SIMULATING REAL PROFILES FOR SHILLING ATTACKS: A GENERATIVE APPROACH

Julio César Barbieri Gonzalez de Almeida

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Geraldo Zimbrão da Silva

Rio de Janeiro
Agosto de 2024

SIMULATING REAL PROFILES FOR SHILLING ATTACKS: A GENERATIVE APPROACH

Julio César Barbieri Gonzalez de Almeida

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientador: Geraldo Zimbrão da Silva

Aprovada por: Prof. Geraldo Zimbrão da Silva
Prof. Geraldo Bonorino Xexéo
Prof. Eduardo Soares Ogasawara
Prof. Ronaldo Ribeiro Goldschmidt
Prof. Julio Cesar Duarte

RIO DE JANEIRO, RJ – BRASIL
AGOSTO DE 2024

Barbieri Gonzalez de Almeida, Julio César

Simulating real profiles for shilling attacks: a generative approach/Julio César Barbieri Gonzalez de Almeida. – Rio de Janeiro: UFRJ/COPPE, 2024.

XVIII, 136 p.: il.; 29, 7cm.

Orientador: Geraldo Zimbrão da Silva

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2024.

Referências Bibliográficas: p. 102 – 116.

1. Recommendation systems. 2. Collaborative filtering. 3. Shilling attack. 4. Variational autoencoder. I. Zimbrão da Silva, Geraldo. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

To everyone who supported me
throughout these past years.

Acknowledgments

I would like to start by thanking my entire family, especially my mother, who has always provided invaluable support and encouraged me to do my best. I am also deeply grateful to my Mariana Kosiba Furtado, who supported me through the most challenging moments of this journey. Without her, this research would have been significantly more difficult.

I want to thank my advisor, Geraldo Zimbrão da Silva, for the wealth of knowledge shared during this entire period under his guidance. I must also acknowledge the contributions of Filipe Braida do Carmo and Leandro Guimarães Marques Alvim; without their support, this work would not have been possible.

Additionally, I extend my gratitude to my research colleagues and friends who supported the first three years of this work with advice, ideas, and meaningful conversations on a wide range of topics: Ygor de Mello Canalli and Christian da Silva Cabral Cardozo. I also want to thank my longtime friends Yuri Almeida de Rezende, Vinicius Costa de Lima, and Felipe Bonomi de Lima for their enduring friendship and support during the challenging periods of this work.

Special thanks go to the examination board for accepting the invitation and for their contributions to this work.

I also want to thank the professors, staff members, and my colleagues from PESC and Universidade Federal do Rio de Janeiro (UFRJ).

Last but not least, I am grateful to Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) for funding this research.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

SIMULANDO PERFIS REAIS PARA SHILLING ATTACKS: UMA ABORDAGEM GENERATIVA

Julio César Barbieri Gonzalez de Almeida

Agosto/2024

Orientador: Geraldo Zimbrão da Silva

Programa: Engenharia de Sistemas e Computação

Filtragem Colaborativa (CF) é vulnerável a *Shilling Attacks*, onde usuários mal-intencionados injetam perfis falsos para manipular recomendações. Modelos atuais frequentemente usam técnicas estatísticas simples, resultando em perfis com padrões de avaliação distintos dos dados reais, o que facilita a detecção e requer um número maior de perfis para ser eficaz. Para resolver esse problema e criar perfis mais realistas, propomos o uso de um modelo generativo, Variational Autoencoder (VAE), que mapeia a distribuição dos dados reais. O VAE gera novos perfis baseados em dados reais sem copiar diretamente as avaliações, e esses perfis gerados são transformados em perfis maliciosos ao adicionar a avaliação do item alvo. Validação da proposta foi feita com diferentes bases de dados, comparando nosso modelo com os da literatura. Os resultados mostram que nosso modelo supera outros, especialmente em ataques menores (3% a 5%) no MovieLens 100k. A análise dos perfis gerados revelou que eles têm padrões de avaliação semelhantes aos perfis reais. Experimentos subsequentes com técnicas de detecção confirmaram que nosso modelo é menos detectável no MovieLens 100k. Também verificamos que abordagens de limpeza de dados são eficazes quando o administrador tem um conjunto confiável de usuários. Nosso modelo representa um avanço em *Shilling Attacks*, oferecendo resultados superiores e maior indistinguibilidade, sendo útil para testar técnicas de detecção e outras tarefas na área.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

SIMULATING REAL PROFILES FOR SHILLING ATTACKS: A GENERATIVE APPROACH

Julio César Barbieri Gonzalez de Almeida

August/2024

Advisor: Geraldo Zimbrão da Silva

Department: Systems Engineering and Computer Science

Collaborative Filtering (CF) systems are vulnerable to Shilling Attacks, where malicious users inject fake profiles to manipulate recommendations. Existing Shilling Attack models often use simplistic statistical templates, resulting in profiles with distinct rating patterns that are easier to detect and require more profiles to be effective. To address this issue, we propose using a Variational Autoencoder (VAE) to generate profiles that closely resemble real system data. Unlike traditional methods, VAE maps the original data distribution to create new profiles without directly copying actual ratings. These generated profiles are then modified to become malicious by altering the target item's rating. We validated our approach across different datasets and compared it with existing attack models. Our results demonstrate that our model outperforms others in model-based CF systems, especially with smaller attack sizes (3% to 5%) using the MovieLens 100k dataset. A correlation analysis of the generated profiles shows they have rating patterns similar to real profiles, and detection experiments confirm our model is less detectable. Furthermore, we found that Data Cleansing approaches can mitigate attacks when the system administrator has a reliable user set. Our model advances Shilling Attack techniques by producing more realistic profiles and achieving superior results, making it a valuable benchmark for testing detection methods and exploring further research in Shilling Attacks.

Contents

List of Figures

List of Tables

List of Symbols

R	Set of ratings
U	Set of users
I	Set of items
r_{ui}	Actual rating of a user u to an item i
$\hat{r}_{ui}$	Model prediction of a rating from a user u to an item i
w_{uv}	Similarity between users u and v
w_{ij}	Similarity between items i and j
$N_i(u)$	k nearest neighboors of user u that rated the item i
$\bar{r}_u$	Average rating for a user u
$\bar{r}_i$	Average rating for an item i
p_u	k latent variables of a user u
q_i	k latent variables of an item i
λ	Regularization term
γ	Learning rate
$\|U_i\|$	Number of users that rated item i
$\|U_{i,j}\|$	Number of users that rated both i and j items
x_i	Ratings for user profile i
z	Sample drawn from encoder
q_Θ	Encoder
p_Θ	Decoder
$p(z)$	Distribution to be approximated
$\mathbb{KL}(* \mid *)$	KL-divergence between two distributions
$\|i\|$	Number of items within the data set
i_{LF}	Lowest factor of the number of items in data set
$p_{u,i}$	Prediction before an attack
$p'_{u,i}$	Prediction after an attack
U_T	Set of tested users
I_T	Set of tested items

$\Delta_{u,i}$	Prediction shift of an user u related to the target item i
Δ_i	Prediction shift for all tested items
R_u	Top-N recommendation list for an user u
H_{ui}	Scoring function returning 1 if i is in R_u and 0 otherwise
HR_i	Hit Ratio for a given target item i
$\overline{HR}$	Average Hit Ratio over all target items
τ	threshold

List of Achronyms

CF	Collaborative Filtering
CA	Cosine Association
GAN	Generative Adversarial Network
GCN	Graph Convolutional Network
$IRSVD$	Improved Regularized SVD
MLP	Multilayer Perceptron
KNN	k-Nearest Neighboors
PIA	Power Item Attack
PUA	Power User Attack
$PIA-NR$	PIA Number of Ratings
$PUA-NR$	PUA Number of Ratings
SVD	Singular Value Decomposition
VAE	Variational Autoencoder

Chapter 1

Introduction

In this chapter, we introduce and contextualize our study, outline its primary objectives, discuss the hypothesis addressed, and outline the structure of this work.

1.1 Contextualization

Recommendation systems popularity has grown exponentially in the last decades with many companies such as *Amazon.com*[1], *Netflix*[2] and *Youtube*[3] relying on these systems as a business competitive differential. These systems' goal is to recommend products and services to its users in order to increase the amount of sales for its companies (ZHOU *et al.*, 2010).

In order to construct a recommendation system, the recommendation problem may be addressed in many ways. Among the most popular approaches to deal with it, collaborative filtering emerged as one of the most successful in both academia and industry with its so mentioned and used memory-based and model-based algorithms (SCHAFER *et al.*, 1999; HUANG & GONG, 2008). For example, the competition named *Netflix Prize* drove improvements in the accuracy of model-based algorithms and popularized its use in the 2000s (KOREN *et al.*, 2009).

One of the discussed issues in collaborative filtering is shilling attacks, also referred to as malicious noise. These attacks may be performed by malicious people or organizations, seeking to inject a large number of malicious profiles into a system to somehow manipulate its recommendations, promote their products, or degrade a competitor's product (BURKE *et al.*, 2006). Usually, attack models are engineered towards memory-based algorithms and rely on simple heuristics, trying to mimic the actual systems rating distribution, e.g., high or low rating on the target item, while selecting items at random and rating them with values around the system

[1] http://www.amazon.com
[2] http://www.netflix.com
[3] http://www.youtube.com

mean. There is an indication that while detection approaches may easily detect these classical shilling attack-crafted malicious profiles, they struggle to detect real-world malicious profiles (ZHANG *et al.*, 2018).

Moreover, these naïve attack models need a substantial number of malicious profiles required to achieve the desired effect. It is often unrealistic to assume that injecting 10% of the database volume can go unnoticed (WILLIAMS *et al.*, 2007), which underscores the need for more sophisticated attack models that accurately replicate the subtleties of legitimate user behavior while evading detection algorithms. By better understanding and simulating how malicious actors might manipulate with these systems, researchers can develop more robust defenses and detection strategies to safeguard against evolving threats in recommendation systems.

Variational autoencoders, introduced in KINGMA & WELLING (2013), are a powerful generative tool able to learn a probability distribution from data and generate new samples from it. It has been used to replicate complicated distributions and generate new data from noise in many tasks, such as handwritten digits (KINGMA & WELLING, 2013), faces (REZENDE *et al.*, 2014), images (GREGOR *et al.*, 2015), semantic segmentation (SOHN *et al.*, 2015) and even forecasting from static images (WALKER *et al.*, 2016). Using this generative model, it may be possible to mimic existing users' rating patterns to create a more realistic attack instead of simple statistical templates.

While introducing more realistic attack models is indispensable for advancing the security and reliability of collaborative filtering systems, ways of evaluating attack effectiveness that are not commonly used should also be explored alongside standard detection techniques to provide alternative defense options. In this context, various methods for handling label noise in collaborative filtering systems have been developed, offering possible solutions such as data cleansing and robust approaches. Label noise is relevant in a collaborative filtering setting because the entire recommendation process relies on user ratings and the quality of this data. While similar to malicious noise, label noise refers to naturally occurring misclassified data that can affect model accuracy (ZHU & WU, 2004). Robustness, on the other hand, denotes a model's resilience to data corruption, which is crucial for maintaining stable predictions (HUBER, 2004). Given these similarities, it is essential to investigate whether techniques for handling label noise can also effectively prevent malicious noise, just as they are used to addressing natural noise.

Given the discussed issues, making meaningful advances in the protection of collaborative filtering systems against shilling attacks, a novel model that moves away from straightforward statistical templates and can deceive detection methods is necessary. Using a generative-based attack model, more realistic and less detectable profiles can be crafted, and further advances can be made regarding the protection

of these systems, either by using detection approaches or robust models. These challenges are directly related to the objectives of this work and are formalized in the following sections.

1.2 Problem Definition

Based on the contextualization presented in the previous section, assuming the context of machine learning, collaborative filtering, and shilling attack, we present the following hypothesis to be evaluated in this thesis:

Hypothesis 1. *It is possible to develop a shilling attack model that deceives literature detection approaches.*

In addition to the hypothesis, the premise that realistic users can be crafted by utilizing existing users' profile information is also considered.

1.3 Objectives

Based on the problem definition, the main objective of this work is to create a new shilling attack model able to craft realistic malicious profiles. This is done in order to provide a new tool for the evaluation of robust algorithms and detection approaches. A secondary objective is to critically assess shilling attack models across various scenarios, encompassing both traditional detection methods and label noise-robust models. Summarizing, our general objectives are presented as follows:

i) **Propose a realistic attack model:** A novel attack model based on the generative power of a variational autoencoder is proposed. We aim to mimic the original data rating patterns in order to generate malicious profiles as close as possible to real ones to efficiently attack model-based collaborative filtering systems. Analysis indicates that the proposed model is effective in learning original rating patterns;

ii) **Empirical analysis of data cleansing and label noise-robust proposals:** We experiment with our method at MovieLens 100k, and Yahoo! Music datasets and compare these results against attack models in the literature. Attacks of different attack sizes are crafted using existing users on these datasets, and data cleansing and label noise-robust proposals are used to defend the system. Experiments indicate that data cleansing can be useful to correct malicious ratings, but label noise is not as successful;

iii) **Empirical analysis of Shilling Attack detection models:** We perform experiments to validate if our proposal is less likely to be detected by the literature shilling attack detection techniques using the aforementioned datasets. In order to have baseline metrics of acceptable levels of detectability, we also carry out experiments using the Amazon product review dataset, a dataset containing malicious profiles crafted by real-world attackers. Experiments indicate that our approach can craft attack profiles that are less detectable, and closer to real-world attackers than other attack models in the MovieLens dataset.

1.4 Contributions

This thesis is contextualized in the field of recommender systems, more precisely in collaborative filtering and shilling attack. The full list of contributions in this area is enumerated as follows:

i) A paper proposing a novel attack model (BARBIERI *et al.*, 2021);

ii) Experiments analyzing the rating patterns distribution of literature attack models and the proposal;

iii) Experiments comparing our proposed model against literature attack models in both memory-based and model-based algorithms;

iv) Experiments investigating the performance of data cleansing approaches against shilling attacks in two different scenarios;

v) Experiments investigating the performance of label noise-robust algorithms in the presence of malicious noise;

vi) Experiments comparing the performance of shilling attack detection techniques when applied to literature attack models;

vii) Experiments comparing the performance of shilling attack detection techniques when applied to attackers in a real-world dataset.

1.5 Document Structure

This work is structured as follows: In Chapter 2, we discuss the theoretical concepts that will give a basis to understand this work, that is, recommender systems, collaborative, shilling attack, machine learning, neural networks, and autoencoders;

In Chapter 3, we discuss the issues with current attack models, present our proposition to use generative models to produce malicious profiles in the shilling attack context, and evaluate it against well-known shilling attack models; In Chapter 4, we present the concept of label noise, review some solutions proposed to deal with this problem, and evaluate how they behave in the presence of malicious noise; In Chapter 5, we present the most common classification of detection methods, some classic and recent approaches to detect shilling attacks, and test them against well-known attack models from the literature; Finally, in Chapter 6, we present our final considerations and future research directions.

Chapter 2

Shilling Attacks in Collaborative Filtering

This chapter provides a theoretical background on recommendation systems, shilling attacks, machine learning, neural networks, and variational autoencoders. This foundational knowledge is essential for a comprehensive understanding of our proposal and the methodology used to validate it through experiments.

2.1 Recommender Systems

The emergence of recommendation systems dates from the beginning of the history of computing itself (EKSTRAND *et al.*, 2011) when a system named *Grundy* was proposed in 1979 (RICH, 1998). This system incorporates some concepts intrinsically linked with the recommendation as we know it today, modeling users through a short interview and using pre-codified stereotypes of book preferences to make recommendations.

Later, the first recommendation system arrived in the early 1990s, *Tapestry* (GOLDBERG *et al.*, 1992), an email system projected to deal with an excess of online information. It allowed users to manually create filters to receive emails according to their preferences. The relationship between the message and its response was modeled using a concept known as collaborative filtering, which would later be one of the main classes of recommender system algorithms.

The late 1990s showed a massive increase in the popularity of recommendation systems, and commercial implementations began to emerge. Systems such as *Amazon.com*[1] began to use purchase history, browsing history, and currently browsed items in order to make recommendations of new items to the user (EKSTRAND *et al.*, 2011). The field became more important over the years, with studies show-

[1] http://www.amazon.com

Table 2.1: Example of a user-item ratings matrix fragment for a movie recommender system.

User	Star Wars	Harry Potter	The Shining	The Avengers
John Smith	5	3	∅	1
Jane Doe	∅	4	5	4
Mary Sue	3	5	∅	5
Gary Stu	3	1	5	∅

ing how recommender systems impacted positively item sales (ZHOU *et al.*, 2010), mainly on less popular products, such as technical books (CHEN *et al.*, 2004). Currently, besides *Amazon.com*, many other companies use recommendation as a business differential, such as *Netflix*[2] and *Youtube*[3].

Finally, in the mid-2000s, ADOMAVICIUS & TUZHILIN (2005) formally formulated the recommendation problem as: Let U be the set of all users of the system, and I be the set of all items that can be recommended in the system, such as books, movies, or music. Let r be the utility function able to measure the usefulness of item i to the user u, this is, $r : U \times I \rightarrow R$, where R represents the sorted set of user preferences for each item present in I. In equation 2.1 we show it more formally.

$$\forall u \in U,\ i'_u = \arg \max_{i \in I} r(u, i). \tag{2.1}$$

Each element of the user set U needs to contain at least a unique user ID for identification purposes. However, it does not prevent the definition of a profile containing more information, such as age, gender, and monthly income, among others. Similarly, each element of the item set I should have a set of features with a unique item ID as well.

Usually, in recommender systems, the utility of an item is represented by a rating, which indicates the level of appreciation of a user for a particular item, e.g., a user rated the movie "Star Wars" by a value of 4 (out of 5). The main problem is that the utility function u is commonly not defined on the whole $U \times I$ space, but on some subset of it, which means that the main task is to extrapolate it to the whole space $U \times I$. Table 2.1 shows an example of a user-item preference matrix fragment for a movie recommendation system, where the rating scale varies between 1 and 5 and the symbol ∅ represents that the user did not assign a rating to this particular item. Therefore, the recommender system should be able to predict ratings for unassigned user-item pairs and present its recommendations based on these predictions.

Predictions can be made through two different approaches, according to ADOMAVICIUS & TUZHILIN (2005): using heuristics to define and validate empirically

[2]http://www.netflix.com
[3]http://www.youtube.com

the utility function, or estimating the utility function through optimization of some performance metric. Besides predicting user rantings, we have another common problem discussed in recommender systems, which is the prediction of the system items' ranking for the users. In other words, one must provide a list of top-N recommendations to a given user in order of importance (HERLOCKER *et al.*, 2004).

BURKE (2007) proposed a taxonomy presenting five different classes of recommender systems. Later, RICCI *et al.* (2011) extended their work, proposing a new class along with the previous five:

- **Content-based:** Learning is given by the recommendation of items similar to those that users liked in the past; this is, using content information about users and items.
- **Collaborative Filtering:** Learning is given based on the past preferences of most similar users to the user in question.
- **Demographic:** A recommendation given by the demographic profile of each user. In other words, the system assumes that people from the same demographic niche share similar tastes.
- **Knowledge-based:** Learning is given by domain-specific knowledge, which tries to infer user preferences from a particular domain.
- **Community-based:** Models social relationships by recommending items based on user friend preferences.
- **Hybrid Recommender Systems:** Learning given by combinations between classes listed above, e.g., a system mixing concepts of collaborative filtering and content-based.

In the next subsection, we will present collaborative filtering and some of your main algorithms.

2.1.1 Collaborative Filtering

Collaborative filtering is a class of recommendation algorithms that mimics a concept used by humans for centuries: searching for other people's opinions for recommendations that will help in their own decision-making process regarding an item. SCHAFER *et al.* (2007) described collaborative filtering as the process of filtering or evaluating items using opinions provided by others.

For example, if John has sufficient acquaintances who like a particular product, John must be inclined to consume it, and, if his acquaintances dislike it, John may

be influenced by them to not consume it. Another important point is that people have different tastes, so John will receive different feedback from his acquaintances about items, that may interest him or not. Thus, John will realize which people have similar tastes to him, which do not, and which may have mixed tastes between what John likes and what he does not like; hence he will learn who he must seek to get recommendations that will satisfy his tastes.

In other words, using a more formal approach, the utility function $r(u, i)$ of an item i for a user u is estimated based on the utilities $r(u_j, i)$ assigned to the same item i by all users $u_j \in U$ who are most similar to the user u (ADOMAVICIUS & TUZHILIN, 2005).

Collaborative filtering algorithms provide some advantages over other approaches. Unlike content-based recommendation, this class of algorithms did not require any additional information about users or items to perform its recommendations, making its recommendation quality not compromised by the lack of data. This particular property leads to another remarkable advantage over other content-based recommendations, that is, providing unexpected recommendations and avoiding obvious indications. This concept is known as serendipity (GE *et al.*, 2010).

Despite its advantages, collaborative filtering still suffers from some issues and provides some research challenges. One of the main issues is the user-item matrix sparsity, since the recommendation systems' datasets are usually large, but with very few evaluations per user. SARWAR *et al.* (2001) estimates that users rated only 1% of the items presented in the system.

Another drawback regarding sparsity is a common problem named cold-start. This issue refers to a new user in the system who still has not provided evaluations for any item, or a new item that has not received any evaluations yet. Since there is no information regarding the preferences of this user, it is difficult to make recommendations to him in this scenario (SCHEIN *et al.*, 2002).

There are some challenges regarding the performance of the systems, called scalability. As more users and items are included in the system, the computational cost grows as well. That said, a recommendation system with many users and items may suffer great scalability issues (SARWAR *et al.*, 2001).

It is also important to note that recommendation systems are prone to shilling attacks, and as a subclass of it, collaborative filtering is not different. Malicious people or organizations can decide to inject numerous malicious profiles into a system in order to promote their products or degrade a competitor's product (GUNES *et al.*, 2012).

There are several approaches in the literature to address the recommendation problem within the scope of the collaborative filtering class of algorithms. These approaches are commonly divided into two categories: memory-based and model-

based algorithms (ADOMAVICIUS & TUZHILIN, 2005).

Memory-based

Memory-based algorithms, also known as neighborhood-based, are heuristics used to make predictions for a user u based on the collection of his closest neighbors' previous ratings. In other words, the unknown rating value $\hat{r}_{u,i}$, which represents the value that a user u would give to an item i, will be computed as an aggregation of the most similar user's ratings for item i or an aggregation over the most similar users (ADOMAVICIUS & TUZHILIN, 2005)

Equation 2.2 shows how the prediction is made, where the k nearest neighbors of user u are denoted by $N_i(u)$. The unknown rating $\hat{r}_{u,i}$ is then predicted by the average rating given to i by these neighbors. Note that this equation does not take into equation different similarity levels that neighbors have with user u, an issue that can be solved using a weighted average rather than a simple average (RICCI *et al.*, 2011).

$$\hat{r}_{ui} = \frac{1}{|N_i(u)|} \sum_{v \in N_i(u)} r_{vi} \tag{2.2}$$

There are two variants of memory-based collaborative filtering methods: user-based and item-based. In the user-based variant, the system uses the most similar users to a user u to calculate the user rating prediction for an item i. Similarly, for the item-based variant, the system uses the most similar items to an i item to predict a user's u preference for this same item.

Equation 2.3 shows how the prediction is calculated for the used-based case. Let w_{uv} be the similarity between two users u and v, let $N_i(u)$ be the set of k most similar users to the user u and considering that item i has already been rated by a set of users previously, the recommendation is the weighted average of the ratings given by the most similar users to the user u that rated the item i.

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i(u)} w_{uv} r_{vi}}{\sum_{v \in N_i(u)} |w_{uv}|} \tag{2.3}$$

Similarly, equation 2.4 shows how the prediction is calculated for the item-based case. Let w_{ij} be the similarity between two items i and j, and let $N_u(i)$ be the set of k most similar items to the item i that have been rated by user u previously, the recommendation is the weighted average of the ratings given to the items most similar to item i.

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u(i)} w_{ij} r_{uj}}{\sum_{j \in N_u(i)} |w_{uj}|} \tag{2.4}$$

Different users may have different approaches to evaluating items, i.e., each user introduces a personal bias when evaluating every item. However, equations 2.3 and 2.4 does not consider this bias when making predictions. In order to solve this issue, each neighbor rating must be converted to a normalized value according to some normalization form. The most common normalization form reported in the literature is mean centering (RICCI *et al.*, 2011), which seeks to determine if the rating is positive or negative with respect to the nearest neighbors' average rating set, making the algorithm predict the deviation instead of the rating itself. Equations 2.5 for the user-based and 2.6 for the item-based shows how the prediction is computed using this solution.

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in \eta_i(u)} w_{uv}(r_{vi} - \bar{r}_u)}{\sum_{v \in \eta_i(u)} |w_{uv}|} \quad (2.5) \qquad \hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in \eta_u(i)} w_{ij}(r_{uj} - \bar{r}_j)}{\sum_{j \in \eta_u(i)} |w_{uj}|} \quad (2.6)$$

A critical step in building a memory-based recommendation system is the chosen similarity metric. The metric has a significant impact on the quality of the recommendations delivered by the system and its performance, as it will be responsible for the selection of the more reliable neighbors (RICCI *et al.*, 2011). A similarity measure $sim(u, v)$ between a user u and another user v is commonly a distance metric used as a weight in the weighted average during the recommendation process. This measure is usually calculated using only rating values rated in common between the pair of items or users (ADOMAVICIUS & TUZHILIN, 2005). Among the most common measures for this task, we highlight cosine similarity and pearson correlation (RICCI *et al.*, 2011).

Cosine is an angle between two vectors and can be interpreted as a similarity measure between two objects in a vectorial space. In the recommendation scenario, these two objects consist of a representation in the form of two rating vectors x and x. Equation 2.7 shows how the similarity is computed between these two vectors using cosine.

$$sim(x,\, y) \;=\; cos(\vec{x},\, \vec{y}) \;=\; \frac{\vec{x} \bullet \vec{y}}{\|\vec{x}\| \times \|\vec{y}\|} \;=\; \frac{\sum_{s \in r_{xy}} r_{xs}\, r_{ys}}{\sqrt{\sum_{s \in S_{xy}} r_{xs}^2}\, \sqrt{\sum_{s \in S_{xy}} r_{ys}^2}}, \quad (2.7)$$

Despite this usefulness, cosine similarity does not take into account differences between each rating average and variance, needing an additional normalization for that. These undesirable effects are not present in the Pearson correlation, given by equation 2.8.

$$sim(x,\, y) \;=\; \frac{\sum_{s \in S_{xy}} (r_{xs} \;-\; \bar{r}_x)\,(r_{ys} \;-\; \bar{r}_y)}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} \;-\; \bar{r}_x)^2 \sum_{s \in S_{xy}} (r_{ys} \;-\; \bar{r}_y)^2}} \tag{2.8}$$

Model-based

Model-based algorithms use the evaluation set to learn a model that can be used to predict the rating that a user u would give to an item i. Usually, machine learning, data mining, dimensionally reduction and many other techniques are applied to reach that goal and improve performance beyond regular memory-based methods achieved.

One of the most recognized model-based techniques was proposed by FUNK (2011) during the Netflix Prize, a competition hosted in 2006 by Netflix[4] with the aim of improving the state-of-the-art Recommendation Systems. This model relied on a concept named latent factors, which assumes that there is some latent knowledge within the user-item matrix that can be extracted in order to explain the user relationship with the system items. For instance, items' latent factors, in a movie recommendation system context, can explain whether a certain movie is geared towards male or female spectators, or whether characters are better constructed or not. In contrast, users' latent factors in the same context may measure how a user identifies herself with the corresponding latent factor characteristic (KOREN *et al.*, 2009). Figure 2.1 illustrates this concept.

Funk's proposal is one of the first in a class of techniques known as matrix factorization. His method was later named Regularized Singular Value Decomposition by PATEREK (2007). It consists of associating a latent variable vector $q_i \in \mathbb{R}^f$ to each item i and, in the same way, associating a latent variable vector $p_u \in \mathbb{R}^f$ to each user u. Vectors initialization is given by Singular Value Decomposition (SVD) dimensionally reduction technique, where the user-item matrix $m \times n$ is decomposed in PSQ^T, being that P is $m \times m$, i.e., latent factors' matrix of users, Q is $n \times n$, i.e., latent factors' matrix of items, and S the singular values $m \times n$.

Therefore, rating prediction may be performed according to equation 2.9, where q_i and p_j are the K-dimensional latent factors' vectors and the dot product $q_i^T p_u$ between latent factors of user u and item i predicts the rating that user u would give to item i.

$$\hat{r}_{ui} = q_i^T p_u \tag{2.9}$$

The cost function proposed in order to perform training for the model is the squared error between its prediction and the actual rating value. Equation 2.10

[4]http://www.netflix.com

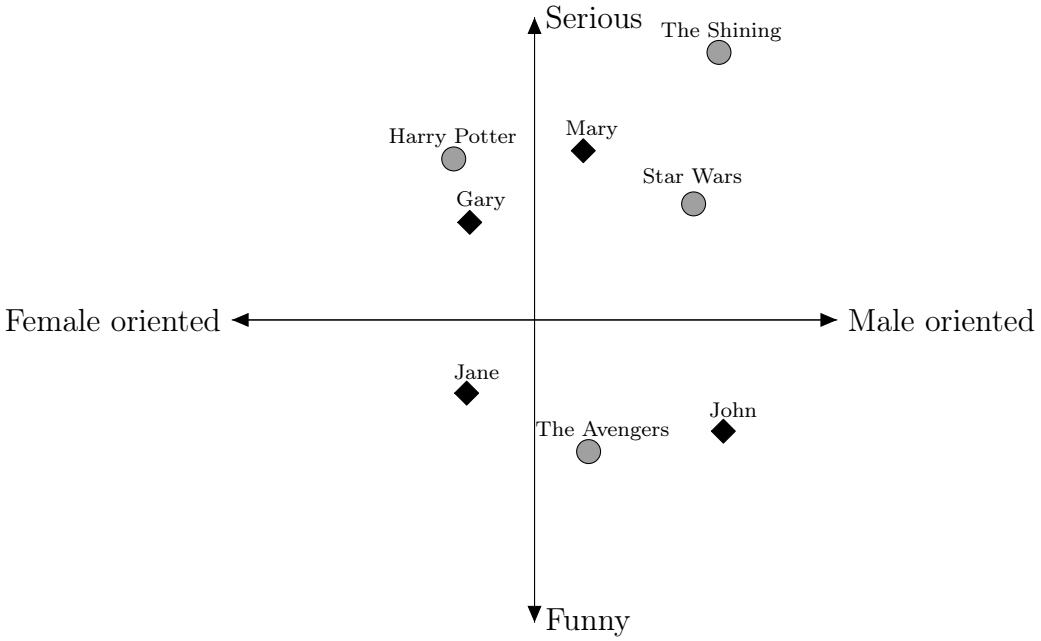


Figure 2.1: Illustration based on latent variables interpretation in a movie dataset context (KOREN *et al.*, 2009).

shows the function that should be minimized during the learning step, where λ is a regularization constant used to prevent overfitting.

$$\min_{q*,p*} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2) \tag{2.10}$$

The model may be trained using a stochastic gradient descent algorithm, i.e., iterating over the training set ratings r_{ui} and calculating the error with respect to the predicted rating r'_{ui}. Equation 2.11 illustrates how it works.

$$e_{iu} = r_{iu} - \hat{r}_{iu} \tag{2.11}$$

Parameters q_i and p_u are adjusted for each training example according to the equations 2.12 and 2.13, where γ is the learning rate, which controls the size of the step for each iteration during optimization.

$$q_{ik} += \gamma(e_{iu}p_{uk} - \lambda q_{ik}) \tag{2.12}$$

$$p_{uk} += \gamma(e_{iu}q_{ik} - \lambda p_{uk}) \tag{2.13}$$

Following this work, the widespread adoption of latent factor models led to the development of various enhancements that built upon the initial proposals made by Funk (KOREN *et al.*, 2009; PATEREK, 2007; TAKÁCS *et al.*, 2008). For instance, PATEREK (2007) proposed adding user and item biases to the cost function of

the model, naming it Improved Regularized Singular Value Decomposition. This approach explores biases in recommender systems that are independent of the interactions within the system itself, e.g., an item with a good critical reception may receive higher ratings from users, while some users may be more or less critical, etc.

Given this rationale, the system aims to determine the portion of the values of $q_i^T p_u$ that can be explained by user or item biases relative to the global average rating μ. This process is illustrated in equation 2.14, where b_u and b_i represent the biases associated with the user and the item, respectively.

$$b_{ui} = \mu + b_u + b_i \tag{2.14}$$

As an example, suppose a movie recommender system needs to predict the rating that a user named Jane would give to the movie *Wonder Woman*. Assume the global average rating μ is 3.5 stars. Since *Wonder Woman* is a well-received movie, it tends to be rated 0.3 stars above the average. Jane is a rather uncritical user who generally rates movies 0.5 stars above the average. Thus, the predicted rating Jane would give to Wonder Woman is 4.3 stars (3.5 + 0.3 + 0.5). Therefore, equation 2.9 is extended to equation 2.15 by incorporating information about user and item biases.

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \tag{2.15}$$

With the addition of the new parameters to the model, the updated cost function is given by:

$$\underset{b_*, q_*, p_*}{\arg\min} \sum_{(u,i,r) \in R} (r_{ui} - \mu - b_u - b_i - q_i^T p_u)^2 + \lambda(\mu + b_u + b_i + ||q_i||^2 + ||p_u||^2). \tag{2.16}$$

The cost is also optimized using gradient descent in the same manner proposed by FUNK (2006). Accordingly, the parameters are updated as follows:

$$b_i += \gamma(e_{ui} - \lambda b_i) \tag{2.17}$$

$$b_u += \gamma(e_{ui} - \lambda b_u) \tag{2.18}$$

$$q_i += \gamma(e_{ui} p_u - \lambda q_i) \tag{2.19}$$

$$p_u += \gamma(e_{ui} q_i - \lambda p_u) \tag{2.20}$$

More recetly, artificial neural networks have been successfully applied in collaborative filtering to model the interaction between users and items. One of the most widely used models is Neural Collaborative Filtering, proposed by HE *et al.* (2017), which is a framework that aims to capture the non-linear relationship between these two entities. This work proposed two models, as well as a combined model that in-

tegrates both, providing the last neural network layer with more information about the relationship between a user and an item.

The first model is a general form of Neural Collaborative Filtering that uses a multilayer perceptron to learn the non-linear relationship between two vectors, each representing a user u and an item i. The unique identifiers of the user and item are used as input features in the input layer, where they are transformed into sparse binary vectors and encoded. Following this layer, there is an embedding layer for each vector, fully connected to the sparse layer, which represents the latent variables of the user and item. These embedding layers are then fed into a multi-layer neural network, which maps the latent variables to predicted ratings. This model can be seem in Figure 2.2.

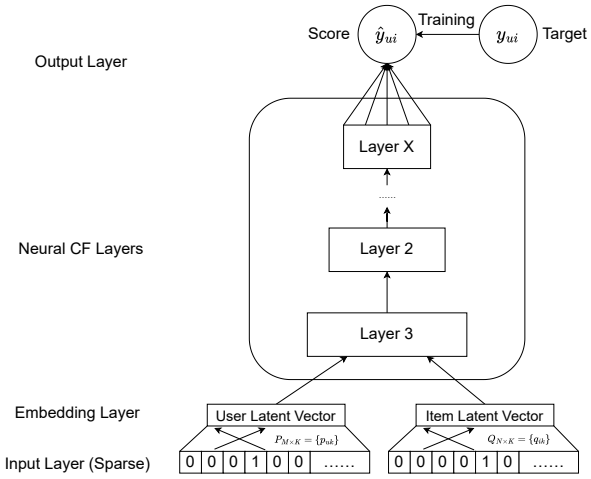


Figure 2.2: Multilayer Perceptron of Neural Collaboraive Filtering (HE *et al.*, 2017).

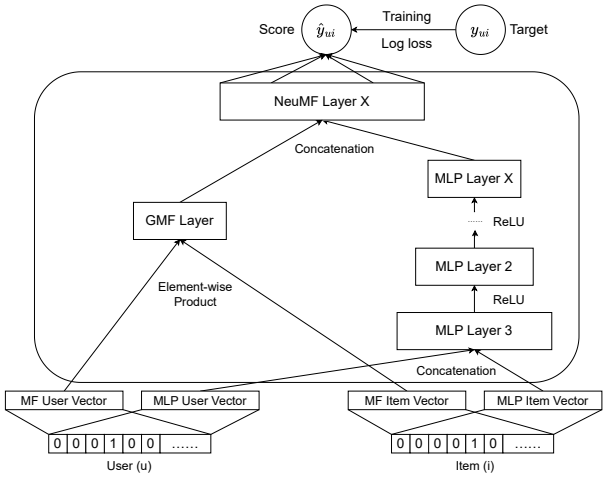


Figure 2.3: Fused model of Neural Collaboraive Filtering (HE *et al.*, 2017).

The second model is a different instantiation of the framework that emulates matrix factorization techniques, termed Generalized Matrix Factorization by the

authors. In this method, the two embedding layers are combined through element-wise multiplication to generate a new vector, which is subsequently fed into a neural network for further processing. This model and the fused model are presented in Figure 2.3.

2.1.2 Datasets

Several datasets have been used for collaborative filtering research over the years. Among them, the MovieLens 100k dataset, collected from the MovieLens[5] (HARPER & KONSTAN, 2015) online recommender system, is one of the most common. It is composed of a total of 100,000 ratings given by 943 users to 1682 movies and the ratings are integers varying from 1 to 5. Besides the ratings, there is additional information regarding users, e.g., age and gender, and movies, e.g., title and release date, which are usually discarded to not mischaracterize collaborative filtering context in the proposal.

It should be noted that this data has been cleaned up, removing users who rated less than twenty movies or did not have complete demographic information. Another important fact is that in the MovieLens system, users are required to rate at least fifteen movies at registration; that is, the rating is given after item consumption. In Figure 2.4, MovieLens 100k histogram.

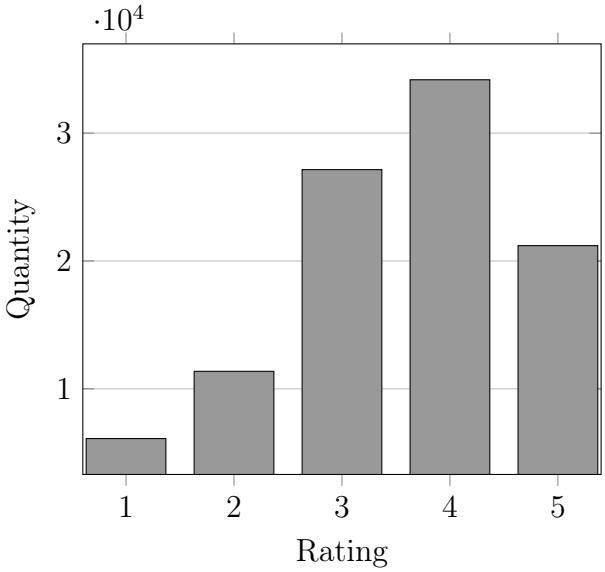


Figure 2.4: The MovieLens 100k dataset histogram.

Another common dataset is the R3 Yahoo! Music ratings for User Selected and Randomly Selected songs, version 1.0, available through Yahoo! Webscope data sharing program[6]. This dataset was collected from two different sources: normal

[5]https://movielens.org/
[6]http://research.yahoo.com/Academic_Relations

interaction with Yahoo! Music services and randomly chosen songs for a survey conducted by Yahoo! Research, which consists of 365,704 ratings given by 15400 users to 1000 songs, with the ratings varying between 1 and 5. In Figure 2.5, R3 Yahoo! Music histogram.

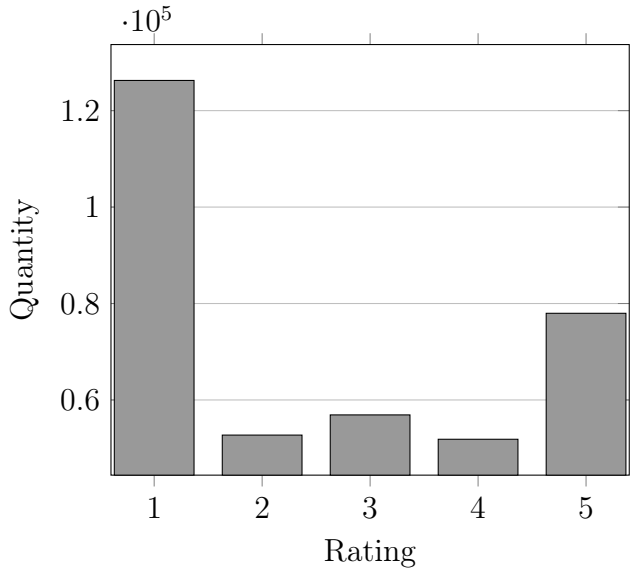


Figure 2.5: The Yahoo! Music dataset histogram.

Additionally, in an e-commerce context, there is the Amazon review dataset (XU *et al.*, 2013), a dataset crawled from Amazon.cn until August 2012. This data is composed of 1,205,125 ratings from 645,072 users on 136,785 items. The distinguishing feature of this dataset is that it labels users as either attackers or genuine users. Labeled data consists of 60,000 ratings from 4,902 users on 21,394 items, with ratings ranging from 1 to 5. In Figure 2.6, we can see the Amazon review histogram.

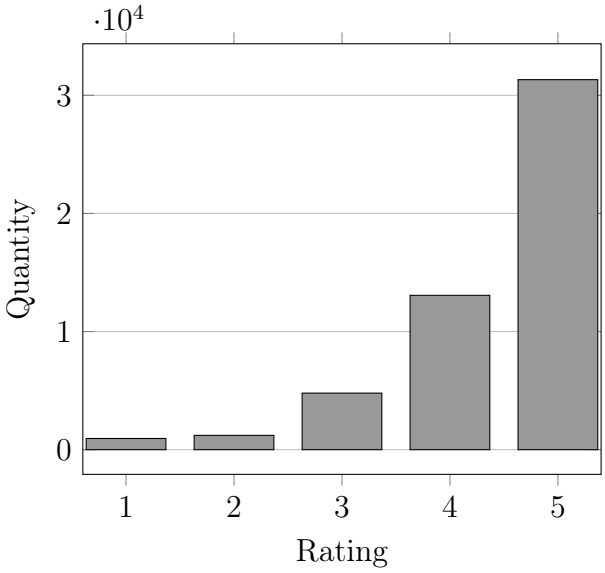


Figure 2.6: The Amazon review dataset histogram.

2.2 Shilling Attack

Collaborative filtering proposals avoid the use of additional information about users or items, which makes this class of algorithms very dependent on data quality in order to provide good recommendations to its users (GUNES *et al.*, 2012). Since we have this clear dependency, it is not unrealistic to state that a system based on collaborative filtering may be vulnerable to some kind of attacks. In the past two decades, various studies have emerged to analyze different aspects of attacks in recommender systems, making the interest on this topic grow and become more relevant (BHAUMIK *et al.*, 2006; MOBASHER *et al.*, 2007a; BURKE *et al.*, 2011).

With a wide range of products available in e-commerce stores, it's natural for companies to want their products to be more often recommended to potential buyers than their rival ones. However, while some companies seek to improve their products to reach their goals, others may go for a more dishonest decision. That said, malicious users or companies may develop and insert malicious profiles into a system in order to manipulate its recommendations and promote their product, demote a rival product, or just create chaos within the system. These attacks are known as shilling attacks (LAM & RIEDL, 2004), and are performed by the injection of fake profiles with similar rating patterns to the users of the real system. These attacks vary among different approaches, intents, and information required from the system itself. There are also adversarial attacks, which focus on manipulating the data stationary assumption by learning and injecting additive perturbations into the system, changing its recommendations in the process (ANELLI *et al.*, 2022).

Therefore, it is important to handle shilling attacks in order to achieve the success of collaborative filtering schemes. In the next subsection, we will present attack

intents, the general form of a malicious profile, and the main literature on classic attack models.

2.2.1 Shilling Attack Types

Shilling attack models are, usually, classified according to two dimensions: attack intent and the amount of knowledge necessary to perform the attack (MOBASHER *et al.*, 2007a). According to GUNES *et al.* (2012), attacks can also be classified by rating types, application, and collaborative filtering algorithms, while LAM & RIEDL (2004) uses three more different dimensions beyond the two previously mentioned: targets, cost, algorithm dependence, and detectability. However, we will primarily focus on the MOBASHER *et al.* (2007a) categorization since it is the most used in the literature.

