Adversarial Attacks on Recommendation Systems

A PROJECT REPORT
SUBMITTED IN PARTIAL FULFILMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
Master of Technology
IN
Artificial Intelligence

BY

G. Aravind

भारतीय विज्ञान संस्थान

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

June, 2022

Declaration of Originality

I, **G. Aravind**, with SR No. **18132** hereby declare that the material presented in the thesis titled

Adversarial Attacks on Recommendation Systems

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2020-2022**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Advisor Signature

DEDICATED TO

The IISc Community

Acknowledgements

I take this opportunity to express a deep sense of gratitude towards my advisor
Prof. Shirish K. Shevade, for providing excellent guidance, encouragement and inspiration throughout the project work. Without his invaluable guidance, this work would never have been possible. I would also like to thank all my classmates for their valuable suggestions and helpful discussions.

Abstract

Deep learning models have advanced the state of the art on many recommendation system tasks. There has been a surge of interest in exploring the adversarial robustness of deep learning models in other domains, some notably using hyperparameter-tuning techniques. We propose a novel meta-gradient based approach to perform poisoning attacks on recommendation systems, by treating the graph-structured data itself as a hyperparameter and tuning it to drop the recommendation model's generalization performance. We performed many experiments under a variety of settings, including the case when the attacker is not even aware of the target model's parameters or exact architecture, and explored various systematic ways of performing adversarial perturbations on recommendation datasets, such as adding new interactions between users and items, as well as adding new users onto the system. These results can serve as a motivating point for more research into strategies for robustness and defense against such gradient-based attacks on recommendations.

Contents

Chapter 1

Introduction

Recommendation systems attempt to model the user-item interactions in the system, which can be seen as a bipartite graph of users and items, and edges representing these interactions. Recommendation systems use collaborative information to learn meaningful embeddings for representing the users and items in the system, which can later be used for predicting if a user is likely to interact with an item. But, like machine learning models in other domains, recommendation systems can also be susceptible to targeted/untargeted attacks on the model itself. Thus, it is important to understand how recommendation systems could be attacked through adversarial techniques.

An adversarial attack against a recommendation system can be of the form of injecting fake user-item iteractions into the system, or injecting new fake users altogether. We study attacks of both types - fake user-item edges and fake users. Adversarial attacks on recommenders could be either *targeted* towards particular users and items, or *global*, i.e., with the aim of reducing the overall generalization performance of the recommendation model. Global attacks on recommendation systems have been the primary focus of this research.

There have been studies on adversarial attacks in other domains which use graph-structured data, such as attacks on graph neural networks based node-classification systems [1]. We present a novel approach, where we treat the underlying user-item interaction data in a recommendation system as a hyperparameter, and tune it using gradients w.r.t the graph structure, referred to as *meta-gradients*, in order to drop the generalization performance of the recommendation system. We show that even under restrictions such as attacker not having access to the target model's parameters, our attacks are able to significantly impact the performance of the recommendation system.

Chapter 2

Related Work

In our context, adversarial attacks can be either *poisoning* attacks or *evasion* attacks. In poisoning, attacks are performed by injecting false data points into the training data with the aim of degrading the performance of the corresponding ML model, whereas evasion attacks do not interfere with training data and inject malicious samples during inference.

Attacks can also be either *targeted* or *global* - targeted attacks are attacks which target performance degradation for specific users and items, whereas global attacks aim to reduce the overall generalization performance of the model.

There have been some studies on adversarial attacks specific to recommendation systems. However, this research work is primarily inspired by the application of meta-gradients in other domains such as graph neural networks [1].

In [1], the authors have proposed a novel approach to perform global, poisoning attacks on GNN-based node-classification tasks. They have shown that the use of meta-gradients w.r.t the entries of the adjacency matrix used in training the GNN classifier model can be used to find which edges of the graph have maximum impact on classification performance, upon perturbation. They have focused on attacking a 2-layer Graph Convolution Layer (GCN) classifier [2], and their meta-gradient attacks have been shown to be effective on small network-based datasets such as the CiteSeer [3] and Cora-ML [4] datasets. One key difference is that they have addressed a node-classification problem whereas our focus is on addressing an edge-prediction problem.

When it comes to recommendation systems, the most common types of attacks are the fast gradient sign attack (FGSM) [5] and Carlini and Wagner (C&W) attacks [6], which belong to l_∞ and l_2 norm attack types respectively. There are many recent works which make use of these two attack types.

