collected data, we compare how well we can estimate the similarity between users in the target system. We employ two different standard similarity measures [6, 7], namely Pearson's correlation and cosine similarity. Our analysis particularly stresses the data collection phase, which is often overlooked or given for granted by most of the literature about attacks on RSs [8, 9], i.e., they assume knowing the full user-item rating matrix.

The described attack represents only one possible attack to RSs. The profile injection attack (also known as the shilling attack) is undoubtedly the most discussed type of attack in literature [10–12]. As the name suggests, the profile injection attack seeks to mislead the RS by injecting well-crafted fake users into the system. The type of damage provoked by fake user profiles depends on the attacker's goal. There are three common goals: (i) increase the popularity of some targeted items (push attack); (ii) decrease the popularity of some targeted items (nuke attack); (iii) deterioration of the performance of the system. Previous works have shown that the more knowledge used by the attacker, the higher the rate of success [13]. While these concerns are reasonable, the limitation of the attacker must also be considered.

This paper aims to fill the gap between the potential threat and the actual feasibility of a concrete attack to a recommendation service. We first examine different ways to collect public information through web crawling, assessing which strategies are the most efficient and effective. Considering the way items are displayed online, we also propose a crawling strategy, called backlink$_{++}$, that extends the classical backlink and is more effective. Additionally, we study how these crawling strategies behave in the case of performing a shilling attack. In the experiments, we considered a set of standard profile injection attacks, comparing the attacks' success rate when the fake users are crafted based on the crawled information. We also checked whether the fake profiles are easy to detect using standard detection measures. As a target system, we employed a classical k-nearest neighbor recommender [4]. In our research, we assume an attacker with no prior information about the target system. The experimental results show that crawling can allow competitors to gather valuable information, e.g., partially reconstructing the user/item neighborhood, while it is usually not enough to mount an effective shilling attack using standard strategies.

This paper is an extension of a recent work [14]. Specifically, this extended version adds the following contributions:

- We added a new shilling attack, i.e., the sampling attack. This attack showed very different behaviors with respect to the other considered attacks, which led to new interesting considerations;
- We included three new datasets in our analysis, namely `Filmtrust`, `BookCrossing` and `Goodreads`. These datasets have been chosen for their characteristics in terms of dimension, distribution, and density. The inclusion of smaller datasets (e.g., Filmtrust and BookCrossing) showed that some of the studied attacks may work in practice, still with some limitations;
- We improved several aspects of the experimentation procedure: (i) all the previous experiments have been applied to the new datasets, (ii) we improve the analysis on the success rate of shilling attacks, and (iii) we added more details and results about the shilling attack detection.

The rest of the paper is structured as follows. Section 2 presents the notation we use and crawling approaches. Section 3 describes the related work underlining the main differences with our analysis. Section 4 describes the methodology and assumptions used in our analysis. Section 5 describes the details of the datasets we used and presents our experimental results, along with a thorough discussion. Finally, Section 6 wraps up the main results of the paper with some insights about possible future research directions.

2. Background

In this section, we summarize the notation (Section 2.1) and the background knowledge on crawling (Section 2.2) used throughout the paper.

2.1. Notation

We refer to the set of users of an RS with $\mathcal{U}$, where $|\mathcal{U}| = n$. Similarly, the set of items is denoted by $\mathcal{I}$, such that $|\mathcal{I}| = m$. The set of ratings is denoted by $\mathcal{R} \equiv \{(u,i,r) \mid u \in \mathcal{U} \wedge i \in \mathcal{I}, u \text{ rated } i \text{ with rating } r\}$. We add a subscription to both user and item sets to indicate, respectively, the set of items rated by a user u ($\mathcal{I}_u$), and the set of users who rated the item i ($\mathcal{U}_i$). We refer to the rating matrix with $\mathbf{R} \in \mathbb{R}^{n \times m}$ such that $\mathbf{r}_{ui}$ is the rating given by u to i. Finally, with $G(\mathcal{U},\mathcal{I},\mathcal{R})$, we indicate the weighted bipartite graph representing the rating matrix. Nodes are users and items, while the edges (between users and items) are weighted by the rating. A graph G is said to be directed if edges have a direction, undirected otherwise. Given a node in a directed graph, we will call in-degree the number of entering edges, while out-degree the number of outgoing edges. When clear from the context, we simply use the letter G to indicate the graph.

2.2. Crawling a Recommendation-based Website

Personalized recommender systems are usually offered by online services, such as e-commerce (e.g., Amazon), streaming services (e.g., Netflix, Spotify), or social networks (e.g., Facebook, Instagram). The information about users, items, and ratings own by these services are usually (partially) publicly available. For example, in amazon.com, we can browse through the products' pages and see the users' reviews. It is also possible to visit the user page and check his/her previous public reviews. This allows a malicious user to automatically collect (for example, via a crawling bot) such information to design

an attack against the recommendation service. Today's online services are aware of this concern, and they defend their websites against automatic crawling. The most gentle countermeasure is responding with a control web page to check whether the requests come from a human or a machine. These control pages usually contain a captcha-based query [15], or tasks that are very simple for humans but "hard" for a bot[2]. Other, more severe countermeasures are temporary IP blacklisting, or in the extreme case, an indefinite ban of the IP address [16].

On the other hand, attackers can try to circumvent such defenses by using, e.g., VPN, proxy, and TOR[3]. Still, modern online services are nowadays equipped to fight against such strategies [16]. For these reasons, crawling a (large) website completely can be expensive or even infeasible. Thus, an attacker has to rely on incomplete information collected through a crawling process. Given this restriction, the crawling strategy must be as effective as possible, minimizing the number of requests (and, generally, the crawling cost) while maximizing the amount of collected knowledge. A direct way of minimizing such cost is to design a crawling algorithm that maximizes the collected information while making as few requests as possible. This optimization problem can be cast into a well-known computational problem, i.e., the Online Graph Exploration Problem. The Online Graph Exploration Problem (OGEP) considers visiting all graph nodes and returning to the starting node with the minimum total traverse cost. The main issue in this problem is that only the already visited sub-graph is known. Hence only "local" decisions can be made. It is worth noticing that in the OGEP, there are constraints that do not apply to the problem at hand. While crawling an RS's website, we are not obliged to follow a path, i.e., we can jump from a node (web page) to another even if they are not directly linked. Moreover, we do not have to go back to the starting node. We can further assume that each item (e.g., web page) contains links to all the users who rated it and vice-versa. So, the graph at hand is an undirected (bipartite) graph.

Nonetheless, maximizing the knowledge does not simply mean collecting as much data as possible but gathering the most informative data for the attacking purposes. This further adds complexity to the crawling task.

2.2.1. Crawling Strategies

In the most general case, the crawling problem has already been studied by researchers in the context of search engines [17]. Even though the final aim is different, the optimization problem is the same. This problem can be seen as an unconstrained Travel Salesman Problem with incomplete information (i.e., partial knowledge of the graph). Thus, it is safe to assume that it is NP-hard. However, there are heuristic-based algorithms that allow crawling the graph efficiently. In particular, Cho et al. proposed the following crawling strategies [17]:

- **random**: the algorithm randomly chooses its next node from the known (but unseen) set of nodes;
- **random$_=$**: this strategy is similar to the random but, it first flips a coin to decide whether to pick a user's node or an item's node and then picks uniformly from the selected set of known but unseen nodes. This strategy aims at avoiding biases towards the most numerous set between users and items;
- **breadth-first**: the algorithm chooses its next node according to the First In, First Out (FIFO) policy, i.e., when a new node is visited, its neighbors are added to the queue (without a specific order);
- **backlink**: the algorithm chooses the (unvisited) node with the highest in-degree according to the known graph. In the case of the undirected graph, in-degree and out-degree are the same;

[2] https://www.w3.org/TR/turingtest/
[3] https://www.torproject.org/

- **PageRank** [18]: the algorithm chooses the (unvisited) node with the highest PageRank score according to the known graph.

3. Related Works

In this section, we discuss the related works regarding web crawling and the shilling attack.

3.1. Crawling

Crawling the web is almost as old as the World Wide Web itself [17, 19–21]. The automatic algorithm that systematically browses the World Wide Web is called a web crawler, or spider/spiderbot. Search engines have been the first technology to rely on such an algorithm to index the web. The term crawler comes from the first search engine: the "WebCrawler". Since their first appearance, many efforts have been devoted to increasing the crawling procedure's efficiency and effectiveness [22–24]. Focused web crawling [25] is one of the main strategies to improve the crawling quality in specific contexts. Focused web crawling is a procedure that collects Web pages satisfying some specific properties by prioritizing the so-called crawl frontier. The crawl frontier is the set (usually implemented as a queue) of known but not yet visited web pages. However, it is not always easy or feasible to define properties that can help focus the crawling. Since our analysis focuses only on the graph's structure, one of the most promising properties is the PageRank [18] of a page. Unfortunately, PageRank is useful when computed on a complete graph, while on a partially known graph is not so accurate [26, 27].

Crawling algorithms have also been influenced by artificial intelligence (AI). iRobot [28] has been one of the first crawler-based on AI technologies. iRobot uses clustering combined with a Prim-like algorithm [29] to reconstruct the sitemap and select optimal traversal paths. The optimal traversal path is defined in terms of the informative content of the pages. Other examples of "intelligent" crawlers are ACHE [30] and HIFI [31]. Our analysis focuses on classical crawling strategies because of their universality and fast (and thus cheap) implementation.