Despite attack goals being related to the manipulation of the recommendation results of a system, different attacks may have different intentions behind them. The most common intents of an attack model are *push* and *nuke* (LAM & RIEDL, 2004). Push attacks refer to these attacks that seek to *push* an item to be recommended to more users inside the system, i.e., increase this particular item's popularity, while nuke attacks refer to attack models that seek to make an item to be recommended to fewer users, i.e., decrease this particular item's popularity (MOBASHER *et al.*, 2007b).

According to the amount of knowledge needed to mount the attack, approaches may be classified as a low-knowledge attack, a high-knowledge attack, and an informed attack. Low-knowledge indicates that the attack requires system-independent knowledge, i.e., knowledge that might be acquired from public sources, while high-knowledge requires much more detailed information regarding the system and its rating distribution (MOBASHER *et al.*, 2007b). Lastly, informed attacks require the most information possible regarding the system, needing high amounts of domain knowledge in order to mount an effective attack (BURKE *et al.*, 2011). Commercial recommender systems often make specific information regarding their ratings database available to the user, such as the average rating of an item. The more information available to mount the attack, the more likely the attack will be to succeed.

2.2.2 Mounting Attacks

Generally, a shilling attack is performed by the injection of malicious profiles into a system in order to mislead it and achieve its intent. According to GUNES *et al.* (2012), this can be done by malicious users or companies acting as authentic users, creating multiple fake profiles on the recommender system she wants to attack, and

sending the fake information to its database. Each attack profile contains biased user preferences about items and is denoted by a vector. The number of malicious profiles injected is denoted as a proportion of the number of profiles in the system before an attack, e.g., if the system contains 1,000 user profiles, an attack of 1% will be composed of 10 malicious profiles (MOBASHER *et al.*, 2007a). Figure 2.7 shows a general form of an attack profile, which consists of four sets of items: selected items, filler items, unrated items, and target item (BHAUMIK *et al.*, 2006; MOBASHER *et al.*, 2007a).

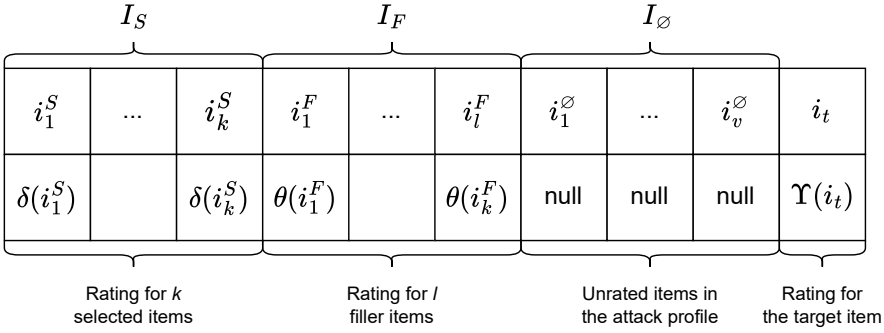


Figure 2.7: A general form of an attack profile (GUNES *et al.*, 2012).

Selected Items

This set I_S is responsible for forming the characteristics of the attack. Frequently, the items contained in it are selected carefully to make the malicious profile similar to the real profiles the attacker wants to mislead. Its ratings are assigned using a rating function δ.

Filler Items

This set I_F obstructs the detection of the attack by preventing the profile from containing only the items used to characterize it. Usually, the items contained in it are selected randomly and their ratings are assigned using a rating function θ. Filler item set size can be considered a parameter and is commonly reported as a proportion over the size of I, i.e., the total number of items (MOBASHER *et al.*, 2007a).

Unrated Items

This set $I_\varnothing$ represents the remaining items in the system, that are left unrated in the malicious profiles crafted.

Target Item

Normally, a single target item i_t is chosen, i.e., the item that the attacker wants to nuke or push, and its ratings are assigned using a rating function Υ. Recently, some works proposing multiple target items emerged as well (SEMINARIO & WILSON, 2014a).

2.2.3 Attack Models

Several attack models have been proposed over the years, and almost all of them rely on simple heuristics to mount their attacks. Most of them can be classified as low-knowledge attacks in the amount of knowledge dimension, despite a few more effective as a whole being classified as high-knowledge attacks. Regarding intent, some attacks are tailored to be used only for push or nuke attacks. However, others can actually be used for both intents. Table 2.2 shows attack models classified according to intent and required knowledge.

Table 2.2: Attack models classified according to intent.

Attack model	Intent		Knowledge	
	Push	Nuke	Low	High
Random	✓	✓	✓	
Average	✓	✓		✓
Bandwagon	✓		✓	
Segment	✓		✓	

Below we will present the most common attack models used in the literature: random attack, average attack, bandwagon attack, and segment attack.

- **Random:** Also known as RandomBot attack (LAM & RIEDL, 2004), this attack model is easy to implement, despite not being very effective (MOBASHER *et al.*, 2007a,b). Attacks crafted using this model randomly choose filler items around the system's overall mean and assign $r_{\max}$ or $r_{\min}$, depending on the intent, push, or nuke, respectively, to the target item.

- **Average:** Also known as AverageBot attack (LAM & RIEDL, 2004), this attack required high-level knowledge, such as, all system items mean and standard deviation, in order to be implemented (MOBASHER *et al.*, 2007b). Using this model, attacks can be crafted by randomly choosing filler items around the item mean and assigning $r_{\max}$ or $r_{\min}$, depending on the intent, push, or nuke respectively, to the target item.

- **Bandwagon:** This low-knowledge attack model aims to increase the similarity between fake profiles and the real ones through well-known popular items (O'MAHONY *et al.*, 2005). That said, the selected item set for its model is filled with popular items with $r_{\max}$ as the assigned rating value; the filler item set is filled with randomly chosen items around the system's overall mean, and the the target item is set to $r_{\max}$ since it is a push-only attack model.

- **Segment:** Also known as segmented attack, it aims to target a particular group of users that would be likely to buy a particular niche product. The selected items set is filled with high ratings on items that this group of users would like while the filler items set is filled with randomly chosen items with low rating values. The target item is set to $r_{\max}$ since it is a push-only attack model.

Table 2.3 shows a summary of the attack models presented in this section.

Table 2.3: Attack models summary.

Attack model	I_S		I_F		$I_\varnothing$	i_t
	Items	Rating	Items	Rating		
Random	Not used		Random	System mean	$I - I_F$	$r_{\max}$ / $r_{\min}$
Average	Not used		Random	Item mean	$I - I_F$	$r_{\max}$ / $r_{\min}$
Bandwagon	Popular items	$r_{\max}$	Random	System mean	$I - \{I_F \cup I_S\}$	$r_{\max}$
Segment	Segment items	$r_{\max}$	Random	$r_{\min}$	$I - \{I_F \cup I_S\}$	$r_{\max}$

2.2.4 Metrics

To evaluate the performance of attack models, several metrics have been used over the years. The most commonly employed in this area are three key robustness metrics: Prediction shift, hit ratio, and average rank (BURKE *et al.*, 2011; SEMINARIO & WILSON, 2014b).

Consider U_T as a set of users in the test data and I_T as a set of items in the same data. Prediction shift is denoted by:

$$\Delta_{u,i} = p'_{u,i} - p_{u,i}, \tag{2.21}$$

such that p and p' are the pre-and-post-attack predicted ratings, respectively, for each user u and item i pair (u, i). We can compute the average prediction shift for an item i over all users, as shown:

$$\Delta_i = \sum_{u \in U_T} \Delta_{u,i} / |U_T|, \tag{2.22}$$

and for all items tested, as shown:

$$\bar{\Delta} = \sum_{i \in I_T} \Delta_i / |I_T|. \tag{2.23}$$

This metric denotes the difference between the model before and after an attack, e.g., in push attacks, if the value is positive, it means that the attack was successful and the item is more positively rated (BURKE *et al.*, 2011).

Despite being a good indicator of the attack effect on an item prediction, prediction shift will not give any information regarding performance in the recommendation list. To get an insight into it, another metric can be used: Hit Ratio. Let R_u be the top-N recommendation list for a user u and i be the target item, Hit Ratio (HR) is denoted by equation 2.24, where H_{ui} is a scoring function that returns 1 if i appeared in R_u for user u, and 0 otherwise.

$$HR_i = \sum_{u \in U_T} H_{ui} / |U_T| \tag{2.24}$$

Usually, it is computed as the Average Hit Ratio ($\overline{HR}$), which is the sum of the hit ratio after an attack for every target item i averaged over the number of target items selected. Equation 2.25 shows how it is computed.

$$\overline{HR} = \sum_{i \in I_T} HR_i / |I_T| \tag{2.25}$$

Higher values of Hit Ratio, i.e., close to 1, indicate that our target item is on the recommendation list of many users in the test set, while low values, e.g., close to 0, indicate that our target item is on the recommendation list of only a few users in the test set. Push attacks aim to achieve higher values of Hit Ratio, while nuke attacks target lower values of this same metric.

More recently, the target item Rank in the recommendation list after an attack began to be used as a robustness metric (SEMINARIO & WILSON, 2014b). The Average Rank is the sum of each target item's rank averaged over the number of target items selected.

2.3 Artificial Neural Networks

Artificial neural networks are a learning model inspired mainly by the observation of the human biological learning model. It consists of interconnected units, namely neurons, where each neuron's output is the direct input of another neuron.

The origins of this model date to the early 1940s, when MCCULLOCH & PITTS (1943) were the first that tried to model an artificial neuron. However, the first

successful model came with ROSENBLATT (1958), which modeled a simple artificial neuron for binary classification problems. In Figure 2.8, we show an example of this model, named Perceptron.

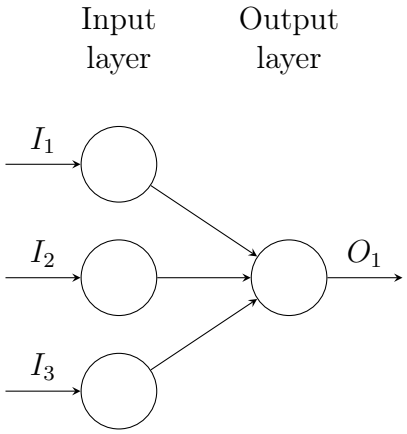


Figure 2.8: An example of a perceptron.

The Perceptron model consists of receiving a vector of real-valued inputs x_1 to x_n, calculating a linear combination of these input vectors. Therefore, if the resulting value is above a certain threshold, the output $o(x_1, ..., x_n)$ will be 1 and -1 otherwise. Equation 2.26 illustrates that process, where weights w_i determine the input x_i contribution level to the output value.

$$o(x_1, ..., x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n > 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.26}$$

Usually, we can simplify equation 2.26 by adding a constant input $x_0 = 1$, called bias, allowing us to write this equation as $\sum_{i=0}^{n} w_i x_i > 0$, or in a vectorized form $\vec{w}.\vec{x} > 0$ (MITCHELL, 1997). The activation function is then written as:

$$o(\vec{x}) = sgn(\vec{w}.\vec{x}),$$

where

$$sgn(y) = \begin{cases} 1 & \text{if } y > 0 \\ -1 & \text{otherwise.} \end{cases}$$

The perceptron model is trained by choosing values for weights $w_0, ..., w_n$. Many ways may be used to achieve that goal, MITCHELL (1997) highlights initialization of weights by random values and iterating over the training set modifying weights each time one training example is missclassified. Equations 2.27 and 2.27 shows this process, where t is the actual value for the training example output, o is the output value predicted for the same example, and γ is the learning rate that controls the size of the step for each iteration during the training process.

$$\Delta\ w_i = \gamma(t - o)x_i \tag{2.27}$$

$$w_i \leftarrow w_i + \Delta\ w_i \tag{2.28}$$

As previously stated, a perceptron model is unable to perform well enough on non-linear data, since it is designed to approximate only linearly separable functions. Despite this, if we use multiple neurons, interconnecting them, we may learn more complex functions. Figure 2.9 shows an example of a multilayer neural network, also known as a multilayer perceptron.

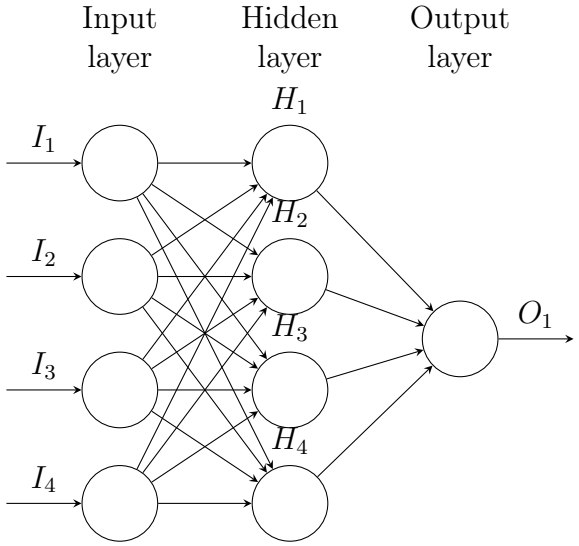


Figure 2.9: An example of multilayer artificial neural network.

Besides adding neurons to create a more complex architecture, it is important to use a non-linear activation function since the activation function used by the perceptron would still only be able to approximate linear functions independently of the network architecture. That said, we could use a non-linear function such as sigmoid, as shown in equation 2.29.

$$f(z) = \frac{1}{1 + exp(-z)} \tag{2.29}$$

The training algorithm is also modified to perform a more efficient approximation of non-linear functions. The most popular algorithm used, namely Backpropagation, minimizes the cost function by computing the gradient of it for each weight and updating it (MITCHELL, 1997). Several gradient descent types can be used, such as Adam (KINGMA & BA, 2014), and Root Mean Square Prop (RMSProp) (GRAVES, 2013), among others.

Artificial neural networks need to rely on regularization techniques in order to avoid overfitting. One of these techniques is dropout, which consists in dropping out random neurons from a neural network to deal with that issue. During the training

phase, each neuron from the network is retained or dropped with a fixed probability p independently of the other neurons. In general, p is set in 0.5 to be close to the optimal value associated with a wide range of applications of the network for different tasks. Thus, during the test time the neuron dropped is always present, and the weights are multiplied by the probability p (SRIVASTAVA *et al.*, 2014).

2.3.1 Convolutional Networks

Fully connected architectures may be sufficient for some problems; however, the topology of the input is ignored. Convolutional Neural Networks address this problem for data, such as images, that have a strong 2D local structure, allowing the network to extract and combine these features (LECUN & BENGIO, 1998). Despite being initially most applied to image problems, convolutional neural networks are currently being used in diverse areas such as Speech Processing (PALAZ *et al.*, 2013) and Natural Language Processing (KIM, 2014).

Convolutional networks are very similar to the regular multilayer perceptron in their general structure. However, their layers are slightly different from their interconnected counterparts. There are various proposed architectures, but most of them rely on three types of layers: convolutional, pooling, and standard fully interconnected.

A convolutional layer consists of several convolutional kernels used to compute different feature maps, this process is done by the convolution of all spatial locations of the input with the learned kernel and applying a non-linear activation function. Equation 2.30 shows how to calculate feature value at (i, j) in the feature map k of the l-th layer (GU *et al.*, 2018), where w_k^l are the vector of weights, b_k^l is the bias term for filter k of layer l and x is the input on location (i, j) of layer l.

$$z_{i,j,k}^l = w_k^{l^T} x_{i,j}^l + b_k^l \tag{2.30}$$

Figure 2.10 shows how convolution works using a 3×3 kernel applied to a 5×5 input padded with a 1×1 border of zeros using 2×2 strides. Note that strides can be seen as a form of subsampling and padding allows this kernel to be applied to the wedges of the data (DUMOULIN & VISIN, 2016).

Besides convolutional, we could also have a deconvolutional layer or transposed convolution layer, which works like a backward pass of the convolutional layer. Thus, it associates a single activation with multiple output activations, upsampling the input by a factor of the stride value with padding before performing the convolution on it (GU *et al.*, 2018).

Being $a(.)$ the non-linear activation function, we can compute the activation of value $a_{i,j,k}^l$ of the obtained convolutional feature $z_{i,j,k}^l$ by equation 2.31.

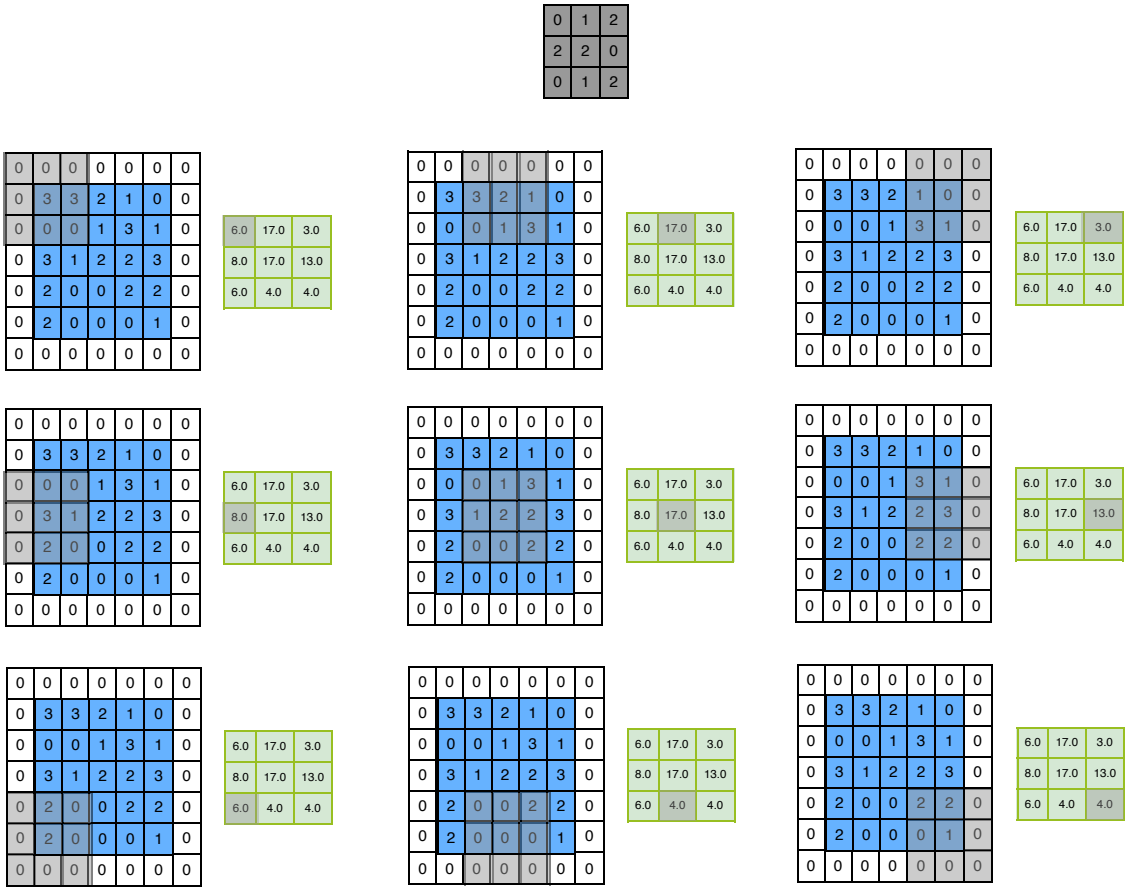


Figure 2.10: Example of convolution using 3×3 kernel applied to a 5×5 input padded with a 1×1 border of zeros using 2×2 strides (DUMOULIN & VISIN, 2016). Note that the white area around the original data (blue) is the padding added in order to apply the kernel (gray) using 2×2 strides.

$$a^l_{i,j,k} = a(z^l_{i,j,k}) \tag{2.31}$$

The regular sigmoid function can be used as an activation function for convolutional neural networks; however, as the number of layers increases, this function suffers from a saturation problem, where the gradient starts to get close to zero (LECUN *et al.*, 2012). Thus, currently, the most used functions include rectified linear unit (ReLU) and leaky rectified linear unit (Leaky ReLU) (MAAS *et al.*, 2013) which are computed faster and do not suffer from this problem. Equations 2.32 and 2.33 show these functions applied to a convolutional network.

$$a^l_{i,j,k} = \max(z^l_{i,j,k}, 0) \tag{2.32}$$

$$a^l_{i,j,k} = \max(z^l_{i,j,k}, 0) + \lambda \min(z^l_{i,j,k}, 0) \tag{2.33}$$

Between two convolutional layers, pooling layers are commonly used to reduce the resolution of feature maps in order to achieve shift-invariance. Equation 2.34 shows how it works, where *pool* is the pooling function, $\mathbf{R}_{ij}$ is the local neighborhood

around the location (i, j).

$$y^l_{i,j,k} = pool(a^l_{i,j,k}), \forall (m, n) \in \mathbf{R}_{ij} \tag{2.34}$$

On top of a stack of these layers, it is usually placed an interconnected layer and, afterward, the output layer, just like a regular neural network (GU *et al.*, 2018).

2.3.2 Autoencoders

Autoencoders are neural networks where $x^{(i)} = y^{(i)}$, which means that training examples and labels are the same. This type of network is trained in an unsupervised way in order to learn a latent representation of the data fed to it. Figure 2.11 shows an example of an Autoencoder with one hidden layer, where the leftmost layer is the input layer, the center layer is the hidden layer, and the rightmost layer is the output layer.

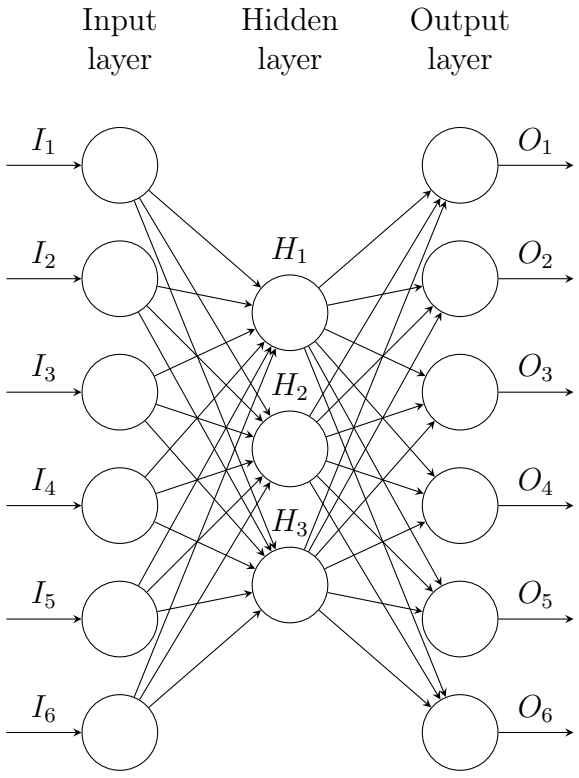


Figure 2.11: An example of the autoencoder.

More formally, an autoencoder may be seen as a non-linear generalization of Principal Component Analysis (PCA). It consists of a multilayer encoder capable of transforming high dimensional data into low dimensional code, and a decoder, with aims to reconstruct the original data from the code, ultimately learning the identity function (HINTON & SALAKHUTDINOV, 2006). The smaller representation learned from data can be seen as latent variables.

The encoder layer consists in

$$y = f_\theta(x) = s(Wx + b).$$

Parametrized by $\theta = W.b$, where W is its $d' \times d$ weight matrix and b is its bias unit. The decoder layer can be expressed by

$$z = g_\theta(y) = s(W'y + b').$$

Similarly, parametrized by $\theta = W'.b'$.

The parameters of the model can be optimized by minimizing the *average reconstruction error* through the *backpropagation* algorithm, like in a regular neural network:

$$\theta^*, \theta'^* = \underset{\theta,\theta'}{\operatorname{argmin}} = \frac{1}{n}\sum_{i=1}^{n}\left(x^{(i)}, z^{(i)}\right)$$

$$= \underset{\theta,\theta'}{\operatorname{argmin}} = \frac{1}{n}\sum_{i=1}^{n}\left(x^{(i)}, g_{\theta'}(f_\theta(x^{(i)}))\right)$$

where L is the cost function such as *squared error*, as show in equation 2.35, or *cross entropy cost*, as shown in equation 2.36 (VINCENT *et al.*, 2008).

$$L(x, z) = ||x - z||^2 \tag{2.35}$$

$$L(x, z) = -\sum_{k=1}^{d}[x_k \log z_k + (1 - x_k)\log(1 - z_k)] \tag{2.36}$$

2.3.3 Variational Autoencoders

A standard autoencoder can efficiently learn a smaller representation of the input data and reconstruct it. However, it is very limited when it comes to generating new data since each entry in the original data is mapped to a single data point in the latent space. A variational autoencoder (KINGMA & WELLING, 2013) is a generative model designed to overcome this drawback, allowing a sampling process directly from its learned distribution without the need to use Markov chain Monte Carlo methods as performed in (BENGIO *et al.*, 2013).

Although the mathematical reasoning behind variational autoencoders is slightly different from regular ones, it can be viewed as such, since it is also composed of an encoder and a decoder. Its encoder, however, is modified to output a vector of means μ and a vector of standard deviations σ rather than an encoding vector like its standard counterpart. Figure 2.12 shows an example of a variational autoencoder.

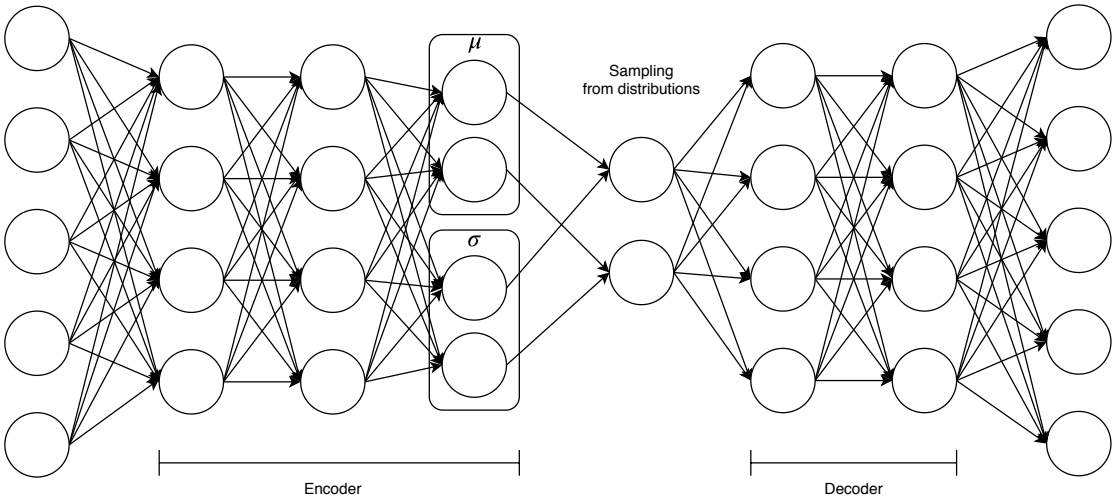


Figure 2.12: An example of variational autoencoder. Note that the encoder defines the latent distributions μ and σ, while the decoder samples from them instead of reconstructing the code like a regular one.

Another important difference lies in the cost function. Besides the regular cost present in the standard autoencoder consisting of the reconstruction error, variational autoencoder, in order to regularize the latent space and be able to generate meaningful new data, incorporates a new term named Kullback–Leibler divergence (KL divergence), which is, intuitively, a standard measure of how a distribution diverges from another (KULLBACK & LEIBLER, 1951). Equation 2.37 shows the variational autoencoder cost function.

$$\mathcal{L}(\Theta, \phi) = -\mathbb{E}_{z \sim q_\phi(z|x_i)}[\log_{p_\Theta}(x_i \mid z)] + \mathbb{KL}(q_\Theta(z \mid x_i) \mid p(z)) \tag{2.37}$$

The first term represents the reconstruction error, which encourages the encoder to accurately reconstruct the original data. The second term is the KL divergence, which measures the distance between encoder output $q_\Theta(z \mid x_i)$ and a given $p(z)$ distribution. Please note that large values of this term may indicate strong regularization effects, which may be undesirable (HOFFMAN & JOHNSON, 2016). Typically, the Gaussian distribution is chosen as $p(z)$.

KL divergence is used to ensure the regularity of the latent space by making sure that, once decoded, every data point may be meaningful (HOFFMAN & JOHNSON, 2016). Let u and v be users of a system. It is possible to assume that u likes horror movies, while v likes thrillers based on their ratings. Without KL divergence as a regularization term, these two users may be mapped into two completely distant probability distributions, and then, if a data point is sampled in latent space between these two distributions, we may end up with a meaningless user after decoding, since both distributions would be too apart from each other. Therefore, KL divergence stimulates the returned distributions to overlap in the latent space, making them

closer to each other in order to address these cases and make sure that meaningful data can be sampled thereafter. This process is carried out by feeding the decoder with a new vector sampled from the latent space, which is then mapped to the same format as the data used to train the model, thereby mimicking the source data.

2.3.4 Generative Adversarial Networks

Generative adversarial networks (GAN) (GOODFELLOW *et al.*, 2014) are a type of generative model based on an adversarial process, where two neural networks, called the generator and the discriminator, are trained simultaneously. The generator network creates synthetic data from random noise without explicit knowledge of the original data distribution. Its goal is to produce data as similar as possible to the real data and to trick the discriminator into classifying the generated data as real. The discriminator network evaluates data samples and attempts to distinguish between real data and data generated by the generator, outputting a probability indicating whether a given sample is real or fake. Given this architecture, both networks compete against each other during the training process to improve their performance, which defines the adversarial process.

In order to train this model, let p_g denote the distribution of the samples generated by $G(z;\theta_g)$, where G is the generator with parameters θ_g, and z is the input noise. G is denoted by a differentiable function represented by a neural network. The discriminator is a second neural network, denoted as $D(x;\theta_d)$, with parameters θ_d, and x is the sample to be evaluated. D outputs a scalar value representing the probability that a sample x belongs to the real dataset rather than fabricated by G. Thus, D is trained to maximize the probability of correctly classifying both real and generated samples, while G is trained simultaneously to minimize $\log(1 - D(G(z)))$. The full cost function can be seen in equation 2.38, where D and G play a two-player minmax game.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[1 - \log D(G(z))] \tag{2.38}$$

To better grasp this architecture, consider an analogy where the generator network is akin to a forger attempting to create counterfeit items that closely mimic real ones. Meanwhile, the discriminator network functions as an evaluator tasked with identifying and distinguishing between the authentic items and those produced by the forger.

Chapter 3

Malicious Profiles Using Generative Models

This chapter will address the main issues with proposed attack models in shilling attacks. Initially, we will elucidate the most common flaws found in the literature regarding the most widely used attack models. Later, we will review the key works that propose solutions to mitigate these issues. Finally, we will introduce a new attack model based on a variational autoencoder, which aims to address many of these problems, and perform an experimental evaluation to validate this proposal.

3.1 Attack Models Issue

As stated in Chapter 2, recommender systems are prone to attacks either to push or nuke a product, or just to create chaos in the system and disrupt it. So, it is very important to understand these attacks and study ways to better protect different systems. One way to do this is to propose novel attack models and investigate their impact on different recommendation algorithms in order to study new ways to improve prediction algorithms robustness (O'MAHONY *et al.*, 2002) and detection of more complex attack approaches.

Several attack models have been proposed and evaluated over time. However, most of them were designed to attack memory-based recommender systems, where users are grouped by their similarities and the predictions are given by the weighted average over the ratings of the most similar users (RICCI *et al.*, 2011). That is, memory-based algorithms have many differences from model-based ones, indicating that attacks designed for memory-based algorithms may not work correctly on model-based algorithms. In addition, GUNES *et al.* (2012) identified a lack of works proposing attacks on model-based algorithms such as Regularized SVD.

Another issue regarding current attack models is that most attack models need to

inject too many profiles to have the desired effect on a system. In a real scenario, it is very unrealistic to inject a lot of profiles, e.g., 10% of the database, into a system. The attack may be easily detected or the systems may have countermeasures to ignore ratings given by spamming-like profiles; that is, the attack becomes too easy to spot (WILLIAMS *et al.*, 2007).

There is also an issue related to the rating distribution between real users and malicious profiles. As stated in CHEN *et al.* (2019), attack models rating patterns usually differ by a large margin when compared with real users, making these attacks more easily to be detected by many techniques.

Considering the points mentioned earlier, developing an attack model that addresses all the listed issues is crucial for better understanding how to protect various systems more effectively. This is important whether the focus is on the robustness of prediction algorithms or the accuracy of detection algorithms.

3.2 Related Work

Several works proposed solutions to some of the issues reported above. RAY & MAHANTI (2009) proposed different strategies to mount attacks against user-based and item-based collaborative filtering algorithms. These strategies consist of varied methods to perform filler item selection in classic attack models using target item rating distribution, where the target item can be categorized into two distinct categories:

- T_L: Most of the ratings are located at the lower end of the rating range (items with 60% in the lower range).
- T_H: Most of the ratings are located at the higher end of the rating range (items with 60% in the higher range).

Note that range varies from dataset to dataset, e.g., in a dataset with a rating range of 1 to 5, a lower range implies ratings 1 or 2, and a higher range implies ratings of 4 or 5. After categorizing the target item, several strategies can be employed:

- **UL:** Focused on attacks on user-based collaborative filtering systems, this strategy involves randomly selecting a filler item and assigning it the average rating given by users who rated the target item in the lower range.
- **UH:** Focused on attacks on user-based collaborative filtering systems, this strategy involves randomly selecting a filler item and assigning it the average rating given by users who rated the target item in the higher range.

- **IL:** Focused on attacks on item-based collaborative filtering systems, this strategy involves randomly selecting filler items highly rated by users who rated the target item in the lower range and assigning the maximum rating to these filler items.

- **IH:** Focused on attacks on item-based collaborative filtering systems, this strategy involves randomly selecting filler items highly rated by users who rated the target item in the higher range and assigning the maximum rating to these filler items.

- **SUL:** Focused on attacks using the Segment attack model on user-based collaborative filtering systems, this strategy involves randomly selecting a filler item and assigning it the average rating given by users who rated any of the items in the segment, as well as the target item, in the lower range.

- **SUH:** Focused on attacks using the Segment attack model on user-based collaborative filtering systems, this strategy involves randomly selecting a filler item and assigning it the average rating given by users who rated any of the items in the segment, as well as the target item, in the higher range.

- **SIL:** Focused on attacks using the Segment attack model on item-based collaborative filtering systems, this strategy involves randomly selecting a filler item and assigning it the average rating given by users who rated any of the items in the segment, as well as the target item, in the lower range.

- **SIH:** Focused on attacks using the Segment attack model on item-based collaborative filtering systems, this strategy involves randomly selecting a filler item and assigning it the average rating given by users who rated any of the items in the segment, as well as the target item, in the higher range.

WILSON & SEMINARIO (2013), in order to create more real attack profiles, studied the concept of *power users* in the context of collaborative filtering, defining it as "those who can exert considerable influence over the recommendations presented to other users". They proposed an attack model named Power User Attack, tailoring three variations of it based on different ways to select *power users*:

- **InDegree:** Power users selected as users appearing in the highest number of other users' neighborhoods, a.k.a In-Degree Centrality, using significance weighting to encourage strong connections between users with commonly rated items.

- **Number of Ratings:** Power users are selected as users with the highest number of ratings in the system.

- **Aggregated Similarity:** Power users are selected as users with the highest aggregate similarity to other users.

They found out that using Number of Ratings or InDregree was the best way to select *power users*. Another interesting finding is that inserting power users' profiles with biased ratings can be an effective way to attack a system with a user-based collaborative filtering algorithm. The authors extended this work with many others (SEMINARIO & WILSON, 2014b; WILSON & SEMINARIO, 2014; SEMINARIO & WILSON, 2015; WILSON *et al.*, 2015) proposing a methodology to create synthetic power user profiles and finding out that Power User Attacks could be successful in SVD-based algorithms, though they did not achieve the same success in a user-based collaborative filtering algorithm. Despite the advantages, the authors did not present a way to create power user profiles from scratch, that is, using as little system knowledge as possible, which may classify this approach as an informed attack.

Following up the research based on *power users*, SEMINARIO & WILSON (2014a) investigated the concept of *power items*, defined as influential items in the system. They proposed the same selection methods used in the previous works to select power items in a novel attack model named Power Item Attack. The rating value for each selected power item is selected randomly from a normal distribution over the item mean and standard deviation in the whole dataset. The results indicated that Power Item Attack is effective against user-based collaborative filtering and SVD-based algorithms. In order to be effective against item-based collaborative filtering, the authors performed an additional experiment using multiple target items.

FANG *et al.* (2018) studied attacks against graph-based collaborative filtering systems in an adversarial attack setting. The authors formalized attacks as an optimization problem, where the filler item ratings are optimized in order to maximize the target item hit ratio. Although their results outperforms all approaches in the adversarial setting, the *black-box* setting, where the optimization to generate profiles is performed using a collaborative filtering algorithm and the actual attack is executed with a different one, making it closer to a standard shilling attack problem, shows results close to classical attack models, such as Average and Bandwagon.

CHEN *et al.* (2019) performed a rated item correlation analysis, using Cosine Association (CA), in real users and artificial profiles generated by classic attack models in order to show the difference in rating patterns between each other. CA is an association rule in data mining and can be applied to measure the correlation between two items, which indicates the possibility of a user rating a pair of items at the same time. This function can be defined by equation 3.1 for items i and j, where $|U_{i,j}|$ denotes the number of users who rated both i and j while $|U_i|$ denotes the number of users that rated i.

$$CA(i,j) = \sqrt{\frac{|U_{i,j}|^2}{|U_i||U_j|}} \tag{3.1}$$

Based on the results of this analysis, they proposed an attack model that takes into account rated item correlation and item popularity in order to choose the items that would be rated in the generated profiles. In the proposed approach, filler items are chosen using the rated item correlation of real users instead of randomly. A parameter λ is used as a trade-off between the influence of rated item correlation and popularity on item selection. The authors performed experiments mounting attacks against user-based collaborative filtering. Additionally, they evaluated how classic shilling attack detection approaches performed against their attack model and compared them with classic attack models.

CHRISTAKOPOULOU & BANERJEE (2019) proposed a formulation of a machine-learned adversarial attack on a collaborative filtering system. It consisted of using GAN as a first step to generate profiles similar to real ones, then perform a repeated general-sum game between an adversary and the recommender, using an algorithm based on zero-order optimization techniques to compensate for the fact that the recommender's gradient is unknown to the attacker. Note that the recommender is not aware of the attacker's existence. The authors targeted top-K recommendations or subsets of users or items in the experiments and identified some vulnerabilities in model-based recommender system.

ANELLI *et al.* (2020) proposed SAShA, an attack strategy that leverages semantic features from knowledge graphs. SAShA enhances the baseline attack models by exploiting semantic information to manipulate user interaction patterns, rather than merely sampling random filler items. The study provides a comprehensive experimental evaluation comparing SAShA to baseline attacks, demonstrating a slight improvement in the evaluated metrics.

LIN *et al.* (2020) presented Augmented Shilling Attack (AUSH), a framework designed to attack recommender systems based on GAN and capable of tailoring attacks against specific user segments. It is done by sampling item templates from real users and using a framework generator to produce ratings for these selected items. Authors mounted attacks against matrix factorization and autoencoder-based recommender systems, reporting competitive results.

At the time of our paper's publication (BARBIERI *et al.*, 2021), this was the state of the literature regarding shilling attack models. Since then, several attack models based on similar reasoning have been developed, as shown below. Note that some of these subsequent works cite our paper.

WU *et al.* (2021) proposed Graph cOnvolution-based generative shilling ATtack (GOAT) which aims to strike a balance between the simplicity of primitive shilling

attacks and the sophistication of advanced methods. It utilizes a GAN to produce fake ratings and incorporates a graph convolution structure to refine these ratings based on co-rated item correlations. The experimental evaluation provides insights into the model's technical feasibility and suggests directions for further investigation into preventive measures.

WANG *et al.* (2022) presented the GSA-GANs model, a gray-box shilling attack approach utilizing generative adversarial networks to craft fake user profiles. The authors proposed two GAN modules, each designed to enhance the similarity of generated fake profiles to real ones and to make these profiles more detrimental to recommendation results. Experimental results on three public datasets show improvements in attack effectiveness, transferability, and camouflage compared to baseline models.

REN *et al.* (2022) studied the impact of semantic information on attack effectiveness, proposing the Semantic Shilling Attack (SSA), a heuristic approach designed to exploit high-level semantic information in heterogeneous information network (HIN) recommender systems. The approach focuses on injecting specific scores into items based on their semantic relationships to the target, showing improved effectiveness compared to existing baseline algorithms.

A different scenario was explored by HUANG & LI (2023), involving a single fake user profile, as opposed to multiple profiles. The proposed SUI-Attack conceptualizes the attack as a node generation problem within the user-item bipartite graph of the recommender system. By generating user features and connections that integrate the fake profile into the existing graph, SUI-Attack seeks to enhance stealthiness. The authors report competitive results for this single-user attack scenario.

The work of LIN *et al.* (2024) introduces Leg-UP, an approach utilizing GANs to create fake user profiles. The authors aimed to develop an attack model capable of mounting successful and less detectable attacks across various recommendation algorithms. Leg-UP generates fake profiles by learning from real user behavior and optimizes the generator using a surrogate recommendation system to improve transferability. A discriminator is employed to mitigate the detectability of the generated profiles.

Finally, LU & GAO (2024) proposed the use of Variational Graph Auto-encoders in a similar fashion of our work, i.e., aiming to create user profiles that align with original rating patterns. This approach utilizes spectral clustering to select users from the original dataset as templates, dispersed according to a specified attack scale. These templates are then matched with similar fake users within the reconstructed profiles. The next step involves inserting ratings for popular items and target items into these fake profiles. The final step involves generating and injecting these fake profiles into the recommendation system to promote target items to users.

In the previous section, we presented some of the issues with the current attack models, lack of experiments with model-based collaborative filtering algorithms (I1), weak results with less attack profiles injected into the system (I2), and non realistic rating patterns (I3). Considering the works published up to the time of our paper, the general status of the literature indicates that very few studies have addressed all the enumerated issues, either by mentioning them or demonstrating significant improvements in their experimental evaluation. WILSON & SEMINARIO (2013) and his subsequent works (SEMINARIO & WILSON, 2014b; WILSON & SEMINARIO, 2014; SEMINARIO & WILSON, 2015; WILSON *et al.*, 2015) address all the issues; however, they do so by directly copying influential realistic profiles to mount an attack. In contrast, LIN *et al.* (2020) can be considered the only work directly comparable to ours in terms of addressing all the listed issues. The table 3.1 shows the proposed works and the issues they address. Note that, although related, we did not include the work of CHRISTAKOPOULOU & BANERJEE (2019) in the comparison because it focuses on adversarial attacks rather than shilling attacks.

Table 3.1: Status of the literature on shilling attack models and the associated issues.

Work	I1	I2	I3
(RAY & MAHANTI, 2009)		✓	
(WILSON & SEMINARIO, 2013)	✓	✓	✓
(SEMINARIO & WILSON, 2014a)		✓	
(FANG *et al.*, 2018)	✓		✓
(CHEN *et al.*, 2019)		✓	✓
(ANELLI *et al.*, 2020)	✓		✓
(LIN *et al.*, 2020)	✓	✓	✓

3.3 Proposal

In this section, we will discuss our proposed approach to crafting profiles closer to real ones and mount an attack to convert these profiles to malicious ones. In order to simplify the understanding of our proposal, we divided our method into two main steps: the simulation of profiles and the construction of an attack. In the first step, a generative model is trained using real data to learn how to simulate profiles whose behaviors resemble those of real users, while in the second step, we construct an attack using the profiles generated in the first step. Figure 3.1 illustrates how our method works.

The novel attack model is expected to replicate real profiles accurately, while strongly impacting the recommendation result on model-based systems, but without

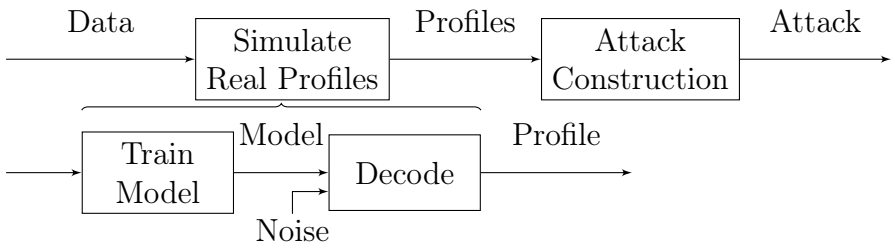


Figure 3.1: Our proposal to generate malicious profiles closer to real ones. The first step is to simulate real profiles using a generative technique, and the second one focuses on converting the generated profiles into malicious ones.

losing efficiency on memory-based systems. We will briefly describe each step before going into details:

(i) *Simulate real profiles*: Step denoting the process for creating real-like profiles. It is subdivided into two stages:

(i.i) *Train model:* Pre-processed data is used for training our proposed model in order to learn the data probability distribution.