In particular, adversarial personalized ranking (APR) [7] was one of the first works which made use of FGSM-based attacks on collaborative filtering models, which examines the robustness of Bayesian Personalized Ranking (BPR) [8] to adversarial perturbation on embedding vectors of users and items. Anelli et al. [9] studied the impact of iterative perturbations based on FGSM attacks, and demonstrated the inefficacy of APR [7] in protecting the recommendation system from multi-step attacks, compared to single-step FGSM attacks.

Another problem in adversarial learning is *unnoticeability*, where the attacker has to ensure that adversarial examples are not too easily distinguished by the target model, which would make it easy for the target to defend against such attacks. Some algorithms such as [1] impose constraints on the attacker about what kinds of adversarial examples are allowed, to ensure that adversarial datapoints are not too different from original datapoints, whereas other works such as [10] make use of GANs [11] to generate adversarial examples which have similar distribution as the original examples.

In this project, we have explored the use of the concept of meta-learning, which is the task of optimizing the learning algorithm itself, as explained in [12] and [13]. Bengio has demonstrated in [14] how gradients w.r.t hyperparameters can be used to optimize the learing algorithm itself, and showed its effectiveness in linear models. Later, Zügner et al. [1] demonstrated how this concept of hyperparameter gradients could be used to generate adversarial samples, and not just improve the model. Our aim in this project has been to use this idea of meta-gradients of hyperparameters, to generate adversarial examples for recommendation systems, and to explore the effectiveness of such attacks.

Chapter 3

Problem Formulation

We consider the task of collaborative-filtering based recomendations, which we have formulated as an edge-prediction task between users and items. And our objective is to reduce the global generalization performance of the collaborative-filtering based edge-prediction model, by suitably modifying the training data, which is represented in the form of a graph.

3.1 Key Idea

The key ideas in our proposed method are:

- The attacker's goal is to reduce the quality of recommendations, by misclassifying less-relevant items as relevant to the user.
- Our algorithm is for global attacks on the recommendation system, and not for targeted attacks. We need to modify/poison the bipartite training data in a way that the performance of the target model *after* training will deteriorate.
- We have used the idea of meta-learning to optimize the graph structure suitably, in order to decrease the system's performance.

3.2 Problem Definition

We are given a user-item interaction matrix. From this matrix, we can construct a bipartite graph $G(V, E)$ with V consisting of users and items as vertices, and E denoting the interactions between them. Let us assume that E is split into two subsets, namely, train set E_{tr} and test set E_{te}, and the recommendation model M, trained on graph $G(V, E_{tr})$ gives performance P on the recomendation task. Here, the recomendation task is predicting missing links between users and items with probabilities which can be used to rank and recommend items to each user. This performance is evaluated in metric μ on $G(V, E_{te})$.

Our aim in this project is as follows. We want to take the graph $G(V, E_{tr})$ and modify it to $G(V', E'_{tr})$ where $V' = V \cup V_{new}$ and $E'_{tr} = E_{tr} \cup E_{new}$ under some constraint such that when the same model M is trained on the graph $G(V', E'_{tr})$ on the same recommendation task, we get performance P' in μ and $(P - P') > s$ which is significant. Here, V_{new} denotes newly added users and E_{new} denotes newly added edges between users and items.

Moreover, we assume that we do not know the underlying recommendation model M, so we need to take a proxy model M', also referred to as a *surrogate*, that generalizes other recomendation models.

3.3 Attacker's Capabilities

- The attacker has no knowledge of the target classification model's architecture or weights, but has complete knowledge of the bipartite graph structure G.
- Attacker can perform at most Δ perturbations on G, between users U and items I.
- Attacker should ensure that the resulting perturbed graph is also strictly bipartite.

Notations

- U = Set of users in the system
- I = Set of items or products in the system
- D = Dimension of the user and item embeddings
- $A \in \{0,1\}^{|U|\times|I|}$ = Adjacency matrix of bipartite interaction graph (where '1' indicates observed interaction between a user and an item)
- $X_U \in \mathbb{R}^{|U|\times D}$ = Matrix containing user embeddings
- $X_I \in \mathbb{R}^{|I|\times D}$ = Matrix containing item embeddings
- $G = (A, [X_U, X_I])$ = The bipartite graph, defined by adjacency A and node embeddings
- A_{train} = Entries of A available during training
- $A_{\text{test}} = A \setminus A_{\text{train}}$ = Entries of A not available during training
- θ = Parameters of a neural network model
- $\theta*$ = Optimal parameters for a given model and data
- θ_t = Parameters θ after performing t gradient-based updates on initial θ
- f_θ = Parameterized function which maps each $(u, i) \in U \times I$ to $\{0, 1\}$
- y_{ui} = Interaction between user $u \in U$ and item $i \in I$, equal to $A(u, i)$
- $\hat{y}_{ui}$ = Interaction between user u and item i as predicted by the model f_θ
- $\mathcal{L}_{\text{train}}$ = Training loss of a recommendation model
- $\mathcal{L}_{\text{atk}}$ = Attacker's loss function
- Δ = Maximum perturbations in bipartite graph G
- $\hat{G}$ = Graph G with some perturbations
- $G^{(k)}$ = Graph G with k perturbations
- $\Phi(G)$ = Set of admissible perturbed versions of G