3.2. Shilling Attack

The profile injection attack (also known as the shilling attack) is by far the most popular attack against recommender systems [11, 12, 32]. A shilling attack consists of the injection of well-crafted fake users into the system. The goal of the attacker is usually one of the following: (i) increase the popularity, expressed in terms of rating or number of ratings, of some targeted items (push attack); (ii) decrease the popularity/rating of some targeted items (nuke attack); (iii) deterioration of the performance of the system, i.e., the recommendation engine is "fooled" by the fake users. For simplicity, in this paper, we focus on the push attack, but all the final considerations also apply to the nuke attack. For a literature review on shilling attacks, we refer interested readers to [11, 12, 32]. Besides the standard shilling attacks, there are also attacks designed for specific types of recommendation methods, such as [33] for graph-based models, [34] for memory-based models, and [35] for factorization-based models. However, the details behind a recommendation engine are usually unknown, which cripples the applicability of the approaches mentioned above. More recently, sophisticated attacks like [8] and [9] assume knowing the whole rating matrix, which is, in most cases, not realistic. Knowing the rating matrix would mean having access to the full purchases/rating history of a system, which cannot be assumed in a general use case. For all the above considerations and to generalize to the most reasonable scenario, we will

consider standard shilling attacks on systems for which we do not know the rating matrix. Details about the considered attacks are reported in Section 4.2.

The literature also offers studies about the effectiveness of shilling attacks under different constraints or scenarios. Burke et al. made an analysis related to the one we propose in this paper [13]. In their study, the attacker has limited knowledge about a target user. Our results confirm some of the drawn conclusions in [13]. Still, our analysis is broader and with a different goal. Moreover, we also cover a new attack scenario, which includes a potential competitor aiming to take advantage of the target system's collected knowledge. In [36], a cost-benefit analysis about a shilling attack is performed. However, the only conclusion the authors draw is that the higher the number of available items in the catalog, the higher the attack cost. Nevertheless, some of our conclusions support their results. It is worth mentioning that we have taken into account the considerations made in [36] when we designed the experiment. The size of the attacks, which is directly related to the cost, has been calibrated to mimic a real-world case. Deldjoo et al. studied the attack's effectiveness on different groups of users (more/less active) [37]. They had quite different results on the two tested datasets, namely Movielens and Yelp. Still, they found that BPR-MF [38] seems to be more resistant than the other tested recommendation approaches.

Finally, there is a large body of research devoted to studying methods to detect shilling attacks. Our analysis does not directly focus on profile injection detection. However, we also conducted some experiments about the detectability of the performed attacks. For a comprehensive study on the detection traits of shilling attacks, we refer the reader to [39].

4. Methodology

We will consider two main attack scenarios that can threaten a recommender system. These two attacks are independent of each other, but both rely on the information gathered through a starting crawling phase (Section 4.1). The two considered type of attacks are:

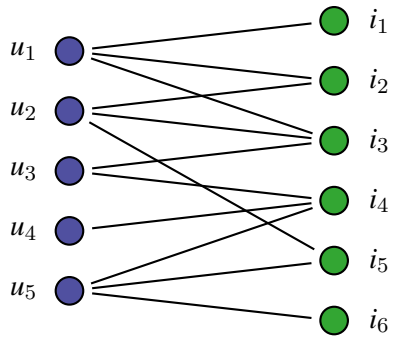
shilling attack (Section 4.2) the standard profile injection attack. The attacker crafts the fake profiles exclusively using the crawled information. The attacker aims to promote (push) or demote (nuke) a specific target item;

neighborhood reconstruction (Section 4.3) the attacker aims at collecting valuable information about the system. The collected information may be valuable for the attacker to improve its own business or simply study its competitors. We assess the informative content of the crawled data by reconstructing the neighborhood of a target node. The higher the overlap w.r.t. the actual neighborhood (computed with the complete knowledge of the graph), the more effective the crawling process. Besides the quality, we also assess the quantity of information each crawling strategy can collect.

4.1. Crawling

Besides the classical crawling techniques described in Section 2, we propose a variation of the *backlink* strategy. In this new strategy, called **backlink**$_{++}$, the known degree for a node is the actual degree in the full graph. Backlink$_{++}$ aims to take advantage of the additional information about the graph structure, i.e., the actual out-degree, provided by the targeted website. Although it might be impractical in the general case, there are many e-service websites where the (public) degree information is accessible without visiting the page corresponding to the node. For example, in the `booking.com` search page, the number of reviews (i.e., the out-degree of the item node) is reported before visiting the item page.

Fig. 1. Example from `booking.com` of the availability of the (full) *public* degree information (the red box) before requesting the item webpage.



ideal : i_4, u_3, u_5, i_3, u_1 (5)
breadth : $i_4, u_3, u_4, u_5, i_3, i_5, i_6, u_1$ (8)
backlink : $i_4, u_3, i_3, u_4, u_1, i_1, u_5$ (7)
backlink$_{++}$: $i_4, u_5, i_5, u_2, i_3, u_1$ (6)
PageRank : $i_4, u_3, i_3, u_4, u_1, i_1, u_5$ (7)

Fig. 2. Example of exploration strategies on a bipartite (recommendation) graph. All the explorations start from the target node i_4. Inside the parenthesis $(\cdot)$ we report the number of a node visited by the strategy. Note that in the graph, we omitted the weight of the edges.

Figure 1 shows an example from booking.com. The same considerations can be done for other services like `amazon.com` or `ebay.com`.

Figure 2 shows an example of the application of all the strategies mentioned in Section 2, including backlink$_{++}$, on a small bipartite graph. The crawling strategies are applied to crawl the entire graph. Note that the algorithms stop when all the graph is known, but it is not required to visit every single node. It is enough to discover the structure (and possibly weights) of the graph without visiting all of them for our purposes.

The crawling phase for collecting the rating information is performed starting from a target node (either a user or an item). Figure 2 shows an example of crawling starting from the item i_4. For our purposes, the starting node is also the target one that is used for the reconstruction of its neighborhood (discussed in Section 4.3) or to make a push shilling attack (discussed in Section 4.2). Algorithm 1 provides the pseudo-code of a general crawling procedure.

In our simulation, a node in the (unknown) user-item rating graph (excluding the starting node) passes through three states (depicted in Figure 3):

unknown The node exists in the whole graph but is currently unknown.

discovered The node has been discovered through another just visited node linked to it. Discovered but not visited nodes can be considered in the frontier of the graph exploration. A discovered node may carry information about its out-degree.

visited The node has been visited, allowing the discovery of (potentially) new nodes. The visiting of a node simulates the request of its web page.

Algorithm 1: Crawling procedure.

Input: $G(U, I, E)$: user-item rating bipartite graph; p: percentage of node to visit, $p \in [0, 1]$; x: target node (user/item)

Output: G': explored sub-graph ($G' \subseteq G$)

$n \leftarrow |U| + |I|$
$U_{G'}, I_{G'}, \mathcal{E} \leftarrow \emptyset, \emptyset, \emptyset$
add node x to either $I_{G'}$ or $U_{G'}$ on the basis of its type
$Q \leftarrow [x]$
while $|Q| > 0$ *and* $|U_{G'}| + |I_{G'}| < pn$ **do**
  $x \leftarrow pop(Q)$
  update $I_{G'}$ or $U_{G'}$ on the basis of the type of x
  update Q with $\{y|(x, y) \in E\}$
  update $\mathcal{E}$ with $\{e \in E|e = (x, y) \in E\}$
  sort Q according to the ordering policy of the algorithm
end
$G'(U_{G'}, I_{G'}, \mathcal{E})$
return G'

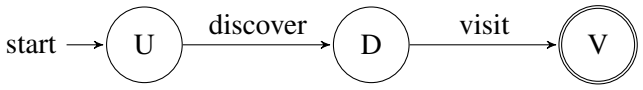


Fig. 3. Possible states of a node (with the exception of the starting node) during the crawling procedure. U = unknown, D = discovered, and V = visited.