(i.ii) *Decode:* The sampling process consists of feeding noisy inputs into the model decoder. The network decoder maps these inputs into synthetic profiles.

(ii) *Attack construction:* Synthetic profiles are generated to be identical to the real ones. Thus, to construct an attack, it is necessary to add an intent and convert them into malicious profiles.

The next subsections will give a motivation and describe these two steps.

3.3.1 Simulate Real Profiles

Most current shilling attack models focus on defining an attack strategy in order to push or nuke a certain item in the system, which already has a particular data distribution that characterizes it. In the past, real users present on this system defined how this data distribution looks like by rating items, in a natural interaction between customers and products. Commonly, these strategies do not take into account how their crafted malicious profiles will fit into system data, that is, their behaviors are completely different from authentic users. In addition, injecting profiles that deviate from the dataset pattern may significantly change this distribution, hinting that these new data are not coming from authentic users (ZHANG *et al.*, 2006) and making them easy to detect by attack detection approaches (CHEN *et al.*, 2019).

Thus, in order to create malicious profiles focused on solving the previously listed issues, we must ensure that they appear like real ones, that is, they need to look like they came from the same distribution as the actual data to remain unnoticed.

Authentic user rating patterns are the key to constructing a malicious profile that fits into real data without raising suspicion. Advances in machine learning allow the learning of these patterns, to make it possible to apply them to construct malicious profiles. As explained in the previous section, discriminative models seek to learn a predictive model that may be possibly used to predict the future behavior of the users within the system. Our intent; however, is to learn the distribution and replicate similar natural behaviors, making generative models more suitable for this task.

A variational autoencoder, as previously mentioned, is a generative model that instead of learning an encoding vector, learn a vector of means and a vector of standard deviations. This property makes it possible to generate new data for the mean and standard deviations learned, which makes it a powerful generative tool. Recently, it has been used to replicate distributions and generate new data from noise in many tasks, such as handwritten digits (KINGMA & WELLING, 2013), faces (REZENDE *et al.*, 2014), images (GREGOR *et al.*, 2015), semantic segmentation (SOHN *et al.*, 2015) and even forecasting from static images (WALKER *et al.*, 2016).

Based on it, we propose a variational autoencoder with convolution layers as an automatic generator of simulated profiles. Our intent is to learn the probability distribution of real profiles to be able to generate our counterfeit ones to look as real as possible and remain undetected by attack detection approaches. Convolutional layers are being applied to shilling attack showing promising results (CHRISTAKOPOULOU & BANERJEE, 2019; TONG *et al.*, 2018).

Collaborative filtering problem data may be seen as an undirected bipartite graph (MELLO *et al.*, 2010), where the vertex represents users or items and the edges are relations between them. In this context, it is reasonable that another option to model this problem is using Graph Convolutional Networks (GCN), which introduces a graph to model user-item interactions within the system (ZHANG *et al.*, 2020), instead of simple convolutional layers. This investigation; however, is left for future work.

In order to prepare training data to train our proposed model, we need to create a rating matrix from it. In this matrix, the users are represented by rows and the items are represented by columns, while missing ratings are represented by zeros. In addition, each row x ratings are rescaled to the 0-1 range. After matrix creation, it needs to be converted into a tensor in order to be in suitable format for the convolutional layers. This process is done by reshaping each row into a matrix that

has its dimensions given by

$$|x|_{LF} \times \frac{|x|}{|x|_{LF}}, \tag{3.2}$$

where $|x|$ is the number of elements of row x and x_{LF} is the lowest factor of $|x|$. For numbers with a square root, $\sqrt{|x|} \times \sqrt{|x|}$ may be used instead. An example of the data pre-processing can be seen in Figure 3.2.

Figure 3.2: An example of pre-processing of the training data.

Afterward, the sampling process is fairly simple: Latent-space-sized vectors are sampled around normal distribution and fed to the decoder. Our model is able to turn these factors into meaningful profiles with behaviors and characteristics similar to the real ones.

3.3.2 Attack Construction

After the first step, we need to convert our simulated real profiles into malicious ones in order to mount an efficient attack. The generative power of a variational autoencoder may be able to generate profiles that look like a real one; however, it will not be a malicious profile since, by default, the first step of our method is unable to generate profiles with an attack intent. Thus, we need to post-process our generated profiles in order to add intent and affect the attacked system in the way intended.

As previously stated, an attack intent is usually to make an item more, push, or less, nuke, recommended within the system. However, there are other intents, such as: improving or degrading a particular item's performance inside a user segment.

To make it simple and evaluate the generative power of variational autoencoder itself, rather than the post-process strategy, we chose a naïve approach: add a target item and rate it according to our intent, e.g., max rating value for push attack or min value for nuke attacks. In this work, we evaluated push attacks only; however, similar results could be obtained by analyzing nuke attacks. A straightforward strategy similar to the majority of attack models proposed in the literature (MOBASHER *et al.*, 2007b). A study of more efficient methods to post-process our simulated profiles is out of the scope of this work and left for future work.

3.4 Empirical Results

In this section, we detail our experiments, results of our attack model, and compare with well-known methods.

Our experiments are carried out to assert three main assumptions: Our model can create malicious profiles closer to the real ones than other approaches; our model is able to outperform other approaches in most of the cases attacking model-based algorithms, and our model achieves competitive results in most of the cases against other approaches in memory-based algorithms. Firstly, we analyze how close our and other approaches' malicious profiles are to real ones in terms of correlated rated items. Secondly, we compare our approach performance in model and memory-based systems against several methods, including: Random, Average, Bandwagon, Segment, Power User NumRatings (PUA-NR), and Power Item NumRatings (PIA-NR). In addition, we trained a GAN model and used its generator as a generative model in your workflow in order to compare its generative power to our proposal.

3.4.1 Setup

We will carry out several experiments to validate our proposal in different scenarios. We compare our attack model performance against classic and some of the proposed attack models in model-based and memory-based collaborative algorithms.

That said, our first experiment will be a rated item correlation analysis using cosine association, in order to evaluate how close malicious profiles generated using our approach are to the real profiles in real-world datasets.

Afterward, we will perform two more similar experiments to evaluate our attack model. The first one will compare our proposed model against several models with respect to its effectiveness at attacking a model-based algorithm. The second one will compare our approach against the same attack models; however, this time, attacking a memory-based algorithm in order to evaluate how our proposal performs in this kind of system.

In both experiments, we will compare our attack model with well-known attack models which are: Random, Average, Segment, Bandwagon, PUA-NR, and PIA-NR. Additionally, we will compare it with a GAN generator.

The first experiment will use the whole data in order to generate the malicious profiles. Therefore, the attack size will be set to 100% in order to have the same number of users as the actual data and make the comparison possible. For the MovieLens 100k dataset, using the Power Item Attack, we set the number of power items to 100 to match the number of ratings in our malicious dataset more closely to that in the real profiles. In attack models that require filler size, we set it to 7% for the same reason. The same logic was applied for the For Yahoo! Music R3, where

we set the number of power items to 23, and for attack models using filler size, to 7%.

As well, we will conduct our second and third experiments using a hold-out strategy, that is, partitioning the data into two sets: 70% for training and 30% for testing. The train data is used to create the malicious profiles, while the test data is used only during the evaluation of the attack model.

In order to evaluate the models from different perspectives, we selected 30 distinct target items: ten randomly chosen among all items; ten chosen among established items, that is, the top-50 most-rated items, and ten chosen among new items, that is, items with only one rating (SEMINARIO & WILSON, 2014a). Since our attack intent is push, the target item rating was set to max value (which is 5 regarding the MovieLens 100k dataset). We averaged the results for each category of target item and each chosen metric and reported the results. For the $\overline{HR}$ metric, we computed it using the top 10, 20, 30, 40, and 50 recommendations (N) and reported the average over the results and their individual values (NOH *et al.*, 2014).

Parameters for the variational autoencoder were chosen using hold-out validation. The MovieLens 100k dataset was split into three parts: train, validation, and test. All experiments were conducted using the train and validation partitions, while the test partition was reserved to validate the best parameters identified during the tuning process. In general, it was found that good results may be achieved by varying the learning rate and kernel size. We also compared the model with convolutional layers to the same configuration with dense layers, finding that the convolutional layers provided superior results and faster training times. We used Adam optimizer with a learning rate of 0.0005 and eight latent dimensions. Regarding the network architecture, the encoder was composed of three convolutional layers with 64 filters, with kernel size of four and two strides, while the decoder was composed of three convolutional layers with 64 filters, a kernel size of four and two or one strides, two for the first layer, one for the second and the third layers. The chosen activation function for the model was the leaky rectified linear unit (Leaky ReLU) (MAAS *et al.*, 2013) and dropout was applied to each layer.

Regarding the attack models parameters, we varied the attack size by 1%, 3%, 5%, 10%, and 20%, and filler size, for the models that require it, by 1%, 3%, 5%, 7%, and 10%. In addition, for the selected items set in the Segment attack, we chose the popular horror movie segment (BURKE, 2007) and for the Bandwagon, we selected movies (IDs) {50, 56, 100, 127, 174, 181, 258, 286, 288, 294} as popular movies, which are movies rated by more than 300 users (LI & LUO, 2011). For Power User Attack and Power Item Attack, we chose the *NumRatings* approach for power user or item selection, since it yields similar results to those of *InDegree* approach, but with a better overall performance. In Power Item Attack there is a

parameter that controls the number of power items used to construct each malicious profile, we varied it by 50, 40, 30, 20, 10, 5, 3, and 2. GAN parameters were taken from CHRISTAKOPOULOU & BANERJEE (2019).

For evaluating attack model performance against model-based systems, we chose Improved Regularized SVD (IRSVD)(PATEREK, 2007) as the collaborative filtering algorithm, with 100 latent factors, 20 epochs, and 0.005 learning rate. Being popular among works in the literature, SVD-based algorithms are the choice regarding evaluating model-based approaches against attack strategies by some works in shilling attack area (SEMINARIO & WILSON, 2014b, 2015).

For attack models performance against memory-based systems, we applied a user-based collaborative filtering algorithm, with Pearson correlation as the similarity measure and 40 as the maximum number of neighbors. We choose the user-based variation because the item-based is well-known for its robustness against most shiling attack models (GUNES *et al.*, 2012).

In resume, three experiments are carried out to evaluate the proposed method:

Experiment I

Item correlation analysis (CA) comparing real data with our approach and classical approaches using MovieLens 100k and Yahoo! Music datasets. This analysis may indicate how close our attack model ratings patterns are to the ones in real data. It may also hint at how easily an attack model may be detected by detection approaches CHEN *et al.* (2019).

Experiment II

Comparison with seven shilling attack models: *Random*, *Average*, *Segment*, *Bandwagon*, *PUA-NR*, *PIA-NR* and *GAN* using MovieLens 100k and Yahoo! Music datasets on an IRSVD system.

Experiment III

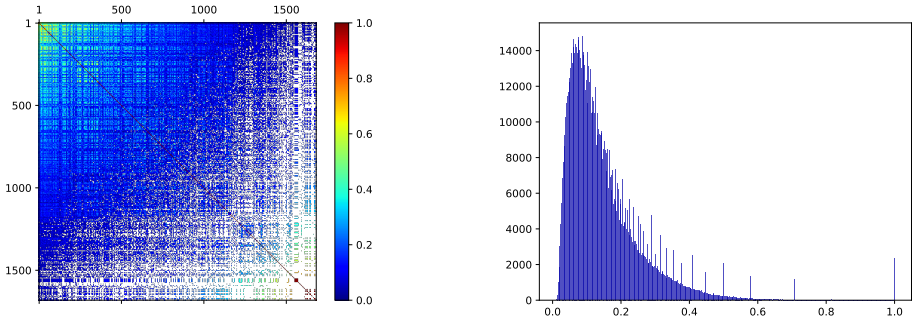
Comparison with seven shilling attack models using MovieLens 100k dataset on a classic user-based collaborative filtering system.

3.4.2 Results

In this subsection, we present and discuss the obtained results from the evaluated experiments. Additionally, we compare the proposed attack model with several well-known shilling attack models.

In *Experiment I* we analyze how close rating patterns from malicious profiles are from real users. That said, Figure 3.3 shows the CA for all item pairs in MovieLens 100k dataset for reference, where both axis represents the movies sorted in ascending order by the times each one is rated, while each point denotes the CA value between

its correspondent pair of movies, with darker red denoting that the pair is highly correlated (CA value equals to 1), darker blue denoting that the pair is almost uncorrelated (CA value close to 0), and white denoting no user rating this item pair at the same time. Note that $CA = 0$ was left white as well in order to illustrate other values more clearly.



(a) Item pair (b) Distribution

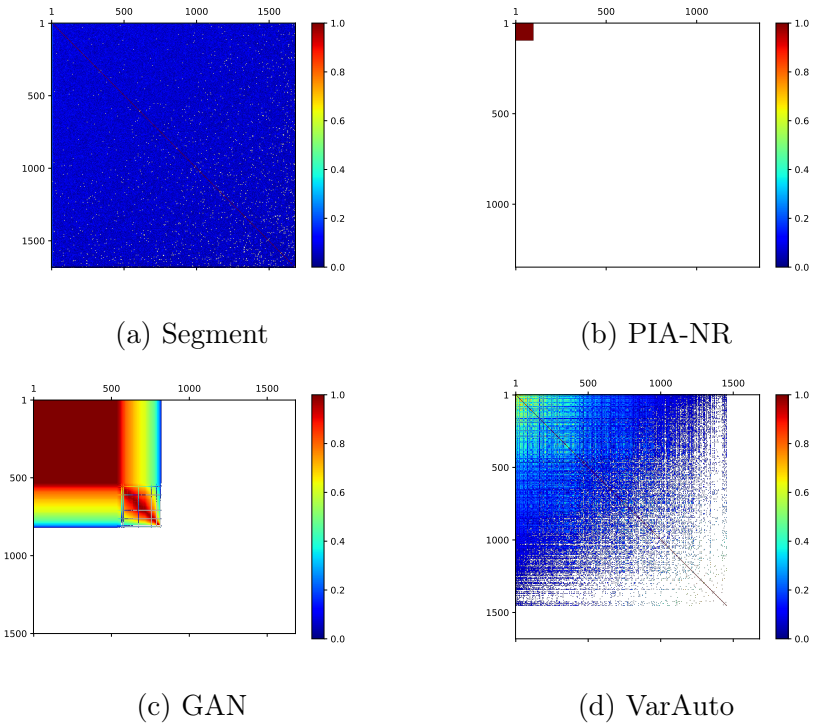
Figure 3.3: CA (a) and CA distribution (b) of item pairs in the MovieLens 100k dataset.

In the top-left corner, this figure indicates that popular items are highly correlated to each other with some yellow and green areas. It is interesting to note as well that there are some unpopular items highly correlated in the bottom-right area showing a similar pattern to the popular ones, despite the lack of ratings.

Besides CA values for each pair, in Figure 3.3b, we show the CA values distribution. These results indicate that the majority of items in the system are not strongly correlated, despite having some item pairs highly correlated. As one may see, the most common value is close to 0.1, decreasing as the correlation increases until around 0.7.

Aiming to compare every malicious profile generated by each attack model, we generated seven different datasets using each presented attack model and performed the same evaluation used in the real data in each of these datasets. Figure 3.4 presents the results obtained for each fake dataset.

In Figure 3.4, we show the CA for each item-pair generated by each attack model. The models are: Random, Average, Bandwagon, Segment (3.4a), PUA-NR, PIA-NR (3.4b), GAN (3.4c), and our model (3.4d). Notice how different each attack behaves in the data, except for PUA-NR (not shown, since its profiles are just copies from the original dataset). Random (not shown), Average (not shown), Bandwagon (not shown) and Segment attack models delivered pairwise correlations in very similar patterns since the filler items are selected randomly, with almost all areas filled with low correlated pairs. On the other hand, malicious profiles constructed using PIA-NR are based on *Power Items* selection. Hence, only a small number of items



(a) Segment (b) PIA-NR

(c) GAN (d) VarAuto

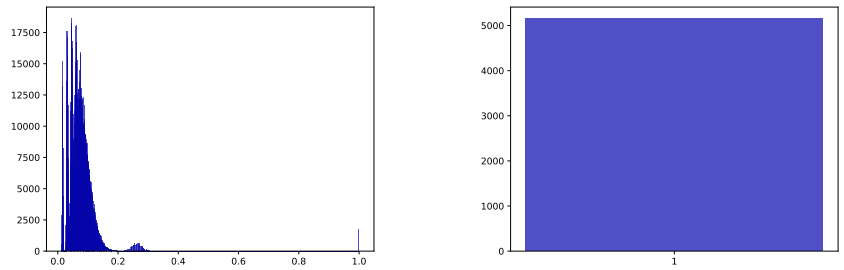
Figure 3.4: CA of item pairs for different shilling attack models. For instance (a) Segment, (b) PIA-NR, (c) GAN, and (d) our model.

is rated and correlated, showing a small area in a dark red color while, no other items are rated. Lastly, the GAN model delivers highly correlated items; however, is unable to resemble the same patterns found in the original data.

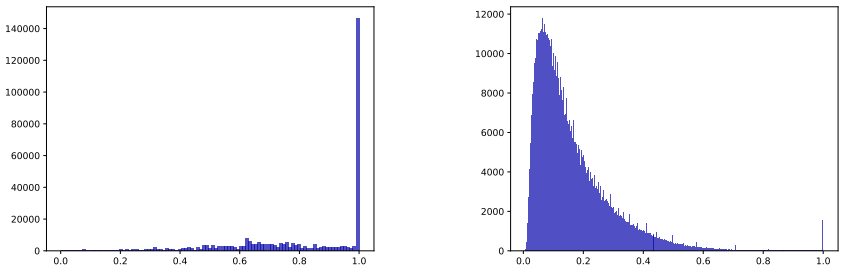
Regarding our approach, despite the correlated area appearing to be smaller, there is a similar pattern compared to the original data. These results indicate that our approach builds profiles that follow similar patterns when compared to the real ones, even without directly copying them as the Power User method. It may also indicate that our malicious profiles may remain unnoticed for detection approaches.

Figure 3.5 shows the distribution of CA values. Note how it actually differs greatly from the actual values depicted in Figure 3.3b. Random (not shown), Average (not shown), Bandwagon (not shown), and Segment (3.5a) also reach their peak at 0.1, but decrease as the correlation increases much faster until around 0.2 for Random and Average; and after 0.3 for Bandwagon and Segment, since its attacks also have the selected items set, which are present in all malicious profiles crafted by these attack models. GAN (3.5c) sampled highly correlated items, showing its peak at the maximum correlation value. On the other hand, our model (3.5d) has the most similar distribution of CA values taken from original MovieLens data.

Figures A.1 and A.2 respectively show the CA for each item pair and the distribution of CA values in the Yahoo! Music dataset. The results are largely similar to those obtained using the MovieLens 100k dataset, except that the correlation area



(a) Segment (b) PIA-NR



(c) GAN (d) VarAuto

Figure 3.5: CA distribution of item pairs for different shilling attack models. For instance (a) Segment, (b) PIA-NR, (c) GAN, and (d) our model.

is sparser in the Variational Autoencoder model rather than smaller.

In both CA values and its distribution for all datasets, PUA-NR manages to faithfully replicate the original data distribution. However, we must emphasize that this attack model consists of directly copying selected power user profiles and then infecting them to achieve the desired effect. Therefore, it was an expected result. In the second experiment, we will evaluate if this strategy is enough to disrupt model-based systems and how it differs from our approach.

Experiment II is performed to evaluate our attack model against several models using Improved Regularized SVD as a collaborative filtering technique. Figure 3.6 shows complete results for (a) random items, (b) the least-rated items, and (c) the most-rated items using the MovieLens 100k dataset. In this overall view, the results show how our attack model performs against the other models for each of these categories of target items. For the least-rated items (Figure 3.6b), it is possible to note that all models behave similarly regarding all three metrics, with our proposal achieving the best results in both metrics, followed closely by Average attack and, surprisingly, in case of hit ratio and average rank, Segment attack (not shown). Regardless of which model has the best performance, the results also indicate that it is easier to push an item with fewer ratings than expected.

For most-rated items (Figure 3.6c), as expected, we can spot a different scenario, where all models have difficulties to push the target item since it already has a lot of

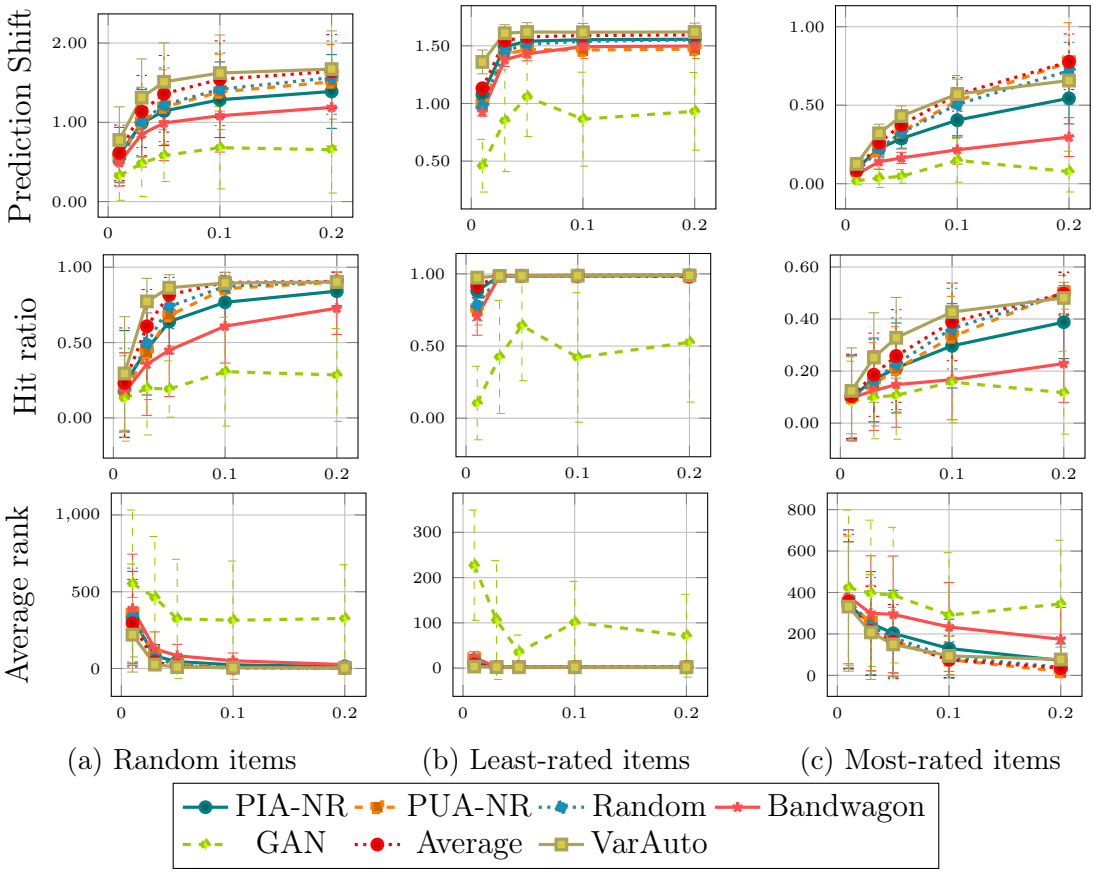


Figure 3.6: Prediction shift, hit ratio and average rank for different attacks sizes using Improved Regularized SVD as collaborative filtering technique. Note that it only shows the best result of filler size among the attack models that require it.

ratings. This time Segment attack is unable to repeat the same results achieved with least-rated items, while our approach consistently improves results for smaller attack sizes, e.g., attack size of 1% to 5%, which was one of our main goals. Although the Average attack model outperforming our approach when a large number of malicious profiles are injected, e.g., attack size of 20%, we can remark that a larger attack is difficult to mount in larger datasets and are more easily spotted.

For random items (Figure 3.6a), which can include any items rated in the system, we found similar observed patterns. Considering the three used metrics, our proposed attack model outperformed the other models, mounting more successful attacks from the attacker's standpoint. PUA-NR and PIA-NR reported results similar to the Random attack model, while the Segment model, once again, failed to perform on par with the best models, as it did with the least-rated items.

Figure A.3 presents results for (a) random items, (b) the least-rated items, and (c) the most-rated items using the Yahoo! Music dataset. These results show that Average, Random, and PUA-NR produce outcomes very similar to our proposal. This suggests that, even without specifically tuning our proposal for the Yahoo!

Music dataset, it can be as competitive as PUA-NR, which directly copies real profiles from the dataset. There is a possibility of improving these results to outperform the current attack model with proper tuning.

In short, for least-rated items, even with 1% attack size, that is 9 profiles, our approach can influence the model-based recommender, giving higher values of prediction shift and hit ratio while giving lower values of rank. For most-rated items, we still have the best performance with lesser attack sizes, indicating that our attack model behaves as intended in all studied scenarios. It is also important to remark that our attack model is capable of achieving good results in model-based systems without directly copying profiles as approaches like the PUA-NR. GAN model on the other hand, for all types of target items, is unable to get consistent results among all metrics. We noted that the model had some trouble balancing generator and discriminator costs in a few scenarios, which may explain the lack of performance compared to the variational autoencoder.

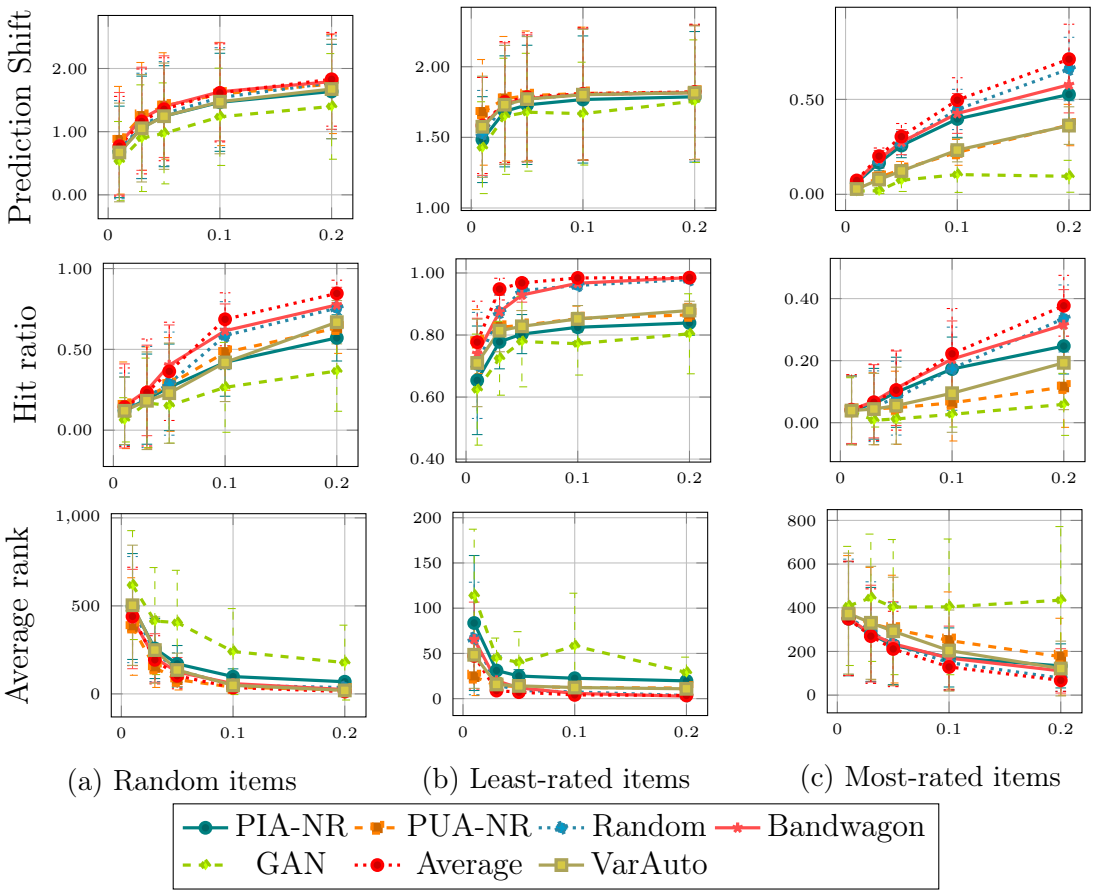


Figure 3.7: Prediction shift, hit ratio and average rank for different attack sizes using User-based as collaborative filtering technique. Note that it only shows the best result of filler size among the attack models that require it.

After the indication that our method can outperform the current attack models when the system attacked is a model-based algorithm, finally, *Experiment III*

is performed to evaluate our attack model against the same previous approaches. However, now we are using user-based collaborative filtering technique to provide recommendations.

For least-rated items (Figure 3.7b), results for prediction shift and average rank metrics indicate that our model may be competitive in memory-based environments, coming very close to the performance of models crafted to attack KNN-based systems and even outperforming some of them, such as PIA-NR. For Hit Ratio, on the other hand, our approach failed to achieve good results against the intended user-based collaborative filtering crafted attacks. Despite achieving results comparable to PUA-NR and PIA-NR, and even slightly outperforming them in some attack size values, our model did not achieve good results against Average, Bandwagon, and Random attack models with this metric.

For most-rated items (Figure 3.7c), a different context emerges where all approaches struggle to push items that already have a large number of ratings. None of the approaches are able to achieve good enough results for all metrics. The best results achieved are when using Average and Random attacks with a 20% attack size, which may be an inefficient and easily detectable mounted attack. Despite the poor performance in Prediction Shift and Hit Ratio, our approach achieves results close to the Bandwagon attack regarding average rank, indicating that we may have room for improvement.

For random items (Figure 3.7a), the results were very close to the reported for least-rated items. Variational autoencoder-crafted malicious profiles achieve competitive results in terms of prediction shift and average rank without being explicitly engineered toward memory-based systems. For hit ratio; however, the results are on par with PIA-NR and PUA-NR, while Average, Bandwagon, and Random attacks reported the best results.

In short, it is possible to spot that our method achieves good results for average rank; however, the values obtained are almost, always around ten average rank, explaining why other models have an advantage over ours in Hit Ratio. Prediction shift values are also competitive, except for most-rated items. Note that none of the models, including ours, were able to achieve truly meaningful results for the most-rated items, indicating that popular items, i.e., those receiving a high number of ratings, are very difficult to push. It is also important to note that our attack model is capable of achieving results in memory-based systems on par with PIA-NR and PUA-NR, models crafted to mount attacks against memory-based approaches.

In memory-based systems, GAN performance was stronger than previously reported results mounting attacks against IRSVD. However, it was not good enough compared to the other models, indicating that this model may be more sensitive to hyperparameters than our proposal since we did not perform extensive hyperparam-

eter tuning in our model as well.

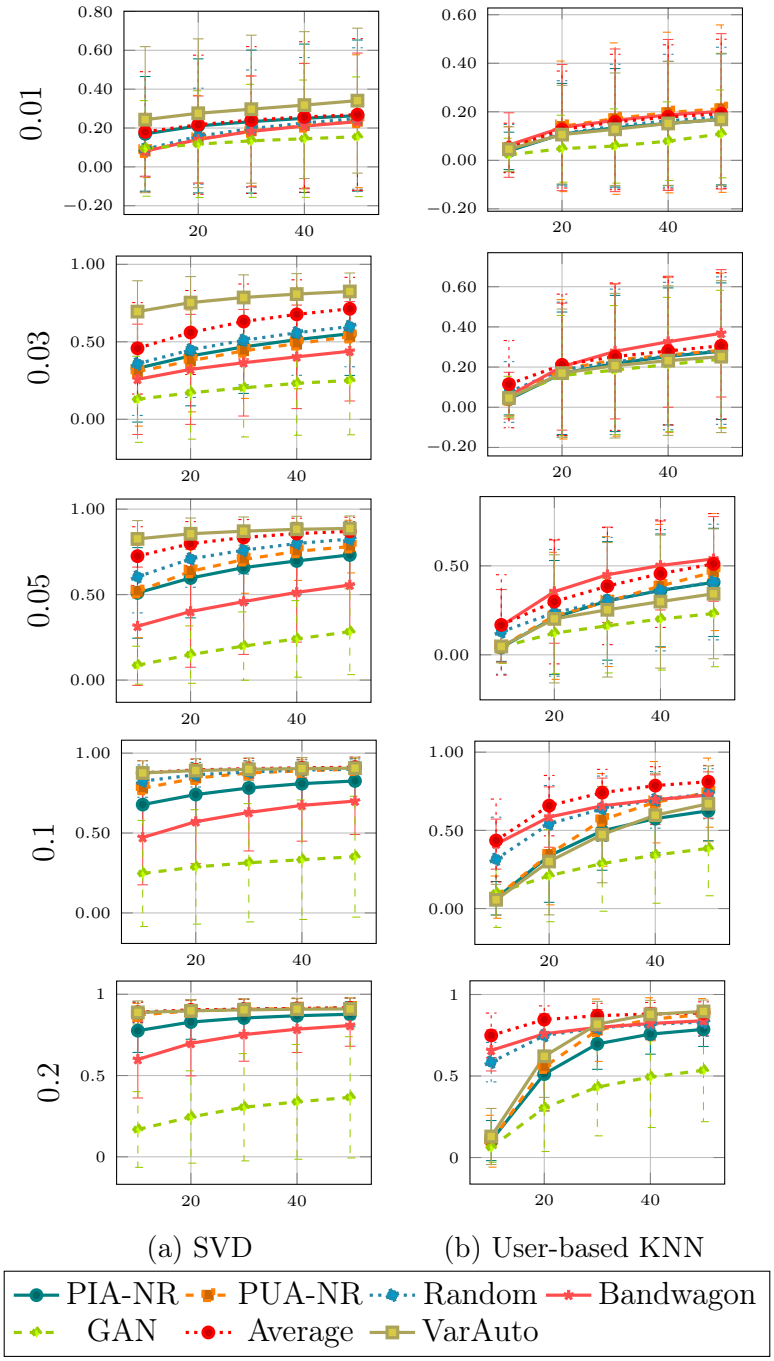


Figure 3.8: Hit ratio for different ranks and attack sizes using both SVD and User-based as collaborative filtering technique. Note that it only shows the best result of filler size among the attack models that require it.

Finally, we present results for different top-N values of the Hit Ratio. For SVD as collaborative filtering technique (Figure 3.8a), results, one more time, corroborate how our approach is able to outperform state-of-the-art attack models considering all attack sizes. One may note the superior performance of our method using only 3% attack size, greatly outperforming all methods in all cases. For User-based

KNN as collaborative filtering technique (Figure 3.8b), it is possible to see where our approach may be improved. We achieved results closer to the state-of-the-art technique with 3% attack size and even outperformed all models using 20% attack size and larger recommendation lists; however, in other scenarios, Bandwagon attacks still achieve the best results of Hit Ratio for these kinds of systems.

Chapter 4

Shilling Attack Perspective as Label Noise

This chapter introduces the concept of label noise in the collaborative filtering context, comparing it with shilling attack, also considered malicious noise, and discusses how the proposed techniques in this area can be applied to mitigate this issue. Initially, we will describe each solution and how its authors envisioned mitigating the label noise problem. Later, we will apply each solution to the malicious noise problem, in order to check if these algorithms can be used as such. Finally, we will analyze the results and show our considerations about this important topic.

4.1 Noise

The performance of a learning model is typically measured by its ability to accurately predict new instances, which often depends on the quality of the data used to train the model. According to ANGLUIN & LAIRD (1988), real-world data is highly susceptible to noise or corruption, e.g., training examples misclassified, which may degrade the quality of the data, thus degrading the learning model's performance. Noise is common in real-world datasets and can be considered, anything that hinders the relationship between attributes and their respective label.

According to ZHU & WU (2004), there are two categories of noise: label noise and attribute noise. Label noise refers to manually misclassifying data, which can occur when instances are labeled incorrectly or when the same instances are given different labels. Attribute noise is related to errors during the data collection process, leading to erroneous, missing, or incomplete attribute values. Due to the presence of only one label instead of multiple attributes, label class is generally considered more harmful than attribute noise (ZHU & WU, 2004; SÁEZ *et al.*, 2014).

FRENAY & VERLEYSEN (2014) introduces label noise as an inherently stochas-

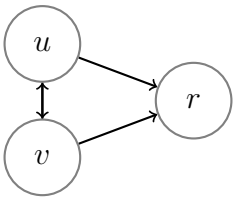


Figure 4.1: A probabilistic model illustrating the cyclic dependency between users, items and ratings, as seen in collaborative filtering problems (CARMO, 2018).

tic process, disregarding cases where errors in the labeling process are maliciously introduced by external agents. On the other hand, VAN DEN HOUT *et al.* (2002) shows label noise as a stochastic process in an application where noise can be generated intentionally to protect the privacy of users.

In a noise context, robustness indicates the capacity of a model to be insensitive to data corruption, thus suffering less from the effects of noise (HUBER, 2004). The more robust the algorithm is, the more similar the model will be to one trained without noise.

Moving on to the recommendation context, noisy instances affect collaborative filtering models differently. In collaborative filtering, the rating is unique and depends on both user u and item v entities, while there is also a relationship between the user and the item, since the evaluation will depend on the opinion of a user regarding an item. Figure 4.1 illustrates this cyclic dependency.

Thus, collaborative filtering systems use a different learning structure than the one used on supervised learning problems, where labels are not used to predict other instances. In a collaborative filtering scheme, the label, user ratings on items, is used to predict another user's ratings, which means that a single noisy rating can indirectly disturb numerous relationships, impacting users and items not directly related to this particular noisy instance. In short, it is possible to conclude that the collaborative filtering scheme is inherently noisy, with noisy instances having a larger impact when compared to supervised models (CARMO, 2018).

Another form of noise in collaborative filtering is the shilling attack, which is considered malicious noise and differs from label noise, as defined in this thesis, because it is not a natural occurrence. Therefore, the main research question of this chapter is to evaluate label noise proposals on malicious noise and analyze if it can mitigate it.

4.2 Methods to Deal with Label Noise

According to FRENAY & VERLEYSEN (2014), there are three ways to deal with label noise. The first approach is to rely on algorithms that are naturally more

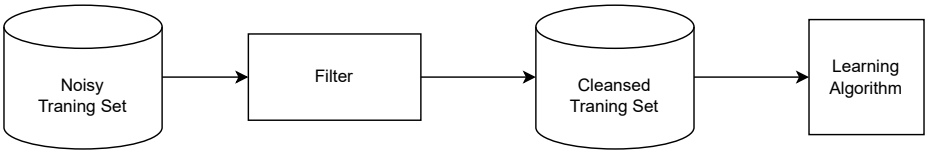


Figure 4.2: Learning procedure for dataset cleaning models (BRODLEY & FRIEDL, 2011).

robust to the noise, which means that the algorithm is not too sensitive when trained on the presence of label noise. The main idea of these models is that predictions remain stable even in the presence of noisy instances, i.e., training data corrupted by label noise. These approaches are also referred to as robust algorithms.

The second one focuses on filtering the data on the training set in order to improve its quality. In these kinds of approaches, mislabeled instances are usually dropped or relabeled according to some procedure. The downside of this approach is that it may result in the removal of large amounts of data during the filtering process. These approaches are referred to as data cleansing.

Lastly, there are algorithms that model the label noise directly or have been modified to consider it. FRENAY & VERLEYSEN (2014) argue that, in this way, it is possible to include information about the nature of the label noise, since the classification model and the label noise model are separated.

In the following subsections, we will present additional concepts regarding data cleansing and label noise-robust learning algorithms as well as some solutions proposed in the literature that fall into these categories.

4.2.1 Data Cleansing

Data cleaning refers to solutions that focus on cleaning the training data, being similar to an outlier or anomaly detection approaches (FRENAY & VERLEYSEN, 2014). The whole process can be seen in Figure 4.2, instances of the noisy training set are filtered before being inputted into the learning algorithm. This subsection describes several methods that apply this concept to detect and remove mislabeled instances from the recommendation perspective.

O'MAHONY *et al.* (2006) proposed a framework to detect noise in Collaborative Filtering datasets, considering two classes of noise: natural and malicious. Let U be the set of users of the system, $U_g \subset U$ be the set of genuine user profiles in a system, and $U_a \subset U$ be the set of attack profiles present. The proposed approach consists of training a recommendation algorithm G with a training set $T \subset U_g$ to verify the consistency of the actual rating for a user-item pair compared with the prediction from this given model. The consistency c of a rating $r_{u,i}$ is defined as the Mean Absolute Error between the actual rating and the G prediction:

$$c(G,T)_{u,i} = \frac{|r_{u,i} - \hat{r}_{ui}|}{r_{\max} - r_{\min}},$$

where $\hat{r}_{ui}$ is a prediction given by the model G to a user u and an item i, while $r_{\max}/r_{\min}$ is the maximum and minimum ratings allowed in the system. A rating $r_{u,i}$ is perceived as noise and removed from the recommendation, if:

$$c(G,T)_{u,i} > th,$$

where th is a threshold value. The intuition behind the approach is that G will be accurate within a threshold th and the $c(G,T)_{u,i}$ for every user-item pair above this value indicates the presence of noise in the rating.

Another data cleansing approach applied in collaborative filtering is the one proposed by TOLEDO *et al.* (2015) which focus on both detection and correction of the noise. It consists of classifying both users and items based on their ratings and assuming that a contradiction may represent evidence of noise. This process assumes that each user has their own tendency when rating items, while, in a similar manner, each item has its tendency to receive ratings. For user tendencies, they formulated the following categories: Positive, Average, Negative, and Hesitating. For item tendencies, the authors proposed a similar scheme: Preferred, Av-Preferred, No-Preferred, and Doubtful.

Later, each user rating for an item is classified into three different categories based on its value:

- Weak: if $r_{ui} < k$.
- Mean: if $k \leq r_{ui} < v$.
- Strong: if $r_{ui} \geq v$.

The constants k and v are, respectively, a weak-average threshold and an average-strong threshold, where both values must satisfy the $k < v$ condition. Being U the set of users and I the set of items, the authors proposed grouping the preferences for each user u in the following sets W_u, A_u, and S_u.

- Set of weak ratings of the user u, W_u: $W_u = \{r_{ui} | \forall i \in I \text{ where } r_{ui} < k_u\}$
- Set of average ratings of the user u, A_u: $A_u = \{r_{ui} | \forall i \in I \text{ where } k_u \leq r_{ui} < v_u\}$
- Set of strong ratings of the user u, S_u: $S_u = \{r_{ui} | \forall i \in I \text{ where } r_{ui} \geq v_u\}$

Similarly, for each item i, the preferences are grouped in the following sets W_i, A_i, and S_i.

- Set of weak ratings of the item i, W_i: $W_i = \{r_{ui} | \forall u \in U \text{ where } r_{ui} < k_i\}$

- Set of average ratings of the item i, A_i: $A_i = \{r_{ui} | \forall u \in U \text{ where } k_i \leq r_{ui} < v_i\}$
- Set of strong ratings of the item i, S_i: $S_i = \{r_{ui} | \forall u \in U \text{ where } r_{ui} \geq v_i\}$

Considering these and the previously mentioned rating classes, the authors classified each user according to their biases: Benevolent, Average, Critical, and Variable. And similarly, they defined a classification for items as follows: Strongly-preferred, Averagely-preferred, Weakly-preferred, and Variably-preferred.

Each user or item is, then, classified according to the cardinality of each of the sets: W_u, A_u, S_u or W_i, A_i, S_i. Once this classification is performed, this information can be used to identify contradictions in the classes of the data and mark rating as noise. Algorithm 1 summarizes this approach to noise detection.

Algorithm 1 Detection of noisy ratings, as proposed by TOLEDO *et al.* (2015).