3.4 The Collaborative Filtering System

We primarily experiment with Neural Collaborative Filtering (NCF) [15] based architectures for our target model. The NCF prediction model is as follows:

$$\hat{y}_{ui} = f_\theta(X_U^T v_u, X_I^T v_i)$$

where v_u and v_i are one-hot representations of user $u \in U$ and item $i \in I$ respectively. Also, f_θ in this case is a multi-layer feedforward neural network, and the user/item embeddings X_U and X_I are also learnable. Let $\mathcal{A}^+$ denote the set of observed interactions in A_{train}, and $\mathcal{A}^-$ denote the set of sampled unobserved interactions (all possible user-item interactions not present in the dataset, also referred to as negative samples), present in the training set A_{train}. The training loss $\mathcal{L}_{\text{train}}$ for this NCF model is defined as:

$$\mathcal{L}_{\text{train}}(f_\theta(G)) := -\sum_{(u,i)\in\mathcal{A}^+\cup\mathcal{A}^-} y_{ui}\log\hat{y}_{ui} + (1-y_{ui})\log(1-\hat{y}_{ui})$$

3.5 Attacker's Objective Function

The goal of the attacker can be mathematically formulated as follows:

$$\min_{\hat{G}\in\Phi(G)} \mathcal{L}_{\text{atk}}(f_{\theta^*}(\hat{G})) \quad \text{s.t.} \quad \theta^* = \operatorname*{argmin}_\theta \mathcal{L}_{\text{train}}(f_\theta(\hat{G}))$$

Since the goal of the attacker is to reduce the performance of the target system, we can simply optimize for $\mathcal{L}_{\text{atk}} = -\mathcal{L}_{\text{train}}$ to increase the training loss of the target model.

3.6 Computing Meta-Gradients for Poisoning Attacks

The above equation for $\mathcal{L}_{\text{train}}$ can be re-written using the user-item adjacency matrix A as:

$$\mathcal{L}_{\text{train}}(f_\theta(G)) := -\sum_{(u,i)\in\mathcal{A}^+\cup\mathcal{A}^-} A(u,i)\log\hat{y}_{ui} + (1-A(u,i))\log(1-\hat{y}_{ui}) \tag{3.1}$$

We can write the above equation because $y_{ui} = A(u,i)$. So it would be possible to compute the gradients of $\mathcal{L}_{\text{train}}(f_\theta(G))$ w.r.t A, at least for those entries of A which were sampled to compute $\mathcal{L}_{\text{train}}$. The idea is to use the interaction matrix A of the bipartite graph G as hyperparameters and compute gradients of $\mathcal{L}_{\text{atk}}$ w.r.t the graph structure (∇_G^{meta}) and use it to perturb the adjacency/interaction matrix accordingly. Thus ∇_G^{meta} is defined as follows.

$$\nabla_G^{\text{meta}} := \nabla_A \mathcal{L}_{\text{atk}}(f_{\theta^*}(G)) \quad \text{where} \quad \theta^* = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}_{\text{train}}(f_\theta(G))$$

Since the attacker does not have access to the target model's parameters according to our original problem statement, the attacker can use a surrogate NCF model $f_{\theta'}$ and perform T gradient-based updates on its parameters θ' to get the attacker's own loss, $\mathcal{L}_{\text{atk}}(f_{\theta'_T}(G))$. Then we use automatic differentiation to compute gradients of $\mathcal{L}_{\text{atk}}(f_{\theta'_T}(G))$ w.r.t A to get attacker's meta-gradients, ∇_A^{meta}. We use ∇_A^{meta} and ∇_G^{meta} interchangeably, since the graph G is defined by its adjacency A.

3.7 Perturbing Graph using Meta-Gradients

To find and perturb the edge corresponding to the largest impact on model performance, we can search for the user-item edge which is not present in the bipartite graph dataset, and has the largest meta-gradient value in ∇_A^{meta}.

$$e_{\text{best}} = \underset{e=(u,i)}{\operatorname{argmax}}\, \nabla_{A(u,i)}^{\text{meta}} \quad \forall\ (u,i) \in A_{\text{train}} \quad \text{s.t.} \quad A_{\text{train}}(u,i) = 0$$

We perform one such meta-update at a time, and again compute θ_T before performing the next meta-update. After performing Δ such meta-updates, we obtain the final perturbed graph $G^{(\Delta)}$.