4.2. Shilling Attack on Recommender Systems

The core difference between profile injection attacks lies in the way fake profiles are designed. The injected profiles must be carefully designed to avoid being detected. Profiles that highly differ from the "average" profile can be easily spotted and marked as suspicious (or even blocked), making the attack harmless. The number of fake profiles injected into the targeted system is usually called the **attack size**, while the **filler size** refers to the number of ratings each attack profile has to assign. The filler size is usually set to 1–20% of the database size. Adding ratings has a relatively lower cost for attackers, w.r.t. the cost of creating additional attack profiles. An effective attack size highly depends on how well the target system has been developed. A reasonable amount of fake profiles is 1–15% of the number of users in the system; otherwise, the associated cost of creating such additional profiles could be prohibitive. However, considering the size of nowadays e-services, this percentage can be easily considered $\ll 1\%$. It is clear that the bigger the system, the harder is to affect its recommendations. In a standard shilling attack [32], a malicious profile can be defined by four disjoint set of items, i.e., $(\mathcal{I}_T, \mathcal{I}_S, \mathcal{I}_F, \mathcal{I}_\emptyset)$ such that $\mathcal{I} \equiv \mathcal{I}_T \cup \mathcal{I}_S \cup \mathcal{I}_F \cup \mathcal{I}_\emptyset$:

			ratings		
Attack	$\mathcal{I}_S$	$\mathcal{I}_F$	S	F	T
Random	$\emptyset$	$\text{sam}(\mathcal{I} \setminus \mathcal{I}_T, f)$	-	$\mathcal{N}(\bar{r}_\mathcal{I}, s_\mathcal{I})$	r_{max}
Average	$\emptyset$	$\text{sam}(\mathcal{I} \setminus \mathcal{I}_T, f)$	-	$\mathcal{N}(\bar{r}_i, s_i)$	r_{max}
Bandwagon rand.	$\text{sam}(pop, s)$	$\text{sam}(\mathcal{I} \setminus \mathcal{I}_T, f)$	r_{max}	$\mathcal{N}(\bar{r}_\mathcal{I}, s_\mathcal{I})$	r_{max}
Bandwagon avg.	$\text{sam}(pop, s)$	$\text{sam}(\mathcal{I} \setminus \mathcal{I}_T, f)$	r_{max}	$\mathcal{N}(\bar{r}_i, s_i)$	r_{max}
Sampling attack	sample real users from the system				r_{max}

Table 1

Summary of the diverse attack models. Note that the filler size (f) and the selection size (s) are attack parameters. $\bar{r}_\mathcal{I}$ and $\bar{r}_i$ respectively indicate the average rating over all items, and the average rating of i over all users. $s_\mathcal{I}$ and s_i are the corresponding standard deviations. *pop* stands for popular items, and $\text{sam}(X, n)$ is a random sampling function over X of dimension n. The items in the set $\mathcal{I}_\emptyset$ are associated to a missing rating (i.e., *null*).

target item(s) the set of target items, $\mathcal{I}_T$, along with a rating function γ, which assigns a rating based on the goal of the attack (e.g., in a push attack the maximum rating value);

selected items set of items $\mathcal{I}_S$ useful to support the attack. Often items in $\mathcal{I}_S$ are related (e.g., bought together) to items in $\mathcal{I}_T$. In most of the standard Sybil attack, $\mathcal{I}_S$ is the empty set; The number of selected items is called **selection size**;

filler items set of items, $\mathcal{I}_F$, used to "camouflage" the fake profile to make it less detectable. Usually, ratings are randomly selected;

unrated items all the remaining set of items for which the fake profile does not give any rating $\mathcal{I}_\emptyset \equiv \mathcal{I} \setminus (\mathcal{I}_T \cup \mathcal{I}_S \cup \mathcal{I}_F)$.

We tested five shilling attacks [11, 12] in our analysis:

- **Random attack**: it is the easiest attack to implement; the fake profiles have ratings randomly chosen around the system overall mean, and the maximum rating (push) is assigned to the target item.
- **Average attack**: the fake profiles have ratings randomly chosen around the item means, and the maximum rating (push) is assigned to the target item.
- **Bandwagon attack (BW)**: an attacker generates profiles with high ratings to popular items and the highest possible rating to the target item. The way filler items are rated discriminate between BW random (BWR) and BW average (BWA).
- **Sampling attack**: the profiles are generated from samples of actual profiles by augmenting the target item's rating.

The way user profiles are crafted in the attacks above is summarized in Table 1.

4.3. Neighborhood Reconstruction

In competitive scenarios, collecting as much data as possible may not be the most effective and efficient strategy. It might be more useful to collect less but more informative data. To this end, we try to assess the quality of the collected knowledge by comparing how close are the most similar users/items computed with the crawled data w.r.t. the ones computed with the whole dataset. This comparison is based on the fact that the most popular recommendation engines are neighborhood-based [4, 6, 7]. Neighborhood-based systems rely on the concept of users/items similarity to compute the recommendations. Thus, if the neighborhood reconstruction is accurate enough, we can affirm that the collected knowledge has a competitive value. For example, if a competitor can match one of its users' identities with one of the target systems, it can use the knowledge about the user's neighborhood to improve the personalization quality, or it can be used to avoid the cold-start problem [4].

4.3.1. Similarity Measures

The neighborhood of a node (user/item) is determined in terms of a similarity measure between nodes. Our analysis employed two of the most widely used similarity measures in the recommender system community [4]: Pearson's correlation and cosine similarity. These measures are formally defined as follows:

Pearson's correlation :

- user-based

$$\text{Pearson}\left(\mathbf{r}_u, \mathbf{r}_v\right) = \frac{\sum_{i \in \mathcal{I}_{uv}} \left(r_{ui} - \bar{r}_u\right)\left(r_{vi} - \bar{r}_v\right)}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} \left(r_{ui} - \bar{r}_u\right)^2 \sum_{i \in \mathcal{I}_{uv}} \left(r_{vi} - \bar{r}_v\right)^2}};$$

- item-based

$$\text{Pearson}\left(\mathbf{r}_i, \mathbf{r}_j\right) = \frac{\sum_{u \in \mathcal{U}_{ij}} \left(r_{ui} - \bar{r}_i\right)\left(r_{uj} - \bar{r}_j\right)}{\sqrt{\sum_{u \in \mathcal{U}_{ij}} \left(r_{ui} - \bar{r}_i\right)^2 \sum_{u \in \mathcal{U}_{ij}} \left(r_{uj} - \bar{r}_j\right)^2}};$$

where $\bar{r}_{(\cdot)}$ is the average ratings of the user/item.

Cosine similarity :

- user-based

$$\cos\left(\mathbf{r}_u, \mathbf{r}_v\right) = \frac{\sum_{i \in \mathcal{I}_{uv}} r_{ui} \cdot r_{vi}}{\sqrt{\sum_{i \in \mathcal{I}_u} r_{ui}^2 \sum_{j \in \mathcal{I}_v} r_{vj}^2}}$$

- item-based

$$\cos\left(\mathbf{r}_i, \mathbf{r}_j\right) = \frac{\sum_{u \in \mathcal{U}_{ij}} r_{ui} \cdot r_{uj}}{\sqrt{\sum_{u \in \mathcal{U}_i} r_{ui}^2 \sum_{u \in \mathcal{U}_j} r_{uj}^2}}.$$

In our experiments, to avoid any bias, similarities have been computed only between users/items with support greater or equal than 5, that is, given $u, v \in \mathcal{U}$, $|\mathcal{I}_{uv}| \geqslant 5$, and, similarly, given $i, j \in \mathcal{I}$, $|\mathcal{U}_{ij}| \geqslant 5$, where $\mathcal{I}_{uv} \equiv \mathcal{I}_u \cap \mathcal{I}_v$, and $\mathcal{U}_{ij} \equiv \mathcal{U}_i \cap \mathcal{U}_j$.

The detailed description of the neighborhood reconstruction procedure is summarized in Algorithm 2.

Given the data crawled by a specific crawling strategy, the similarity matrix between users/items is computed and compared with the same matrix computed on the entire dataset. The reconstruction quality is measured in terms of the size of the overlap between the set of the k most similar users/items computed with the crawled data and the whole dataset. The higher the overlap's size, the higher the value of the crawled information.

Algorithm 2: Neighborhood Reconstruction Evaluation

Input: **R**: rating matrix;*crawler* : crawling algorithm, p: percentage of node to visit, $p \in [0,1]$; *sim*: similarity measure, x: target node (user/item)
Output: overlap percentage $\forall k \in \{10, 20, 50, 100, 200\}$

construct G, the user-rating bipartite graph, from **R**
/* extracts the sub-graph of G using the algorithm *crawler* starting from node x visiting 100p% nodes of G */
$G' \leftarrow crawler(G, p, x)$
/* computes the similarity, according to *sim* between x and all the other nodes, of the same type, in the graph. */
$S_G \leftarrow \text{argsort } sim(x, G)$
$S_{G'} \leftarrow \text{argsort } sim(x, G')$
$\mathbf{O} \leftarrow []$
for $k \in \{10, 20, 50, 100, 200\}$ **do**
  $\mathcal{S}_k \leftarrow set(S_G[:k])$
  $\mathcal{S}'_k \leftarrow set(S'_G[:k])$
  $O_k \leftarrow \frac{|\mathcal{S}_k \cap \mathcal{S}'_k|}{k}$
end
return O

4.4. Threat Model

We assume the following attacker's capabilities:

- The attacker can access only the public (e.g., accessible through the system's website) information of the target service. The data collection is performed by crawling the website, as described in Section 2.2.
- The crawling methods leverage only information about the user-item rating graph ignoring all side information like figures, description, or comments.
- The information of a user/item is gathered when the corresponding page is requested (i.e., visited). A page's visit also discloses the linked/rated items (users) but not their details.
- A discovered item, i.e., an item linked by a user/item's page, also carries information about its out-degree (see Section 4.1);
- The attacker targets a particular node in the graph (either user or item), which is also its starting node for the crawling.
- The attacker has the resources to perform a (push) shilling attack. The size of the attack is calibrated concerning the size of the target system.