Input: $R = \{r_{ui}\}$ – set of available ratings, k_u, v_u, k_i, v_i, k, v, - classification thresholds
Output: possible noise = $\{r(u,i)\}$ – set of possible noisy ratings
$W_u = \{\}$, $W_i = \{\}$, $A_u = \{\}$, $A_i = \{\}$, $S_u = \{\}$, $S_i = \{\}$
possible_noise = {}
for *rating* $r_{ui} \in R$ **do**
  if $r_{ui} < k_u$ **then**
    Add r_{ui} to the set W_u
  else if $r_{ui} \geq k_u$ *and* $r_{ui} < v_u$ **then**
    Add r_{ui} to the set A_u
  else
    Add r_{ui} to the set S_u
  end
  if $r_{ui} < k_i$ **then**
    Add r_{ui} to the set W_i
  else if $r_{ui} \geq k_i$ *and* $r_{ui} < v_i$ **then**
    Add r_{ui} to the set A_i
  else
    Add r_{ui} to the set S_i
  end
end
for *each user* u *and item* i **do**
  if $|W_u| \geq |A_u| + |S_u|$ **then**
    Classify u as *critical*
  else if $|A_u| \geq |W_u| + |S_u|$ **then**
    Classify u as *average*
  else if $|S_u| \geq |W_u| + |A_u$ **then**
    Classify u as *benevolent*
  else
    Classify u as *variable*
  end
  if $|W_i| \geq |A_i| + |S_i|$ **then**
    Classify i as $weakly - preferred$
  else if $|A_i| \geq |W_i| + |S_i|$ **then**
    Classify i as $averagely - preferred$
  else if $|S_i| \geq |W_i| + |A_i|$ **then**
    Classify i as $strongly - preferred$
  else
    Classify i as $variably - preferred$
  end
end
for *rating* $r_{ui} \in R$ **do**
  if u *is critical,* i *is weakly-preferred, and* $r_{ui} \geq k$ **then**
    Add r_{ui} to the set *possible_noise*
  if u *is average,* i *is averagely-preferred, and* $r_{ui} < k$ *or* $r_{ui} \geq v$ **then**
    Add r_{ui} to the set *possible_noise*
  if u *is benevolent,* i *is strongly-preferred, and* $r_{ui} < v$ **then**
    Add r_{ui} to the set *possible_noise*
end

The next step formulated by the authors is the correction of the detected noisy ratings. The proposed algorithm is similar to the one proposed by O'MAHONY

et al. (2006), where a threshold τ is used to determine if a rating will be replaced by the one predicted using the model. Algorithm 2 shows how the process is carried out.

Algorithm 2 Filtering process for noisy ratings, as proposed by TOLEDO *et al.* (2015)

Input: $R^{'} \subseteq R$ - set of noisy ratings, τ - threshold, φ - predictive model.
Output: R^* - noiseless set of ratings.

```
R* ← {}
for (u, i, r) ∈ R' do
    r̃ ← φ(u, i)
    if |r̃ − r| < τ then
        R* ← R* ∪ {(u, i, r)}
    else
        R* ← R* ∪ {(u, i, r̃)}
    end
end
```

In order to determine the classification thresholds κ_u, ν_u, κ_i e ν_i, that are highly domain-dependant, TOLEDO *et al.* (2015) proposed three different methods. The first idea is named *global-pv*, where the threshold values are selected according to the data divided into three equal bins. In this approach, the threshold τ is defined as the difference between two ratings. The equations 4.1 and 4.2 shows how to calculate the values of all κ and ν, using the *round* function to determine the nearest value for n.

$$\kappa = \kappa_u = \kappa_i = min(P) + \lfloor \frac{1}{3} \cdot (max(P) - min(P)) \rceil \tag{4.1}$$

$$\nu = \nu_u = \nu_i = max(P) - \lfloor \frac{1}{3} \cdot (max(P) - min(P)) \rceil \tag{4.2}$$

The next approach chooses values based on the distribution of each user and item; that is, the thresholds vary according to the distribution of each user or item and are calculated using the mean $\bar{x}$ and the standard deviation σ. Using this approach, the thresholds are defined as follows:

$$\kappa_u = \bar{x}_u - \sigma_u \tag{4.3}$$

$$\nu_u = \bar{x}_u + \sigma_u \tag{4.4}$$

$$\kappa_i = \bar{x}_i - \sigma_i \tag{4.5}$$

$$\nu_i = \bar{x}_i + \sigma_i \tag{4.6}$$

This approach considers two alternatives to calculate the values of κ and ν: user-based ($\kappa = \kappa_u$ e $\nu = \nu_u$), named as *user-based-pv*, and item-based ($\kappa = \kappa_i$ e

$\nu = \nu_i$), named as *item-based-pv*. Finally, the threshold τ is defined as $\tau = \sigma_u$ for the *user-based-pv* method and $\tau = \sigma_i$ in the case of the *item-based-pv* method.

4.2.2 Label Noise-Robust Models

Label noise-robust refers to algorithms that are naturally robust to label noise, i.e., they suffer less influence in the presence of label noise than others (FRENAY & VERLEYSEN, 2014). These models have a learning process that is assumed to be not too sensitive to the presence of noisy instances, which means that they are less influenced by it. This subsection describes several methods that are robust to mislabeled instances.

GOLDBERGER & BEN-REUVEN (2016) models the noise as an additional softmax layer, namely a noise channel, on top of the network output. SUKHBAATAR & FERGUS (2014) propose a similar approach; however, their training process has two steps instead of the regular one commonly used in neural networks.

Being $p(y = i|x; w)$ a multi-class neural network classifier, where x is the feature vector, w the network weights, and i an element of the set of classes $1, ..., k$, they assume that, in the training process, the network can only directly observe a noisy version z of the correct label y. This noise distribution is unknown and the proposal consists of learning it during the training phase using the Expectation Maximization (EM) algorithm.

During the training process, the model receives n feature vectors $x_1, ..., x_n$ and its corresponding noisy labels $z_1, ..., z_n$, which are the noisy versions of the correct labels $y_1, ..., y_n$. Equation 4.7 shows the log-likelihood of the model parameters.

$$L(w, \theta) = \sum_{t=1}^{n} \log(\sum_{i=1}^{k} p(z_t|y_t = i; \theta)p(y_t = i|x_t; w)) \tag{4.7}$$

From those inputs, the model aims to maximize with respect to the noise distribution θ and the model parameters w. The EM algorithm is, then, applied to estimate the hidden true labels $y_1, ..., y_n$ and update the neural network and the noisy channel parameters. Each E-step of an iteration is responsible for the former and may be defined as:

$$c_{ti} = p(y_t = i|x_t, z_t; w_0; \theta_0), i = 1, ..., k, t = 1, ..., n, \tag{4.8}$$

where w_0 and θ_0 are the current parameters. On the other hand, each M-step of an iteration is responsible for the latter and thus can be defined in two different phases. To update the noise distribution, the following function can be used:

$$\theta(i, j) = \frac{\sum_t c_{ti} 1_{z_t=j}}{\sum_t c_{ti}}, i, j \in 1, ..., k, \tag{4.9}$$

where the $k \times k$ matrix θ can be viewed as a confusion matrix between the estimated true labels c_{ti} and the noisy labels z_t. The second phase is to update the neural network parameters. For this end, the following function is maximized:

$$S(w) = \sum_{t=1}^{n} \sum_{i=1}^{k} c_{ti} \log p(y_t = i | x_t; w), \tag{4.10}$$

where h is the final hidden layer and $u_1, ..., u_k$ are the parameters of the softmax output layer.

REED *et al.* (2015) proposes different approaches consisting of modifying the cost function to deal with noisy labels. One of their proposals, referred to as bootstrapping, avoids the direct modeling of the noise distribution by using a convex combination of the training labels and the model's predictions during a given model iteration. The main idea of this approach is that the model improves over time, making its predictions more reliable and, therefore, hindering the effect of the noisy label by giving less importance to them. In practice, they extended the cross-entropy cost function to generate new regression targets for each SGD mini-batch based on the model's current state.

The authors evaluated two types of bootstrapping. The soft bootstrapping generates regression targets for each batch using the predicted class probabilities q as follows,

$$\mathcal{L}_{soft}(q, t) = \sum_{k=1}^{L} [\beta t_k + (1 + \beta) q_k] \log(q_k). \tag{4.11}$$

The hard bootstrapping modifies the regression targets using the MAP estimate of q given x as follows,

$$\mathcal{L}_{hard}(q, t) = \sum_{k=1}^{L} [\beta z_k + (1 + \beta) q_k] \log(q_k), \tag{4.12}$$

where β is a scalling factor, and $z_i = \mathbf{1}$, $i = \text{argmax } \tilde{y}_i, i = 1...C$ with C standing for the number of labels.

PATRINI *et al.* (2016a) proposed two domain-independent procedures for loss correction in order to make supervised models more resistant to noise. However, both approaches need knowledge about the noise distribution to work properly, i.e., it needs a stochastic square matrix T showing the probability of one class being flipped into another. The main idea is to use those procedures along with a learning model, e.g., artificial neural network, to increase its robustness.

Considering the asymmetric noise, where each training set instance label $\boldsymbol{y}$ may be flipped into $\tilde{\boldsymbol{y}} \in \mathcal{Y}$ with a probability of $p(\tilde{\boldsymbol{y}}|\boldsymbol{y})$, while the feature vectors remain

unaltered. The sample distribution may be calculated as follows:

$$p(\boldsymbol{x}, \tilde{\boldsymbol{y}}) = p(\tilde{\boldsymbol{y}}|\boldsymbol{x})p(\boldsymbol{x}) = \sum_{\boldsymbol{y}\in\mathcal{Y}} p(\tilde{\boldsymbol{y}}|\boldsymbol{y})p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x}).$$

Thus, the noise transition matrix $T \in [0,1]^{c\times c}$ can be defined, denoting the probability of one label i being flipped into another label j. This row stochastic matrix is defined as:

$$T_{ij} = p(\tilde{\boldsymbol{y}} = \boldsymbol{e}^j | y = \boldsymbol{e}^i),$$

where $\boldsymbol{e}^i$ means the i-th standard canonical vector $\mathbb{R}^c$, i.e., $\boldsymbol{e}^i \in \{0,1\}^c, \mathbf{1}^\top \boldsymbol{e}^i = 1$.

The basis of both noise correction methods proposed by PATRINI *et al.* (2016b) is to incorporate matrix T into the loss function ℓ, e.g., cross entropy for artificial neural network models. There are two processes: the backward process and the forward process.

In the backward process, an unbiased estimator using the corrected loss function equals the original loss computed in clean data. The theorem by (PATRINI *et al.*, 2016b), which is based on previous work by PATRINI *et al.* (2016a), shows that, under specific characteristics, the minimizers will be the same for the data with or without the noise. In practice, the corrected loss can be considered as a linear combination of the loss values for each row of T^{-1}. One of the drawbacks of this method is that, in practice, T must be invertible.

Alternatively, the forward procedure focuses on correcting the model's predictions themselves. The process is formalized as follows:

$$\ell^{\rightarrow}(p(\boldsymbol{y}|\boldsymbol{x})) = \ell(T^T p(\boldsymbol{y}|\boldsymbol{x})) \tag{4.13}$$

To analyze the behavior of this process, we will introduce concept of a family of losses called *proper composite* (REID & WILLIAMSON, 2010). Given an invertible *link* function $\boldsymbol{\psi} : \Delta^{c-1} \rightarrow \mathbb{R}^c$, the cost function, will be considered a *composite loss* and denoted by $\ell_{\boldsymbol{\psi}} : \mathcal{Y} \times \mathbb{R}^c \rightarrow \mathbb{R}$, if it can be expressed using $\boldsymbol{\psi}^{-1}$, which is,

$$\ell_{\boldsymbol{\psi}}(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) = \boldsymbol{\ell}(\boldsymbol{y}, \boldsymbol{\psi}^{-1}(\boldsymbol{h}(\boldsymbol{x}))) \ . \tag{4.14}$$

For the cross-entropy loss, the softmax function is the invertible *link* function. A *composite loss* may also be a proper loss, i.e., proper composite, which means that their minimizer has the particular shape of the link function applied to the class-conditional probabilities $p(\boldsymbol{y}|\boldsymbol{x})$:

$$\operatorname*{argmin}_{\boldsymbol{h}} \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}\ \ell_{\boldsymbol{\psi}}(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) = \boldsymbol{\psi}(p(\boldsymbol{y}|\boldsymbol{x})) \ . \tag{4.15}$$

Cross-entropy and square functions are proper composite losses, for example. The forward procedure uses the same minimizers using appropriate composite cost functions.

Both approaches suppose the knowledge of a noise transition matrix; however, in real-life problems, this matrix is usually unknown. PATRINI *et al.* (2016b) estimates this matrix, assuming that there is a perfect example for each class in the training data; however this solution is unsuitable for the collaborative filtering problem since user preferences are subjective and can change depending on many factors, such as social influences and context. CARMO (2018) extended PATRINI *et al.* (2016b) work and proposed an efficient way to construct this noise transition matrix for the collaborative filtering problems.

Since there is no way to determine a perfect example for collaborative filtering problem, CARMO (2018) approach focuses on selecting ratings with a low degree of inconsistency, i.e., ratings given by a user that are close to their actual values. Formally:

Definition 1. Being a rating r_{uv} given by a user u to an item v and $\breve{r}_{uv}$ his actual preference to this item, r_{uv} is considered an *anchor*, if and only if,

$$|\breve{r}_{uv} - r_{uv}| < \tau \ , \tag{4.16}$$

where τ is a threshold close to zero.

To select the ratings to be part of the *anchors* set, the user's actual preferences are unknown and must be estimated. In order to estimate the unknown value of $\breve{r}_{uv}$, CARMO (2018) proposed the use of a probability estimator. The procedure can be seen in the algorithm 3.

Algorithm 3 Determine anchors for the collaborative filtering problem.

Input: ratings set - R, threshold - τ, prediction model - φ.
Output: anchors set - $\mathcal{A}$.

$\mathcal{A} = \{\}$
for $(u, i, r) \in R$ **do**
  $\tilde{r} \leftarrow \varphi(u, i)$
  if $|\tilde{r} - r| < \tau$ **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(u, i, r)\}$
  end
end

Using the obtained *anchors* set, it is possible to calculate an approximation of the noise transition matrix. The proposal uses a probabilistic estimator to calculate each anchor probability. The procedure is summarized in the algorithm 4.

Algorithm 4 Estimating transition matrix T proposed by CARMO (2018).

Input: $\mathcal{A}|\mathcal{A} \subset R$ – set of anchor's ratings, P – ratings set, $\hat{p}$ – probability estimator

Output: T – noise transition matrix

for $c \in P$ **do**
  $\mathcal{A}_c \leftarrow \{\forall_{(u,i,r)} \in \mathcal{A}|r = c\}$ **for** $w \in P$ **do**
    for $(u, i, r) \in \mathcal{A}_c$ **do**
      $T \leftarrow \hat{p}(\tilde{y} = c|u, i, w) \cdot ||\mathcal{A}_c||^{-1}$
    end
  end
end

4.3 Empirical Results

In this section, we detail our experiments and results using label noise applied to the malicious noise problem.

Our experiments are carried out to assert our two main assumptions: data cleansing can detect and filter malicious instances and diminish the impact of a shilling attack, and label noise approaches are robust to malicious noise and can be applied successfully to the shilling attack problem in collaborative filtering. Firstly, we analyze how data cleansing approaches can filter noisy instances and mitigate shilling attacks and how these results can be compared to the ones obtained previously. Secondly, in the same way, we compare each label noise-robust learning algorithm against each other, evaluating its performance and determining if it is a feasible alternative in the presence of a shilling attack as it is in the presence of natural noise. In experiments, we use the following attack models to evaluate the data cleansing and Robust techniques: Variational Autoencoder, Random, Average, Bandwagon, Segment, PUA-NR, and PIA-NR.

The methodology used here to carry out our experiments will closely follow the one presented in section 3.4.

4.3.1 Metrics

In order to keep the consistency between all experiments carried out on this work and facilitate comparisons, we will be using the same metrics set used in the subsection 2.2.4: prediction shift, hit ratio, and average rank.

4.3.2 Setup

We will carry out two different experiments to verify the effectiveness of some label noise proposals when attacked using several well-known attack models on different scenarios. Therefore, two similar experiments will be performed to evaluate those label noise approaches. The first one will use different attack models to mount attacks against a multilayer perceptron (MLP)-based recommender system (HE *et al.*, 2017), using data cleansing methods to clean the training set in order to compare these methods' efficiency on malicious data. The second one will compare several label noise-robust algorithms by mounting attacks against MLP-based systems using the modifications proposed by each of these works.

In both experiments, we will attack the MLP system using well-known attack models, including Random, Average, Segment, Bandwagon, PUA-NR, and PIA-NR. Additionally, we will attack it with the Variational Autoencoder, which is our proposal from the previous chapter.

Both experiments will be conducted using a hold-out strategy, that is, partitioning the data into two sets: 70% for train and 30% for testing. The train data is used to create the malicious profiles, while the test data is used only during the evaluation of the attack model. For the first experiment, we will assume two different scenarios: the first will assume no knowledge about malicious users and the entire training set will be used to perform the data cleansing approach, while the second will assume that the system administrator has a set of trusted users and only the real users will be used to perform data cleansing.

Similarly to section 3.4 experiments, in order to evaluate the models from different perspectives, we selected the same 30 distinct target items from these experiments: ten randomly chosen among all items, ten chosen among established items, that is, top-50 most-rated items, and ten chosen among new items, that is, items with only one rating (SEMINARIO & WILSON, 2014a). Since our attack intent is to push, the target item rating was set to the maximum value, which is 5 in the MovieLens 100k and Yahoo! Music datasets. We averaged the results for each category of target item and each chosen metric, and reported the results.

The MLP architecture was composed of two hidden layers with 128 neurons and a dropout of 0.1 in each. We used the Adam optimizer and stopped the training if no improvement was detected for five consecutive epochs. The batch size chosen was 128 samples per batch. We modified the cross-entropy cost function according to each proposal to increase the label-noise tolerance of the model.

Regarding the attack models parameters, we varied the attack size by 1%, 3%, 5%, 10%, and 20%, and filler size, for the models that require it, will be fixed in 7% for the MovieLens 100k dataset and 3% for the Yahoo! Music dataset. We

chose this approach because it means sampling 118 items as filler items in case of MovieLens 100k and 25 items in case of Yahoo! Music, which is around the average of items composing a genuine profile in each of these datasets. In addition, for the selected items set, in segment attack, only carried out for MovieLens 100k, we selected popular horror movies segment (BURKE, 2007) and for the bandwagon attack, we selected movies (IDs) {50, 56, 100, 127, 174, 181, 258, 286, 288, 294} as popular movies for MovieLens 100k, which are movies rated by more than 300 users (LI & LUO, 2011), while opting for the 10 most rated songs for Yahoo! Music. For *Power User Attack* and *Power Item Attack*, we chose the *NumRatings* approach for power user/item selection, since it yields similar results to those of *InDegree* approach, but with better performance overall. In *Power Item Attack* there is a parameter that controls the number of power items used to construct each malicious profile, we varied it by 50, 40, and 30. Parameters for the Variational Autoencoder were the same chosen in the previous chapter.

Regarding the data cleansing parameters, O'Mahony used a threshold th obtained from previous experiments conducted with the same methodology described in subsection 3.4.1 for selecting parameters for the Variational Autoencoder. Specifically, a threshold of 0.37 was chosen for MovieLens 100k, and 0.15 was chosen for Yahoo! Music. For Toledo, the global approach was used for both datasets, which means the threshold $th = 0.25$, $\kappa = 0.25$, and $\mu = 0.25$.

In short, two experiments are carried out to evaluate data cleansing and label noise-robust methods:

Experiment I

Comparison with seven shilling attack models: *Variational Autoencoder*, *Random*, *Average*, *Segment*, *Bandwagon*, *PUA-NR* and *PIA-NR* using MovieLens 100k and Yahoo! Music datasets on two *MLP*-based system. One with noisy instances treated using O'MAHONY *et al.* (2006) method, while the other uses TOLEDO *et al.* (2015) proposal as the noise treatment.

Experiment II

Comparison with seven shilling attack models: *Variational Autoencoder*, *Random*, *Average*, *Segment*, *Bandwagon*, *PUA-NR* and *PIA-NR* using MovieLens 100k and Yahoo! Music datasets on *Carmo* (using the forward method), *Noise Channel*, *Bootstrapping Soft* and *Bootstrapping Hard* systems.

4.3.3 Results

In this subsection, we present and discuss the obtained results from the evaluated experiments. Additionally, we compare data cleansing and label noise-robust approaches against several well-known shilling attack models.

Experiment I

In *Experiment I* we analyze how data cleansing approaches are able to protect collaborative filtering-based recommender systems from shilling attacks in two different scenarios. The first scenario presented will consider that the system does not have access to a set of trusted users.

Figures 4.3, A.4, and A.5 shows the comparison between (a) MLP without any data cleansing step applied, (b) MLP with O'MAHONY *et al.* (2006) applied before training the model, and (c) MLP with TOLEDO *et al.* (2015) applied using MovieLens 100k dataset with, respectively, selected random target items, the least-rated target item, and the most-rated target items. For all these experiments, it is possible to see relevant results from multiple approaches in all metrics, such as PIA-NR, Variational Autoencoder, PUA-NR, and Average attacks, while Bandwagon shows weak results, but still causes a lesser impact on the recommender. Regarding the treatments applied, in comparison, O'Mahony (b) and Toledo (c) approaches do not seem to alleviate the effects of any attack models under these conditions, reporting results similar to the baseline without any data cleansing treatment (a).

As expected, the results indicate that data cleansing approaches are highly dependent on the quality of the data used to train the estimator, which is then used to predict and replace the noisy ratings in the dataset. Consequently, when the data is already contaminated with attacks, the predictions made by the estimator for the attacked items are close to the biased injected values and fall below the selected threshold. This results in nearly identical outcomes regardless of the treatment applied. The conclusion thus far is that data cleansing approaches are ineffective as a defense mechanism for the MovieLens 100k dataset in the absence of a set of trusted users.

Following the evaluation using the Movielens dataset, the Figures 4.4, 4.4, and A.8 shows the same comparative results between (a) the baseline MLP, (b) O'Mahony applied to the data before running th MLP, and (c) Toledo applied under the same circunstances. However, using the R3 Yahoo! Music dataset with, respectively, selected random target items, the least-rated target item, and the most-rated target items. Please note that due to the lack of information about the items, it was not possible to evaluate the segment attack using this data. For all three methodologies for selecting target items, the baseline shows that PUA-NR stands out and causes significant damage to the system, while the other attacks have a lesser effect. The situation remains the same for both the O'Mahony and Toledo approaches, as they are unable to mitigate the effects of the attacks, similar to what was observed during the MovieLens 100k evaluation.

Similar to the previous experiment with the MovieLens 100k data under the same

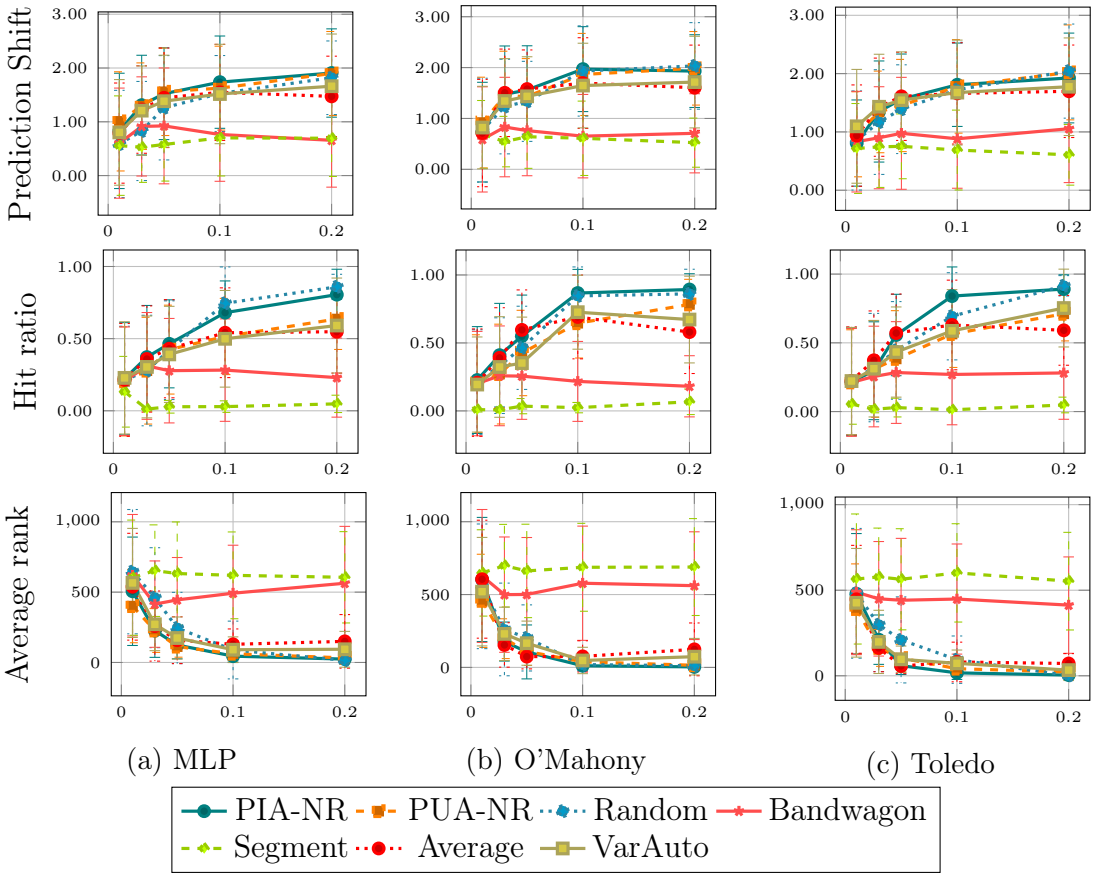


Figure 4.3: Prediction shift, hit ratio, and average rank for different attack sizes comparing the baseline with O'Mahony's and Toledo's Data Cleansing approach using random target items from MovieLens 100k data set and with attackers injected in the base estimator training data. Note that it only shows the best result of filler size among the attack models that require it.

conditions, the reported results reinforce the high dependency on the data, indicating that this setup will not produce favorable outcomes regardless of the domain. In all scenarios studied, the differences in the reported results are minimal. Thus, it can be concluded that data cleansing approaches are ineffective in defending the system against malicious noise in the music domain as well under these conditions.

Given the results obtained in all scenarios using both datasets, it is possible to argue that these data cleansing approaches cannot prevent shilling attacks at all given a scenario where the system administrator does not have access to a set of trusted users. With the training data already biased by attack profiles inserted in the ratings database, the results indicate that data cleansing schemes have difficulties filtering malicious instances from the data.

After the initial evaluation and conclusion that data cleansing methods are ineffective without a set of trusted uses, the second scenario assumes that the system has access to these trusted consumers in its database. In this case, the estimator is

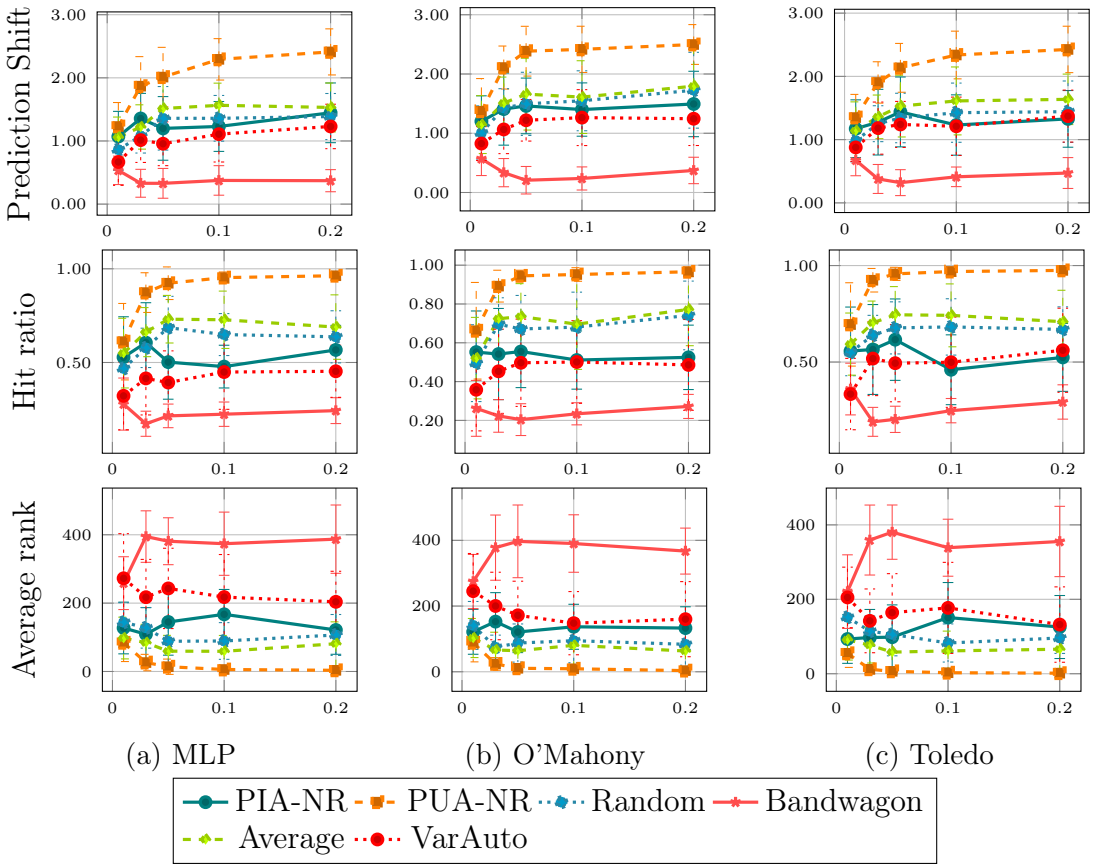


Figure 4.4: Prediction shift, hit ratio, and average rank for different attack sizes comparing the baseline with O'Mahony's and Toledo's Data Cleansing approach using random target items from Yahoo! Music data set and with attackers injected in the base estimator training data. Note that it only shows the best result of filler size among the attack models that require it.

trained using only the set of actual users, excluding the attackers. In Figures 4.5, A.6, and A.7, we display (a) the baseline results, (b) the results applying O'Mahony's method, and (c) the results applying Toledo's approach, using the MovieLens 100k dataset with random, least-rated, and most-rated target items selected, respectively. This time; however, it is possible to note how O'Mahony's technique is able to effectively reduce the effects of the attack to the bare minimum. The prediction shift is near zero, and all attack models have difficulties pushing the item up the recommendation list. Toledo's technique, on the other hand, does not show major improvements over the baseline, reporting similar results to the ones found earlier.

Based on the results obtained, there is strong indication that the simple detection approach formulated by O'Mahony is more effective for detecting malicious noise compared to the classification-based detection proposed by Toledo. In shilling attack schemes, the entire malicious profile can be considered noise; however, data cleansing approaches might need to focus on detecting and correcting only the target item

to mitigate the effects of the attack. Analyzing Toledo's algorithm, for a rating to be considered noise in our push attack scenario, the user-item pair must meet specific criteria. Therefore, it is reasonable to suggest that malicious ratings for the target item may be rarely detected, depending on the user and item individual classification.

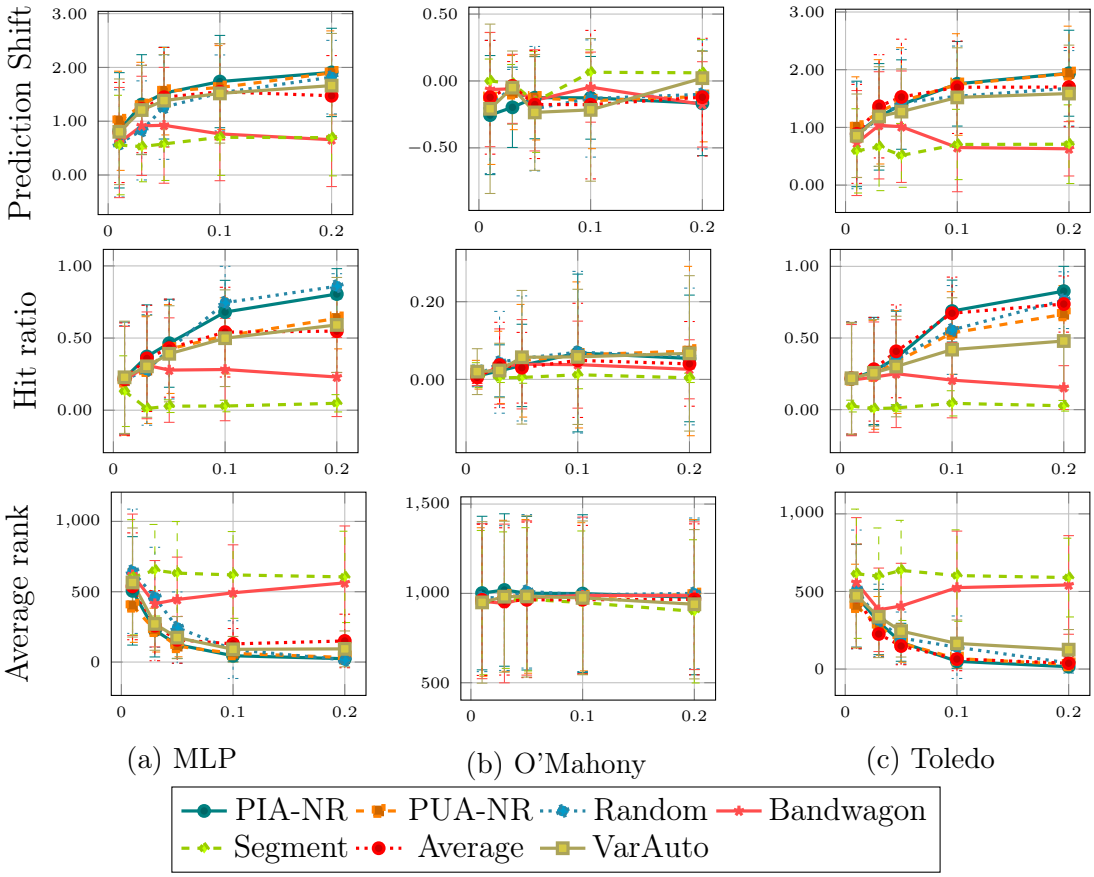


Figure 4.5: Prediction shift, hit ratio, and average rank for different attack sizes comparing the baseline with O'Mahony's and Toledo's Data Cleansing approach using random target items from MovieLens 100k data set and no attackers in the base estimator. Note that it only shows the best result of filler size among the attack models that require it.

After the evaluation with the MovieLens 100k dataset, we will move on to evaluate the same experiment using Yahoo! Music dataset. The figures 4.6, A.10, and A.11 displays the comparative results for (a) baseline, (b) O'Mahony, and (c) Toledo, this time, for Yahoo! Music dataset and, respectively, randomly selected, randomly selected among the least-rated, and randomly selected among the most-rated target items. All figures show that the baseline and Toledo method report similar results, with PUA-NR causing the most damage to the system. However, the O'Mahony approach reduces the impact of all attack models to minimal levels, with a prediction shift of at most 0.4. This indicates that, under the right conditions, this approach

can effectively protect the recommender system also with music domain data.

Please note that the same behavior observed during the MovieLens 100k evaluation is also noted here with a dataset from a different domain and distribution. This suggests that having too many rules for classifying users and items may hinder the identification of malicious noise using Toledo's global approach. Thus, we can conclude that a simpler ruleset is more effective for addressing the shilling attack problem in the settings presented in this experiment.

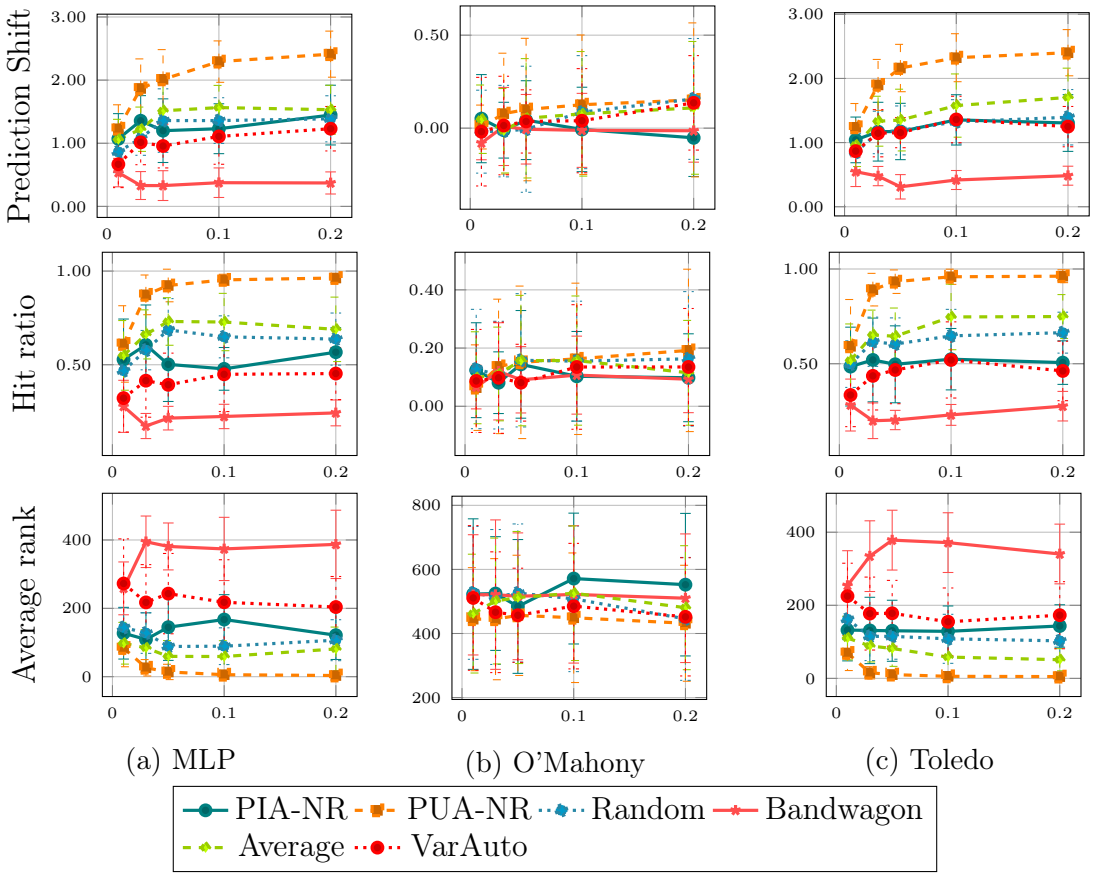


Figure 4.6: Prediction shift, hit ratio, and average rank for different attack sizes comparing the baseline with O'Mahony's and Toledo's Data Cleansing approach using random target items from Yahoo! Music data set and no attackers in the base estimator. Note that it only shows the best result of filler size among the attack models that require it.

In short, having a set of trusted users is essential for achieving good results with data cleansing approaches. While Toledo's approach yields weak results, O'Mahony's approach effectively nullifies the effects of the selected attack model, proving to be an effective method for protecting a recommender system against attacks, provided the system's administrator has access to a set of trusted users. In more realistic scenarios, where it is not possible to identify trusted users a priori, these techniques are unable to distinguish attackers from genuine users. While we

cannot pinpoint exactly why Toledo's approach did not achieve results as competitive as O'Mahony's, it is possible to argue that a different set of rules tailored to better address various types of malicious noise could improve the approach's effectiveness. Experiments to confirm this hypothesis; however, are left for future work.

Experiment II

In *Experiment II* we analyze how well label noise-robust techniques maintain their predictions stable in the presence of shilling attacks. In Figures A.12 and A.13, we show comparative results for (A.12a and A.13a) baseline without any modifications to deal with label noise in the cost function, (A.13b) Bootstrapping Hard cost function, (A.13c) Bootstrapping Soft cost function, (A.13b) Noise Channel, and (A.13c) the solution presented by CARMO (2018) using the MovieLens 100k dataset with the least-rated target items selected. With this setup, most of the attack models report similar results regarding all three metrics, easily moving the target items up in the recommendation list. Notably, Variational Autoencoder, PUA-NR, PIA-NR, and Average report the best results, while Segment is not able to cause enough damage, which is expected since this attack is designed to attack a subset of the items in the recommender system. Comparing the two Bootstrapping approaches, Noise Channel, and Carmo with the baseline, it is possible to note that the effects of each attack are not alleviated at all, with the modifications in the cost function yielding similar results to the baseline.

These results indicate that the selected robust approaches do not have the desired effect when applied to the malicious noise problem. In general, the only attack that shows any difference between the treatments is Segment, where the Carmo approach seems to reinforce the attack effects.

The comparison for target items sampled from the most-rated items set of the MovieLens 100k is shown in Figures A.14 and A.13, comparing once more results for (A.14a) baseline MLP with the default cost function, (A.14b) Bootstrapping Hard cost function, (A.14c) Bootstrapping Soft cost function, (A.13b) Noise Channel, and (A.13c) Carmo's proposal. As usual, all attack models show some difficulties in pushing already established items up the recommendation list, which explains why the impact is less noticeable when trying to affect items in this set. Despite the difficulties, PUA-NR, PIA-NR, and Random attacks manage to achieve good results, while Variational Autoencoder achieves the best results using fewer profiles despite not being able to improve with more profiles injected. One more time, Segment is unable to shift predictions up. When comparing the modifications to the cost function with the baseline, once more, the two Bootstrapping approaches, and Noise Channel cannot prevent the undesirable effects of the attack models, reporting results on par with the baseline. While all approaches yields unfavorable

results, Carmo's approach actually reinforces the effects of the attacks when applied to malicious noise, showing that this approach ends up achieving the opposite effect under these conditions and confirming our initial analysis.

Finally, the last comparison is using target items sampled from the whole universe of possible items in the MovieLens 100k dataset. The results for the approaches are shown in Figures 4.7 and 4.8, comparing (4.7a) the baseline with (4.7b) the Bootstrapping Hard cost function, (4.7c) the Bootstrapping Soft cost function, (4.8b) Noise Channel layer, and (4.8c) Carmo's approach. In general terms, most of the attack models report high values of prediction shift and hit ratio as the attack size grows while reporting low values of average rank. Note that PUA-NR, PIA-NR, Average, Random, and Variational Autoencoder are able to shift the target items without any issues. When focusing on the comparison between the baseline and the two Bootstrapping approaches, it is noted that Variational Autoencoder shows slightly better results against the Bootstrapping Soft and Hard cost functions, indicating that this solution may reinforce the attack's expected behavior rather than mitigate it. The same can be said for Carmo, though with greater intensity, as observed earlier. This suggests that the anchors algorithm should be reworked to better prevent shilling attacks and replicate the success of the approach against natural noise.

In short, for the MovieLens 100k dataset, it is possible to assume that label noise-robust techniques do not show competitive results when applied against malicious noise, not showing the same results reported against natural noise and even reinforcing attack behaviors in some cases. The results are still limited to the movie domain; however it is possible to argue that the different nature of the malicious noise problem compared with the natural noise plays an important part since, in a traditional natural noise setting, the algorithms are designed to correct local noisy occurences in data, not whole profiles.