3.8 Meta-Attack and Baseline Algorithms

Initially, we consider the setting where the attacker and target models have the same architecture and parameter initialisations. To check whether the drop in performance using meta-attack is significant, we compare it with a reasonable baseline. We have considered random perturbations of negative-edges in the original dataset, as the baseline algorithm. The algorithms for our proposed meta-attack method and baseline method are given in Algorithm 1 and Algorithm 2 respectively.

Algorithm 1: Meta-Attack for Recommendation Systems

```
1  $T \leftarrow$ Number of training iterations for each perturbation
2  $f_{\theta^S} \leftarrow$ Surrogate NCF model (attacker)
3  $P^S \leftarrow$ Performance of surrogate model (same parameters as target model, in this case)
4  $\eta \leftarrow$ Stepsize for gradient descent ($\eta > 0$)
   /* For each perturbation, do the following */
5  for $j \leftarrow 1$ to $\Delta$ do
       /* Train surrogate model */
6      $\theta_0^S \leftarrow$ Weights of surrogate model initialized
7      for $t \leftarrow 0$ to $T-1$ do
8          $\hat{y} \leftarrow f_{\theta_t^S}(U, I)$
9          $\theta_{t+1}^S \leftarrow \theta_t^S - \eta \nabla_{\theta_t^S}$ LogLoss $(\hat{y}, \mathcal{A}^+ \cup \mathcal{A}^-)$
10     end
       /* Calculate performance of surrogate model */
11     $\hat{y} \leftarrow f_{\theta_T^S}(U, I)$
12     $P_j^S \leftarrow$ getAccuracy $(\hat{y}, \mathcal{A}^+ \cup \mathcal{A}^-)$
       /* Compute meta-gradient and perturb the data */
13     $\nabla_{\mathcal{A}^-}^{\text{meta}} \leftarrow \nabla_{\mathcal{A}^-}$ LogLoss $(\hat{y}, \mathcal{A}^+ \cup \mathcal{A}^-)$
14     $e' \leftarrow \text{argmax}_{(u,i)} \nabla_{\mathcal{A}^-}^{\text{meta}}(u, i)$
15     $\mathcal{A}^+ \leftarrow$ Insert edge $e'$ into $\mathcal{A}^+$
16     $\mathcal{A}^- \leftarrow$ Delete edge $e'$ from $\mathcal{A}^-$
17 end
```

Algorithm 2: Baseline-Attack for Recommendation Systems

```
   /* For each perturbation, do the following */
1  for $j \leftarrow 1$ to $\Delta$ do
       /* Train surrogate model */
2      $\theta_0^S \leftarrow$ Weights of surrogate model initialized
3      for $t \leftarrow 0$ to $T-1$ do
4          $\hat{y} \leftarrow f_{\theta_t^S}(U, I)$
5          $\theta_{t+1}^S \leftarrow \theta_t^S - \eta \nabla_{\theta_t^S}$ LogLoss $(\hat{y}, \mathcal{A}^+ \cup \mathcal{A}^-)$
6      end
       /* Calculate performance of surrogate model */
7      $\hat{y} \leftarrow f_{\theta_T^S}(U, I)$
8      $P_j^S \leftarrow$ getAccuracy $(\hat{y}, \mathcal{A}^+ \cup \mathcal{A}^-)$
       /* Pick a negative edge randomly */
9      $e' \leftarrow (u, i) \sim \mathcal{A}^-$
       /* Perturb the data */
10     $\mathcal{A}^+ \leftarrow$ Insert edge $e'$ into $\mathcal{A}^+$
11     $\mathcal{A}^- \leftarrow$ Delete edge $e'$ from $\mathcal{A}^-$
12 end
```

3.8.1 Complexity Analysis

The asymptotic time and space complexities for our proposed Meta-Attack algorithm are $O(n^2mT)$ and $O(nmT)$ respectively, where m denotes the number of parameters in the attacker model, n denotes the number of samples in training dataset, Δ is the number of perturbations (we consider $\Delta \propto n$) and T is the number of training iterations for the attacker model.