5. Experiments and Results

Each phase of the described methodology has been extensively tested. Specifically:

- we compare the crawling policies in terms of how much they cover, i.e., number of discovered nodes, the recommendation graph after visiting a fixed number of nodes;

- we assess whether standard shilling attacks based on the crawled information are effective in practice (Section 5.3). We test both the effectiveness as well as the detectability of the injected user profiles;
- we measure the value of the crawled data in understanding the underlying recommendation model. Assuming the system is based on a k-nearest neighbor algorithm, we try to reconstruct a target node's neighborhood, either user or item. If the reconstruction is similar to the one computed over all the graph, then the collected information has collaborative value, and competitors might maliciously leverage it (Section 5.4).

5.1. Datasets

In our experiments, to emulate a real e-service (e.g., e-commerce) recommendation-based website, we use seven small- to large-scale datasets commonly used as a benchmark in the RSs community [4] (details are summarized in Table 2). In particular:

Filmtrust [40] It is a small dataset crawled from the entire FilmTrust website in June 2011. Filmtrust uses a 5-star rating system.

BookCrossing [41] Dataset collected by Cai-Nicolas Ziegler in a 4-week crawl in 2004 from the Book-Crossing community. The dataset has been pre-processed to keep only users with at least ten ratings.

MovieLens [4] This dataset contains users (5-stars) ratings collected from a movie recommendation service designed by the GroupLens Research. In our experiments, we used three different versions of the dataset with an increasing number of ratings, users, and items, namely `ml100k`, `ml1m`, and `ml20m`.

Goodreads [42] This dataset contains reviews and ratings (5-stars) from the Goodreads book review website.

Netflix this is the user-movie (5-stars) ratings data from the Netflix Prize[5]. The main difference with the `Movielens` datasets is its sparsity, that is five times the most sparse `Movielens` dataset (i.e., `ml20m`).

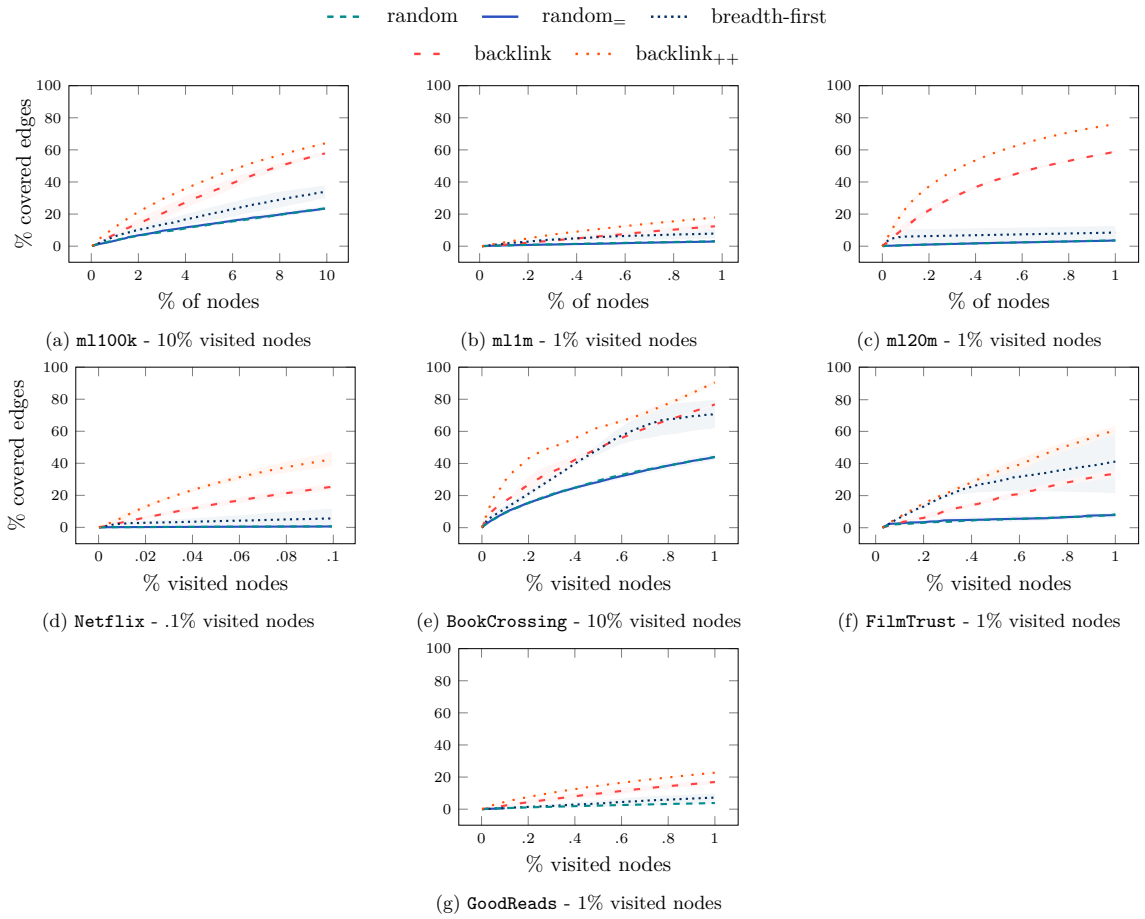
Dataset	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{R}\rvert$	Avg. u deg.	Avg. i deg.	Density
`Filmtrust`	1 507	2 071	35 485	23.5 $\pm$ 23.7	17.1 $\pm$ 91.7	1.13%
`BookCrossing`	1 891	17 631	92 784	49.1 $\pm$ 5.8	5.2 $\pm$ 20.6	0.28%
`ml100k`	943	1 639	99 955	165.6 $\pm$ 192.7	270.9 $\pm$ 384.4	6.47%
`ml1m`	6 040	3 691	1 000 192	105.9 $\pm$ 100.7	60.9 $\pm$ 80.8	4.49%
`Goodreads`	18 891	25 475	1 378 001	72.9 $\pm$ 102.6	54.1 $\pm$ 103.3	0.29%
`ml20m`	138 493	26 164	19 999 645	209.2 $\pm$ 230.2	764.4 $\pm$ 3 117.8	0.55%
`Netflix`	480 188	17 770	100 462 736	144.4 $\pm$ 302.2	5 653.5 $\pm$ 16 909.2	1.18%

Table 2

Datasets information: number of users, number of items, number of interactions (i.e., ratings), average number of ratings per user, and number of ratings per item.

[4] https://grouplens.org/datasets/movielens/

[5] http://www.netflixprize.com/



(a) `ml100k` - 10% visited nodes
(b) `ml1m` - 1% visited nodes
(c) `ml20m` - 1% visited nodes
(d) `Netflix` - .1% visited nodes
(e) `BookCrossing` - 10% visited nodes
(f) `FilmTrust` - 1% visited nodes
(g) `GoodReads` - 1% visited nodes

Fig. 4. Graph coverage (% of discovered nodes) of the crawling algorithms across different datasets.

5.2. *Crawling the Recommendation Graph*

To mimic a real attacking scenario, we test the crawling algorithms assuming that only a small subset of nodes can be visited. We fixed this number of nodes with respect to the size of the datasets. The experiments have been performed on the datasets described in Table 2. The results reported in Figure 4 show the coverage of each crawling algorithm in terms of how many nodes they can discover. We do not report the PageRank algorithm because of its computational complexity, making it less reasonable to be used in practice.

In all datasets, both random and random$_=$ perform poorly. This is expected since the exploration does not follow any "smart" policy. Moreover, it seems that neither of the random strategies prevails over the other. Also, the breadth-first algorithm shows, on average, very weak coverage. However, this is not surprising since its ordering policy is not informed. It simply visits the first node in the frontier queue, which is not a better prioritization method w.r.t. the random one. Clearly, this consideration holds for this set of experiments, but we expect very different performance in the successive experiments. Nonetheless, on `Filmtrust` breadth-first achieves good coverage even though with a very high variance. We argue that this is due to the size of the dataset (the smallest one) and the way ratings are distributed [43].

Regarding the backlink strategy, both the standard and the backlink$_{++}$ achieve very good coverage, almost always over 60% in the small datasets. The superiority of backlink$_{++}$ can be attributed to the fact that it uses the information about the nodes' global connectivity instead of the one related to the crawled graph.

It is worth noticing that the percentage of nodes visited is not a direct indicator of how much the crawling procedure can cover the graph. A clear example is the difference between `ml1m` and `ml20m` where on both 1% of the nodes are visited. In this case, the difference lies in the graph's connectivity (see Table 2). Even though `ml20m` is more sparse than `ml1m`, it has, on average, a connectivity but with a higher variance. This reveals the presence of many *hub nodes* [44, 45] that allows a single visit to cover many edges explaining the huge difference in the resulting coverage.

5.3. Performing Shilling Attacks Using Crawled Information

This set of experiments aim to assess whether performing a shilling attack using crawled information is as effective as using the entire dataset. In particular, we test whether crafting malicious user profiles using crawled data harms standard shilling attacks' effectiveness. To run these tests, we needed to build a recommendation engine. We chose to employ the popular k-nearest neighbor [4] recommender (k=40) because it represents the main off-the-shelf alternative. We considered only push attacks on a single target item selected from the most popular items' second quintile. Both the attack size, i.e., the number of fake profiles, and the filler size have been set to 5% of the entire dataset, which is usually used in the literature [13, 37].