Following the MovieLens 100k dataset evaluation, we move on to evaluate if label noise-robust techniques are able to keep predictions stable in the presence of shilling attacks using Yahoo! Music dataset. In Figures A.16 and A.17, we show comparative results for (A.16a) baseline without any modifications to deal with label noise in the cost function, (A.16b) Bootstrapping Hard cost function, (A.16c) Bootstrapping Soft cost function, (A.17b) Noise Channel and (A.17c) the solution presented by Carmo using Yahoo! Music dataset with least-rated target items selected. Please note that, once again due to the lack of information about the items in this dataset, the Segment attack could not be carried out. PUA-NR achieves the best results, followed by the Average attack. Except for Bandwagon, which fails to push the target item, all other models show acceptable performance against the baseline, Bootstrapping-based techniques, and the Noise Channel. This indicates that neither modification

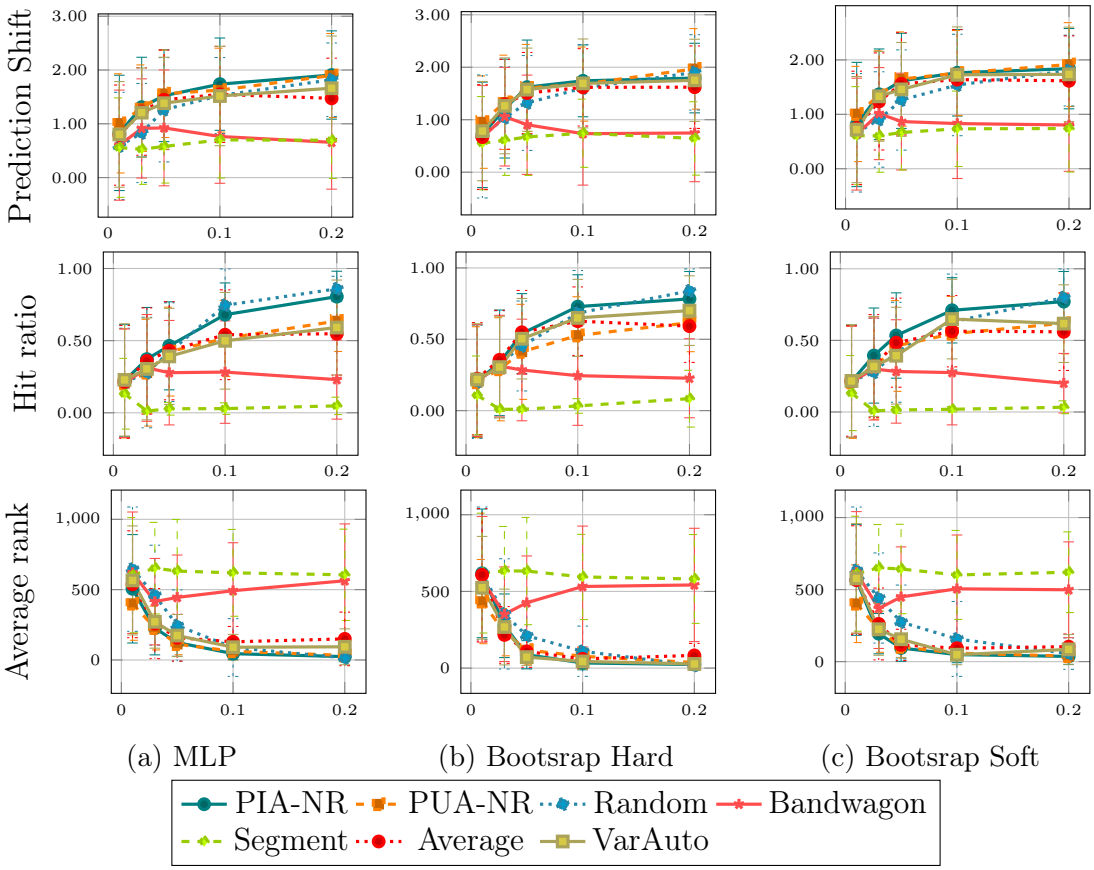


Figure 4.7: Prediction shift, hit ratio, and average rank for different attack sizes comparing a regular MLP with Bootstrapping techniques using random target items from MovieLens 100k data set. Note that it only shows the best result of filler size among the attack models that require it.

of the cost functions alleviates the effects of the attack models in the music domain. The approach introduced by Carmo, on the other hand, reinforces the impact of all attacks, resulting in considerably worse performance compared to the baseline, Bootstrapping methods, and the Noise Channel.

The comparison for target items sampled from the most-rated items set of the Yahoo! Music is shown in figures A.18 and A.19, comparing again the results for (A.18a) baseline MLP with the default cost function, (A.18b) Bootstrapping Hard cost function, (A.18c) Bootstrapping Soft cost function, (A.19b) Noise Channel and (A.19c) Carmo's proposal. Once more, PUA-NR attack model stands out, this time being the only attack able to obtain results above the average levels. However, it is important to note that in MovieLens 100k, none of the approaches managed to reach these levels with items already established in the dataset. Regarding the comparison between the baseline and the Bootstrapping approaches, channel, and Carmo, there is not much difference between the treatments, supporting the initial claim that Label Robust techniques do not seem to make a difference when applied

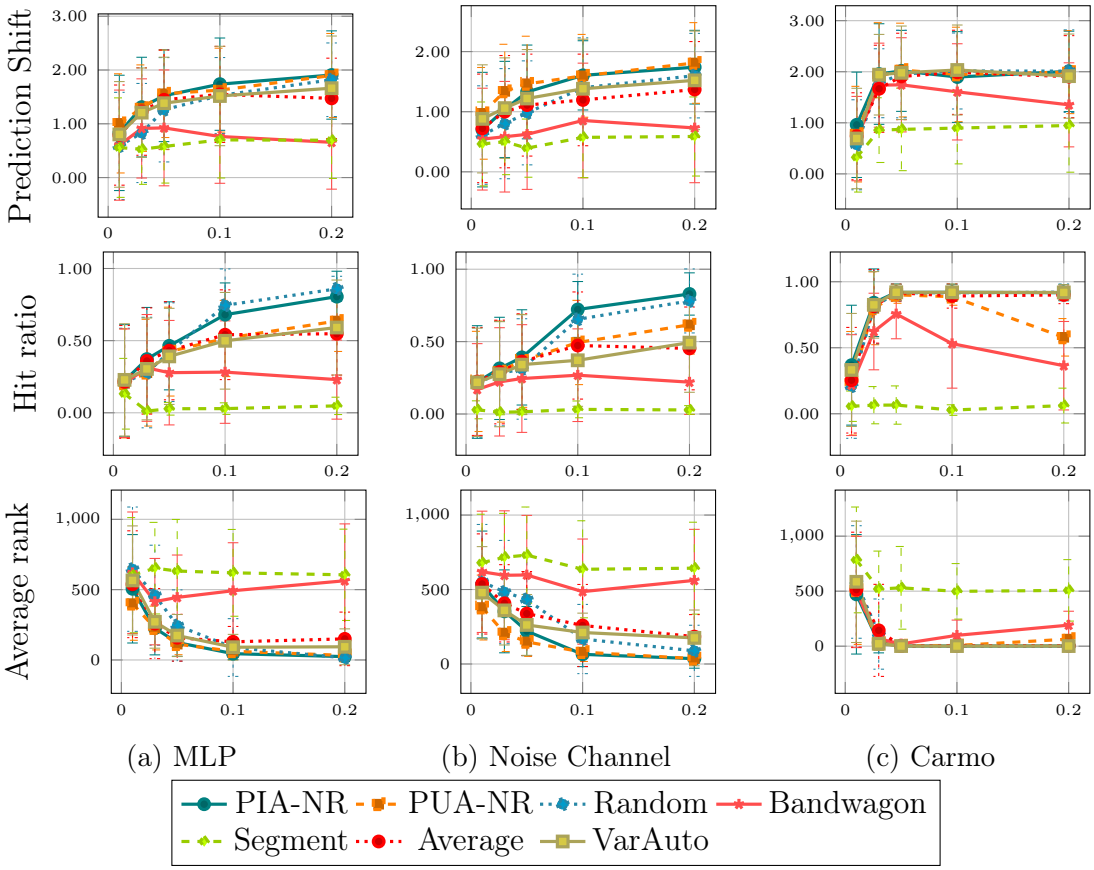


Figure 4.8: Prediction shift, hit ratio, and average rank for different attack sizes comparing a regular MLP with Noise Channel and Carmo techniques using random target items from MovieLens 100k data set. Note that it only shows the best result of filler size among the attack models that require it.

to malicious noise. The comparison between the baseline and Carmo's proposal shows similar results to the ones reported earlier, with the second reinforcing the effects of the attack models.

Lastly, we will compare using target items sampled from the whole universe of possible items of the Yahoo! Music dataset. For the first two approaches, the results are shown in figures 4.9 and 4.10, with the comparison of (4.9a) the baseline with (4.9b) the Bootstrapping Hard cost function, (4.9c) the Bootstrapping Soft cost function, (4.10b) the Noise Channel layer, and (4.10c) Carmo's approach. Similar to the previously reported results, the PUA-NR attack models achieve the best results across all treatments with minimal differences, indicating that in the music domain, Bootstrapping-based approaches and the Noise Channel are also ineffective at preventing the effects of shilling attacks. When comparing the baseline with Carmo's approach (see Figure 4.10c), the same pattern is observed: Carmo's approach reinforces the effects of the attacks, with only Bandwagon failing to cause damage to the system in this case.

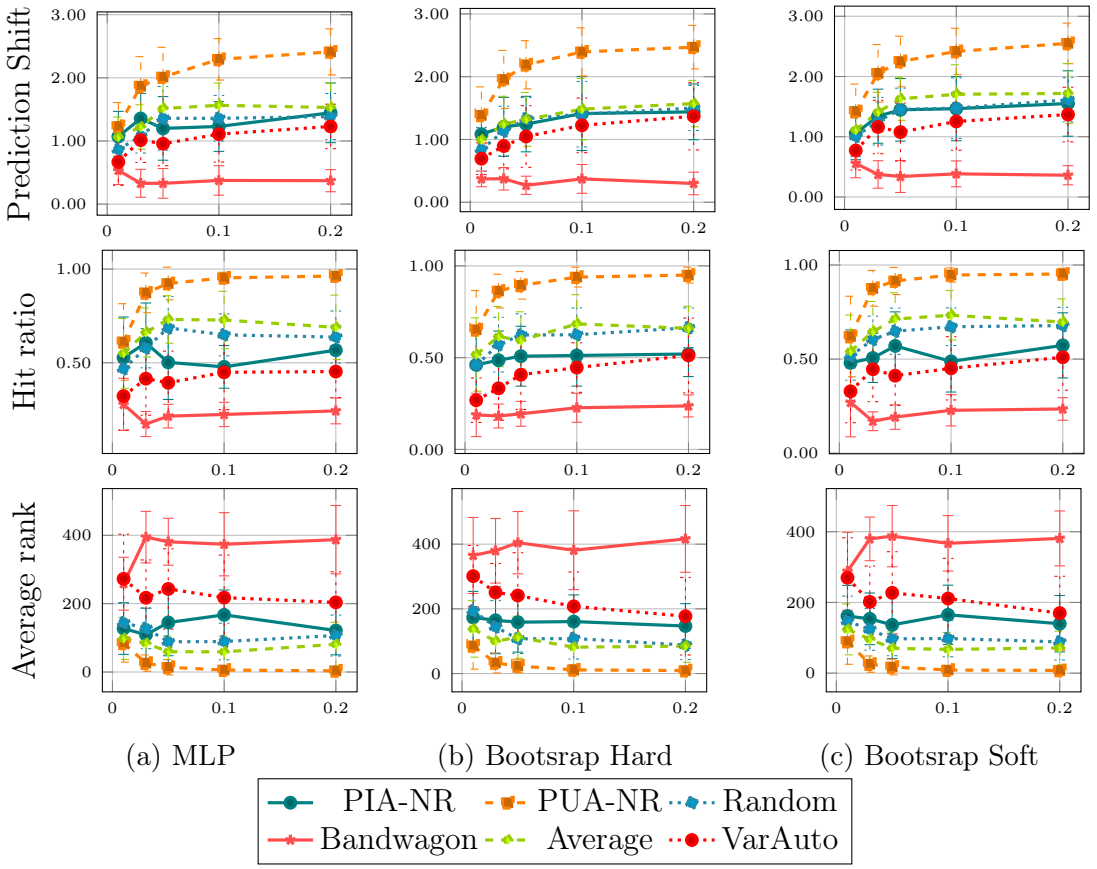


Figure 4.9: Prediction shift, hit ratio, and average rank for different attack sizes comparing a regular MLP with Bootstrapping techniques using random target items from Yahoo! Music data set. Note that it only shows the best result of filler size among the attack models that require it.

In general, it is evident that power users can push target items further up the recommendation list on Yahoo! Music compared to MovieLens 100k. This may be attributed to the unique characteristics of music recommendations, where users tend to either strongly like or dislike a song, often assigning the minimum or maximum rating. Another important point is that, while other approaches do not achieve the same level of impact as the PUA-NR attack, the results are still significant. They indicate that none of the collaborative filtering robust algorithms presented in this chapter are robust against malicious noise.

Additionally, these results highlight fundamental differences between natural and malicious noise problems. Natural noisy data in a collaborative filtering setting might include a misclicked rating or preferences saved by different family members using the same profile. In contrast, malicious noise often involves an entire set of multiple profiles used to inject noise in a coordinated way, making it challenging for label noise-robust models to handle. These models are designed to correct small perturbations in the data, rather than deal with multiple coordinated ratings for

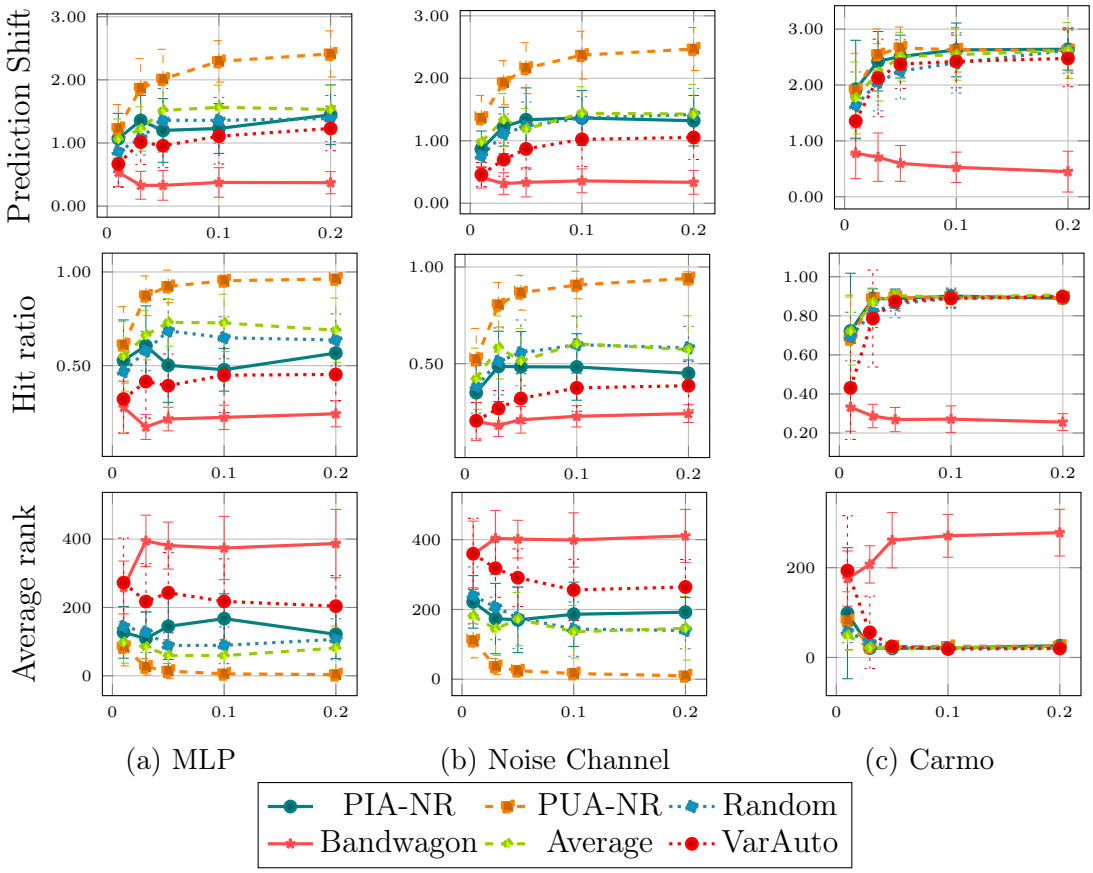


Figure 4.10: Prediction shift, hit ratio, and average rank for different attack sizes comparing a regular MLP with Noise Channel and Carmo techniques using random target items from Yahoo! Music data set. Note that it only shows the best result of filler size among the attack models that require it.

the same item. Specifically, for Carmo's MLP method, shilling attacks not only avoid detection but also exacerbate the effects of the attack, regardless of the domain. This occurs because the algorithm treats the ratings of the malicious target items as anchors and calculates the probabilities in the transition matrix based on them, effectively telling the model that these ratings are close to their true values. Therefore, this approach is currently unsuitable for preventing shilling attacks.

Chapter 5

Evaluating Detection Models for Shilling Attacks

This chapter introduces and reviews shilling attack detection models, one of the main strategies to diminish the effects of attacks. At the end of the chapter, we will apply the presented detection models to well-known shilling attacks in the literature, as well as, our proposal presented in Chapter 3. We will also discuss and compare these results with results obtained by applying the same detection techniques to a real-world dataset, with real attackers.

5.1 Detection Models

Detection models are important to protect recommender systems against attacks and mitigate their effects. According to GUNES *et al.* (2012), detection models accounted for 29% of the published works until 2012, being one of the most researched topics in shilling attack. There have been various models proposed for detection over the years, such as statistical techniques, classification, unsupervised clustering, and variable selection (GUNES *et al.*, 2012). In the following subsections, we will present and discuss several detection approaches separated into three distinct categories: supervised, semi-supervised, and unsupervised techniques (LI *et al.*, 2016).

5.1.1 Supervised

In general, supervised detection models involve deriving features for each profile based on user rating patterns and training a classifier to predict whether a profile is an attacker. This subsection will present several supervised methods for detecting shilling attacks.

DegreeSAD (LI *et al.*, 2015) is a supervised approach that proposes running a classifier on top of statistical features derived from the data. The authors presented

a set of three features based on the popularity degree distribution of items rated by real users: User's popularity average (MUD), User's popularity extreme difference (RUD), and User's popularity upper quartile (QUD).

MUD represents the average popularity degree of the items rated by each user in the whole dataset. It can be calculated as follows, where d'_k is an element of the user u popularity vector, and G_u is the list of the popularity of each item u rated:

$$MUD_u = \frac{\sum_{k-1}^{G_u} d'_k}{G_u}, u = 1, 2, 3, ..., N \tag{5.1}$$

RUD represents the difference between the maximum and minimum values in each user's popularity vector, where $d'_{\max}$ represents the maximum popularity value on the set of items rated by u and $d'_{\min}$ represents the minimum popularity value on the set of items rated by u:

$$RUD_u = d'_{\max} - d'_{\min}, u = 1, 2, 3, ..., N. \tag{5.2}$$

The last metric, QUD, is the upper quartile of the popularity vector for each user, where d'_k represents user u popularity vector from the 4th quartile onwards, sorted in ascending order:

$$QUD_u = d'_k, u = 1, 2, 3, ..., N. \tag{5.3}$$

Each of these features is calculated for each user in the dataset, and a supervised classifier is trained on top of the generated data.

PopSAD (LI *et al.*, 2016) is another supervised approach focusing on feature extraction and detection through machine learning-supervised models. Like DegreeSAD, this work also proposes exploring popularity-based features, this time using selected user patterns, in order to differentiate shilling profiles from real ones. Basically, the authors argue that attackers and real users rate items using different mechanisms, where real users rate according to their preferences while fake profiles logically use rules determined by the attack models themselves.

The authors first defined a user u popularity profile PP_u as a set of item popularity values for rated items, i.e., $PP_u = (d_{u,1}, d_{u,2}, ..., d_{u,N_u})$, where N_u is the total number of items rated by user u and $d_{u,i}$ is the popularity of the i-th rated item in user u's profile. Then, they introduced the popularity distribution of a user, which is the probability distribution of item popularity values in a user's popularity profile, i.e., each element in the distribution is a probability of items in PP_u that is equal to a value. This probability is calculated as $p_{u,i} = N_{d=i}N_u$, where N_u is the number of items in the user's popularity profile, and $N_{d=i}$ is the number of elements with the popularity of i. For a user u then, the probability distribution is defined as

$D_u = (p_{u,1}, p_{u,2}, ..., p_{u,d_{max}})$, where d_{max} is the maximum popularity value within the recommender system.

In order to define the width of the intervals to transform the probability distribution into usable features, the authors proposed using the lower boundary of the Mean Popularity of a User (MPU) to divide it into bins, which means the mean value of the popularity profile:

$$MPU_u = \frac{\sum_{u=1}^{N_u} d_{u,i}}{N_u}, \tag{5.4}$$

where N_u is the number of items rated by user u and $d_{u,i}$ is the popularity of the i-th rated item for the user u. With the interval bins defined as features, a supervised classifier is trained on top of these extracted features.

CoDetector (DOU *et al.*, 2017) is a supervised approach that focuses on exploiting interactions between users and items. The core concepts of this approach involve matrix factorization (MF) and user embedding techniques, which jointly decompose both the rating matrix and the user-user co-occurrence matrix, sharing latent factors between the two.

As already mentioned in section 2.1.1, matrix factorization is a class of collaborative filtering techniques based on the extraction of underlying interactions between users and items, through latent factor extraction, which maps users and items into a low-dimensional space (KOREN *et al.*, 2009).

Word embedding refers to models where each word in a phrase is embedded into a continuous vector space, while context is defined by the words surrounding a center word. Bridging the two concepts, SGNS (skip-gram neural embedding model) (MIKOLOV *et al.*, 2013) is a neural model trained with a negative sampling procedure, equivalent to the factorization of a word-context matrix, where the rows are the pointwise mutual information (PMI) between each word and its respective context (LEVY & GOLDBERG, 2014). The PMI of a word w and a context c can be calculated as follows, where $\#(i, j)$ is the number of times word j is in the context of the word i, and $|D|$ is the total number of word-context pairs.

$$PMI(i, j) = \log \frac{\#(i, j)\dot{|}D|}{\#(i)\dot{\#}(j)} \tag{5.5}$$

LEVY & GOLDBERG (2014) proposed SPPMI (Shifted Positive PMI), which improves the embedding using different negative sample count k as follows.

$$SPPMI(i, j) = max\{PMI(i, j) - \log k, 0\} \tag{5.6}$$

In the context of recommender systems, the context of a user u is defined as the set of other users who have rated the same items as user u. Specifically, if another

user j has rated the same items that user u has rated, then user j is considered part of user u's context. Thus, the user-user SPPMI matrix $M \in R_{m \times m}$ can be constructed by computing $\#(i, j)$, the number of items that both users i and j rated. User embedding is obtained by factorizing M.

Putting it all together, the CoDetector cost function can be defined as follows, where p_u is the shared user latent factors, m_{uj} is the shifted positive point-wise mutual information between users u and j, g_j is the context of user u and w_u and c_j are, respectively, the biases of the user and the context:

$$\begin{aligned} L = & \sum_{u,i}(y_{ui} - p_u^T q_i)^2 + \sum_{u,j}(m_{uj} - p_u^T g_j - w_u - c_j)^2 \\ & + \lambda(\sum_u ||p_u||^2 + \sum_i ||q_i||^2 + \sum_j ||g_j||^2) \end{aligned} \tag{5.7}$$

The model parameters can be updated by using the stochastic gradient descent as normal. The updated rules are denoted as follows:

$$\begin{aligned} \frac{\partial L}{\partial p_u} &= \lambda p_u - (y_{ui} - p_u^T q_i)q_i - (m_{uj} - p_u^T g_j - w_u - c_j)g_j \\ \frac{\partial L}{\partial q_i} &= \lambda q_i - (y_{ui} - p_u^T q_i)p_u \\ \frac{\partial L}{\partial g_j} &= \lambda g_j - (m_{uj} - p_u^T g_j)p_u \\ \frac{\partial L}{\partial w_u} &= m_{uj} - p_u^T g_j - w_u - c_j \\ \frac{\partial L}{\partial c_j} &= m_{uj} - p_u^T g_j - w_u - c_j \end{aligned} \tag{5.8}$$

5.1.2 Semi-Supervised

Semi-SAD (CAO *et al.*, 2013) is a semi-supervised detection technique that combines a naïve Bayes classifier with an EM-λ algorithm to improve the initial results. The proposal's first step is to define multiple metrics that will be used to construct the classification dataset from rating data to train the model and predict if a profile is an attacker or not. The authors selected multiple well-established metrics proposed in previous works, such as Entropy, DegSim, LengthVar, RDMA, and FMTD (BURKE *et al.*, 2006; CHIRITA *et al.*, 2005; WILLIAMS & MOBASHER, 2006).

Entropy is used to measure the level of dispersal or concentration of a profile (CAO *et al.*, 2013). Attack profiles' ratings are often generated using a Gaussian distribution, which means that the entropy of attackers tends to be smaller than real users. Given $X_u = n_i, i = 1, 2, ..., r_{max}$ as a statistical set for user u, where n_i is the time when the i-th rating occurred for this particular user, the entropy of u can be calculated as follows:

$$H(u) = -\sum_{i=1}^{r_{max}} \frac{n_i}{S} \log_2 \frac{n_i}{S}, \text{ where } S = \sum_{i=1}^{r_{max}} n_i \tag{5.9}$$

Degree of Similarity with Top Neighbors (DegSim) represents the average similarity of a profile's top-K neighbors. This metric can be computed as follows:

$$DegSim_u = \frac{\sum_{v \in neighbors(u)} W_{u,v}}{k} \tag{5.10}$$

Length Variance (LengthVar) was proposed to measure the difference between the length of a given profile and the average length of the profiles within a system. Larger profiles tend to be constructed by attackers, since it is unlikely that a user has entered all that rating data manually (BURKE *et al.*, 2006). This metric is computed using the following equation:

$$LengthVar_u = \frac{|n_u - \bar{n_u}|}{\sum_{u \in U}(n_u - \bar{n_u})^2} \tag{5.11}$$

Rating Deviation from Mean Agreement (RDMA) (CHIRITA *et al.*, 2005) represents the average deviation of ratings assigned by a user compared to the mean rating of the item given by other users, weighted by the total number of ratings given by that user. It can be calculated by the following equation:

$$RDMA_u = \frac{\sum_{i=0}^{N_u} \frac{|r_{i,u} - Avg_i|}{NR_i}}{N_u} \tag{5.12}$$

Filler Mean Target Difference (FMTD) was proposed to detect Bandwagon and Segment attacks, where an attacker assigns the maximum rating value to a selected group of items. This metric focuses on detecting the difference between these profiles from the average rating received by the item in question. It can be calculated as follows:

$$FMTD_u = \left| \left(\frac{\sum_{i \in P_{u,T}} r_{u,i}}{|P_{u,T}|} \right) - \left(\frac{\sum_{k \in P_{u,F}} r_{u,k}}{|P_{u,F}|} \right) \right| \tag{5.13}$$

After the metrics definition and extraction, a naïve Bayes classifier is trained using the small set of labeled data available, estimating its initial parameters, i.e., the mean and the standard deviation of the probability distribution from both attackers and normal users classes. For this learning process, the authors assumed that the i-th metric from the set of extracted metrics M_i follows a Gaussian distribution with a mean of μ_i and a standard deviation of ρ_i. This way, the probability $P(x_i|C)$ represents the probability of a user belonging to class C if its i-th metric is $M_i = x_i$.

$$P(x_i|C) = g(x_i, \mu_{C_i}, \rho_{C_i}), \text{ where } g(x, \mu, \rho) = \frac{1}{\sqrt{2\pi}\rho} e^{-\frac{(x-\mu)^2}{2\rho^2}} \tag{5.14}$$

Where μ_{C_i} is the mean and ρ_{C_i} is the standard deviation of the i-th metric in class C and $g(x_i, \mu, \rho)$ the probability density function of the gaussian distribution. Thus, the probability of a user belonging to class C may be given as follows:

$$P(u|C) = \prod_{i=1}^{n} P(x_{ui}|C) \tag{5.15}$$

The final phase consists of using EM-λ (NIGAM *et al.*, 2000) to take advantage of the unlabeled data and improve the naïve Bayes classifier parameters. EM basically re-estimates parameters using two steps, the E-step, and the M-step.

The E-step is used to calculate the probability of a user belonging to any of the classes, where probability $P(C)$ represents the probability of the sample being an attacker or not, which can be uniform or estimated based on the cardinality of the classes in the labeled set:

$$P(u_k \in C) = P(c|u_k) = \frac{P(C)P(u_k|C)}{P(u_k)} \tag{5.16}$$

The M-step calculates the estimated parameters based on the probability calculated from the previous step as follows. The mean of the i-metric of the data that belongs to the C class can be computed as:

$$\mu_{C_i} = \frac{1}{|C|} \sum_{u=1}^{|C|} \omega_u x_{ui} \tag{5.17}$$

While the standard deviation of the i-metric of the data that belongs to the C class can be computed as:

$$\rho_{C_i} = \sqrt{\frac{1}{|C|} \sum_{u=1}^{|C|} \omega_u^2 (x_{ui} - \mu_{C_i})^2} \tag{5.18}$$

In both mean and standard deviation, each class number $|C|$ is computed using weight ω_u, where $|L_c|$ is the number of labeled C users in L.

$$|C| = |L_C| + \sum_{u=1}^{|U|} \omega_u \tag{5.19}$$

The weighting factor ω_u can be computed as follows:

$$\omega_u = \Lambda(u) \frac{P(u \in C)}{\sum_j P(u \in C_j)} \tag{5.20}$$

Please note that the weighting factor Λ controls the influence of the unlabeled data in the results.

5.1.3 Unsupervised

PCAVarSelect (MEHTA & NEJDL, 2009) is an unsupervised detection technique proposed to exploit the high correlation between attack profiles to find such a group of profiles in a CF dataset. As previously discussed, attack profiles are usually correlated in order to cause the desired effect in the recommendation algorithm, so it is possible to use Principal Component Analysis (PCA) to compute principal components, i.e., orthogonal linear combinations, and, therefore, select those which shows the highest covariance with all other dimensions.

PCA is a linear dimensionality reduction technique that computes and uses the principal components to perform a change of basis in data. The process is equivalent to an eigendecomposition of the original data covariance matrix. More formally, suppose that $X^{m \times n}$ is a matrix representing the input data, where each column is an observation $x_i = \{x_{i,1}, x_{i,2}, ..., x_{i,m}\}$. Note that, for simplifying purposes, we assume that the data is zero-centered, thus the covariance matrix C is defined as:

$$C = \frac{1}{n-1} X \dot{X}_T, \tag{5.21}$$

The symmetric covariance matrix C can be decomposed using spectral decomposition theorem (JOLLIFFE, 2002), where *Lambda* is a diagonal matrix of eigenvalues of C and U are the eigenvectors:

$$C = U \lambda U_T, \tag{5.22}$$

hence, the principal components are given by the rows of the S matrix, as follows:

$$S = U_T X, \tag{5.23}$$

the principal components can be sorted in ascending order, where the ith row will represent the i principal component. To accomplish this, the rows of U are ordered according to the eigenvalues of C.

The full process is carried out according to Algorithm 5 below. First, the data is normalized using *z-scores*. Next, the resulting matrix is transposed, and the covariance matrix is calculated. Then, eigendecomposition is performed to extract the first three principal components.

Algorithm 5 Calculating distances between users using PCA MEHTA & NEJDL (2009).

Input: D – ratings matrix ($user \times item$)

Output: T – r users with smallest *distance* values

$D = \text{z-scores}(D)$

$D = D^T$

$COV = D^T D$

$U\lambda U^T = \text{Eigen-value-Decomposition}(COV)$

$PCA_1 = U(:, 1)$

$PCA_1 = U(:, 2)$

$PCA_1 = U(:, 3)$

for $user \in D$ **do**

| $Distance(user) = PCA_1(user)^2 + PCA_2(user)^2 + PCA_3(user)^2$

end

$Sort(Distance)$

Fraudulent Action Propagation (FAP) (ZHANG *et al.*, 2015) is an unsupervised detection framework designed to identify shilling attacks without relying on specific details about various attack strategies. The authors noted that many existing detection methods are typically based on identifying particular characteristics of known attack strategies. In contrast, their proposal focuses solely on the ratings that malicious profiles assign to target items.

Given a bipartite graph $G = (\mathcal{U}, \mathcal{P}, \mathcal{E})$, where $\mathcal{U}$ is the set of users, $\mathcal{P}$ is the set of items, and $\mathcal{E}$ is the set of edges $\langle u_i, p_j \rangle$, and a set $\mathcal{U}_s$ of seed attackers detected and labeled manually, the proposed framework consists on a label propagation algorithm to estimate the probability $P(u_i)$ of a user $u_i \in \mathcal{U}$ being an attacker.

In order to apply it to the rating data, the bipartite graph G needs to be constructed using a methodology to estimate the weight of each edge w_{ij}, which links a particular user u_i and a particular item p_j. Each weight is estimated considering the rating count, user rating bias, item rating bias, and global rating bias, as follows:

$$w_{ij} = 1 + \left| \frac{r_{ij} - \hat{r_i}}{\hat{r_i}} \right| + \left| \frac{r_{ij} - \hat{r_j}}{\hat{r_j}} \right| + \left| \frac{r_{ij} - \hat{r}}{\hat{r}} \right| \tag{5.24}$$

To propagate the probability of a user/item being an attacker, it is necessary to estimate the user-to-item transition probability $t_{u_i p_j}$ and the item-to-user transition probability $t_{p_j u_i}$ using the bipartite graph G. This probability is calculated by dividing a particular edge weight by the total edge weight of a user as follows:

$$t_{u_i p_j} = \frac{w_{ij}}{\sum_{i' : \langle u_i, p_{j'} \rangle \in \mathcal{E}} \mathcal{W}_{ij'}} \tag{5.25}$$

In the same way, it can be calculated for items:

$$t_{p_j u_i} = \frac{w_{ij}}{\sum_{i':\langle u_{i'}, p_j \rangle \in \mathcal{E}} \mathcal{W}_{i'j}} \tag{5.26}$$

If there are no edges connecting the user u_i and the item p_j, we have $t_{u_i p_j} = t_{p_j u_i} = 0$.

Supposing $m = \mathcal{U}$ and $n = \mathcal{P}$, the user-to-item transition matrix can now be defined as $T_{up} = (T_{u_i p_j})_{m \times n}$, while the item-to-user transition matrix is $T_{pu} = (T_{p_j u_i})_{n \times m}$.

The user probability vector P_u and the item probability vector P_p are constructed using the respective probabilities of each user and item being attackers as follows:

$$P_u = [P(u_1), P(u_2), P(u_3), ..., P(u_m)]^T \tag{5.27}$$

$$P_p = [P(p_1), P(p_2), P(p_3), ..., P(p_n)]^T \tag{5.28}$$

Finally, the probability vectors can be estimated using the transition matrices, the next equations show how the propagation is performed for the i-th iteration:

$$P_p^i = T_{pu} P_u^{i-1} \tag{5.29}$$

$$P_u^i = T_{up} P_p^i \tag{5.30}$$

One should note that labeled users should have their probabilities set to their initial values, i.e., $P(u_i) = 1$, before each iteration in order to achieve convergence.

UD-HMM (ZHANG *et al.*, 2018) is a shilling attack detection method based on the hidden Markov model (HMM) and hierarchical clustering. It was proposed to overcome the limitations of the existing techniques, which in general require prior knowledge to properly achieve competitive results.

The model consists of two different stages: the first stage uses an HMM to generate the users' preference sequence, calculating a metric called the user's matching degree, while the second stage calculates the entropy of each item, combining it with the user's matching degree to form the item's suspicious degree and, then, calculating the user's suspicious degree from the previous metrics. The last step is to apply hierarchical clustering to the user's suspicious degree and obtain the detection results.

The core idea is based on the difference in the rating patterns of real and shilling users, where the item sequence is the main trait analyzed. For instance, it's possible to define the *User Rating Item Sequence (URIS)* as the observation sequence, where $j^u_{n1,t1}$ represents the item j_{n1} rated by the user u at the time $t1$.

$$URIS_u = j^u_{n1,t1}, j^u_{n2,t2}, ..., j^u_{ns,ts} \tag{5.31}$$

Using the rating sequences, it is possible to train an HMM to generate the user's preference sequence where the items rated in the sequence are the observed variables and the hidden states are regarded as the user's preferences. The number of hidden states N is the number of different interests of users and the number of observations M or T is regarded as the number of items rated by each user.

With the model trained, a metric named *User Matching Degree (UMD)* can be derived from it. This metric aims to measure the difference in rating patterns between real and shilling profiles. For every user in the user's set U, $O_u = O_1, O_2, ..., O_T$ represents the observation sequence of a user u, and $Q_u = q_1, q_2, ..., q_T$ represents the same user hidden state sequence, it is defined as:

$$UMD_u = \sqrt[t]{\hat{\Pi}(q_1)\hat{\beta}(q_1, O_1)\prod_{i=2}^{T}\hat{\alpha}(q_{i-1}, q_i)\hat{\beta}(q_i, O_i)}, \tag{5.32}$$

where T is the number of items rated by user u, $\hat{\alpha}$ is the station transition probability matrix, $\hat{\Pi}$ is the initial state probability matrix, and $\hat{\beta}$ is the observation probability matrix.

Following this step, in order to calculate the *Item Suspicious Degree (ISD)*, it is necessary to define the *Item Entropy (IE)*. Begin H a set of possible ratings provided by users in the system, e.g., 1, 2, 3, 4, 5 for MovieLens dataset, $P_{j,d}$ the probability of all users giving d points for item j, the entropy for item j is calculated as follows

$$IE_j = -\sum_{d \in H} P_{j,d} \log_2 P_{j,d}, \tag{5.33}$$

$$P_{j,d} = \frac{\sum_{r^{t_k}_{u_k,j} \in IRS_j} \gamma(r^{t_k}_{u_k,j}, d)}{\sum_{d \in H}\sum_{r^{t_k}_{u_k,j} \in IRS_j} \gamma(r^{t_k}_{u_k,j}, d)}, \tag{5.34}$$

where IRS_j is the *Item Rating Sequence (IRS)*, i.e., $r^{t_1}_{u_1,j}, r^{t_1}_{u_1,j}, ..., r^{t_m}_{u_m,j}$ a series of item j's ratings given by users $u_1, u_2, ..., u_m$ at the time $t_1, t_2, ..., t_m$ and, $\gamma(r^{t_k}_{u_k,j}, d)$ is the discriminator function.

The *Item Suspicious Degree (ISD)* of an item j can be calculated using the normalized representation of the reciprocal of user u's matching degree Φ_u and, the normalized representation of the reciprocal of item j's matching degree ζ_j:

$$ISD_j = \frac{\sum_{u \in U_H} \Phi_u}{|U_H|} \times \zeta_j, \tag{5.35}$$

$$\Phi_u = \frac{1/UMD_u - 1/UMD_{\max}}{1/UMD_{\min} - 1/UMD_{\max}}, \tag{5.36}$$

$$\zeta_j = \frac{1/IE_j - 1/IE_{\max}}{1/IE_{\min} - 1/IE_{\max}}, \tag{5.37}$$

where U_H is the set of users who gave high ratings to the item j, e.g., greater than 3 for MovieLens dataset, UMD_u is the matching degree for the user u, $UMD_{\max}$ and $UMD_{\min}$ are, respectively, the maximum and minimum values computed for the UMD, while, in the same way, IE_j is the item entropy for item j and, $IE_{\max}$ and $IE_{\min}$ are the maximum and minimum computed values for the IE. A user is regarded as more suspicious as greater is the deviation of her preferences.

Suspicious Degree Range of Items (SDRI) of an user u is defined by subtracting the minimum $ISD^u_{\min}$ from the maximum $ISD^u_{\max}$ value within the items the user u have rated.

$$SDRI_u = ISD^u_{\max} - ISD^u_{\min} \tag{5.38}$$

Finally, the *User Suspicious Degree (USD)* for an user u is defined as the linear weighted combination of Φ_u, the normalization of reciprocal user matching degrees, and ϕ_u, the normalizing representation of the suspicious degree range of items rated by u, with α as a weight factor and, $SDRI_{\max}$ and $SDRI_{\min}$ as the maximum and minimum values of SDRI rated by these users.

$$USD_u = \alpha \times \Phi_u + (1 - \alpha) \times \phi_u \tag{5.39}$$

$$\phi_u = \frac{SDRI_u - SDRI_{\min}}{SDRI_{\max} - SDRI_{\min}} \tag{5.40}$$

After the metrics are calculated for all users, a hierarchical clustering technique can be applied. The cluster with the highest mean is chosen as the shilling attack users cluster.

5.2 Detection Models Evaluation

In this section, we detail our experiments and results comparing different state-of-the-art shilling attack detection approaches.

Our experiments are focused on asserting if the selected state-of-the-art detection techniques can successfully detect shilling attackers and prevent significant damage to the recommender system. A secondary goal is to assess the number of genuine profiles that are misclassified as attackers, as this could be problematic when applying these techniques in real-world scenarios. Additionally, we can compare real-world attacks with the shilling attacks described in the current literature and evaluate the effectiveness of the techniques. In our experiments we compare the following shilling attack detection approaches against each other, evaluating its assertiveness: CoDetector, DegreeSAD, PopSAD, FAP, PCAVarSelect, SemiSAD, and UD-HMM. In the same way, the following attack models are used to evaluate these techniques:

Variational Autoencoder, Random, Average, Bandwagon, Segment, PUA-NR, and PIA-NR.

5.2.1 Metrics

For the upcoming experiments, we use precision, recall, and F1-measure to evaluate the detection approaches reviewed in this chapter. These are classic metrics when evaluating detection algorithms, and are defined as follows:

$$Precision = \frac{tp}{tp + fp} \tag{5.41}$$

$$Recall = \frac{tp}{tp + fn} \tag{5.42}$$

$$F1 - measure = \frac{2tp}{tp + fp + tp + fn}. \tag{5.43}$$

Additionally, we use the False Alarm Rate (FAR), which is a metric being used in similar experiments in previous works CHUNG *et al.* (2013). This metric captures the real profiles falsely detected as attackers:

$$FAR = \frac{fp}{fp + tn}, \tag{5.44}$$

where tp is the number of true positives, which, in a shilling attack context means the total attack profiles correctly detected, fp is the number of false positives, i.e., real profiles misclassified as attackers, fn are false negatives, the shilling attackers misclassified as real profiles, and tn is the number of true negatives, which means the real profiles correctly classified.

5.2.2 Setup

We will carry out an experiment to verify the effectiveness of the shilling attack detection approaches. This experiment will compare several state-of-the-art detection algorithms using the previously presented datasets, including Amazon review, a dataset containing real-world spammer users.

Due to the different nature of the tested models, experiments for supervised and semi-supervised detection techniques require a different setup compared to experiments for unsupervised detection techniques. For the former, the experiment uses a hold-out strategy, partitioning the users into two sets: 50% for training and 50% for testing. Both partitions are used to create malicious profiles, which are then injected into their respective partitions. For the latter, the entire dataset is used to create attack profiles, which are injected into the full dataset and subsequently

labeled by the unsupervised technique. Please note that regardless of the technique type, attackers are generated for MovieLens 100k and Yahoo! Music datasets, while for the Amazon dataset, there is no need to generate attackers since this dataset was already attacked previously, and the users are labeled.

Different from the previous chapters' experiments, for MovieLens 100k and Yahoo! Music we selected ten randomly chosen items among all items for these experiments. Please note that the items chosen here are the randomly chosen in the previous experiments. Since our attack intent is to push, the target item rating was set to maximum value (which is 5 in the MovieLens 100k and Yahoo! Music datasets). We averaged the results for each chosen metric and reported the results. The attack model parameters remain the same as reported previously in subsection 4.3.2.

The parameters for each shilling attack detection technique are chosen empirically, using a random partition of the data and a random target item in MovieLens 100k and Yahoo! Music cases. The parameters of the attack models are the same as reported in the previous experiment in subsection 4.3.2.

In resume, two experiments are carried out to evaluate the detection approaches and the attack models:

Experiment I

Comparison with well-known detection approaches: DegreeSAD, PopSAD, CoDetector, SemiSAD, PCAVarSelect, FAP, and UD-HMM (only for MovieLens 100k) running on seven shilling attack models: Variational Autoencoder, Random, Average, Segment, Bandwagon, PUA-NR and PIA-NR using MovieLens 100k and Yahoo! Music datasets.

Experiment II

Comparison with well-known detection approaches: DegreeSAD, PopSAD, CoDetector, SemiSAD, PCAVarSelect, FAP, and UD-HMM (only for MovieLens 100k) running on the Amazon Review dataset, a database containing labeled real-world attackers.

5.2.3 Results

In this subsection, we present and discuss the obtained results from the evaluated experiments. In addition, we compare state-of-the-art shilling attack detection approaches against several well-known shilling attack models.

Experiment I

In *Experiment I*, we analyze how detection approaches perform when faced with well-known shilling attack models. Note that approaches from different categories are not directly comparable.