3.9 Approximate Meta-Attack Algorithm

While optimizing the model parameters θ, since we are performing gradient-based updates, θ_{t+1} naturally depends on θ_t. Hence, in order to compute ∇_A^{meta}, the gradients of the θ parameters all the way upto θ_0 have to be stored in the computational graph, which drastically increases the space complexity of the algorithm. Inspired by [1] about meta-attacks on GCNs, we have also experimented with a heuristic of the meta-gradient, where we assume θ_{t+1} to be independent of θ_t, thus simplifying the calculation of the meta-gradients. This can be achieved in deep-learning frameworks like PyTorch by *detaching* θ_{t+1} from θ_t, to get $\hat{\theta}_{t+1}$. ∇_A^{meta} can be computed as:

$$\nabla_A^{\text{meta}} \approx \sum_{t=1}^{T} \nabla_A \mathcal{L}_{\text{train}}(f_{\hat{\theta}_t}(G))$$

This is our *Meta-Approx* algorithm for recommendation systems. Approximating ∇_A^{meta} this way takes more time because we need to backpropagate gradients for every $t \in \{1, \cdots, T\}$, but requires lesser memory in the GPU as compared to computing the exact meta-gradients. The algorithm for Meta-Approx is given in Algorithm 3.

3.10 Surrogate Meta-Attack for Recommendation Systems

For our perturbations to be of any real use, the perturbed graph must be able to reduce the performance of any target model, even if we don't know the target model's specific architecture or parameters.

Hence, we have also implemented the *surrogate* model approach where we perform perturbations on the dataset using the surrogate model and evaluate its effectivenes on a different target model, on both the training data and unseen data. The surrogate model may have different architectures and/or parameter initialisations compared to the target model. For clarity, the *Surrogate Meta-Attack* algorithm for recommendation systems is given in Algorithm 4.

Algorithm 3: Meta-Approx Algorithm for Recommendation Systems

```
   /* For each perturbation, do the following */
1  for $j \leftarrow 1$ to $\Delta$ do
2      $\nabla^{\text{meta}}_{\mathcal{A}^-} \leftarrow 0$
3      $\theta^S_0 \leftarrow$ Weights of surrogate model initialized
       /* Train surrogate model */
4      for $t \leftarrow 0$ to $T - 1$ do
5          $\hat{y} \leftarrow f_{\theta^S_t}(U, I)$
6          $\theta^S_{t+1} \leftarrow \theta^S_t - \eta \nabla_{\theta^S_t}$ LogLoss $(\hat{y}, \mathcal{A}^+ \cup \mathcal{A}^-)$
           /* Approximate $\nabla^{\text{meta}}_{\mathcal{A}^-}$ as sum of meta-gradients per iteration */
7          $\nabla^{\text{meta}}_{\mathcal{A}^-} \leftarrow \nabla^{\text{meta}}_{\mathcal{A}^-} + \nabla_{\mathcal{A}^-}$ LogLoss $(\hat{y}, \mathcal{A}^+ \cup \mathcal{A}^-)$
           /* Parameters $\theta^S_{t+1}$ are detached to stop chaining of gradients */
8          $\theta^S_{t+1} \leftarrow$ stopGradient$(\theta^S_{t+1})$
9      end
       /* Calculate performance of surrogate model */
10     $\hat{y} \leftarrow f_{\theta^S_T}(U, I)$
11     $P^S_j \leftarrow$ getAccuracy $(\hat{y}, \mathcal{A}^+ \cup \mathcal{A}^-)$
       /* Perturb data using approximated meta-gradient */
12     $e' \leftarrow \text{argmax}_{(u,i)}\, \nabla^{\text{meta}}_{\mathcal{A}^-}(u, i)$
13     $\mathcal{A}^+ \leftarrow$ Insert edge $e'$ into $\mathcal{A}^+$
14     $\mathcal{A}^- \leftarrow$ Delete edge $e'$ from $\mathcal{A}^-$
15 end
```

3.11 Attacking Recommendation Systems using New Users

We also studied the problem of how we can reduce the performance of a recommendation system by injecting a few malicious users into the set of original users, using our meta-attack algorithm. This *New-Users-Attack* algorithm inserts some number of new users (who have not interacted with any item yet) and perturbs the edges corresponding to only the new users, one-by-one, according to the meta-gradient values. For clarity, the algorithm is given in Algorithm 5.