We report the results on the smallest datasets, namely `Filmtrust`, `ml1m`, and `BookCrossing`. For the bigger datasets, all attacks failed using the crawled data. We stopped the crawling algorithm after visiting 0.5% of the graph. We argue that the visited percentage would be much lower in a real setting and with limited resources. We measure the performance of the attacks in terms of *prediction shift*, i.e., how the average rating of the target item changes before and after the attack, and *hit@n* which is 1, if and only if the target item is ranked in the top k items. Formally, given a rank R and an item i, $\text{hit@n}(R, i) = 1$ if and only if i is in the first n positions of R, 0 otherwise. The *hit@k* results are reported in Figure 5.

It is evident from the figure that, in general, in terms of hit@10, having the full knowledge of the rating matrix increases the attack's effectiveness. However, on really small and dense datasets (e.g., `Filmtrust`), all attacks seem to be effective with all the crawling algorithms except for the random. Unfortunately, this scenario is, in reality, less likely since usual real-world rating matrices are large-scale and hugely sparse (density $\ll 1\%$). In fact, `BookCrossing` is the closest to a real scenario, with many more items than users and a density less than 1%. None of the attacks are effective on this dataset.

In terms of prediction shift, all attacks consistently achieve about a 1 point push. Nonetheless, it is not enough to get the target item in the top-10 position of the recommendation in most cases. The `Filmtrust` exception is due to the small number of items and its ratings distribution [43].

In general, the new set of experiments show that shilling attack can be very successful in small size recommendation systems, especially if the rating matrix is relatively dense and not extremely long-tailed.

5.3.1. Shilling Attack Detection

After performing the attack, regardless of the success or not, we checked whether the injected profiles were easy to detect using standard statistical detection mechanisms [46, 47]. In particular, we implemented the following detection strategies:

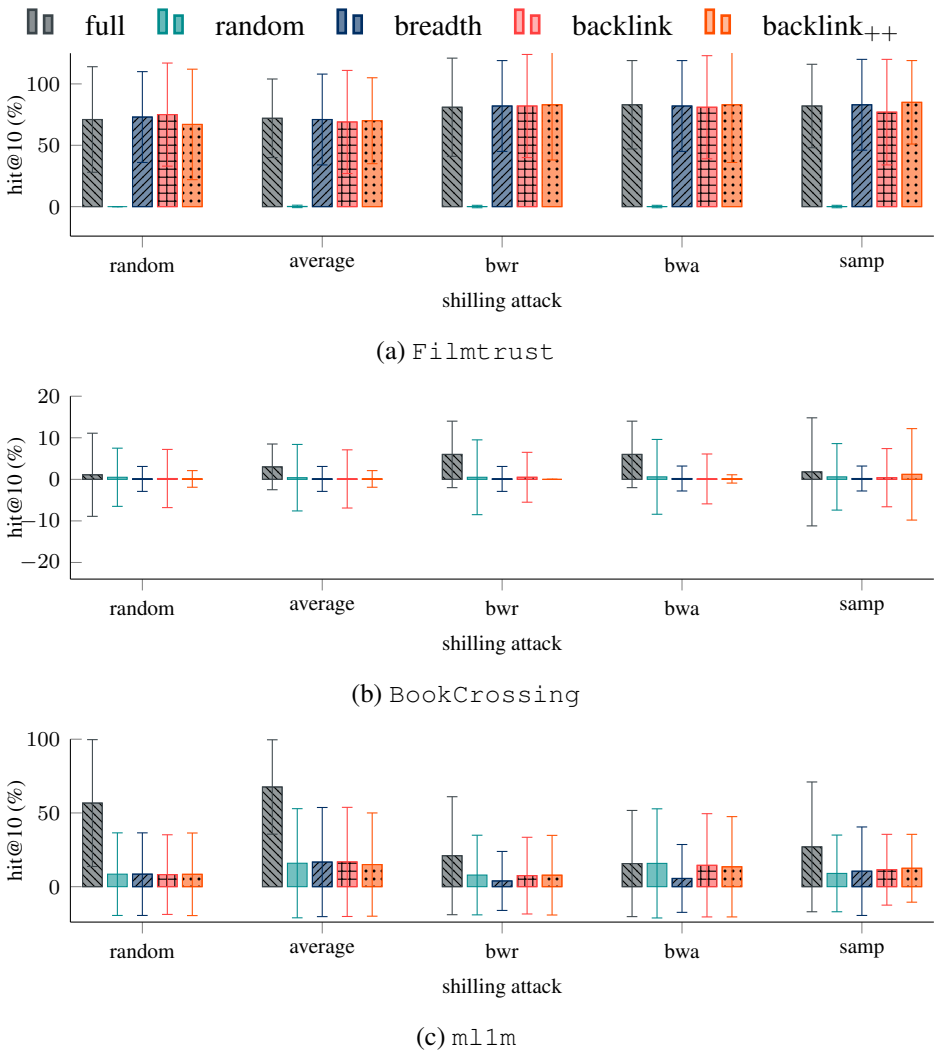


Fig. 5. Comparison of different (push) shilling attacks based on data crawled using different algorithms (and the full dataset). Reported results are in Hit@10 overall percentage users that did not rate the target item. Target item has been randomly selected from the 2nd quintile of the most popular items. On the x axis, *bwr* means Bandwagon Random, *bwa* means Bandwagon Average, and *samp* stands for the Sampling attack.

Rating Deviation from Mean Agreement (RDMA) measure of rating deviation of a user on a set of target items concerning other users, combined with the inverse rating frequency of these items [48].

$$RDMA(u) = \frac{\sum_{i \in \mathcal{I}_u} \frac{|r_{ui} - \bar{r}_i|}{|\mathcal{U}_i|}}{|\mathcal{I}_u|}.$$

Mean Variance (MeanVar) is usually used to detect average attacks. It computes the mean-variance between all the filler items and the overall average. A low variance would indicate the possibility

Attack	all	rand	breadth	backlink	backlink$_{++}$
rnd	100	100	100	100	100
avg	98	84	88	88	98
bwr	98	86	92	92	98
bwa	98	97	98	95	98
samp	10	7	6	6	8

Table 3
Fake profile detection percentage (%) on `ml1m`.

Attack	all	rand	breadth	backlink	backlink$_{++}$
rnd	100	99	98	91	100
avg	100	5	75	91	100
bwr	100	99	96	85	100
bwa	100	100	93	94	100
samp	8	2	3	2	3

Table 4
Fake profile detection percentage (%) on `Filmtrust`.

Attack	all	rand	breadth	backlink	backlink$_{++}$
rnd	100	17	16	32	10
avg	100	13	18	22	0
bwr	100	16	11	25	8
bwa	100	3	8	13	1
samp	7	0	0	0	0

Table 5
Fake profile detection percentage (%) on `BookCrossing`.

of an average attack [49].

$$MeanVar(u) = \frac{\sum_{i \in \mathcal{F}_u} (r_{ui}\ \bar{r}_u)^2}{|\mathcal{F}_u|},$$

where $\mathcal{F}_u = \mathcal{I}_u \setminus \mathcal{I}_T$ are the filler items in the profile of u.

In the experiments, a fake user profile is considered detected if either its MeanVar or RDMA is significantly different from the average computed over all the training users. Tables 3, 4, and 5 show the detection percentage of the injected profiles.

In both MovieLens and Filmtrust, almost all the crafted profiles have been detected for all attacks except the sampling attack. By design, sampling attack is based on true statistics, and it seems that even using a subset (from the crawling) helps the attack be successful, at least in terms of the number of undetected fake profiles. On `BookCrossing` the detection rate is in general very low, and it also seems that the profile crafted on crawled data are also harder to detect w.r.t. the one created using the full data set. On the other hand, the attack itself on `BookCrossing` was not really successful (Section 5.3).

In general, considering both the success rate and the detectability of the attacks, these results further support that performing a shilling attack on crawled data can hardly be effective in large-scale scenarios.