In Figure 5.1, we show the results for the supervised detection techniques: (a) DegreeSAD, (b) PopSAD, and (c) CoDetector using MovieLens 100k dataset. Among these techniques, DegreeSAD (Figure 5.1a) shows competitive results across almost all shilling attack approaches, demonstrating high F1 scores and low False Alarm Rates for every attack size, except for Variational Autoencoder and PUA-NR attacks. These results suggest that both of these approaches are particularly effective at creating profiles that are closer to real ones. The reason for the PUA-NR case is more apparent, as it directly copies real users from the database. However, Variational Autoencoder aims to generate ratings similar to those from real users without directly copying profiles, and the results here support this hypothesis.

The second detection approach evaluated, PopSAD (Figure 5.1b), shows some difficulties with smaller attack sizes across nearly all approaches using the MovieLens 100k dataset, with the exception of PIA-NR. However, it manages to improve results as the number of injected attack profiles increases. From an attack size of 5% of the total number of profiles, i.e., 47 profiles, the results become comparable to those reported by DegreeSAD, with Variational Autoencoder and PUA-NR managing to evade detection of several profiles. PIA-NR, however, avoids detection almost entirely. Since PopSAD focuses on constructing features based on the popularity of items in the database, this attack constructs shilling profiles using the most popular items in the system, which allows it to bypass detection by real users without issues.

The last supervised technique evaluated is the CoDetector (Figure 5.1c), which shows perfect results on all metrics against Segment attacks and good precision levels defending the system from Variational Autoencoder and PIA-NR attacks. On the other hand, recall values are not competitive at all, indicating that some attackers might pass as genuine profiles when using this detection technique. It is also important to note that this technique also fails to detect most of the other approaches, such as Average and Bandwagon attacks. In any case, except for Segment attacks, this approach performs poorly with smaller attacks, i.e., lower than 5% of the total profiles, indicating limitations in the area where Variational Autoencoder typically excel in latent factor-based systems.

These initial reports indicate that our model is able to evade detection by most of the supervised detection approaches on the MovieLens 100k dataset. Additionally, there are strong indications supporting the hypothesis that replicating real users' rating patterns is sufficient to create an attack model capable of avoiding detection.

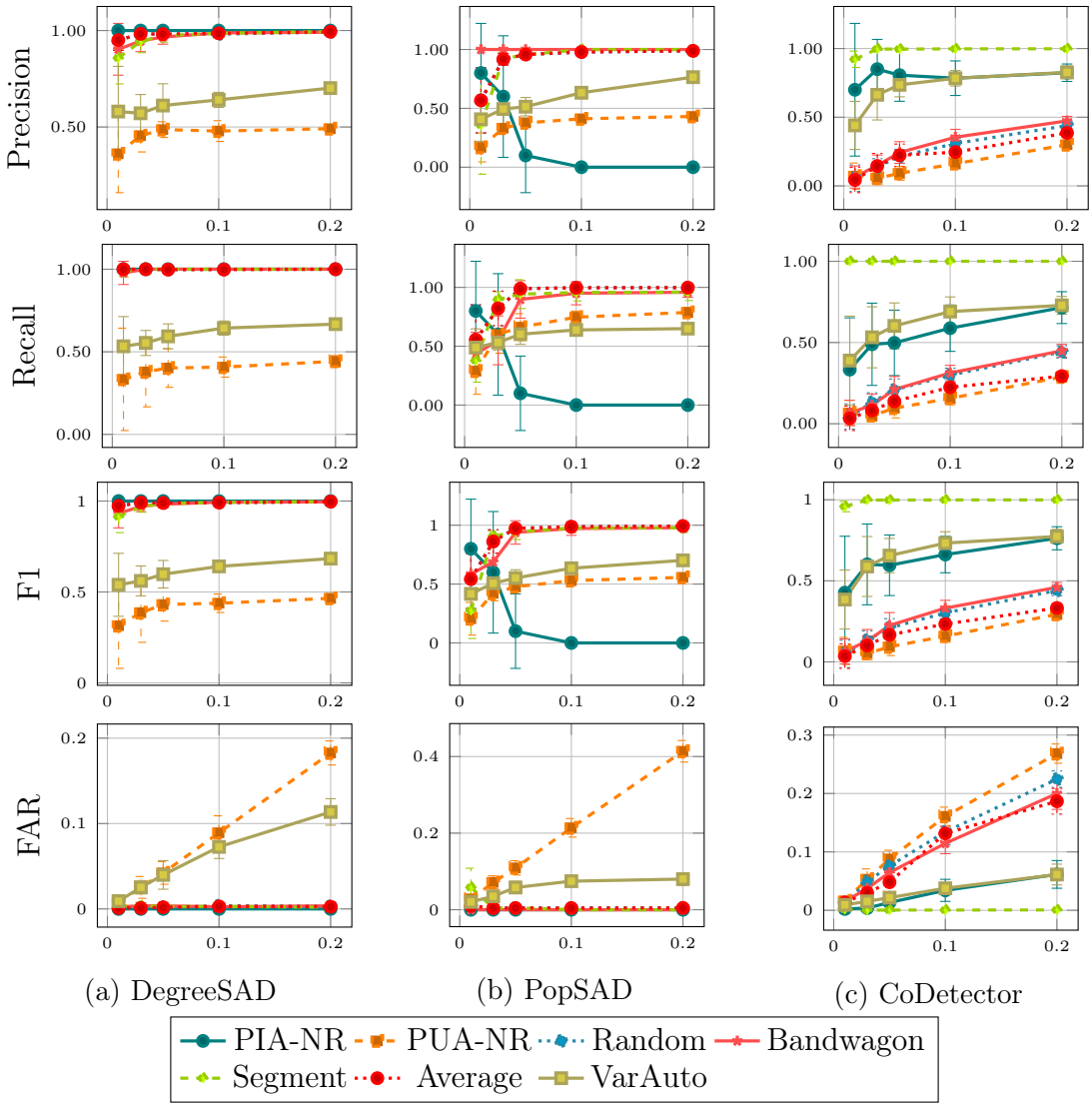


Figure 5.1: Precision, recall, F1, and False Alarm Rate for supervised detection methods using MovieLens 100k data set. Note that it only shows the best result of filler size among the attack models that require it.

In Figure 5.2, we present the results for the unsupervised detection techniques: (a) PCAVarSelect, (b) FAP, and (c) UD-HMM, using the MovieLens 100k dataset. The first technique discussed is PCAVarSelect (Figure 5.2a), which generally struggles to detect all approaches across various attack sizes, exhibiting the poorest performance compared to other shilling attack detection techniques. Notably, the number of false alarms is at most 20%, which, while smaller than that of the supervised techniques, is still relatively high. This indicates that PCAVarSelect is not competitive even against the most basic shilling attacks.

The second approach evaluated is FAP (Figure 5.2b), which achieves good recall results for smaller attack sizes tested, e.g., 1% of the total users in the dataset. However, as the size of the attack set grows, FAP does not improve on these results

and misclassifies several genuine users as attackers, as indicated by the False Alarm Rate. This technique also struggles with PUA-NR and Variational Autoencoder attacks, failing to distinguish between shilling profiles and genuine ones. This is another important result in the comparison between our approach and PUA-NR, indicating that the malicious profiles generated from samples are sufficiently similar to the original data.

Finally, the results for the UD-HMM approach (Figure 5.2c) demonstrate the most competitive performance among the unsupervised techniques. It effectively detects almost all attack sizes from Random and Average approaches without issues and shows competitive, though less effective, results against Segment and Bandwagon attacks. However, when it comes to detecting more realistic profiles, the approach struggles with PUA-NR and Variational Autoencoder, exhibiting high False Alarm Rates and lower precision, recall, and F-measure. A special case occurs with the PIA-NR attack, where UD-HMM reports high recall but low precision, indicating that while attackers are not misclassified as real profiles, a significant number of genuine profiles are incorrectly identified as attackers, which represents a meaningful drawback for this technique.

The reported results for the unsupervised approaches strongly suggest that the hypothesis is valid for the MovieLens 100k dataset, corroborating our previous assumptions. So far, the Variational Autoencoder attack model presents a greater threat than most other attack models due to its ability to remain less detectable and its tendency to provoke a higher rate of false alarms.

The last technique evaluated is the only semi-supervised technique, SemiSAD. Figure 5.3 shows the results reported for this detection approach using MovieLens 100k dataset. Apart from the attack mounted using only 9 profiles (1% of the total users in the dataset), the results indicate that this approach seems to detect effectively most of the attack models, with the exception of, once more, PUA-NR and Variational Autoencoder. Despite good precision results for the Variational Autoencoder, average recall values indicate that several profiles are able to pass undetected to this technique, dragging the F1 metric value down and, thus, compromising using this detection approach against this attack model. It is also possible to highlight the high False Alarm Rate when evaluating the detection technique against the PUA-NR, which is understandable since PUA-NR copies real profiles as previously stated.

Based on the experiments reported above, the hypothesis can be accepted as valid under the conditions presented in this section. The Variational Autoencoder model successfully avoids detection in most scenarios, except when using the SemiSAD approach. However, it is important to note that in this case, the reported recall values are low, indicating that several attack profiles are misclassified as genuine ones.

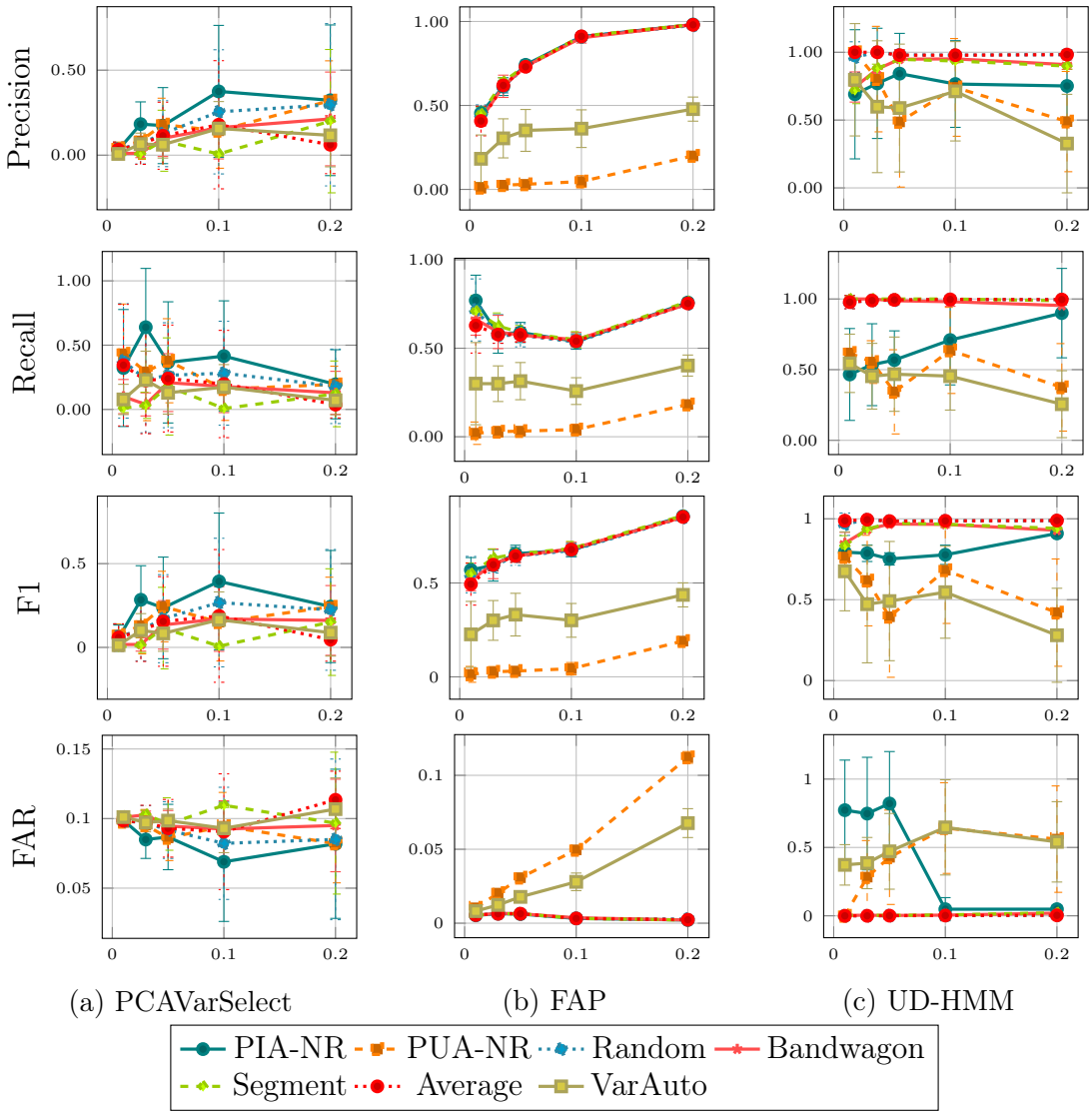
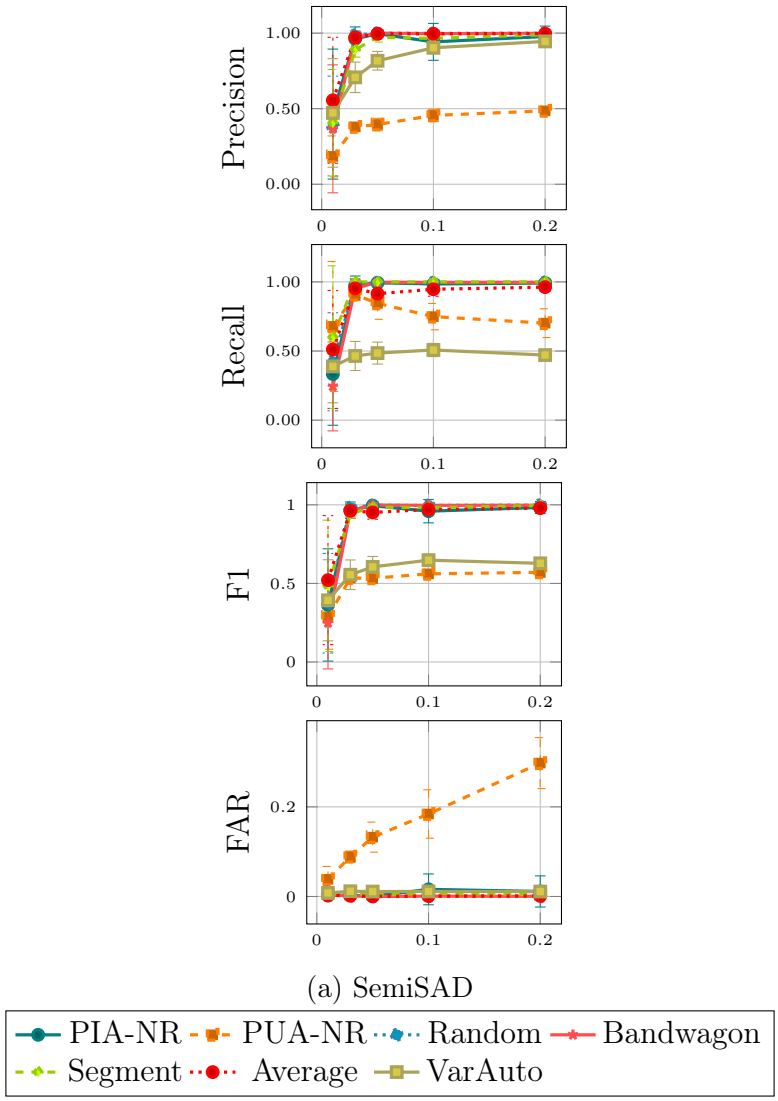


Figure 5.2: Precision, recall, F1, and False Alarm Rate for unsupervised detection methods using MovieLens 100k data set. Note that it only shows the best result of filler size among the attack models that require it.

This suggests that the rating patterns replicated by the Variational Autoencoder are indeed an effective way to avoid detection.

After the evaluation of detection techniques using the Movielens dataset, we will present results for the R3 Yahoo! Music dataset. Thus, in Figure 5.4, we run the comparative results between the supervised techniques (a) DegreeSAD, (b) PopSAD, and (c) CoDetector using this dataset. The first technique evaluated is the DegreeSAD (Figure 5.4a) which shows near-perfect results for the PIA-NR and Bandwagon attack models and competitive results for most other attack models. When dealing with Variational Autoencoder-crafted profiles, DegreeSAD's precision, recall, and F1 scores are slightly less competitive compared to the other attack models, and it also misclassifies a small number of genuine profiles as attackers.



(a) SemiSAD

Figure 5.3: Precision, recall, F1, and False Alarm Rate for SemiSAD using MovieLens 100k data set. Note that it only shows the best result of filler size among the attack models that require it.

Despite this, PUA-NR is the only attack model that manages to evade detection when confronting this approach.

We also conducted experiments using PopSAD for this dataset. The Figure 5.1b shows that the Bandwagon attack is easily detected by this detection method. However, for the Average and Variational Autoencoder attacks, the method requires a large number of injected profiles to avoid incorrectly classifying genuine profiles. The PIA-NR attack remained undetected in our experiments, while the PUA-NR attack resulted in a good number of fake profiles being detected but also produced many false positives, causing confusion. Overall, the approach does not consistently achieve competitive results.

The last supervised technique evaluated is the CoDetector (Figure 5.1c), which

also requires a large number of injected profiles to accurately detect attack profiles and avoid misclassifying genuine profiles. However, for the Variational Autoencoder, the results remain consistent even with a low number of profiles, suggesting that with the right formulation, matrix factorization methods can effectively detect this attack model in music domain datasets. It is important to note that the Variational Autoencoder parameters are not fully optimized for this dataset, which may be one reason why this attack was detected so easily in this case. PUA-NR, once again, manages to bypass this detection approach, which is expected given that this attack model is based on genuine profiles.

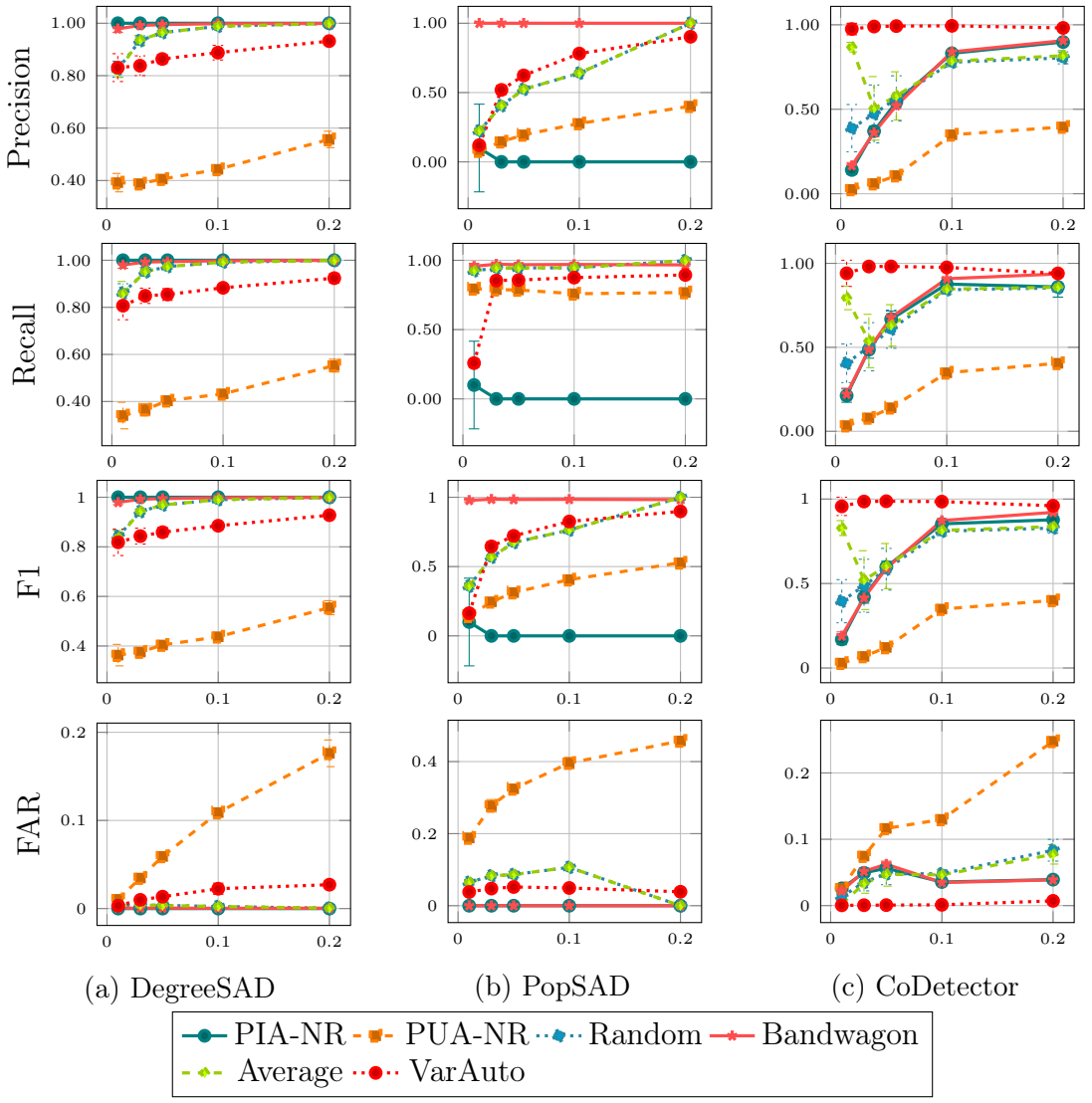


Figure 5.4: Precision, recall, F1, and False Alarm Rate for supervised detection methods using Yahoo! Music data set. Note that it only shows the best result of filler size among the attack models that require it.

In Figure 5.5, we show the results for the unsupervised detection techniques: (a) PCAVarSelect, and (b) FAP using R3 Yahoo! Music dataset. Note that due to the

lack of timestamps on this dataset, we could not run and obtain UD-HMM results to include in the comparison. The first unsupervised detection technique presented is the PCAVarSelect (Figure 5.5a), which does not achieve competitive results at all for the music domain. The best performance is observed for the Bandwagon attack model with 20% attack size, where the technique achieves precision, recall, and F1 scores above 0.5. However, these results are still not competitive, particularly due to the high number of false alarms detected.

This time, the second and last technique evaluated is the FAP (Figure 5.5b), where it is possible to see competitive results for most of the attack models mounting attacks with 7% attack size or above. In our experiments, as the attack size increases, the detection results improve across all reported metrics, indicating that more examples are needed to effectively propagate the correct labels for this technique. The Variational Autoencoder produces results similar to most other attack models. Once more, the only attack model in which this technique is not able to be competitive enough is the PUA-NR, which is understandable given the circumstances where this attack is mounted.

Following the evaluation of supervised and unsupervised techniques, we will now present the only semi-supervised technique, SemiSAD. Figure 5.6 shows the results for this detection approach using the R3 Yahoo! Music dataset. The results indicate that SemiSAD can detect most attack models effectively without major issues, except for PUA-NR and the attack model based on Variational Autoencoders. It is evident that SemiSAD encounters difficulties, resulting in a higher number of false negatives than expected, as reflected in the recall values for these attacks. For PUA-NR, the attack model easily evades detection, which is anticipated due to its use of real user profiles as a basis. The Variational Autoencoder model also poses a challenge for this technique, though its effect is much less pronounced.

While the Variational Autoencoder can confuse some detection techniques, the reported results do not favor it at all. CoDetector and, given enough samples, FAP manages to rather easily detect the profiles crafted using Yahoo! Music dataset. With these results, it is not possible to entirely accept the hypothesis for this second experiment. It is important to note that, the Yahoo! Music dataset has 2.4% sparsity, thus, sparser than MovieLens 100k (6.3%), which in turn makes the profile have fewer ratings available to learn than the movie dataset. This may be a possible cause for the lack of good results using convolutional architecture in this case. Another relevant explanation for this behavor is the lack of proper tuning of the Variational Autoencoder model for the Yahoo Music! dataset.

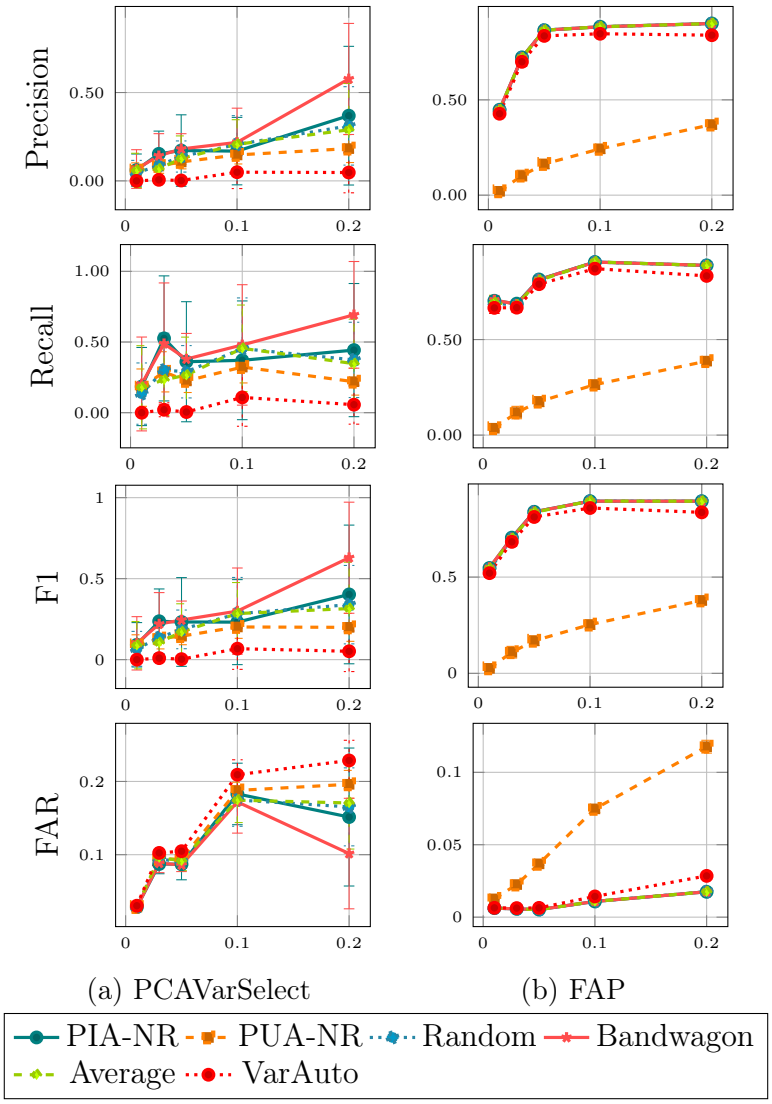
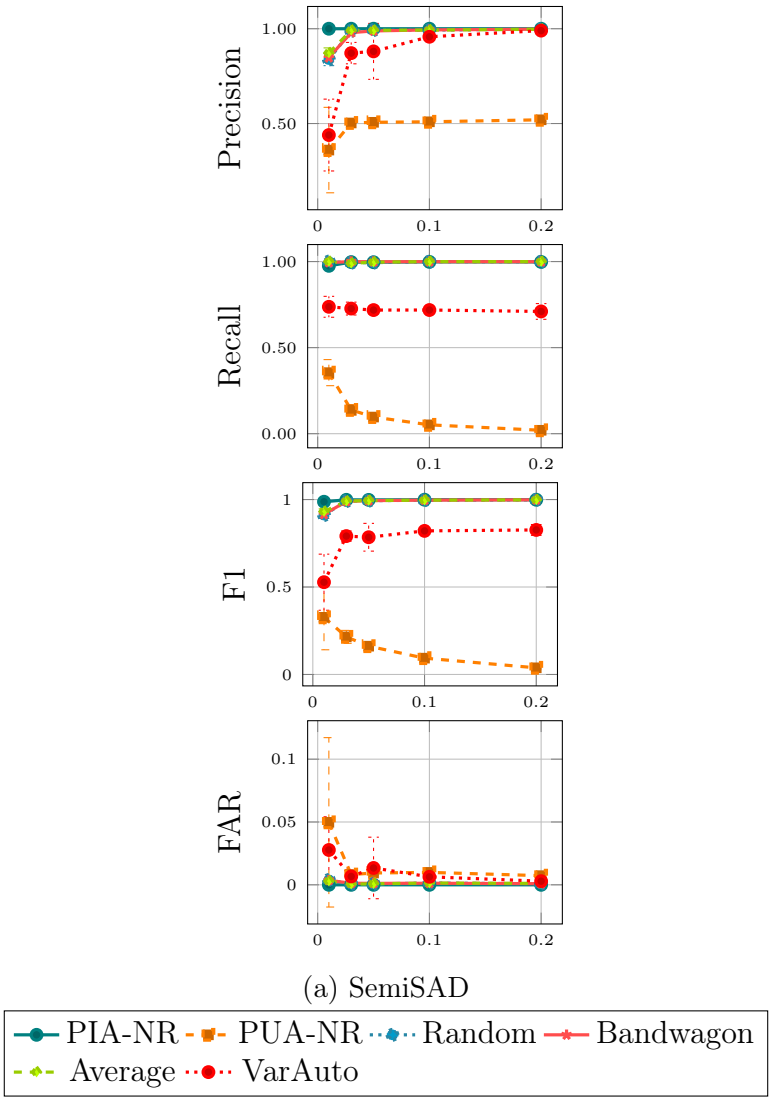


Figure 5.5: Precision, recall, F1, and False Alarm Rate for unsupervised detection methods using Yahoo! Music data set. Note that it only shows the best result of filler size among the attack models that require it.

Experiment II

In *Experiment II* we analyze how detection approaches perform confronted against real-world attackers using Amazon review dataset. Please note that approaches from different categories are not directly comparable.

In the Table 5.1, we present the results for each shilling attack detection technique on the Amazon review dataset. Among the supervised techniques, CoDetector demonstrates the best performance, successfully detecting most of the shilling profiles while misclassifying only 11% of genuine profiles as attackers. This is followed by DegreeSAD, which misclassifies double that amount (22%). The semi-supervised technique, SemiSAD, shows good recall but poor precision, resulting in 37% of profiles being falsely detected. Finally, among the unsupervised ones, which are applied



(a) SemiSAD

Figure 5.6: Precision, recall, F1, and False Alarm Rate for SemiSAD using Yahoo! Music data set. Note that it only shows the best result of filler size among the attack models that require it.

to the full dataset, FAP outperforms PCAVarSelect, showing average performance despite a relatively high number of false alarms. Please note that UD-HMM could not be tested due to the lack of timestamp information available for ratings in this dataset.

The results for the real-world attackers highlight the ongoing challenges that many techniques face in detecting shilling attacks in commercial recommender systems. These findings suggest that further research is needed to develop more effective methods for addressing realistic shilling attacks. Additionally, it is noteworthy that some attack models, such as PUA-NR and our proposal, yield results comparable to real-world attacks. This supports the objective of our work to propose an attack capable of simulating real-world scenarios without explicitly copying real profiles.

Table 5.1: Top results of different shilling attack detection techniques in Amazon review dataset.

Technique	Type	Precision	Recall	F1	False Alarm Rate
CoDetector	Supervised	0.83	0.85	0.84	0.11
DeegreeSAD	Supervised	0.64	0.61	0.62	0.22
PopSAD	Supervised	0.33	0.57	0.42	0.73
SemiSAD	Semi Supervised	0.58	0.8	0.67	0.37
PCAVarSelect	Unsupervised	0.13	0.17	0.15	0.71
FAP	Unsupervised	0.57	0.67	0.62	0.3

In summary, the results of the experimental evaluation in this chapter highlight how our attack model represents an advancement over current models in the literature. Our proposal proves to be a valuable tool for assessing robust models and detection strategies, as it exhibits superior performance in evading detection compared to baseline attack models, akin to attacks encountered in real-world recommender systems.

Chapter 6

Conclusion

This chapter presents the summary of the proposal and results, the contributions, and future works envisioned during the conception of this work.

6.1 Proposal Summary

Shilling attacks are attacks performed by malicious individuals or companies that wants to promote their products or demote rival products within a recommender system. One problem is that most of the attack models currently proposed are based on straightforward statistical templates, which may allow an attack to be easily detected and may take a lot of malicious profiles to achieve its goals. In addition, these models are tailored toward memory-based systems and remain untested in model-based ones. Generative models began to be applied as powerful tools for learning probability distributions and generating new data from them, replicating data with a high level of complexity.

Another interesting aspect of the shilling attack area is the protection of the collaborative filtering systems themselves, either by the robustness of the collaborative filtering system, i.e., the predictions remain stable no matter how noisy the instances in the database, or detection approaches, where statistical, machine learning, etc., are applied to detect attackers among the set of users of the collaborative filtering system.

In this work, we approach the general state of the shilling attack research field, focusing at first on the current attack models issue and evaluating ways to mitigate the models. We propose using a variational autoencoder, a generative model, to learn the rating patterns of real data in order to craft more realistic malicious profiles. In addition, we test our approach against several well-known detection models to check how well it can remain undetected as well as perform experiments with data cleansing and label noise robust algorithms to evaluate if they can be applied to shilling attack.

6.2 Results Summary

In order to evaluate our proposal, we performed experiments with a few well-established data sets: MovieLens 100k, R3 Yahoo! Music and Amazon review. The experiments were divided into several different phases. In the first phase, we conducted experiments using cosine association analysis to compare how realistic malicious profiles are between current attack models and our proposal. In the second one, we compared our approach against other relevant methods using model-based and memory-based collaborative filtering systems. In the third one, we tested the previous tested attack models against data cleansing approaches to clean noisy data before using an artificial neural network-based collaborative filtering system, to see how effective these algorithms are in preventing malicious noise. In the fourth, we implemented four label noise-robust proposals using the same artificial neural network-based collaborative filtering system to evaluate if these methods are effective against malicious noise. The fifth and final experiment compares how less detectable is our proposed attack model using well-known shilling attack detection approaches.

Our results indicate that our approach outperforms current attack models in almost all scenarios for Regularized SVD, a model-based algorithm, for the MovieLens 100k data set. We found that our results are strong and consistent using fewer malicious profiles in comparison with current attack models as intended. Results for Yahoo! Music are not so competitive, which seems to be related to the high sparsity of this data set compared to MovieLens 100k. There are fewer ratings per profile to learn, making the task more challenging than what was expected. Another interesting finding is that, through rated item correlation analysis, our results indicate that variational autoencoder may produce profiles realistic enough and close to the original ones' distribution, hinting that our attack would be more difficult to spot than current approaches. On user-based collaborative filtering, the results were less competitive, however, our model can still perform close to most of the other attack models, even though it is not designed to work against these kinds of systems. An interesting finding regarding memory-based techniques is that our attack model performance is paired with approaches such as PUA-NR, which is designed to use real users as attackers.

Regarding label noise experiments, we found that data cleansing works well when the system has a set of trusted users. Although it is not always a possible scenario, this approach works well to prevent shilling attacks under these conditions. On the other hand, label noise-robust techniques do not achieve the same effect, even reinforcing the effects of the attack in some cases.

The subsequent detection experiments corroborate the vision that our approach is less likely to be detected, indicating that our method can be a good tool to evaluate

current and novel shilling attack detection techniques, at least for the MovieLens 100k data set. For Yahoo! Music, however, our approach reports results closer to the other attack models. Finally, we ran a detection experiment on the Amazon review data set, which focused on evaluating how effective detection approaches are when facing real-world attack profiles and compare the results with the literature models. Only CoDetector showed promising results, with the rest of the approaches falling short and showing results similar to those reported when applied to detect our approach.

In short, the reported results show enough evidence to support the thesis' hypothesis for the MovieLens 100k data set. For the Yahoo! Music dataset, the higher sparsity of this data set is deemed a possible cause; further work needs to be done to properly evaluate these properties.

6.3 Future Works

Despite the advantages of our method, we have some issues that still need to be worked on. Firstly, we can appoint the performance against memory-based collaborative filtering techniques compared with basic approaches such as average and random attacks. Even though our attack model is not designed to attack memory-based collaborative filtering systems, rebuilding the model to also overcome basic attacks may be promising. Another issue is the high dependence on real data for learning our model. It may be feasible to use our approach to mount attacks to evaluate shilling attack detection techniques. However, in other scenarios, it would be necessary to reduce this real data dependency to be able to develop attacks in more real-world-like scenarios. It is also important to note that this work evaluated many of the major attack models up to 2021 across a range of scenarios. Future research should include novel attack models introduced after 2021 to provide a more comprehensive comparison.

There are also interesting contributions to be made regarding the profiles used to train the variational autoencoder. This work utilized all profiles available in the training set split of the dataset; however, employing a different methodology to select profiles that are more likely to affect the system could enhance the results obtained.

Another future work is to model a variational autoencoder architecture based on graphs in order to generate even more realistic profiles. As mentioned before, GCN has been successfully explored in recommender systems recently (ZHANG *et al.*, 2020; FAN *et al.*, 2019) and proved to be well suited to model collaborative filtering schemes. Besides that, it is important to take this opportunity to run a proper hyperparameter tuning comparing the GCN-based variational autoencoder with the

convolutional one, since this work did not invest enough time to study the effects of the model's parameters.

Data cleansing results are promising in this work; however, a future direction is to apply it to more attack models from the literature, e.g., previously mentioned Adversarial Attacks, to see how well the approach can handle a different methodology of attack construction.

A final research direction is to use our attack model to evaluate new approaches for mitigating shilling attacks. One approach is to change the formulation of the noise transition matrix, presented by CARMO (2018), to alleviate the effects of the shilling attacks. The possible drawback is the need for a method to detect malicious profiles beforehand to alleviate the attack effects during the training of the model. However, given the results of the detection experiments, this approach could still yield competitive results.

References

ZHOU, R., KHEMMARAT, S., GAO, L. "The Impact of YouTube Recommendation System on Video Views". In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pp. 404–410, New York, NY, USA, 2010. ACM. ISBN: 978-1-4503-0483-2. doi: 10.1145/1879141.1879193. Disponível em: <http://doi.acm.org/10.1145/1879141.1879193>.

SCHAFER, J., KONSTAN, J., RIEDI, J. "Recommender systems in e-commerce". In: *Proceedings of the 1st ACM conference on Electronic commerce*, pp. 158–166. ACM, 1999. ISBN: 1581131763. Disponível em: <http://dl.acm.org/citation.cfm?id=337035>.

HUANG, C., GONG, S. "Employing rough set theory to alleviate the sparsity issue in recommender system". In: *Machine Learning and Cybernetics, 2008 International Conference on*, v. 3, pp. 1610–1614. IEEE, 2008. ISBN: 9781424420964. Disponível em: <http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=4620663>.

KOREN, Y., BELL, R., VOLINSKY, C. "Matrix Factorization Techniques for Recommender Systems", *Computer*, v. 42, n. 8, pp. 30–37, ago. 2009. ISSN: 0018-9162. doi: 10.1109/MC.2009.263. Disponível em: <http://dx.doi.org/10.1109/MC.2009.263>.

BURKE, R., MOBASHER, B., WILLIAMS, C., et al. "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*, p. 542, 2006. ISBN: 1595933395. doi: 10.1145/1150402.1150465. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.119.3252{&}rep=rep1{&}type=pdfhttp://portal.acm.org/citation.cfm?doid=1150402.1150465>.

ZHANG, F., ZHANG, Z., ZHANG, P., et al. "UD-HMM: An unsupervised method for shilling attack detection based on hidden Markov model and hierar-

chical clustering", *Knowledge-Based Systems*, v. 148, pp. 146–166, may 2018. ISSN: 0950-7051. doi: 10.1016/J.KNOSYS.2018.02.032. Disponível em: <https://www-sciencedirect-com.ez29.capes.proxy.ufrj.br/science/article/pii/S0950705118300959>.

WILLIAMS, C. A., MOBASHER, B., BURKE, R., et al. "Defending recommender systems: detection of profile injection attacks", v. 1, pp. 157–170, 2007. doi: 10.1007/s11761-007-0013-0. Disponível em: <https://link.springer.com/content/pdf/10.1007{%}2Fs11761-007-0013-0.pdf>.

KINGMA, D. P., WELLING, M. "Auto-Encoding Variational Bayes". 2013. Disponível em: <http://arxiv.org/abs/1312.6114>. cite arxiv:1312.6114.

REZENDE, D. J., MOHAMED, S., WIERSTRA, D. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: Xing, E. P., Jebara, T. (Eds.), *Proceedings of the 31st International Conference on Machine Learning*, v. 32, *Proceedings of Machine Learning Research*, pp. 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR. Disponível em: <http://proceedings.mlr.press/v32/rezende14.html>.

GREGOR, K., DANIHELKA, I., GRAVES, A., et al. "DRAW: A Recurrent Neural Network For Image Generation". 2015. Disponível em: <http://arxiv.org/abs/1502.04623>. cite arxiv:1502.04623.

SOHN, K., YAN, X., LEE, H. "Learning Structured Output Representation Using Deep Conditional Generative Models". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, p. 3483–3491, Cambridge, MA, USA, 2015. MIT Press.

WALKER, J., DOERSCH, C., GUPTA, A., et al. "An Uncertain Future: Forecasting from Static Images Using Variational Autoencoders". In: Leibe, B., Matas, J., Sebe, N., et al. (Eds.), *Computer Vision – ECCV 2016*, pp. 835–851, Cham, 2016. Springer International Publishing. ISBN: 978-3-319-46478-7.

ZHU, X., WU, X. "Class Noise vs. Attribute Noise: A Quantitative Study", *Artificial Intelligence Review*, v. 22, n. 3, pp. 177–210, 2004. ISSN: 0269-2821. doi: 10.1007/s10462-004-0751-8.

HUBER, P. *Robust Statistics*. Wiley Series in Probability and Statistics - Applied Probability and Statistics Section Series. Wiley, 2004. ISBN: 9780471650720.

BARBIERI, J., ALVIM, L. G., BRAIDA, F., et al. "Simulating real profiles for shilling attacks: A generative approach", *Know.-Based Syst.*, v. 230, n. C, oct 2021. ISSN: 0950-7051. doi: 10.1016/j.knosys.2021.107390. Disponível em: <https://doi.org/10.1016/j.knosys.2021.107390>.

EKSTRAND, M. D., RIEDL, J. T., KONSTAN, J. A. "Collaborative Filtering Recommender Systems", *Found. Trends Hum.-Comput. Interact.*, v. 4, n. 2, pp. 81–173, fev. 2011. ISSN: 1551-3955. doi: 10.1561/1100000009. Disponível em: <http://dx.doi.org/10.1561/1100000009>.

RICH, E. "Readings in Intelligent User Interfaces". Morgan Kaufmann Publishers Inc., cap. User Modeling via Stereotypes, pp. 329–342, San Francisco, CA, USA, 1998. ISBN: 1-55860-444-8. Disponível em: <http://dl.acm.org/citation.cfm?id=286013.286035>.

GOLDBERG, D., NICHOLS, D., OKI, B. M., et al. "Using Collaborative Filtering to Weave an Information Tapestry", *Commun. ACM*, v. 35, n. 12, pp. 61–70, dez. 1992. ISSN: 0001-0782. doi: 10.1145/138859.138867. Disponível em: <http://doi.acm.org/10.1145/138859.138867>.

CHEN, P.-Y., WU, S.-Y., YOON, J. "The impact of online recommendations and consumer feedback on sales", *ICIS 2004 Proceedings*, p. 58, 2004.

ADOMAVICIUS, G., TUZHILIN, A. "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions", *IEEE Trans. on Knowl. and Data Eng.*, v. 17, n. 6, pp. 734–749, jun. 2005. ISSN: 1041-4347. doi: 10.1109/TKDE.2005.99. Disponível em: <http://dx.doi.org/10.1109/TKDE.2005.99>.

HERLOCKER, J. L., KONSTAN, J. A., TERVEEN, L. G., et al. "Evaluating Collaborative Filtering Recommender Systems", *ACM Trans. Inf. Syst.*, v. 22, n. 1, pp. 5–53, jan. 2004. ISSN: 1046-8188. doi: 10.1145/963770.963772. Disponível em: <https://doi.org/10.1145/963770.963772>.

BURKE, R. "The Adaptive Web". Springer-Verlag, cap. Hybrid Web Recommender Systems, pp. 377–408, Berlin, Heidelberg, 2007. ISBN: 978-3-540-72078-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1768197.1768211>.

RICCI, F., ROKACH, L., SHAPIRA, B., et al. "Recommender Systems Handbook", *Media*, 2011. doi: 10.1007/978-0-387-85820-3. Disponível em: <http://www.springerlink.com/index/10.1007/978-0-387-85820-3>.