Algorithm 4: Surrogate Meta-Attack for Recommendation Systems

```
   /* Notations */
1  $f_{\theta^S}, f_{\theta^M} \leftarrow$ Surrogate and Target NCF models
2  $P^S, P^M \leftarrow$ Performance of surrogate and target models
   /* For each perturbation, do the following */
3  for $j \leftarrow 1$ to $\Delta$ do
       /* Train surrogate model */
4      $\theta_0^S \leftarrow$ Weights of surrogate model initialized
5      for $t \leftarrow 0$ to $T-1$ do
6          $\hat{y} \leftarrow f_{\theta_t^S}(U, I)$
7          $\theta_{t+1}^S \leftarrow \theta_t^S - \eta \nabla_{\theta_t^S}$ LogLoss $(\hat{y}, \mathcal{A}^+ \cup \mathcal{A}^-)$
8      end
       /* Calculate performance of surrogate model */
9      $\hat{y} \leftarrow f_{\theta_T^S}(U, I)$
10     $P_j^S \leftarrow$ getAccuracy $(\hat{y}, \mathcal{A}^+ \cup \mathcal{A}^-)$
       /* Compute meta-gradient and perturb the data */
11     $\nabla_{\mathcal{A}^-}^{\text{meta}} \leftarrow \nabla_{\mathcal{A}^-}$ LogLoss $(\hat{y}, \mathcal{A}^+ \cup \mathcal{A}^-)$
12     $e' \leftarrow \text{argmax}_{(u,i)} \nabla_{\mathcal{A}^-}^{\text{meta}}(u, i)$
13     $\mathcal{A}^+ \leftarrow$ Insert edge $e'$ into $\mathcal{A}^+$
14     $\mathcal{A}^- \leftarrow$ Delete edge $e'$ from $\mathcal{A}^-$
       /* Train target model */
15     $\theta_0^M \leftarrow$ Weights of target model initialized
16     for $t \leftarrow 0$ to $T-1$ do
17         $\hat{y} \leftarrow f_{\theta_t^M}(U, I)$
18         $\theta_{t+1}^M \leftarrow \theta_t^M - \eta \nabla_{\theta_t^M}$ LogLoss $(\hat{y}, \mathcal{A}^+ \cup \mathcal{A}^-)$
19     end
       /* Calculate performance of target model */
20     $\hat{y} \leftarrow f_{\theta_T^M}(U, I)$
21     $P_j^M \leftarrow$ getAccuracy $(\hat{y}, \mathcal{A}^+ \cup \mathcal{A}^-)$
22 end
```

Notations used in New Users Algorithm

- N_U, N_I = Number of original users and items respectively
- N_{new} = Number of new users used for attacking the model
- $U = \{0, 1, ..., N_U - 1\}$ = Set of original users
- $I = \{0, 1, ..., N_I - 1\}$ = Set of items
- $\{0, 1\}$ = Set of binary labels, which indicate whether an edge between a user and an item is present or not
- $D \subseteq U \times I \times \{0, 1\}$ = Original unperturbed dataset, consisting of both positive interactions and negative interactions (through negative sampling).
  Thus, D consists of triplets (u, i, y) where $u \in U$, $i \in I$, and $y \in \{0, 1\}$.
- D_{train} and D_{unseen} = Original training and validation datasets, respectively.
  Thus $D_{\text{train}} \cup D_{\text{unseen}} = D$ and $D_{\text{train}} \cap D_{\text{unseen}} = \phi$
- $U_{\text{new}} = \{N_U, N_U + 1, ..., N_U + N_{\text{new}} - 1\}$ = Set of N_{new} new users
- $D_{\text{new}} = \{(u, i, y) : u \in U_{\text{new}}, i \in I, y \in \{0\}\}$ = Set of all possible edges corresponding to the N_{new} new users (initially all edges are negative)
- $D'_{\text{train}} = D_{\text{train}} \cup D_{\text{new}}$ = Training data with new users
- $\mathcal{A}^+_{\text{train}}, \mathcal{A}^-_{\text{train}}$ = Positive and negative edges in D'_{train}
- $\mathcal{A}^+_{\text{unseen}}, \mathcal{A}^-_{\text{unseen}}$ = Positive and negative edges in D_{unseen}
- Δ = Maximum permissible perturbations
- T = Number of training iterations for each perturbation
- $f_{\theta^S}, f_{\theta^M}$ = Surrogate and Target models
- $P^S_{\text{train}}, P^S_{\text{unseen}}$ = Performance of surrogate model on training and validation data
- $P^M_{\text{train}}, P^M_{\text{unseen}}$ = Performance of target model on training and validation data