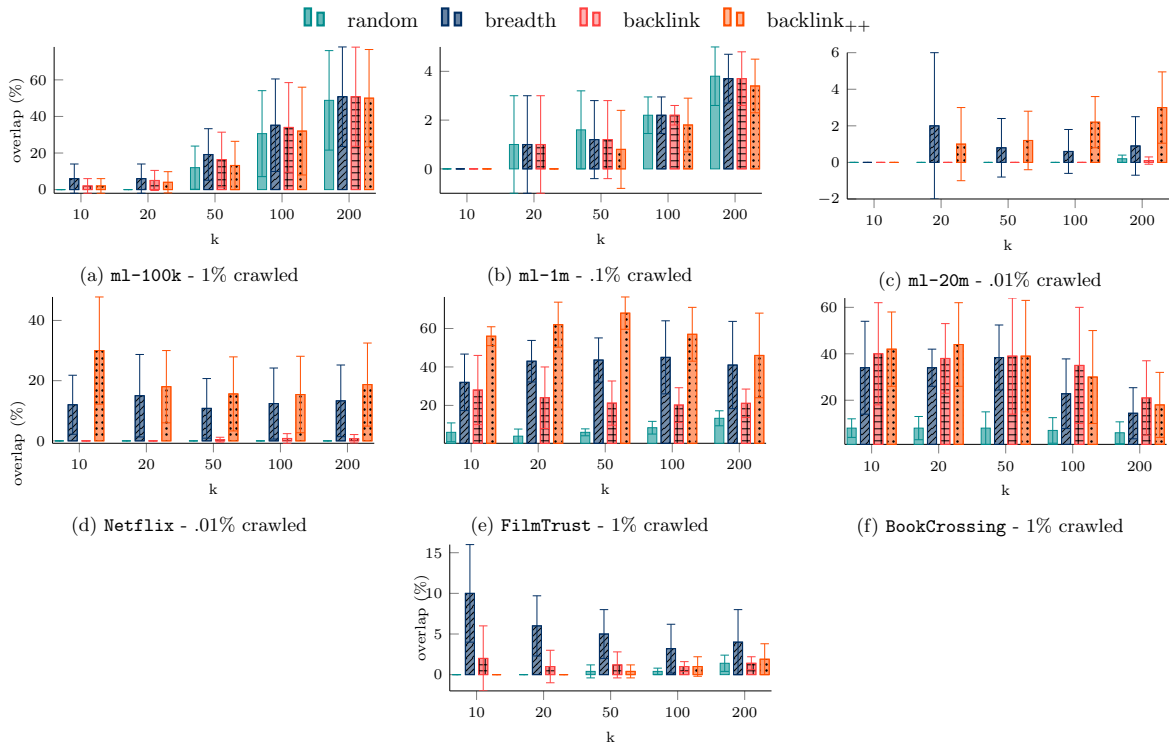


Fig. 6. Neighborhood reconstruction using user-based Pearson's correlation. The results are the average (± standard deviation) over five randomly selected users. *k* on the x-axis is the dimension of the considered neighborhood.

5.4. Neighborhood Reconstruction

With these last set of experiments, we want to test if the crawled information is enough to be considered valuable in terms of competitive knowledge about the underlying recommendation engine. Clearly, different recommender systems use collaborative information in very different ways. Still, the underlying core concept is related to the number of ratings shared by the users (or items). For this reason, we argue that reconstructing the neighborhood gives a good estimate of what we know about the system.

These experiments have been performed following the procedure described in Section 4.3. We tested both user-based and item-based recommendations, using cosine similarity and Pearson correlation.

5.4.1. Neighborhood Reconstruction on User-based Recommenders

Figures 6 and 7 show the overlap percentage of the neighborhood reconstruction using a user-based KNN based on Pearson's correlation and cosine similarity, respectively.

As expected from the crawling experiments results, the random strategies do not allow any kind of reconstruction. This is intuitively reasonable since this strategy does not consider any properties of the nodes/graph to prioritize the nodes.

Contrary, in all but one dataset (i.e., `Goodreads`) backlink$_{++}$ shows to be the most successful strategy, but, as we already mentioned, it may not be applicable. Comparable, but in general, a bit less effective is the backlink strategy, which seems to struggle on bigger datasets, namely `ml20m` and `Netflix`.

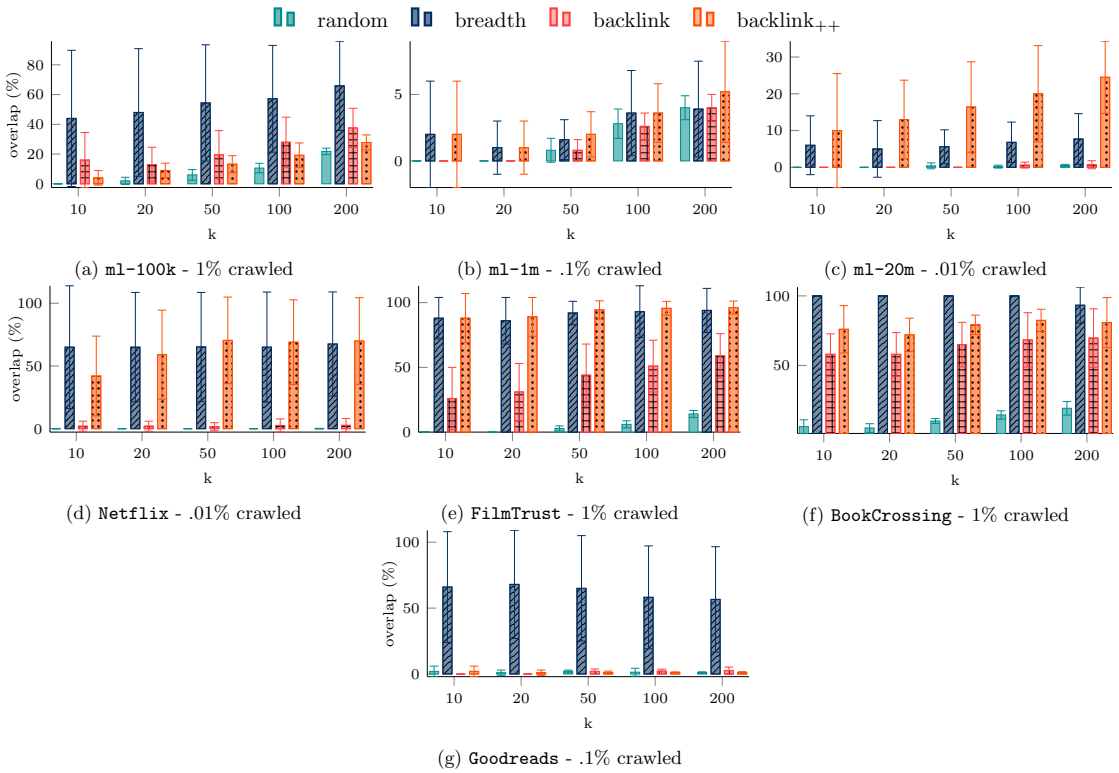


Fig. 7. Neighborhood reconstruction using user-based cosine similarity. The results are the average (± standard deviation) over five randomly selected users. *k* on the x-axis is the dimension of the considered neighborhood.

We argue that this is due to the poor approximation quality of the degree of the nodes, i.e., the degree in the full graph (this is also supported by the higher gap in coverage w.r.t. backlink$_{++}$, see Figure 4).

Surprisingly, breadth-first that has shown not to be a good method in terms of coverage, achieves comparable results w.r.t. backlink$_{++}$. This is justified by the way the neighborhood is constructed. Users are similar if they share many ratings, so if we target a specific user and perform a breadth-first crawling, we are searching on her neighborhood nodes in the bipartite graph. Notable is the reconstruction of breadth-first on `Goodreads` where all other approaches failed. In this particular dataset, we argue that its success is due to balancing the node degrees between users and items. This allows the breadth-first search to cover the neighbor nodes uniformly. In terms of similarity metrics, no particular difference can be deduced from the results.

5.4.2. Neighborhood Reconstruction on Item-based Recommenders

Figures 8 and 9 show the overlap percentage of the neighborhood reconstruction using an item-based KNN based on Pearson's correlation and cosine similarity, respectively. In these experiments, we increased the crawling percentage on `ml20m` and `Netflix`.

The need to increase the crawling percentage underlines the fact that it is harder to reconstruct an item's neighborhood on big systems. This can be due to a longer tail in the long tail distribution. However, on small datasets, the reconstruction is possible even with the random crawling strategy. This is reasonable since we targeted popular items, and hence the crawling has more chance of covering useful nodes.

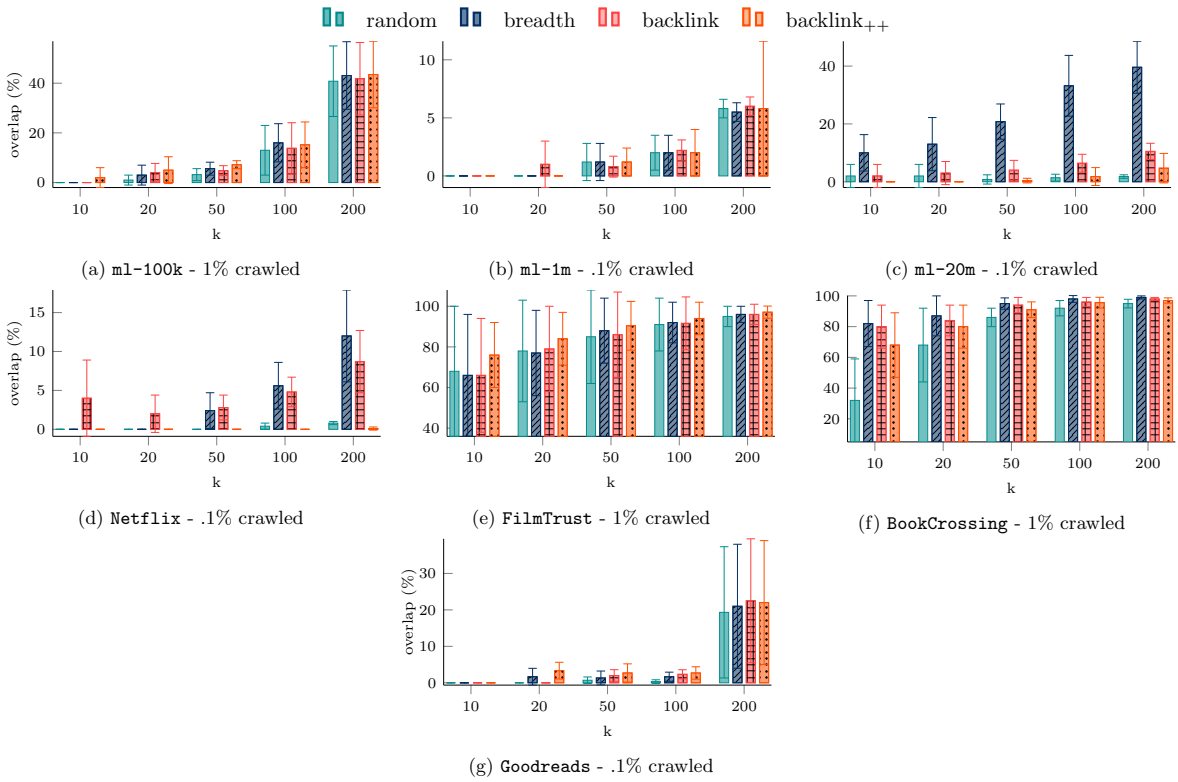


Fig. 8. Neighborhood reconstruction using item-based Pearson's correlation. The results are the average (± standard deviation) over five randomly selected average popular items. *k* on the x-axis is the dimension of the considered neighborhood.