SCHAFER, J. B., FRANKOWSKI, D., HERLOCKER, J., et al. "The Adaptive Web". Springer-Verlag, cap. Collaborative Filtering Recommender Systems, pp. 291–324, Berlin, Heidelberg, 2007. ISBN: 978-3-540-72078-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1768197.1768208>.

GE, M., DELGADO-BATTENFELD, C., JANNACH, D. "Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity". In: *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pp. 257–260, New York, NY, USA, 2010. ACM. ISBN: 978-1-60558-906-0. doi: 10.1145/1864708.1864761. Disponível em: <http://doi.acm.org/10.1145/1864708.1864761>.

SARWAR, B., KARYPIS, G., KONSTAN, J., et al. "Item-based Collaborative Filtering Recommendation Algorithms". In: *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pp. 285–295, New York, NY, USA, 2001. ACM. ISBN: 1-58113-348-0. doi: 10.1145/371920.372071. Disponível em: <http://doi.acm.org/10.1145/371920.372071>.

SCHEIN, A. I., POPESCUL, A., UNGAR, L. H., et al. "Methods and Metrics for Cold-start Recommendations". In: *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, pp. 253–260, New York, NY, USA, 2002. ACM. ISBN: 1-58113-561-0. doi: 10.1145/564376.564421. Disponível em: <http://doi.acm.org/10.1145/564376.564421>.

GUNES, I., KALELI, C., BILGE, A., et al. "Shilling attacks against recommender systems: a comprehensive survey", *Artificial Intelligence Review*, v. 42, n. 4, pp. 767–799, 2012. ISSN: 15737462. doi: 10.1007/s10462-012-9364-9. Disponível em: <https://link.springer.com/content/pdf/10.1007{%}2Fs10462-012-9364-9.pdf>.

FUNK, S. "Netflix update: Try this at home, 2006", *URL http://sifter. org/˜ simon/journal/20061211. html*, 2011.

PATEREK, A. "Improving regularized singular value decomposition for collaborative filtering". In: *Proc. KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pp. 39–42, 2007. Disponível em: <http://serv1.ist.psu.edu:8080/viewdoc/summary;jsessionid=CBC0A80E61E800DE518520F9469B2FD1?doi=10.1.1.96.7652>.

TAKÁCS, G., PILÁSZY, I., NÉMETH, B., et al. "Matrix Factorization and Neighbor Based Algorithms for the Netflix Prize Problem". In: *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pp. 267–274, New York, NY, USA, 2008. ACM. ISBN: 978-1-60558-093-7. doi: 10.1145/1454008.1454049. Disponível em: <http://doi.acm.org/10.1145/1454008.1454049>.

FUNK, S. "Netflix Update: Try This at Home". December 2006. Disponível em: <http://http://sifter.org/~simon/journal/20061211.html>. Acesso em: 2014-06-30.

HE, X., LIAO, L., ZHANG, H., et al. "Neural Collaborative Filtering". 2017.

HARPER, F. M., KONSTAN, J. A. "The MovieLens Datasets: History and Context", *ACM Trans. Interact. Intell. Syst.*, v. 5, n. 4, pp. 19:1–19:19, dez. 2015. ISSN: 2160-6455. doi: 10.1145/2827872. Disponível em: <http://doi.acm.org/10.1145/2827872>.

XU, C., ZHANG, J., CHANG, K., et al. "Uncovering Collusive Spammers in Chinese Review Websites". In: *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, CIKM '13, p. 979–988, New York, NY, USA, 2013. Association for Computing Machinery. ISBN: 9781450322638. doi: 10.1145/2505515.2505700. Disponível em: <https://doi.org/10.1145/2505515.2505700>.

BHAUMIK, R., WILLIAMS, C., MOBASHER, B., et al. "Securing collaborative filtering against malicious attacks through anomaly detection". In: *Proceedings of the 4th Workshop on Intelligent Techniques for Web Personalization (ITWP'06), Boston*, v. 6, p. 10, 2006.

MOBASHER, B., BURKE, R., BHAUMIK, R., et al. "Attacks and remedies in collaborative recommendation", *IEEE Intelligent Systems*, v. 22, n. 3, pp. 56–63, 2007a. ISSN: 15411672. doi: 10.1109/MIS.2007.45.

BURKE, R., O'MAHONY, M. P., HURLEY, N. J. "Robust Collaborative Recommendation". In: Ricci, F., Rokach, L., Shapira, B., et al. (Eds.), *Recommender Systems Handbook*, pp. 805–835, Boston, MA, Springer US, 2011. ISBN: 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_25. Disponível em: <https://doi.org/10.1007/978-0-387-85820-3_25>.

LAM, S. K., RIEDL, J. "Shilling recommender systems for fun and profit", *Thirteenth International World Wide Web Conference Proceedings, WWW2004*, pp. 393–402, 2004. doi: 10.1145/988672.988726.

ANELLI, V. W., DELDJOO, Y., DINOIA, T., et al. "Adversarial Recommender Systems: Attack, Defense, and Advances". In: Ricci, F., Rokach, L., Shapira, B. (Eds.), *Recommender Systems Handbook*, pp. 335–379, New York, NY, Springer US, 2022. ISBN: 978-1-0716-2197-4. doi: 10.1007/978-1-0716-2197-4_9. Disponível em: <https://doi.org/10.1007/978-1-0716-2197-4_9>.

MOBASHER, B., BURKE, R., BHAUMIK, R., et al. "Toward Trustworthy Recommender Systems: An Analysis of Attack Models and Algorithm Robustness", *ACM Trans. Internet Technol.*, v. 7, n. 4, pp. 23–es, out. 2007b. ISSN: 1533-5399. doi: 10.1145/1278366.1278372. Disponível em: <https://doi.org/10.1145/1278366.1278372>.

SEMINARIO, C., WILSON, D. "Attacking item-based recommender systems with power items", *RecSys*, pp. 57–64, 2014a. doi: 10.1145/2645710.2645722. Disponível em: <http://delivery.acm.org/10.1145/2650000/2645722/p57-seminario.pdf?ip=146.164.34.1{&}id=2645722{&}acc=ACTIVESERVICE{&}key=344E943C9DC262BB.36BD8EA867D3A5EB.4D4702B0C3E38B35.4D4702B0C3E38B35{&}{_}{_}acm{_}{_}=1522776643{_}dbbc11bcb85f668cc80f2f9c5a6badf2http://dl.acm.org/c>.

O'MAHONY, M. P., HURLEY, N. J., SILVESTRE, G. C. M. "Recommender Systems: Attack Types and Strategies". In: *IN PROCEEDINGS OF THE 20TH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE (AAAI-05*, pp. 334–339. AAAI Press, 2005.

SEMINARIO, C. E., WILSON, D. C. "Assessing impacts of a power user attack on a matrix factorization collaborative Recommender System", *Proceedings of the 27th International Florida Artificial Intelligence Research Society Conference, FLAIRS 2014*, pp. 81–86, 2014b. Disponível em: <https://pdfs.semanticscholar.org/0764/0f89fe3a97c8b6736f609b7c333be795d69a.pdfhttp://www.scopus.com/inward/record.url?eid=2-s2.0-84923870685{&}partnerID=40{&}md5=e565253f1f23540135a8de7b442220f9>.

MCCULLOCH, W., PITTS, W. "A Logical Calculus of Ideas Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, v. 5, pp. 127–147, 1943.

ROSENBLATT, F. “The perceptron: a probabilistic model for information storage and organization in the brain”, *Psychological Review*, v. 65, n. 6, pp. 386–408, nov. 1958.

MITCHELL, T. M. *Machine Learning*. 1 ed. New York, NY, USA, McGraw-Hill, Inc., 1997. ISBN: 0070428077, 9780070428072.

KINGMA, D. P., BA, J. “Adam: A Method for Stochastic Optimization”. 2014.

GRAVES, A. “Generating Sequences With Recurrent Neural Networks”. 2013.

SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, *J. Mach. Learn. Res.*, v. 15, n. 1, pp. 1929–1958, jan. 2014. ISSN: 1532-4435. Disponível em: <http://dl.acm.org/citation.cfm?id=2627435.2670313>.

LECUN, Y., BENGIO, Y. “Convolutional Networks for Images, Speech, and Time Series”. In: *The Handbook of Brain Theory and Neural Networks*, p. 255–258, Cambridge, MA, USA, MIT Press, 1998. ISBN: 0262511029.

PALAZ, D., COLLOBERT, R., MAGIMAI-DOSS, M. “Estimating Phoneme Class Conditional Probabilities from Raw Speech Signal using Convolutional Neural Networks”, *CoRR*, v. abs/1304.1018, 2013. Disponível em: <http://dblp.uni-trier.de/db/journals/corr/corr1304.html#abs-1304-1018>.

KIM, Y. “Convolutional Neural Networks for Sentence Classification”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1746–1751, Doha, Qatar, out. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. Disponível em: <https://www.aclweb.org/anthology/D14-1181>.

GU, J., WANG, Z., KUEN, J., et al. “Recent Advances in Convolutional Neural Networks”, *Pattern Recogn.*, v. 77, n. C, pp. 354–377, maio 2018. ISSN: 0031-3203. doi: 10.1016/j.patcog.2017.10.013. Disponível em: <https://doi.org/10.1016/j.patcog.2017.10.013>.

DUMOULIN, V., VISIN, F. “A guide to convolution arithmetic for deep learning”. 2016. Disponível em: <http://arxiv.org/abs/1603.07285>. cite arxiv:1603.07285.

LECUN, Y. A., BOTTOU, L., ORR, G. B., et al. “Efficient BackProp”. In: Montavon, G., Orr, G. B., Müller, K.-R. (Eds.), *Neural Networks: Tricks of the Trade: Second Edition*, pp. 9–48, Berlin, Heidel-

berg, Springer Berlin Heidelberg, 2012. ISBN: 978-3-642-35289-8. doi: 10.1007/978-3-642-35289-8_3. Disponível em: <https://doi.org/10.1007/978-3-642-35289-8_3>.

MAAS, A. L., HANNUN, A. Y., NG, A. Y. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*, v. 30, p. 3, 2013.

HINTON, G. E., SALAKHUTDINOV, R. R. "Reducing the Dimensionality of Data with Neural Networks", *Science*, v. 313, n. 5786, pp. 504–507, 2006. ISSN: 0036-8075. doi: 10.1126/science.1127647. Disponível em: <http://science.sciencemag.org/content/313/5786/504>.

VINCENT, P., LAROCHELLE, H., BENGIO, Y., et al. "Extracting and Composing Robust Features with Denoising Autoencoders". In: *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 1096–1103, New York, NY, USA, 2008. ACM. ISBN: 978-1-60558-205-4. doi: 10.1145/1390156.1390294. Disponível em: <http://doi.acm.org/10.1145/1390156.1390294>.

BENGIO, Y., YAO, L., ALAIN, G., et al. "Generalized Denoising Auto-Encoders as Generative Models". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, p. 899–907, Red Hook, NY, USA, 2013. Curran Associates Inc.

KULLBACK, S., LEIBLER, R. A. "On information and sufficiency", *The annals of mathematical statistics*, v. 22, n. 1, pp. 79–86, 1951.

HOFFMAN, M. D., JOHNSON, M. J. "Elbo surgery: yet another way to carve up the variational evidence lower bound". In: *Workshop in Advances in Approximate Bayesian Inference, NIPS*, v. 1, 2016.

GOODFELLOW, I. J., POUGET-ABADIE, J., MIRZA, M., et al. "Generative adversarial nets". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, p. 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

O'MAHONY, M. P., HURLEY, N. J., SILVESTRE, G. C. M. "Towards Robust Collaborative Filtering". In: O'Neill, M., Sutcliffe, R. F. E., Ryan, C., et al. (Eds.), *Artificial Intelligence and Cognitive Science*, pp. 87–94, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN: 978-3-540-45750-3.

CHEN, K., CHAN, P. P., ZHANG, F., et al. "Shilling attack based on item popularity and rated item correlation against collaborative filtering", *Interna-*

tional Journal of Machine Learning and Cybernetics, v. 10, n. 7, pp. 1833–1845, 2019. ISSN: 1868808X. doi: 10.1007/s13042-018-0861-2. Disponível em: <http://dx.doi.org/10.1007/s13042-018-0861-2>.

RAY, S., MAHANTI, A. "Filler item strategies for shilling attacks against recommender systems", *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences, HICSS*, pp. 1–10, 2009. doi: 10.1109/HICSS.2009.217.

WILSON, D. C., SEMINARIO, C. E. "When power users attack: assessing impacts in collaborative recommender systems". In: *[RecSys2013]Proceedings of the 7th ACM conference on Recommender systems*, pp. 427–430, New York, New York, USA, 2013. ACM Press. ISBN: 9781450324090. doi: 10.1145/2507157.2507220. Disponível em: <http://dl.acm.org/citation.cfm?doid=2507157.2507220http://dl.acm.org/citation.cfm?id=2507220{%}5Cnhttp://dl.acm.org/citation.cfm?doid=2507157.2507220>.

WILSON, D., SEMINARIO, C. "Evil twins: Modeling power users in attacks on recommender systems", *User Modeling, Adaptation, and ...*, pp. 231–242, 2014. ISSN: 16113349. doi: 10.1007/978-3-319-08786-3_20. Disponível em: <http://link.springer.com/10.1007/978-3-319-08786-3{_}20http://link.springer.com/chapter/10.1007/978-3-319-08786-3{_}20>.

SEMINARIO, C. E., WILSON, D. C. "Nuke ' Em Till They Go : Investigating Power User Attacks to Disparage Items in Collaborative Recommenders", *RecSys 2015: Proceedings of the 9th ACM conference on Recommender systems*, pp. 293–296, 2015. doi: 10.1145/2792838.2799666. Disponível em: <http://delivery.acm.org/10.1145/2800000/2799666/p293-seminario.pdf?ip=146.164.34.1{&}id=2799666{&}acc=ACTIVESERVICE{&}key=344E943C9DC262BB.36BD8EA867D3A5EB.4D4702B0C3E38B35.4D4702B0C3E38B35{&}{_}{_}acm{_}{_}=1523539577{_}9992d7e4af27e6b61a3800b589cd4cf3>.

WILSON, D. C., SEMINARIO, C. E., CAROLINA, N. "Mitigating Power User Attacks on a User-Based Collaborative Recommender System", pp. 513–518, 2015. Disponível em: <https://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS15/paper/viewFile/10451/10433>.

FANG, M., YANG, G., GONG, N. Z., et al. "Poisoning Attacks to Graph-Based Recommender Systems". In: *Proceedings of the 34th Annual Computer*

Security Applications Conference, ACSAC '18, p. 381–392, New York, NY, USA, 2018. Association for Computing Machinery. ISBN: 9781450365697. doi: 10.1145/3274694.3274706. Disponível em: <https://doi.org/10.1145/3274694.3274706>.

CHRISTAKOPOULOU, K., BANERJEE, A. "Adversarial atacks on an oblivious recommender", *RecSys 2019 - 13th ACM Conference on Recommender Systems*, pp. 322–330, 2019. doi: 10.1145/3298689.3347031.

ANELLI, V. W., DELDJOO, Y., DI NOIA, T., et al. "SAShA: Semantic-Aware Shilling Attacks on Recommender Systems Exploiting Knowledge Graphs". In: Harth, A., Kirrane, S., Ngonga Ngomo, A.-C., et al. (Eds.), *The Semantic Web*, pp. 307–323, Cham, 2020. Springer International Publishing. ISBN: 978-3-030-49461-2.

LIN, C., CHEN, S., LI, H., et al. "Attacking Recommender Systems with Augmented User Profiles". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, p. 855–864, New York, NY, USA, 2020. Association for Computing Machinery. ISBN: 9781450368599. doi: 10.1145/3340531.3411884. Disponível em: <https://doi.org/10.1145/3340531.3411884>.

WU, F., GAO, M., YU, J., et al. "Ready for emerging threats to recommender systems? A graph convolution-based generative shilling attack", *Information Sciences*, v. 578, pp. 683–701, 2021. ISSN: 0020-0255. doi: https://doi.org/10.1016/j.ins.2021.07.041. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0020025521007313>.

WANG, Z., GAO, M., LI, J., et al. "Gray-Box Shilling Attack: An Adversarial Learning Approach", *ACM Trans. Intell. Syst. Technol.*, v. 13, n. 5, oct 2022. ISSN: 2157-6904. doi: 10.1145/3512352. Disponível em: <https://doi.org/10.1145/3512352>.

REN, Y., LI, Z., YUAN, L., et al. "Semantic Shilling Attack against Heterogeneous Information Network Based Recommend Systems", *IEICE Transactions on Information and Systems*, v. E105.D, n. 2, pp. 289–299, 2022. doi: 10.1587/transinf.2021BCP0015.

HUANG, C., LI, H. "Single-User Injection for Invisible Shilling Attack against Recommender Systems". In: *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, CIKM '23, p.

864–873, New York, NY, USA, 2023. Association for Computing Machinery. ISBN: 9798400701245. doi: 10.1145/3583780.3615062. Disponível em: <https://doi.org/10.1145/3583780.3615062>.

LIN, C., CHEN, S., ZENG, M., et al. "Shilling Black-Box Recommender Systems by Learning to Generate Fake User Profiles", *IEEE Transactions on Neural Networks and Learning Systems*, v. 35, n. 1, pp. 1305–1319, 2024. doi: 10.1109/TNNLS.2022.3183210.

LU, Q., GAO, M. "A Research on Shilling Attacks Based on Variational graph auto-encoders for Improving the Robustness of Recommendation Systems". In: *Proceedings of the 2024 International Conference on Generative Artificial Intelligence and Information Security*, GAIIS '24, p. 120–126, New York, NY, USA, 2024. Association for Computing Machinery. ISBN: 9798400709562. doi: 10.1145/3665348.3665370. Disponível em: <https://doi.org/10.1145/3665348.3665370>.

ZHANG, S., CHAKRABARTI, A., FORD, J., et al. "Attack detection in time series for recommender systems". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06*, p. 809, New York, New York, USA, 2006. ACM Press. ISBN: 1595933395. doi: 10.1145/1150402.1150508. Disponível em: <http://portal.acm.org/citation.cfm?doid=1150402.1150508>.

TONG, C., YIN, X., LI, J., et al. "A shilling attack detector based on convolutional neural network for collaborative recommender system in social aware network", *The Computer Journal*, feb 2018. ISSN: 0010-4620. doi: 10.1093/comjnl/bxy008. Disponível em: <https://academic.oup.com/comjnl/advance-article/doi/10.1093/comjnl/bxy008/4835634>.

MELLO, C. E., AUFAURE, M.-A., ZIMBRAO, G. "Active Learning Driven by Rating Impact Analysis". In: *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, p. 341–344, New York, NY, USA, 2010. Association for Computing Machinery. ISBN: 9781605589060. doi: 10.1145/1864708.1864782. Disponível em: <https://doi.org/10.1145/1864708.1864782>.

ZHANG, S., YIN, H., CHEN, T., et al. "GCN-Based User Representation Learning for Unifying Robust Recommendation and Fraudster Detection". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 689–698, 2020.

NOH, G., KANG, Y.-M., OH, H., et al. "Robust Sybil attack defense with information level in online Recommender Systems", *Expert Systems with Applications*, v. 41, n. 4, pp. 1781–1791, mar 2014. ISSN: 0957-4174. doi: 10.1016/J.ESWA.2013.08.077. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417413006982>.

LI, C., LUO, Z. "Detection of shilling attacks in collaborative filtering recommender systems". In: *2011 International Conference of Soft Computing and Pattern Recognition (SoCPaR)*, pp. 190–193, Oct 2011. doi: 10.1109/SoCPaR.2011.6089138.

ANGLUIN, D., LAIRD, P. "Learning from noisy examples", *Machine Learning*, v. 2, n. 1984, pp. 343–370, 1988. ISSN: 08856125. doi: 10.1007/BF00116829.

SÁEZ, J. A., GALAR, M., LUENGO, J., et al. "Analyzing the presence of noise in multi-class problems: alleviating its influence with the One-vs-One decomposition", *Knowledge and Information Systems*, v. 38, n. 1, pp. 179–206, 2014.

FRENAY, B., VERLEYSEN, M. "Classification in the Presence of Label Noise: A Survey", *IEEE Transactions on Neural Networks and Learning Systems*, v. 25, n. 5, pp. 845–869, 2014. doi: 10.1109/TNNLS.2013.2292894.

VAN DEN HOUT, A., VAN DER HEIJDEN, P. G. M., VAN DEN HOUT, A., et al. "Randomized Response, Statistical Disclosure Control and Misclassification: A Review", *International Statistical Review / Revue Internationale de Statistique*, v. 70, n. 2, pp. pp. 269—-288, 2002. ISSN: 03067734. doi: 10.2307/1403910. Disponível em: <http://www.jstor.org/stable/1403910>.

CARMO, F. B. D. *Considerando o ruído no aprendizado de modelos preditivos robustos para a filtragem colaborativa.* Tese de Doutorado, 2018.

BRODLEY, C. E., FRIEDL, M. A. "Identifying Mislabeled Training Data", *CoRR*, v. abs/1106.0219, 2011. Disponível em: <http://arxiv.org/abs/1106.0219>.

O'MAHONY, M. P., HURLEY, N. J., SILVESTRE, G. C. "Detecting noise in recommender system databases". In: *Proceedings of the 11th international conference on Intelligent user interfaces - IUI '06*, p. 109, 2006. ISBN: 1595932879. doi: 10.1145/1111449.1111477. Disponível

em: <https://www.researchgate.net/publication/221607786http://portal.acm.org/citation.cfm?doid=1111449.1111477>.

TOLEDO, R. Y., MOTA, Y. C., MARTÍNEZ, L. "Correcting Noisy Ratings in Collaborative Recommender Systems", *Know.-Based Syst.*, v. 76, n. 1, pp. 96–108, mar 2015. ISSN: 0950-7051. doi: 10.1016/j.knosys.2014.12.011. Disponível em: <https://doi.org/10.1016/j.knosys.2014.12.011>.

GOLDBERGER, J., BEN-REUVEN, E. "Training deep neural-networks using a noise adaptation layer", 2016.

SUKHBAATAR, S., FERGUS, R. "Learning from noisy labels with deep neural networks," arXiv". 2014.

REED, S., LEE, H., ANGUELOV, D., et al. "Training Deep Neural Networks on Noisy Labels with Bootstrapping". 2015.

PATRINI, G., NIELSEN, F., NOCK, R., et al. "Loss factorization, weakly supervised learning and label noise robustness", 2016a. Disponível em: <http://arxiv.org/abs/1602.02450>.

PATRINI, G., ROZZA, A., MENON, A., et al. "Making Deep Neural Networks Robust to Label Noise: a Loss Correction Approach", 2016b. doi: 10.1109/CVPR.2017.240. Disponível em: <http://arxiv.org/abs/1609.03683>.

REID, M. D., WILLIAMSON, R. C. "Composite Binary Losses", *The Journal of Machine Learning Research*, v. 11, pp. 2387–2422, 2010.

LI, W., GAO, M., LI, H., et al. "Shilling attack detection in recommender systems via selecting patterns analysis", *IEICE TRANSACTIONS on Information and Systems*, v. 99, n. 10, pp. 2600–2611, 2016.

LI, W., GAO, M., LI, H., et al. "An shilling attack detection algorithm based on popularity degree features", *Zidonghua Xuebao/Acta Automatica Sinica*, v. 41, n. 9, pp. 1563–1575, 2015.

DOU, T., YU, J., XIONG, Q., et al. "Collaborative shilling detection bridging factorization and user embedding". In: *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pp. 459–469. Springer, 2017.

MIKOLOV, T., CHEN, K., CORRADO, G., et al. "Efficient Estimation of Word Representations in Vector Space". 2013. Disponível em: <https://arxiv.org/abs/1301.3781>.

LEVY, O., GOLDBERG, Y. "Neural Word Embedding as Implicit Matrix Factorization". In: Ghahramani, Z., Welling, M., Cortes, C., et al. (Eds.), *Advances in Neural Information Processing Systems*, v. 27. Curran Associates, Inc., 2014. Disponível em: <https://proceedings.neurips.cc/paper/2014/file/feab05aa91085b7a8012516bc3533958-Paper.pdf>.

CAO, J., WU, Z., MAO, B., et al. "Shilling attack detection utilizing semi-supervised learning method for collaborative recommender system", *World Wide Web*, v. 16, n. 5, pp. 729–748, 2013.

CHIRITA, P.-A., NEJDL, W., ZAMFIR, C. "Preventing Shilling Attacks in Online Recommender Systems". In: *Proceedings of the 7th Annual ACM International Workshop on Web Information and Data Management*, WIDM '05, p. 67–74, New York, NY, USA, 2005. Association for Computing Machinery. ISBN: 1595931945. doi: 10.1145/1097047.1097061. Disponível em: <https://doi.org/10.1145/1097047.1097061>.

WILLIAMS, C., MOBASHER, B. "Profile injection attack detection for securing collaborative recommender systems", *DePaul University CTI Technical Report*, pp. 1–47, 2006.

NIGAM, K., MCCALLUM, A. K., THRUN, S., et al. "Text Classification from Labeled and Unlabeled Documents using EM", *Machine Learning*, v. 39, n. 2/3, pp. 103–134, 2000. Disponível em: <citeseer.nj.nec.com/nigam99text.html>.

MEHTA, B., NEJDL, W. "Unsupervised strategies for shilling detection and robust collaborative filtering", *User Modeling and User-Adapted Interaction*, v. 19, n. 1, pp. 65–97, 2009.

JOLLIFFE, I. T. *Principal component analysis for special types of data*. Springer, 2002.

ZHANG, Y., TAN, Y., ZHANG, M., et al. "Catch the Black Sheep: Unified Framework for Shilling Attack Detection Based on Fraudulent Action Propagation". In: *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, p. 2408–2414. AAAI Press, 2015. ISBN: 9781577357384.

CHUNG, C.-Y., HSU, P.-Y., HUANG, S.-H. "βP: A novel approach to filter out malicious rating profiles from recommender systems", *Decision Support Systems*, v. 55, n. 1, pp. 314–325, 2013. ISSN: 0167-9236. doi: https://doi.org/10.1016/j.dss.2013.01.020. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167923613000481>.

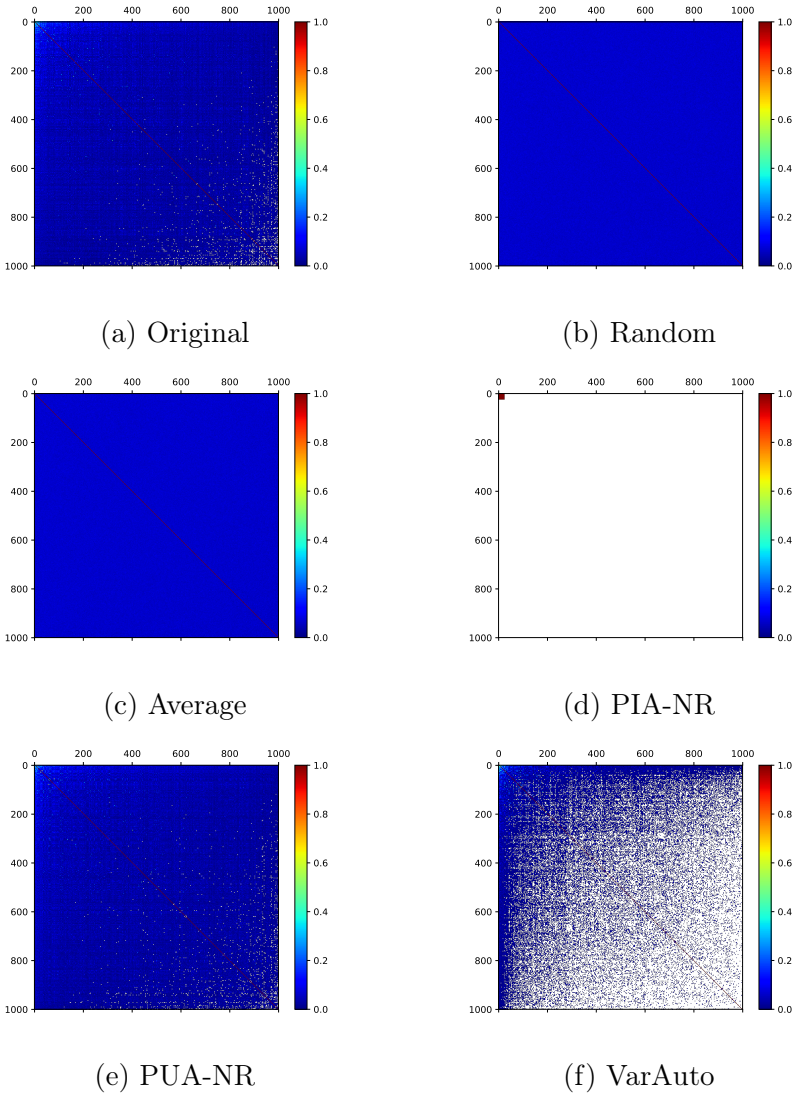
FAN, S., ZHU, J., HAN, X., et al. "Metapath-guided Heterogeneous Graph Neural Network for Intent Recommendation". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, p. 2478–2486, New York, NY, USA, 2019. Association for Computing Machinery. ISBN: 9781450362016. doi: 10.1145/3292500.3330673. Disponível em: <https://doi.org/10.1145/3292500.3330673>.

Appendix A

Additional Experimental Results

A.0.1 Item Correlation Analysis

Yahoo! Music

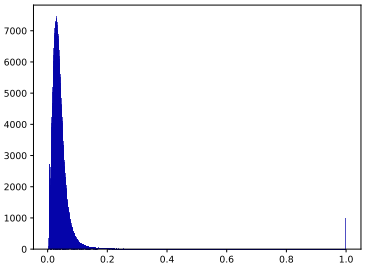


(a) Original (b) Random

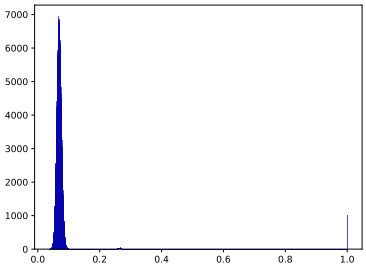
(c) Average (d) PIA-NR

(e) PUA-NR (f) VarAuto

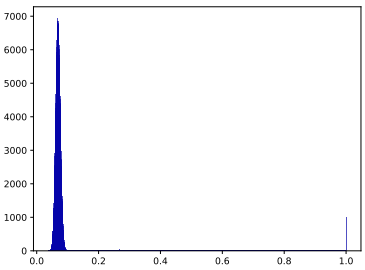
Figure A.1: CA of item pairs of Yahoo! Music (a), and for different shilling attack models (b-e) and the proposal (f).



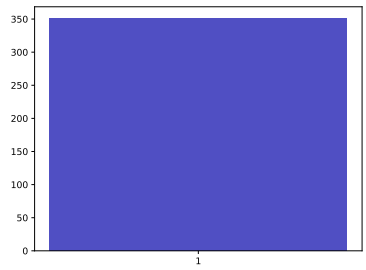
(a) Original



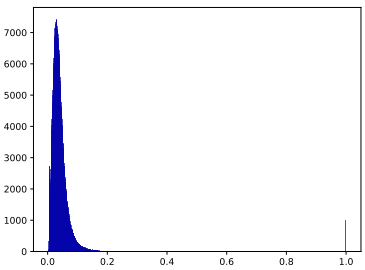
(b) Random



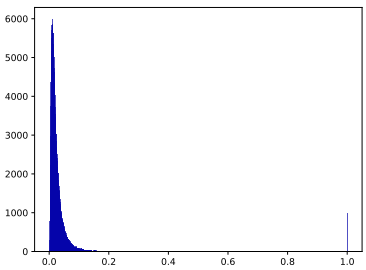
(c) Average



(d) PIA-NR



(e) PUA-NR



(f) VarAuto

Figure A.2: CA distribution of item pairs of Yahoo! Music (a), and for different shilling attack models (b-e) and the proposal (f).

A.0.2 Attack Models Evaluation

Yahoo! Music

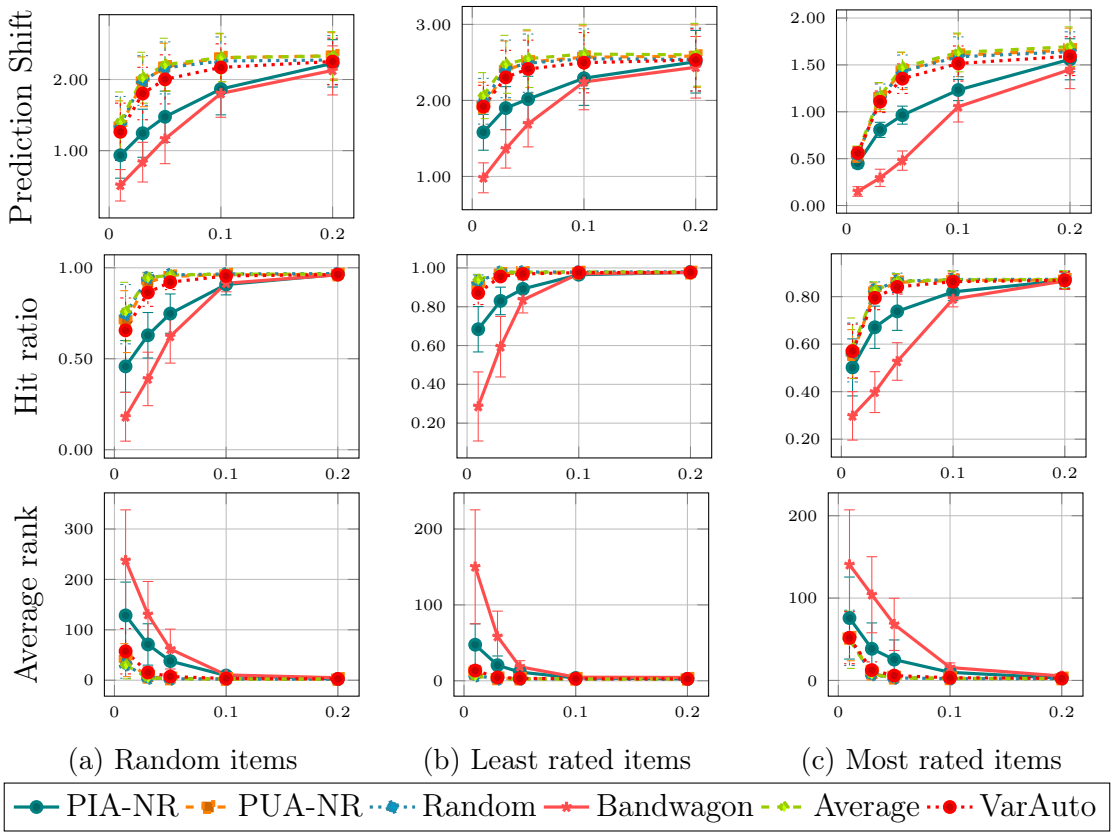


Figure A.3: Prediction shift, hit ratio and average rank for different attacks sizes using Improved Regularized SVD as collaborative filtering technique and Yahoo! Music dataset.

A.0.3 Data Cleansing

Movielens 100k

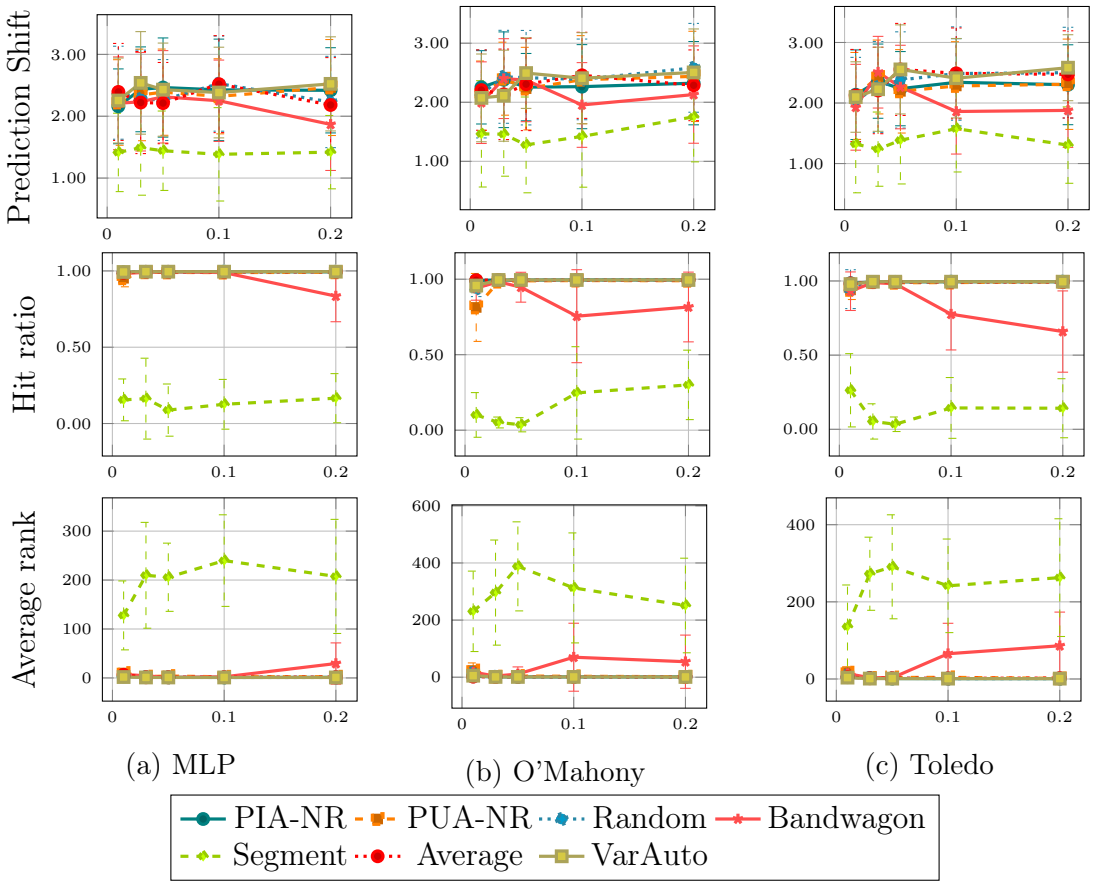


Figure A.4: Prediction shift, hit ratio, and average rank for different attack sizes comparing the baseline with O'Mahony's and Toledo's Data Cleansing approach using least-rated target items from MovieLens 100k data set and with attackers injected in the base estimator training data. Note that it only shows the best result of filler size among the attack models that require it.

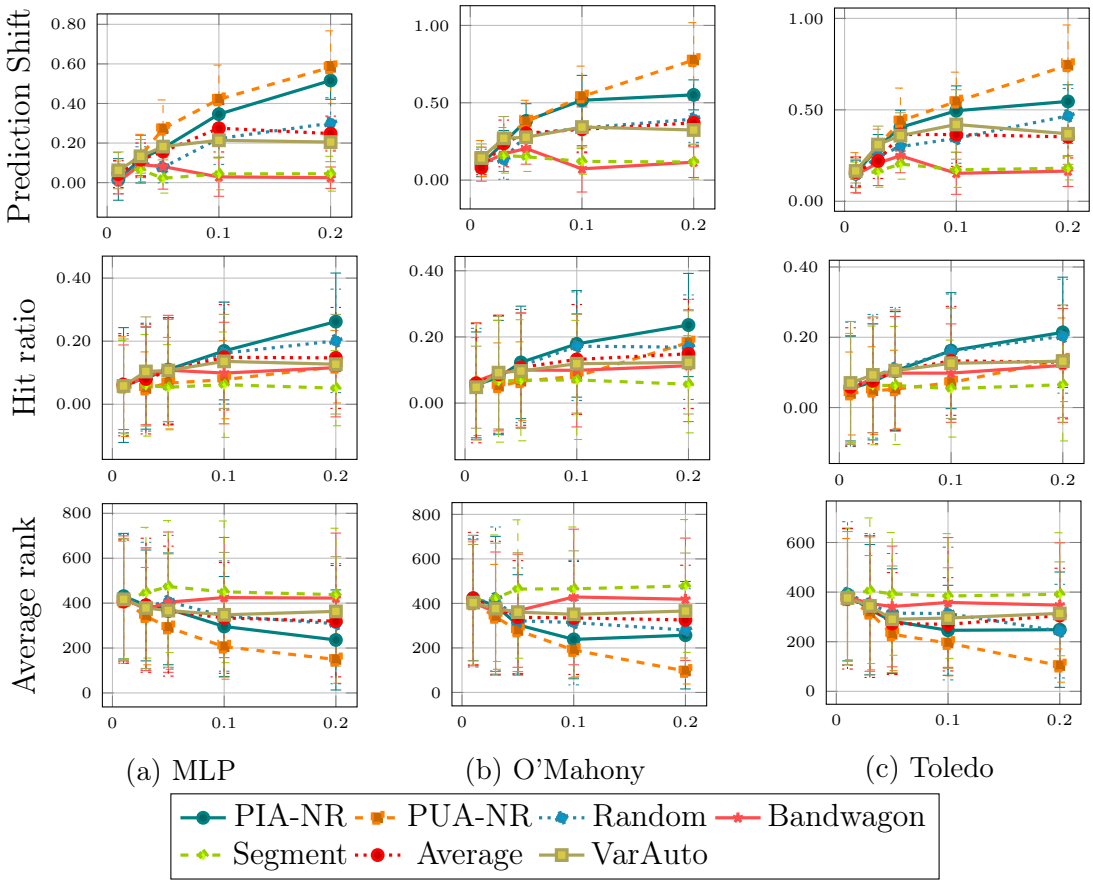


Figure A.5: Prediction shift, hit ratio, and average rank for different attack sizes comparing the baseline with O'Mahony's and Toledo's Data Cleansing approach using most-rated target items from MovieLens 100k data set and with attackers injected in the base estimator training data. Note that it only shows the best result of filler size among the attack models that require it.

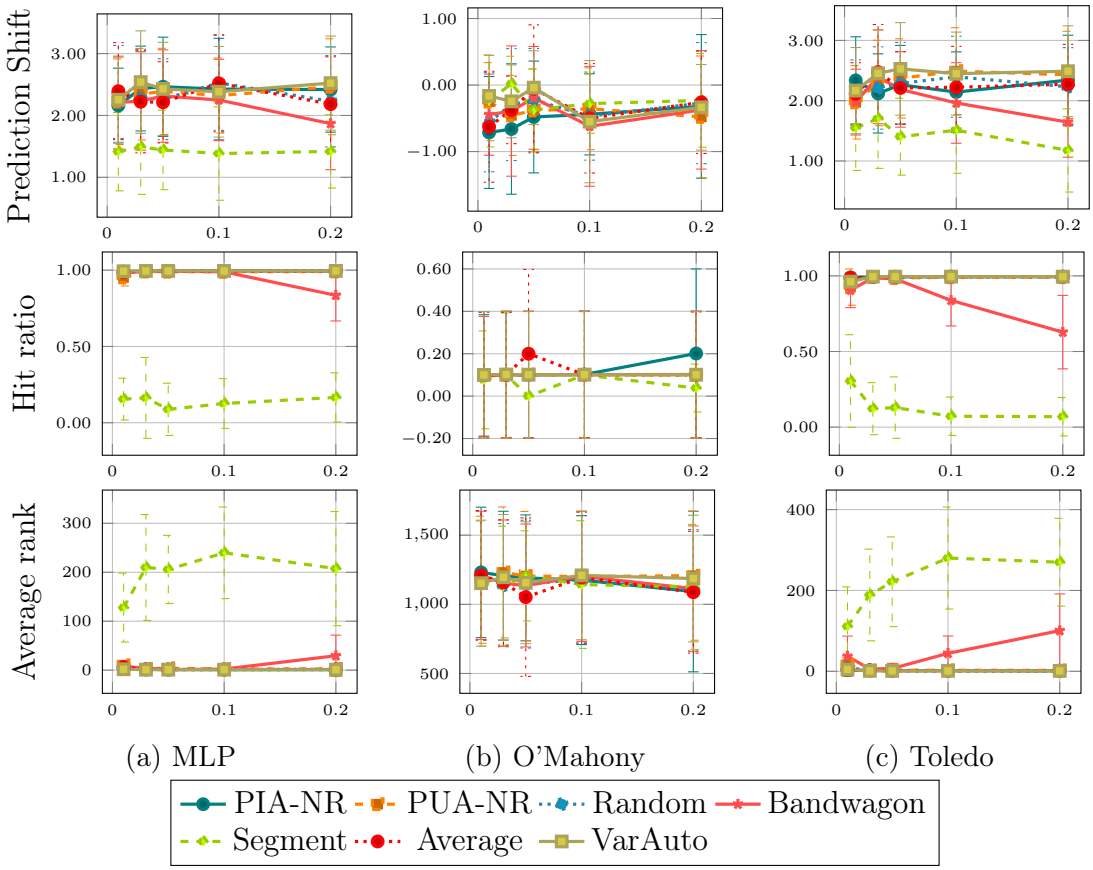


Figure A.6: Prediction shift, hit ratio, and average rank for different attack sizes comparing the baseline with O'Mahony's and Toledo's Data Cleansing approach using least-rated target items from MovieLens 100k data set and no attackers in the base estimator. Note that it only shows the best result of filler size among the attack models that require it.