Algorithm 5: Surrogate Meta-Attack using New Users

```
   /* For each perturbation, do the following */
1  for $j \leftarrow 1$ to $\Delta$ do
       /* Train surrogate model */
2      $\theta_0^S \leftarrow$ Weights of surrogate model initialized
3      for $t \leftarrow 0$ to $T-1$ do
4          $\hat{y} \leftarrow f_{\theta_t^S}(D'_{\text{train}})$
5          $\theta_{t+1}^S \leftarrow \theta_t^S - \eta \nabla_{\theta_t^S}$ LogLoss $(\hat{y}, \mathcal{A}^+_{\text{train}} \cup \mathcal{A}^-_{\text{train}})$
6      end
       /* Compute performance of surrogate on training data */
7      $\hat{y} \leftarrow f_{\theta_t^S}(D'_{\text{train}})$
8      $P^S_{\text{train}} \leftarrow$ getAccuracy $(\hat{y}, \mathcal{A}^+_{\text{train}} \cup \mathcal{A}^-_{\text{train}})$
       /* Compute performance of surrogate on validation data */
9      $\hat{y} \leftarrow f_{\theta_t^S}(D_{\text{unseen}})$
10     $P^S_{\text{unseen}} \leftarrow$ getAccuracy $(\hat{y}, \mathcal{A}^+_{\text{unseen}} \cup \mathcal{A}^-_{\text{unseen}})$
       /* Compute meta-gradient and perturb the data */
11     $\hat{y} \leftarrow f_{\theta_t^S}(D'_{\text{train}})$
12     $\nabla^{\text{meta}}_{\mathcal{A}^-_{\text{train}}} \leftarrow \nabla_{\mathcal{A}^-_{\text{train}}}$ LogLoss $(\hat{y}, \mathcal{A}^+_{\text{train}} \cup \mathcal{A}^-_{\text{train}})$
13     $e' \leftarrow \text{argmax}_{(u,i)} \nabla^{\text{meta}}_{\mathcal{A}^-_{\text{train}}}(u, i), u \in U_{\text{new}}, i \in I$
14     $\mathcal{A}^+_{\text{train}} \leftarrow$ Insert edge $e'$ into $\mathcal{A}^+_{\text{train}}$
15     $\mathcal{A}^-_{\text{train}} \leftarrow$ Delete edge $e'$ from $\mathcal{A}^-_{\text{train}}$
       /* (We don't touch the unseen data during perturbations) */
       /* Train target model on attacked (perturbed) data */
16     $\theta_0^M \leftarrow$ Weights of target model initialized
17     for $t \leftarrow 0$ to $T-1$ do
18         $\hat{y} \leftarrow f_{\theta_t^M}(D'_{\text{train}})$
19         $\theta_{t+1}^M \leftarrow \theta_t^M - \eta \nabla_{\theta_t^M}$ LogLoss $(\hat{y}, \mathcal{A}^+_{\text{train}} \cup \mathcal{A}^-_{\text{train}})$
20     end
       /* Compute performance of target on training data */
21     $\hat{y} \leftarrow f_{\theta_t^M}(D'_{\text{train}})$
22     $P^M_{\text{train}} \leftarrow$ getAccuracy $(\hat{y}, \mathcal{A}^+_{\text{train}} \cup \mathcal{A}^-_{\text{train}})$
       /* Compute performance of target on validation data */
23     $\hat{y} \leftarrow f_{\theta_t^M}(D_{\text{unseen}})$
24     $P^M_{\text{unseen}} \leftarrow$ getAccuracy $(\hat{y}, \mathcal{A}^+_{\text{unseen}} \cup \mathcal{A}^-_{\text{unseen}})$
25 end
```

3.12 Attacking Recommendation Systems using Similar Users

We also studied the problem of attacking a recommendation system using *similar users*, where we inject new fake users into the training dataset such that the new users are in a way, similar to the set of original users. In order to generate similar users, we have experimented with two different methods.

1. Select some users from the existing set of users. and for those users we sample a fraction of items that they have interacted with, and add these new users into the training dataset.
2. Create clusters of users based on their trained user-embeddings from the collaborative-filtering model using K-means clustering, use the means of these clusters as the user-embedding for new similar users, and add these mean embeddings as the latent representations of these new similar users into the embedding matrix of the models, and the training dataset.

Chapter 4

Experimental Results

4.1 Adversarial Attacks on GCN-based Node Classification

Before studying adversarial attacks on edge-prediction based recommendation models, we studied the problem of attacking node classifier models. In [1], Zugner et al. have discussed a way to perform perturbations on the graph data's adjacency matrix using meta-gradients, such that performance of the node classification model can be reduced.

We re-implemented the Meta-Train algorithm presented in [1], and tested its effectiveness on the Cora dataset [4]. In this dataset, it is a 7-class classification problem. The following results are obtained when the number of inner training iterations, T, is set to 100.