It is further confirmed that breadth-first works pretty well, except for `Netflix` (with cosine similarity). Surprisingly, backlink seems to be the best performing strategy, even better than backlink$_{++}$, which is still a good alternative. This behavior is worth being investigated in the future.

We want to emphasize that, on average, the reconstruction over the cosine similarity seems easier than with Pearson's correlation, while in terms of absolute value, the neighborhood reconstruction has a slightly higher rate of success in the user-based setting.

Wrapping up, in general, we can state that a competitor can collect useful knowledge crawling a target e-service, especially in the case of a small size target system. In general, the rate of success highly depends on the size of the target site and available resources to perform the crawling. Empirically, it seems that a standard breadth-first strategy does the job nicely, but when possible, using more information to prioritize the crawling frontier (e.g., backlink$_{++}$) can improve the results.

6. Conclusions and Future Work

Profile injection attacks are by far the most common kind of attacks to recommender systems. In principle, they are easy to apply and have only a limited cost. However, when dealing with real-world, large-scale systems, many challenges must be faced to perform such an attack. This work showed that the first challenge is to gather data to start mounting the attack. We empirically demonstrated strategies to effectively collect a good amount of information (backlink and backlink$_{++}$) with limited resources.

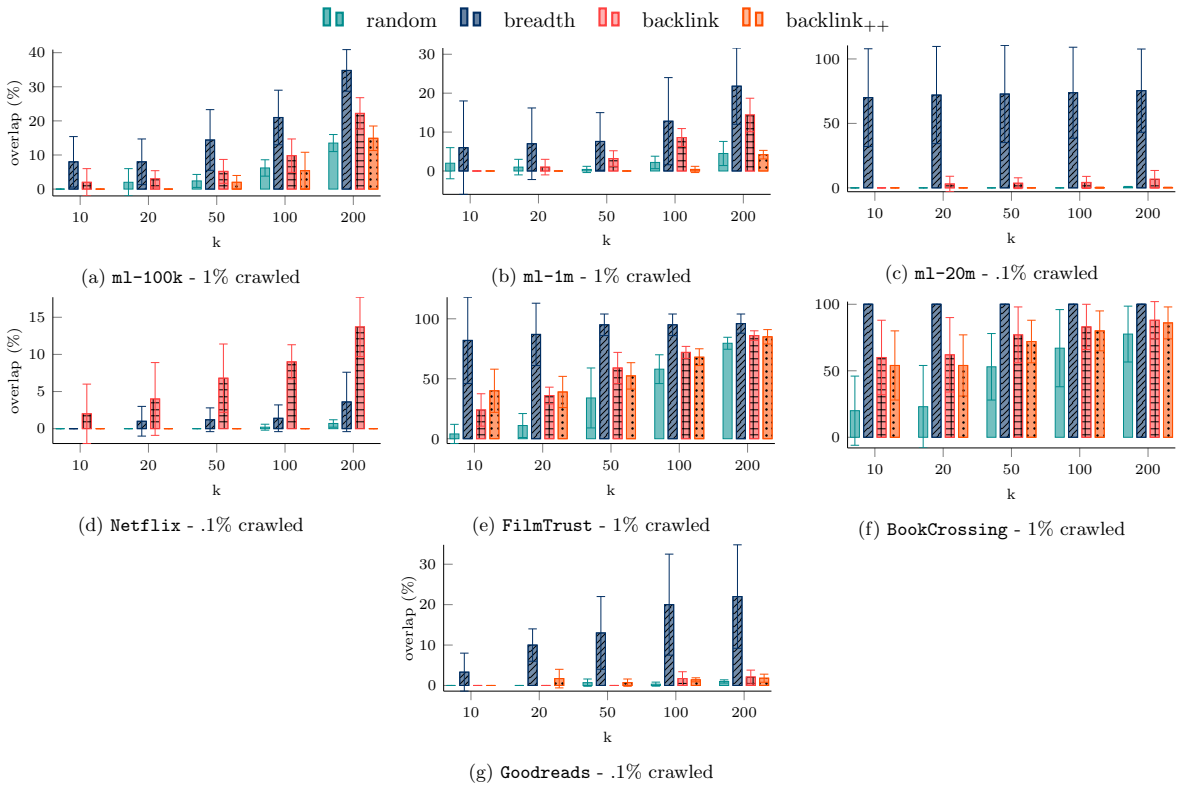


Fig. 9. Neighborhood reconstruction using item-based cosine similarity. The results are the average (± standard deviation) over five randomly selected average popular items. k on the x-axis is the dimension of the considered neighborhood.

However, this might not be enough to successfully attack the system, especially if countermeasures like detection mechanisms are used. On the other hand, the crawled information still brings knowledge about the system that competitors can leverage. In this case, a more "targeted" (e.g., breadth-first) crawling strategy can be effective in covering a specific part of the recommendation graph. Whether the amount of gathered information is valuable or not hugely depends on the size of the system.

In future work, we aim to expand this analysis to other types of attacks. Moreover, it will be worth investigating new crawling policies that also use the content information about the items rather than the graph's mere structural information. Besides, new and more advanced attacking techniques should also be compared with ad-hoc detection mechanisms.

References

[1] W. Deng, Y. Shi, Z. Chen, W. Kwak and H. Tang, Recommender system for marketing optimization, *World Wide Web* **23**(3) (2020), 1497–1517. doi:10.1007/s11280-019-00738-1.

[2] C. Eksombatchai, P. Jindal, J.Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich and J. Leskovec, Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time, in: *Proceedings of the 2018 World Wide Web Conference*, WWW '18, WWW Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2018, pp. 1775–1784–. ISBN 9781450356398. doi:10.1145/3178876.3186183.

[3] C. Gomez-Uribe and N. Hunt, The Netflix Recommender System: Algorithms, Business Value, and Innovation, *ACM Trans. Manage. Inf. Syst.* **6**(4) (2016). doi:10.1145/2843948.

[4] F. Ricci, L. Rokach and B. Shapira, *Recommender Systems Handbook*, 2nd edn, Springer Publishing Company, Incorporated, 2015. ISBN 1489976361.

[5] G. Linden, B. Smith and J. York, Amazon.com recommendations: item-to-item collaborative filtering, *IEEE Internet Computing* **7**(1) (2003), 76–80.

[6] Y. Koren and R. Bell, *Advances in Collaborative Filtering*, in: *Recommender Systems Handbook*, Springer, Boston, MA, 2011, pp. 145–186. ISBN 978-0-387-85820-3. doi:10.1007/978-0-387-85820-3_5.

[7] X. Su and T.M. Khoshgoftaar, A Survey of Collaborative Filtering Techniques, *Adv. in Artif. Intell.* **2009** (2009). doi:10.1155/2009/421425.

[8] K. Christakopoulou and A. Banerjee, Adversarial Attacks on an Oblivious Recommender, in: *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, ACM, 2019, pp. 322–330. ISBN 978-1-4503-6243-6. doi:10.1145/3298689.3347031.

[9] L. Muñoz-González, B. Pfitzner, M. Russo, J. Carnerero-Cano and E.C. Lupu, Poisoning Attacks with Generative Adversarial Nets, *ArXiv* **abs/1906.07773** (2019).

[10] P. Kaur and S. Goel, Shilling attack models in recommender system, in: *2016 International Conference on Inventive Computation Technologies (ICICT)*, Vol. 2, 2016, pp. 1–5. doi:10.1109/INVENTIVE.2016.7824865.

[11] I. Gunes, C. Kaleli, A. Bilge and H. Polat, Shilling attacks against recommender systems: a comprehensive survey, *Artificial Intelligence Review* (2014), 767–799.

[12] K. Patel, A. Thakkar, C. Shah and K. Makvana, A State of Art Survey on Shilling Attack in Collaborative Filtering Based Recommendation System, in: *Proceedings of First International Conference on Information and Communication Technology for Intelligent Systems: Volume 1*, S.C. Satapathy and S. Das, eds, Springer, Cham, 2016, pp. 377–385. ISBN 978-3-319-30933-0.

[13] R. Burke, B. Mobasher and R. Bhaumik, Limited knowledge shilling attacks in collaborative filtering systems, in: *In Proceedings of the 3rd IJCAI Workshop in Intelligent Techniques for Personalization*, 2005.