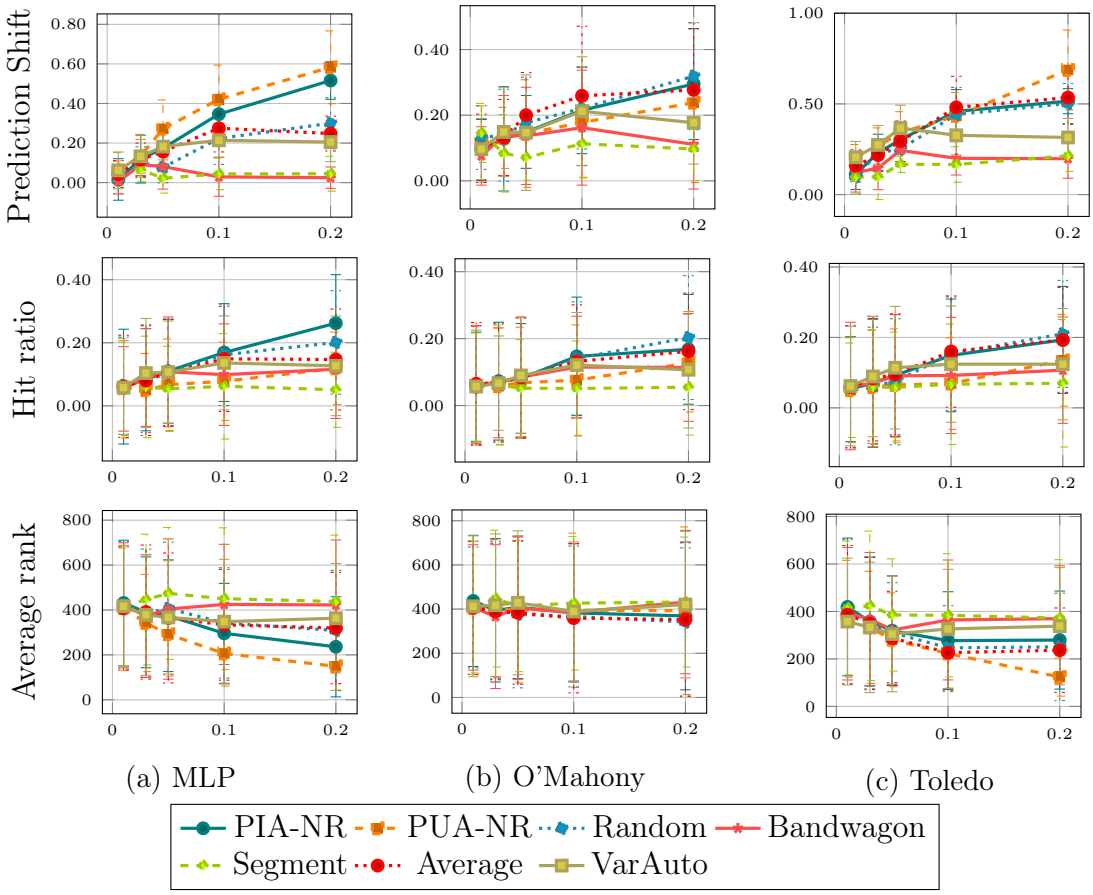


Figure A.7: Prediction shift, hit ratio, and average rank for different attack sizes comparing the baseline with O'Mahony's and Toledo's Data Cleansing approach using most-rated target items from MovieLens 100k data set and no attackers in the base estimator. Note that it only shows the best result of filler size among the attack models that require it.

Yahoo! Music

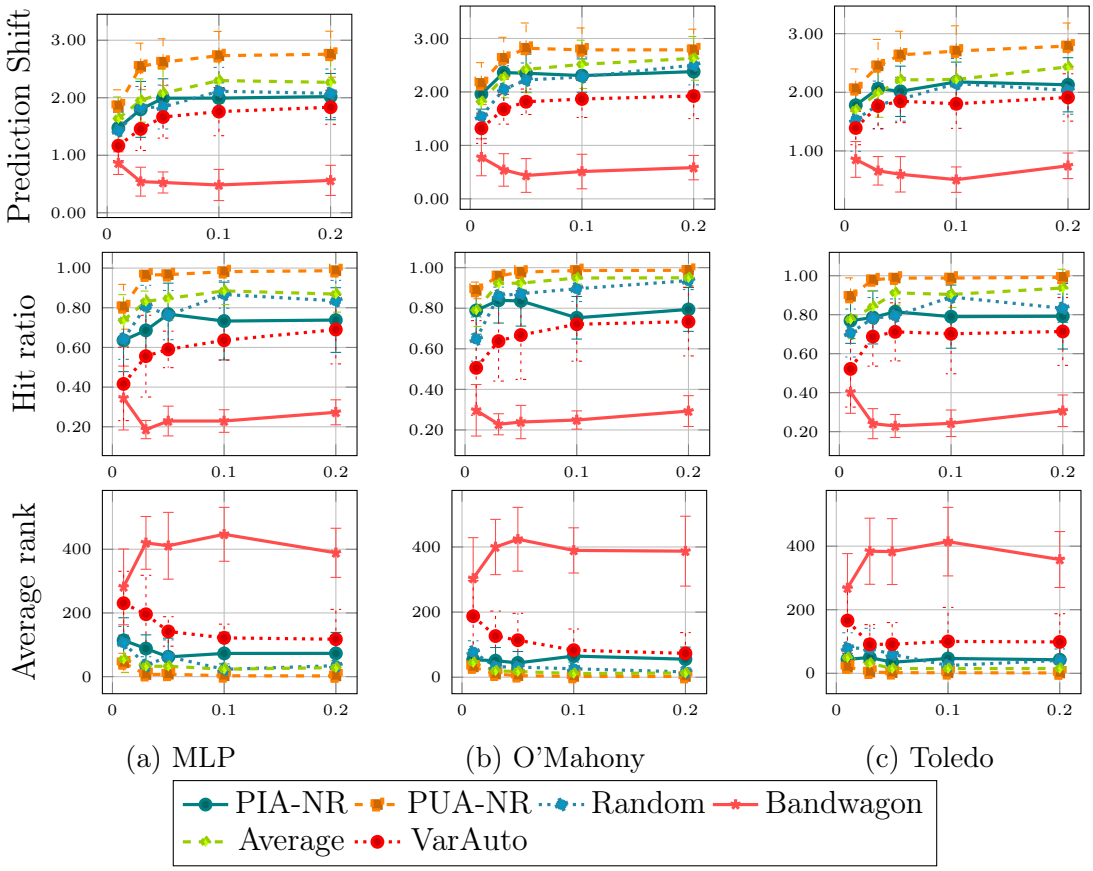


Figure A.8: Prediction shift, hit ratio, and average rank for different attack sizes comparing the baseline with O'Mahony's and Toledo's Data Cleansing approach using least-rated target items from Yahoo! Music data set and with attackers injected in the base estimator training data. Note that it only shows the best result of filler size among the attack models that require it.

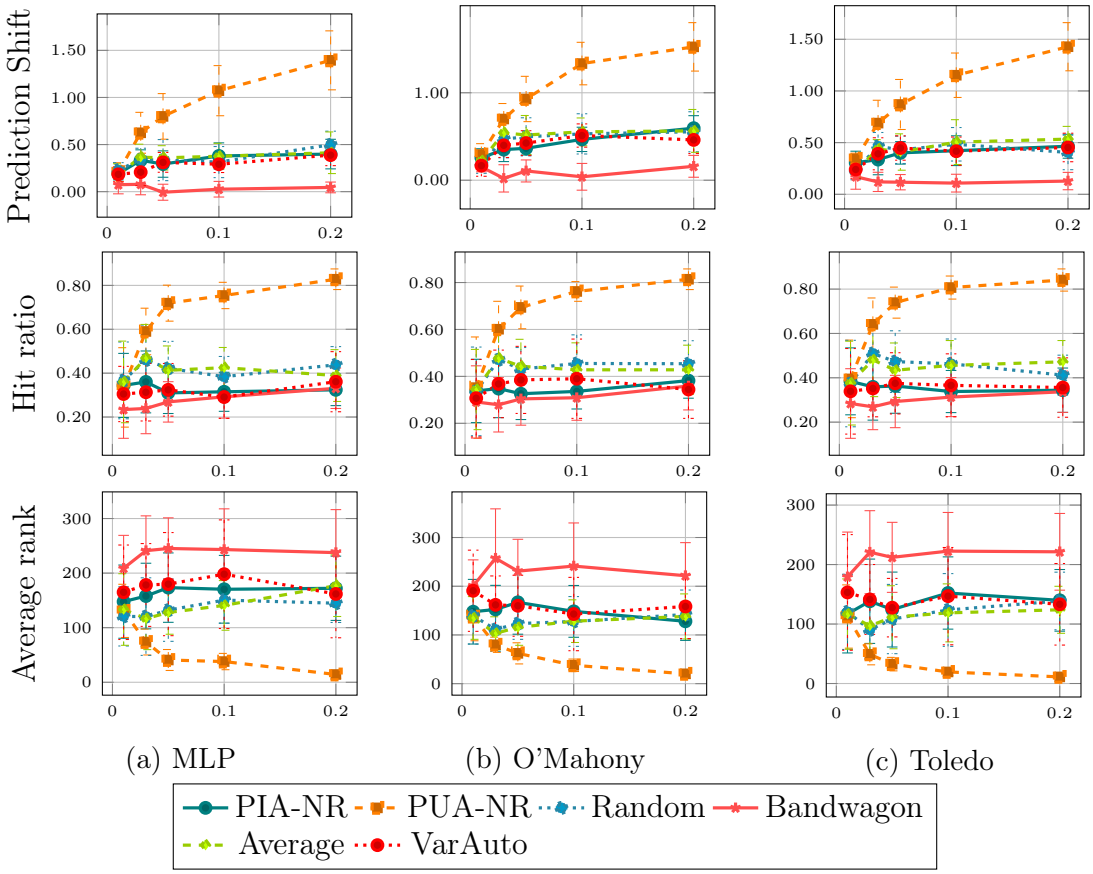


Figure A.9: Prediction shift, hit ratio, and average rank for different attack sizes comparing the baseline with O'Mahony's and Toledo's Data Cleansing approach using most-rated target items from Yahoo! Music data set and with attackers injected in the base estimator training data. Note that it only shows the best result of filler size among the attack models that require it.

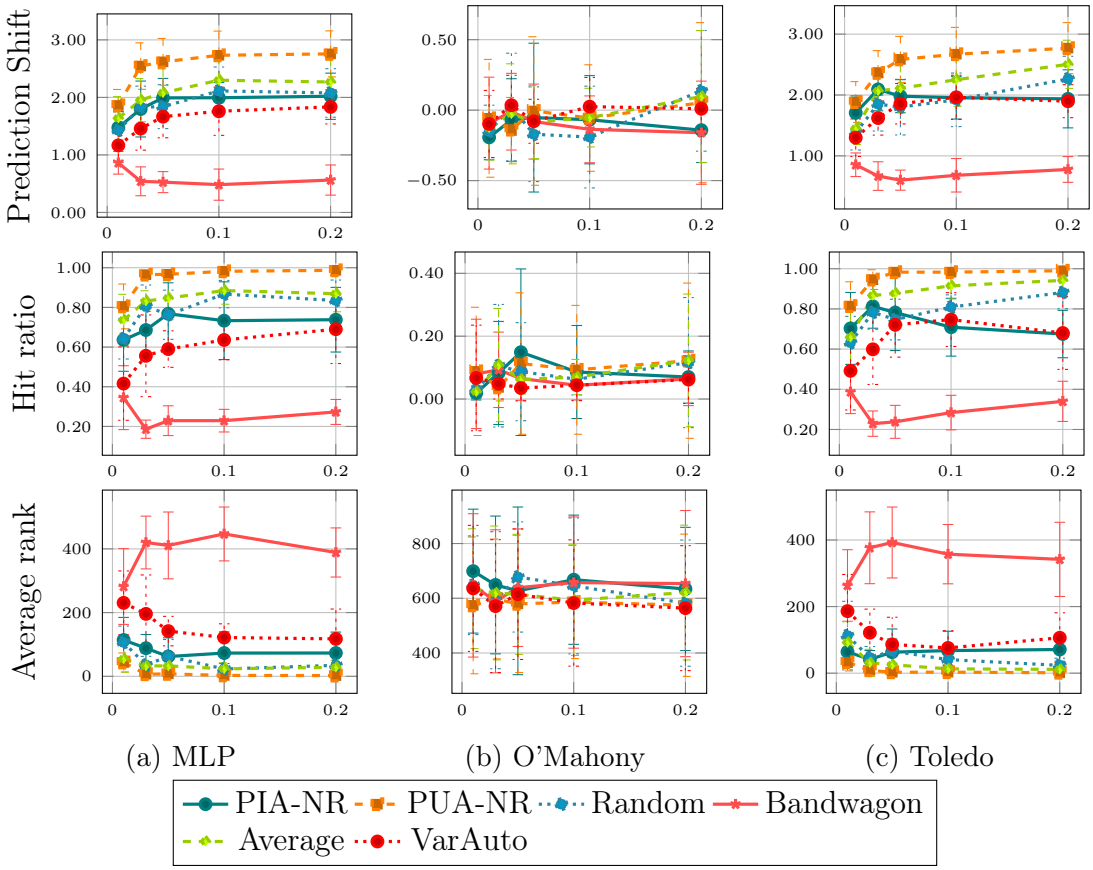


Figure A.10: Prediction shift, hit ratio, and average rank for different attack sizes comparing the baseline with O'Mahony's and Toledo's Data Cleansing approach using least-rated target items from Yahoo! Music data set and no attackers in the base estimator. Note that it only shows the best result of filler size among the attack models that require it.

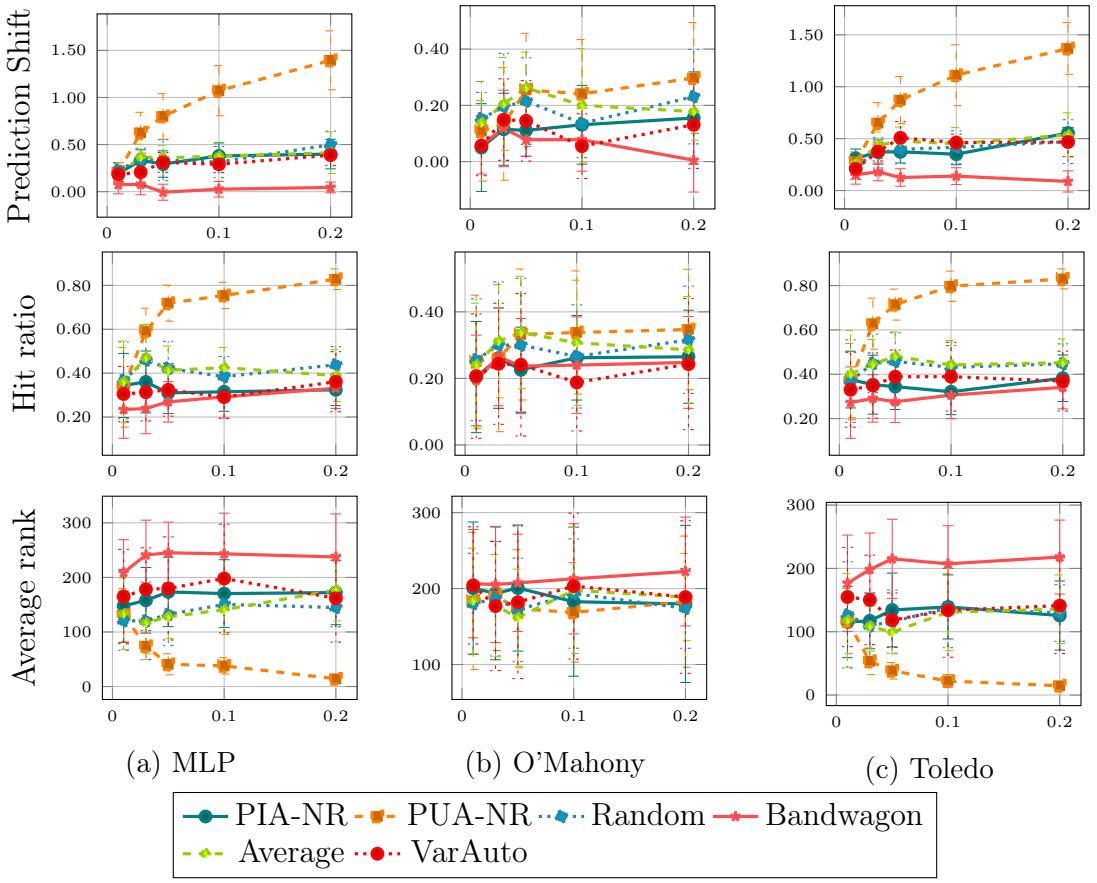


Figure A.11: Prediction shift, hit ratio, and average rank for different attack sizes comparing the baseline with O'Mahony's and Toledo's Data Cleansing approach using most-rated target items from Yahoo! Music data set and no attackers in the base estimator. Note that it only shows the best result of filler size among the attack models that require it.

A.0.4 Label Noise-Robust Algorithms

Movielens 100k

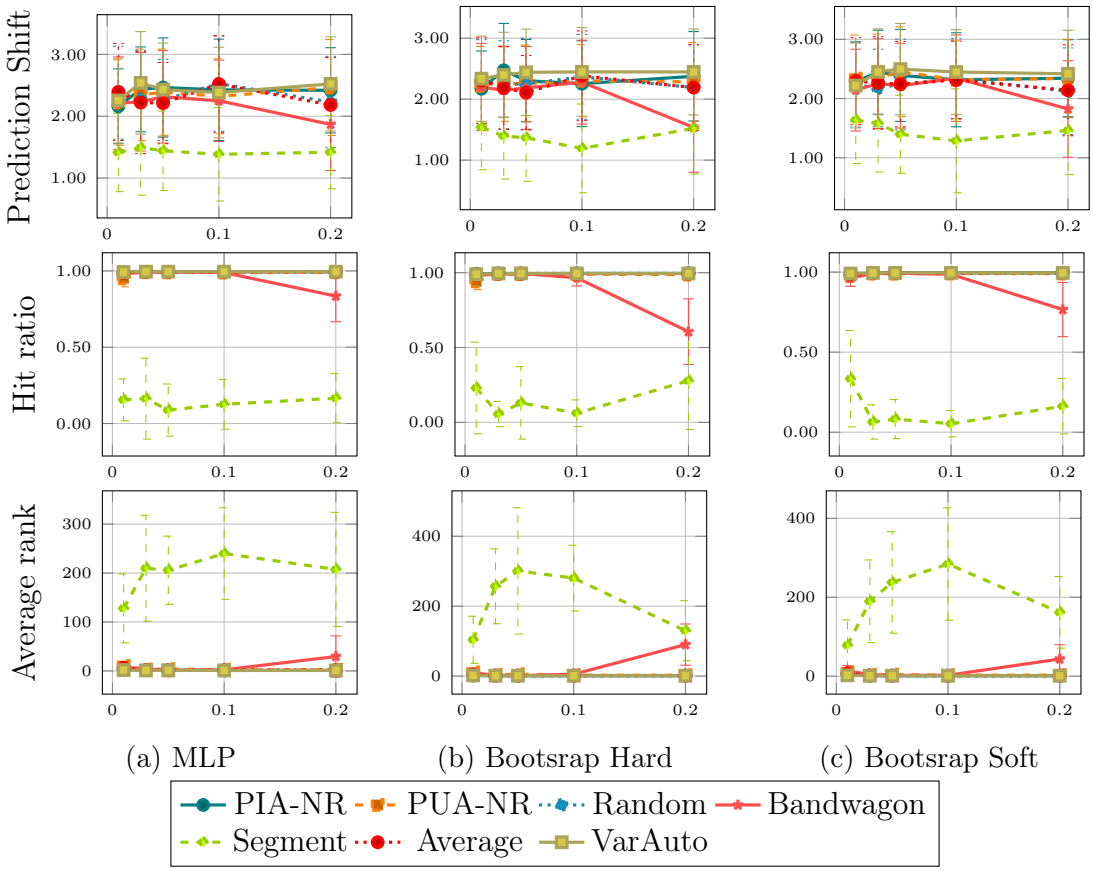


Figure A.12: Prediction shift, hit ratio, and average rank for different attack sizes comparing a regular MLP with Bootstrapping techniques using least-rated target items from MovieLens 100k data set. Note that it only shows the best result of filler size among the attack models that require it.

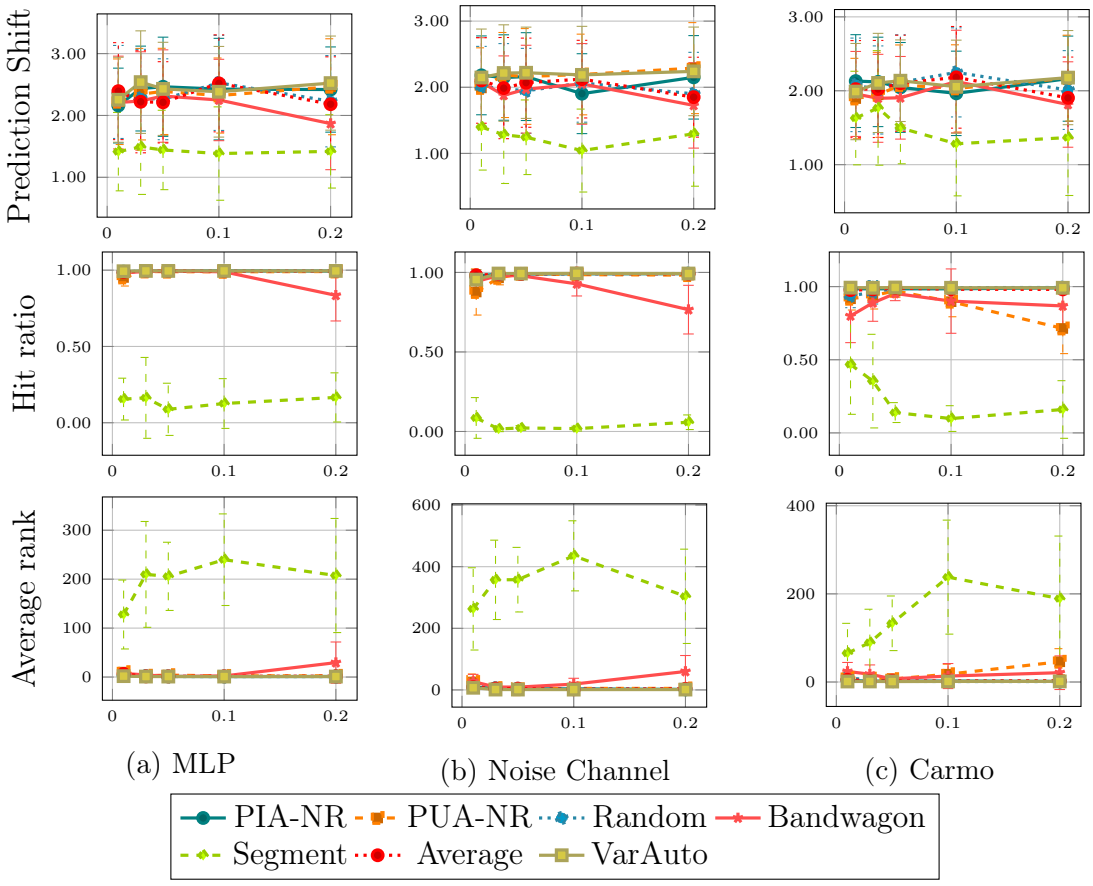


Figure A.13: Prediction shift, hit ratio, and average rank for different attack sizes comparing a regular MLP with Noise Channel and Carmo techniques using least-rated target items from MovieLens 100k data set. Note that it only shows the best result of filler size among the attack models that require it.

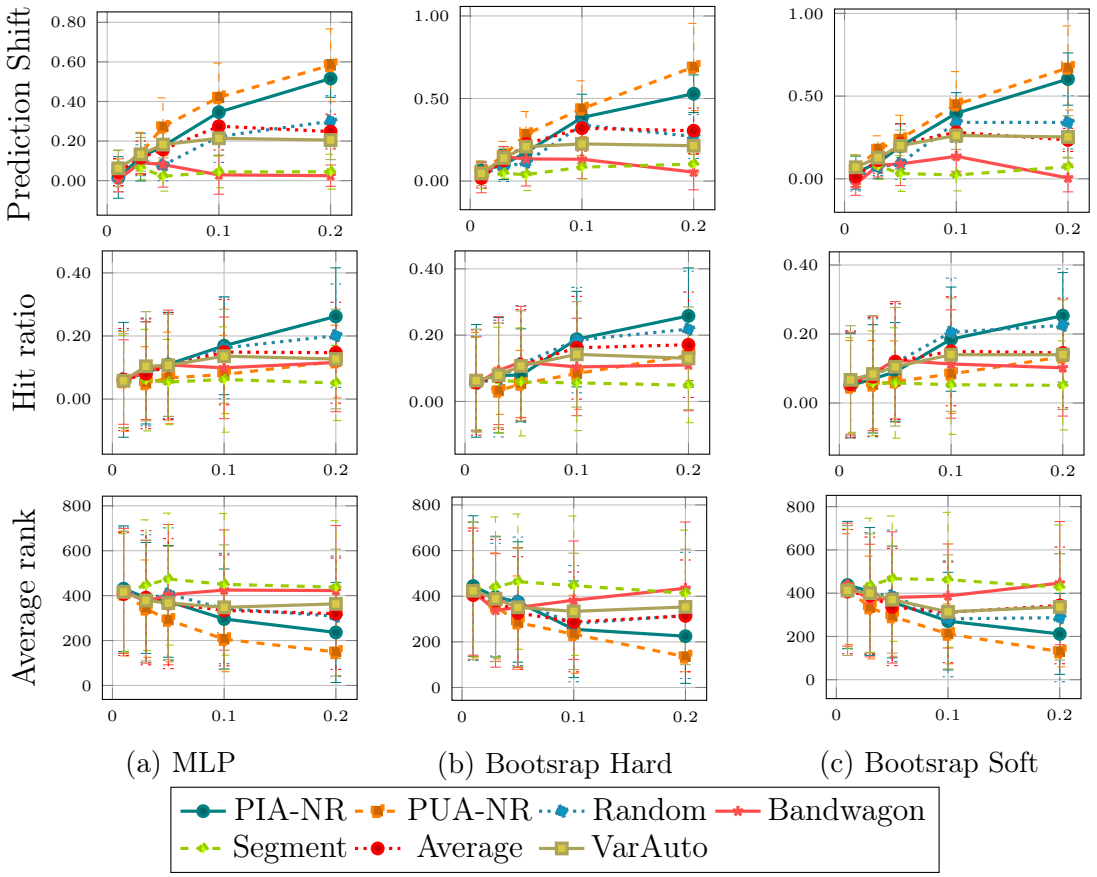


Figure A.14: Prediction shift, hit ratio, and average rank for different attack sizes comparing a regular MLP with Bootstrapping techniques using most-rated target items from MovieLens 100k data set. Note that it only shows the best result of filler size among the attack models that require it.

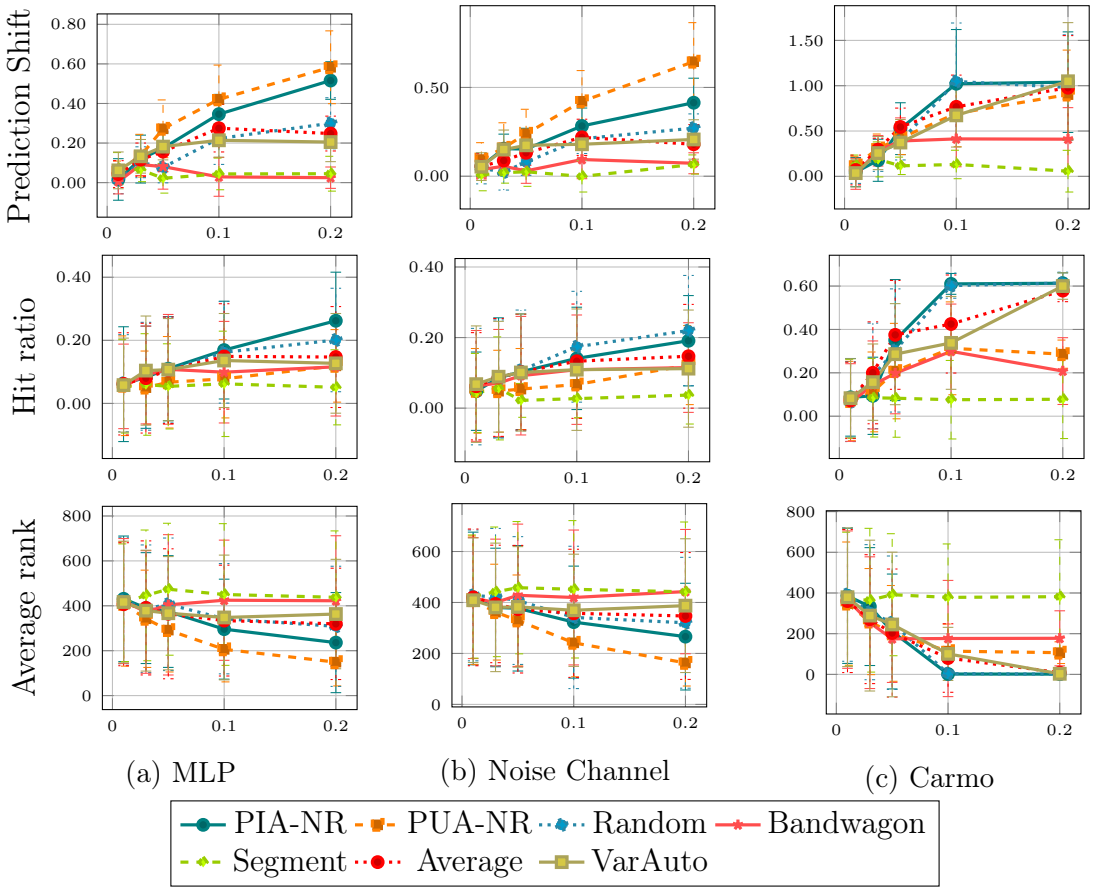


Figure A.15: Prediction shift, hit ratio, and average rank for different attack sizes comparing a regular MLP with Noise Channel and Carmo techniques using most-rated target items from MovieLens 100k data set. Note that it only shows the best result of filler size among the attack models that require it.

Yahoo! Music

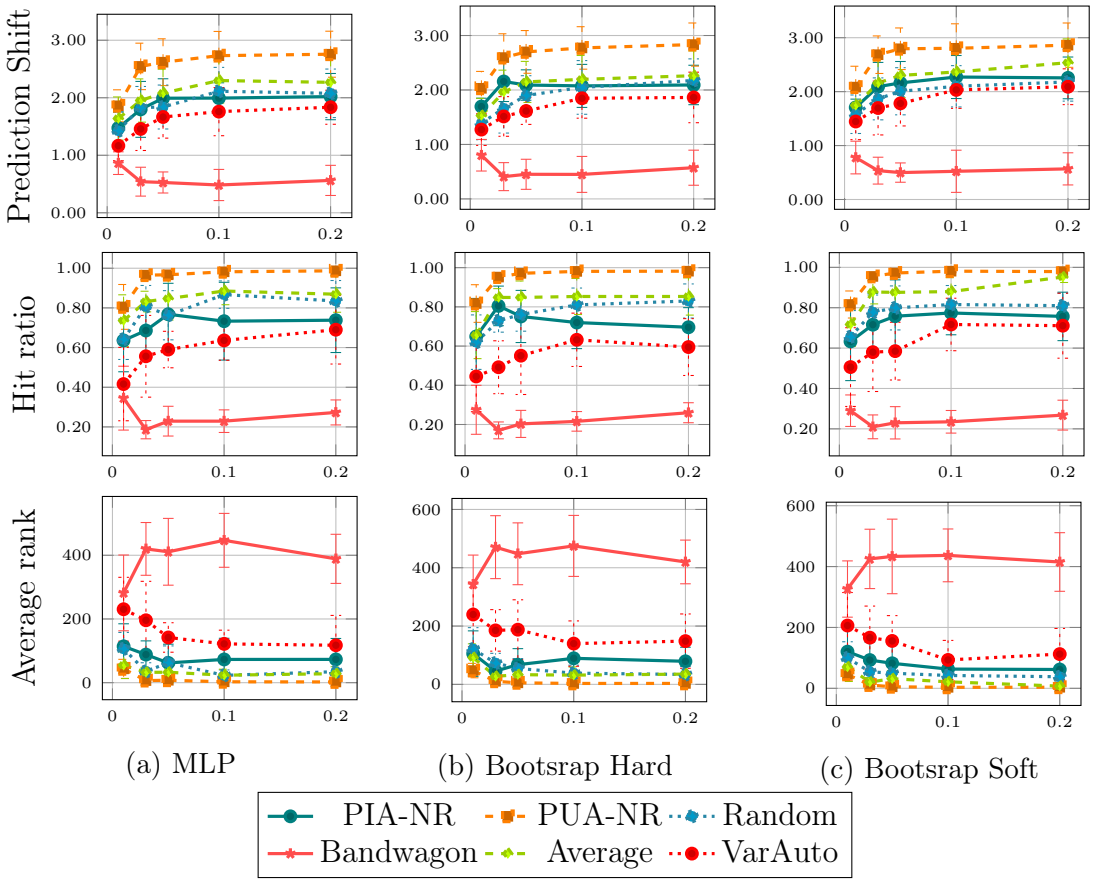


Figure A.16: Prediction shift, hit ratio, and average rank for different attack sizes comparing a regular MLP with Bootstrapping techniques using least-rated target items from Yahoo! Music data set. Note that it only shows the best result of filler size among the attack models that require it.

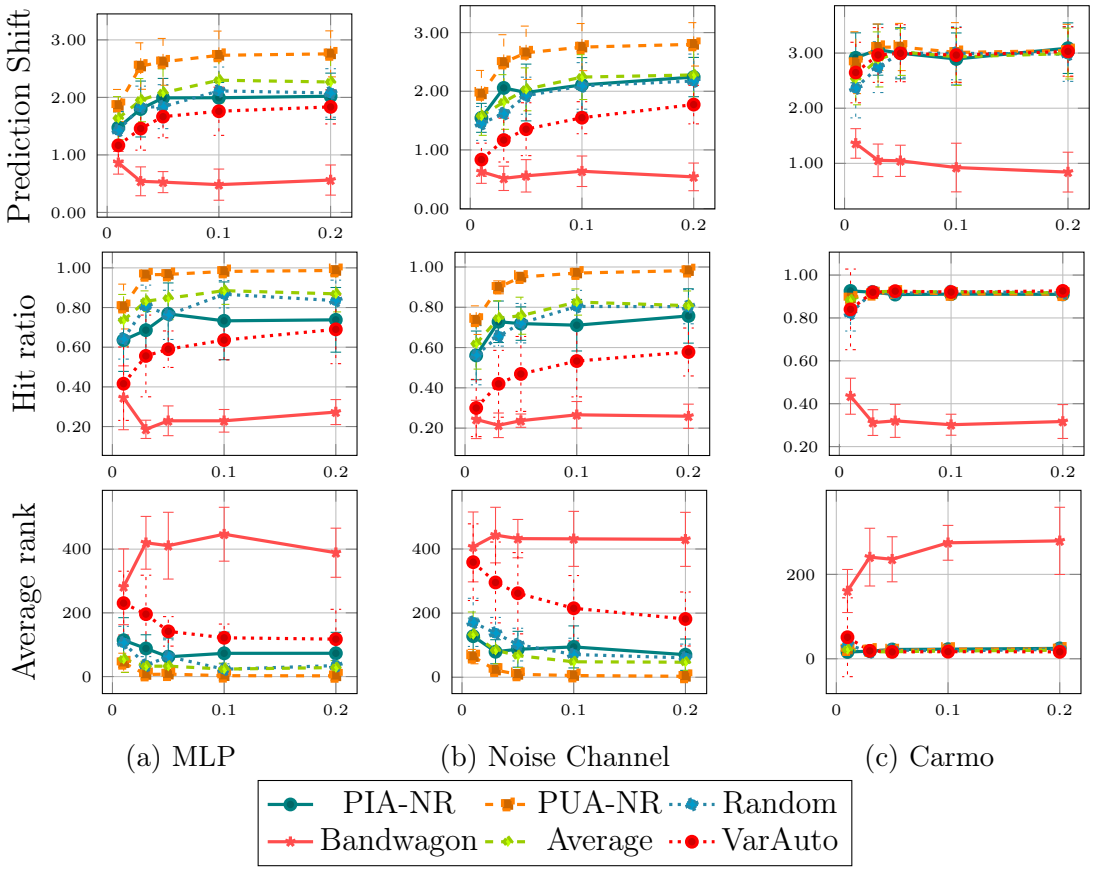


Figure A.17: Prediction shift, hit ratio, and average rank for different attack sizes comparing a regular MLP with Noise Channel and Carmo techniques using least-rated target items from Yahoo! Music data set. Note that it only shows the best result of filler size among the attack models that require it.

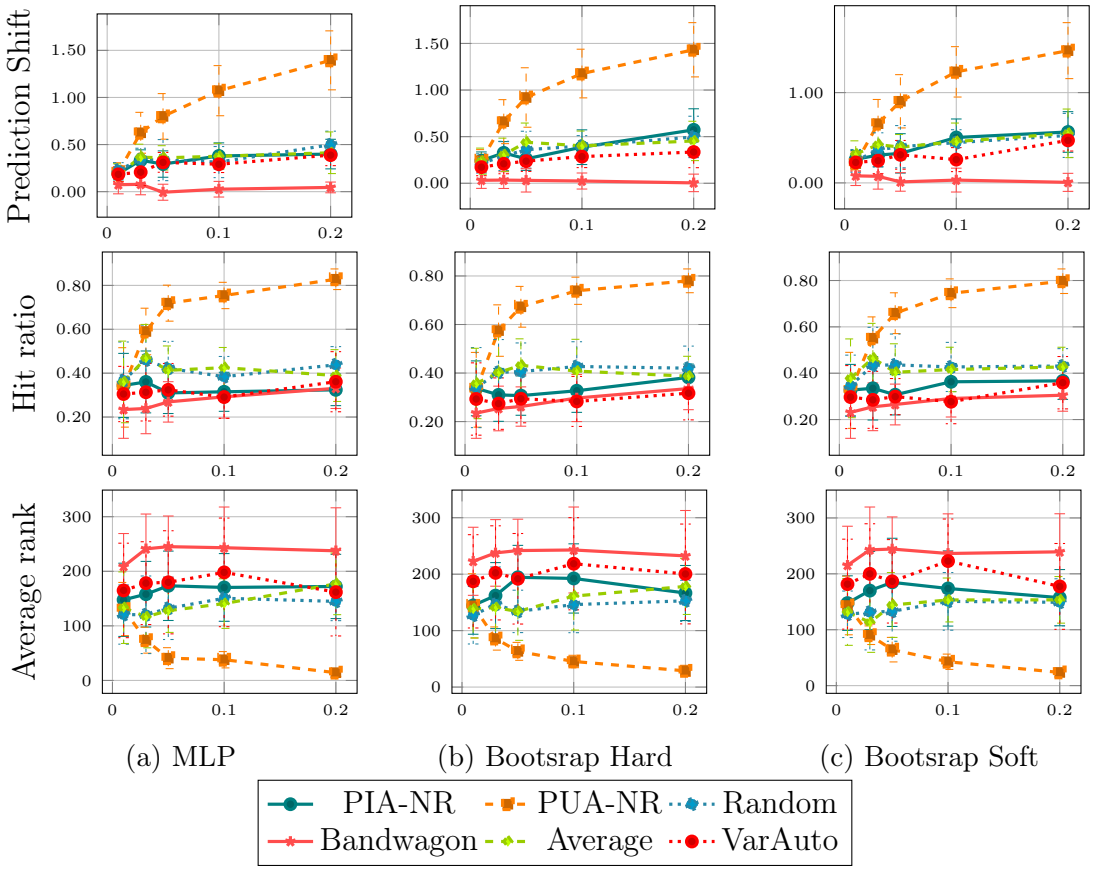


Figure A.18: Prediction shift, hit ratio, and average rank for different attack sizes comparing a regular MLP with Bootstrapping techniques using most-rated target items from Yahoo! Music data set. Note that it only shows the best result of filler size among the attack models that require it.

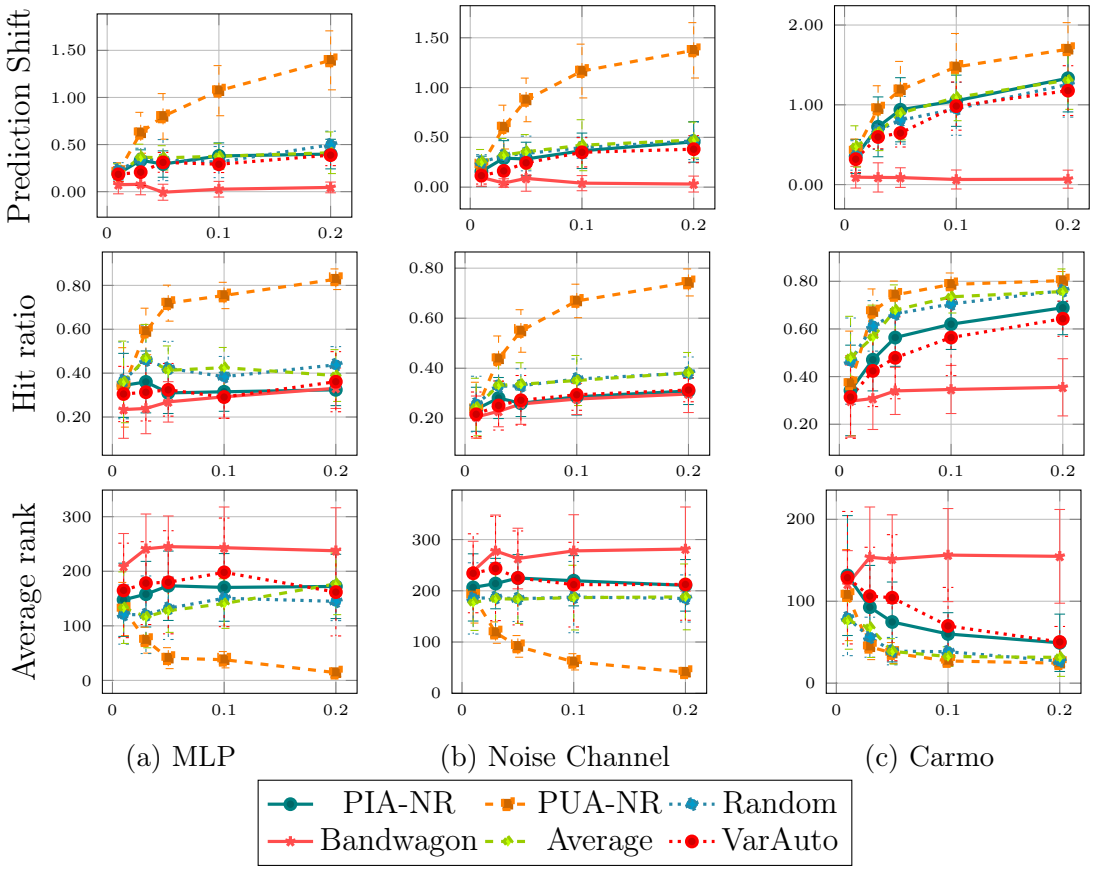


Figure A.19: Prediction shift, hit ratio, and average rank for different attack sizes comparing a regular MLP with Noise Channel and Carmo techniques using most-rated target items from Yahoo! Music data set. Note that it only shows the best result of filler size among the attack models that require it.