[14] F. Aiolli, M. Conti, S. Picek and M. Polato, Big Enough to Care Not Enough to Scare! Crawling to Attack Recommender Systems, in: *Computer Security – ESORICS 2020*, L. Chen, N. Li, K. Liang and S. Schneider, eds, Springer International Publishing, Cham, 2020, pp. 165–184. ISBN 978-3-030-59013-0.

[15] Y. Zhang, H. Gao, G. Pei, S. Luo, G. Chang and N. Cheng, A Survey of Research on CAPTCHA Designing and Breaking Techniques, in: *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2019, pp. 75–84.

[16] K. Turk, S. Pastrana and B. Collier, A tight scrape: methodological approaches to cybercrime research data collection in adversarial environments, in: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, 2020, pp. 428–437. doi:10.1109/EuroSPW51379.2020.00064.

[17] J. Cho, H. Garcia-Molina and L. Page, Efficient crawling through URL ordering, *Computer Networks and ISDN Systems* **30**(1) (1998), 161–172, Proceedings of the Seventh International World Wide Web Conference. doi:https://doi.org/10.1016/S0169-7552(98)00108-1. http://www.sciencedirect.com/science/article/pii/S0169755298001081.

[18] L. Page, S. Brin, R. Motwani and T. Winograd, The PageRank Citation Ranking: Bringing Order to the Web., in: *WWW 1999*, 1999.

[19] M. Koster, Robots in the web: threat or treat?, *ConneXions* **9**(4) (1995).

[20] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson and J. Kleinberg, Automatic Resource Compilation by Analyzing Hyperlink Structure and Associated Text, in: *Proceedings of the Seventh International Conference on World Wide Web 7*, WWW7, Elsevier, NLD, 1998, pp. 65–74–.

[21] S. Brin and L. Page, The Anatomy of a Large-Scale Hypertextual Web Search Engine, in: *Proceedings of the Seventh International Conference on World Wide Web 7*, WWW7, Elsevier, NLD, 1998, pp. 107–117–.

[22] M. Ester, H.-P. Kriegel and M. Schubert, Accurate and Efficient Crawling for Relevant Websites, in: *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, VLDB Endowment, 2004, pp. 396–407–. ISBN 0120884690.

[23] A. Lawankar and N. Mangrulkar, A review on techniques for optimizing web crawler results, in: *2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave)*, 2016, pp. 1–4.

[24] R. Baeza-Yates, C. Castillo, M. Marin and A. Rodriguez, Crawling a Country: Better Strategies than Breadth-First for Web Page Ordering, in: *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, WWW '05, Association for Computing Machinery, New York, NY, USA, 2005, pp. 864–872–. ISBN 1595930515. doi:10.1145/1062745.1062768.

[25] S. Chakrabarti, *Focused Web Crawling*, in: *Encyclopedia of Database Systems*, Springer US, Boston, MA, 2009, pp. 1147–1155. ISBN 978-0-387-39940-9. doi:10.1007/978-0-387-39940-9_165.

[26] H. Holzmann, A. Anand and M. Khosla, Delusive PageRank in Incomplete Graphs, in: *Complex Networks and Their Applications VII*, L.M. Aiello, C. Cherifi, H. Cherifi, R. Lambiotte, P. Lió and L.M. Rocha, eds, Springer International Publishing, Cham, 2019, pp. 104–117.
[27] H. Holzmann, A. Anand and M. Khosla, Estimating PageRank deviations in crawled graphs, *Applied Network Science* **4** (2019), 86–107.
[28] R. Cai, J.-M. Yang, W. Lai, Y. Wang and L. Zhang, IRobot: An Intelligent Crawler for Web Forums, in: *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, Association for Computing Machinery, New York, NY, USA, 2008, pp. 447–456–. ISBN 9781605580852. doi:10.1145/1367497.1367558.
[29] S.I. Gass and M.C. Fu (eds), *Prim's Algorithm*, in: *Encyclopedia of Operations Research and Management Science*, Springer US, Boston, MA, 2013, pp. 1160–1160. ISBN 978-1-4419-1153-7.
[30] L. Barbosa and J. Freire, An Adaptive Crawler for Locating Hidden-Web Entry Points, in: *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, Association for Computing Machinery, New York, NY, USA, 2007, pp. 441–450–. ISBN 9781595936547. doi:10.1145/1242572.1242632.
[31] L. Barbosa and J. Freire, Combining Classifiers to Identify Online Databases, in: *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, Association for Computing Machinery, New York, NY, USA, 2007, pp. 431–440–. ISBN 9781595936547. doi:10.1145/1242572.1242631.
[32] M. Si and Q. Li, Shilling attacks against collaborative recommender systems: a review, *Artificial Intelligence Review* **53** (2020), 291–319.
[33] M. Fang, G. Yang, N.Z. Gong and J. Liu, Poisoning Attacks to Graph-Based Recommender Systems, in: *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 381–392–. ISBN 9781450365697. doi:10.1145/3274694.3274706.
[34] I. Gunes, A. Bilge and H. Polat, Shilling Attacks Against Memory-Based Privacy-Preserving Recommendation Algorithms, *TIIS* **7** (2013), 1272–1290.
[35] B. Li, Y. Wang, A. Singh and Y. Vorobeychik, Data Poisoning Attacks on Factorization-based Collaborative Filtering, in: *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, 2016, pp. 1893–1901. ISBN 978-1-5108-3881-9. http://dl.acm.org/citation.cfm?id=3157096.3157308.
[36] N.J. Hurley, M.P. O'Mahony and G.C.M. Silvestre, Attacking Recommender Systems: A Cost-Benefit Analysis, *IEEE Intelligent Systems* **22**(3) (2007), 64–68.
[37] Y. Deldjoo, T. Di Noia and F.A. Merra, Assessing the Impact of a User-Item Collaborative Attack on Class of Users, in: *In Proceedings of the 13th ACM RecSys Workshop on Impact of Recommender Systems, (ImpactRS@RecSys'19)*, 2019. http://sisinflab.poliba.it/publications/2019/DDM19.
[38] S. Rendle, C. Freudenthaler, Z. Gantner and L. Schmidt-Thieme, BPR: Bayesian Personalized Ranking from Implicit Feedback, in: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, AUAI Press, Arlington, Virginia, USA, 2009, pp. 452–461–. ISBN 9780974903958.
[39] A.P. Sundar, F. Li, X. Zou, T. Gao and E.D. Russomanno, Understanding Shilling Attacks and Their Detection Traits: A Comprehensive Survey, *IEEE Access* **8** (2020), 171703–171715. doi:10.1109/ACCESS.2020.3022962.
[40] G. Guo, J. Zhang and N. Yorke-Smith, A Novel Bayesian Similarity Measure for Recommender Systems, in: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2013, pp. 2619–2625.
[41] C.-N. Ziegler, S.M. McNee, J.A. Konstan and G. Lausen, Improving Recommendation Lists through Topic Diversification, in: *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, Association for Computing Machinery, New York, NY, USA, 2005, pp. 22–32. ISBN 1595930469. doi:10.1145/1060745.1060754.
[42] M. Wan, R. Misra, N. Nakashole and J.J. McAuley, Fine-Grained Spoiler Detection from Large-Scale Review Corpora, in: *ACL*, 2019, pp. 2605–2610. https://doi.org/10.18653/v1/p19-1248.
[43] M. Polato and F. Aiolli, Boolean kernels for collaborative filtering in top-N item recommendation, *Neurocomputing* **286** (2018), 214–225. doi:https://doi.org/10.1016/j.neucom.2018.01.057.
[44] P. Knees, D. Schnitzer and A. Flexer, Improving Neighborhood-Based Collaborative Filtering by Reducing Hubness, in: *Proceedings of International Conference on Multimedia Retrieval*, ICMR '14, Association for Computing Machinery, New York, NY, USA, 2014, pp. 161–168–. ISBN 9781450327824. doi:10.1145/2578726.2578747.
[45] K. Hara, I. Suzuki, K. Kobayashi and K. Fukumizu, Reducing Hubness: A Cause of Vulnerability in Recommender Systems, in: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 815–818–. ISBN 9781450336215. doi:10.1145/2766462.2767823.
[46] W. Bhebe and O.P. Kogeda, Shilling attack detection in Collaborative Recommender Systems using a Meta Learning strategy, in: *2015 International Conference on Emerging Trends in Networks and Computer Communications*, 2015, pp. 56–61.
[47] W. Zhou, J. Wen, Y.S. Koh, Q. Xiong, M. Gao, G. Dobbie and S. Alam, Shilling Attacks Detection in Recommender Systems Based on Target Item Analysis, *PLOS ONE* **10**(7) (2015), 1–26. doi:10.1371/journal.pone.0130968.

[48] P.-A. Chirita, W. Nejdl and C. Zamfir, Preventing Shilling Attacks in Online Recommender Systems, in *WIDM '05*, Association for Computing Machinery, New York, NY, USA, 2005, pp. 67–74. ISBN 1595931945. doi:10.1145/1097047.1097061.

[49] R. Burke, B. Mobasher, C. Williams and R. Bhaumik, Classification Features for Attack Detection in Collaborative Recommender Systems, in: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, Association for Computing Machinery, New York, NY, USA, 2006, pp. 542–547. ISBN 1595933395. doi:10.1145/1150402.1150465.