Perturbations	Paper's results	Reproduced results
0%	0.834	0.845
2.5%	-	0.829
5%	0.788	0.814
10%	0.719	0.792

Table 4.1: Attacks on node classification task

- The paper [1] did not provide results for 2.5% perturbations case, hence the blank space in the results. We could not perform tests with more number of perturbations like 20% due to CUDA memory constraints.

- Unlike the author's implementation of [1], we have not implemented the *unnoticeability constraints* (graph connectivity constraint and log-likelihood constraint for degree distribution) for node classification, since it is not used in our recommendations setting.

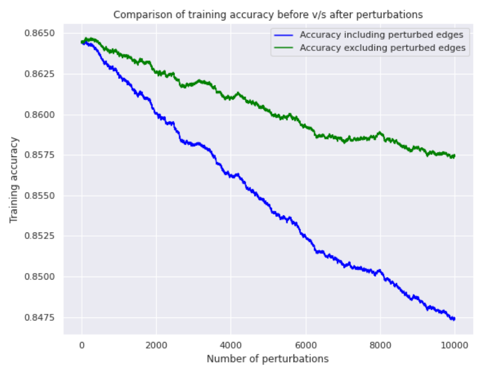
4.2 Meta-Attack vs Baseline: Comparison and Results

As explained in the problem formulation section, we extended the idea of meta-gradient based attacks presented in [1] to the edge-prediction task, but initially without using separate surrogate and target models. For the target model, we used the Generalized Matrix Factorization (GMF) architecture presented in [15], with 2 hidden layers.

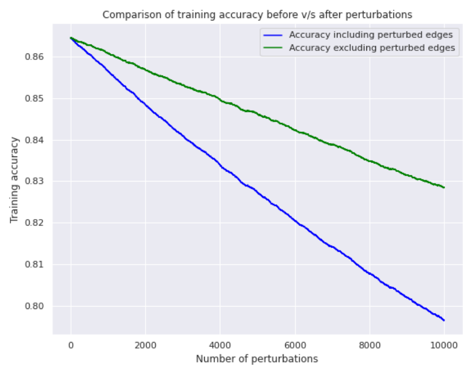
We tested this algorithm on multiple datasets – Movielens dataset [16], Steam Games dataset [17], Netflix dataset [18] and Anime Reviews dataset [19]. We had to do random sampling and reduce the size of the Netflix and Anime datasets in order for our algorithm to run efficiently (since the time and space complexities of the meta-gradient computation is directly proportional to number of entries in the user-item adjacency matrix). More details about the datasets are provided in the Datasets section. The results are given below (N_p denotes the number of perturbations, and the percentage of '0'-edges which are perturbed, is also provided).

N_p	N_p (%)	Accuracy (%)		Loss	
		Baseline	Meta-Attack	Baseline	Meta-Attack
0	0%	86.4430	86.4430	0.367983	0.367983
2,000	2%	86.2620	85.6833	0.370208	0.371127
4,000	4%	86.1047	84.9638	0.373078	0.384024
6,000	6%	85.9229	84.2186	0.377565	0.398227
8,000	8%	85.8784	83.4875	0.380229	0.409323
10,000	**10%**	**85.7438**	**82.8459**	**0.381340**	**0.422425**

Table 4.2: Results on Movielens dataset



(a) Baseline

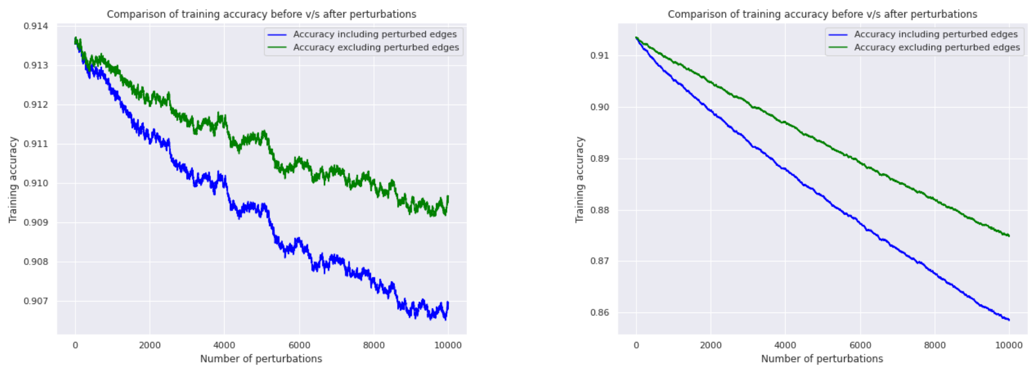


(b) Meta-Attack

Figure 4.1: Accuracy w.r.t perturbations on Movielens dataset

N_p	N_p (%)	Accuracy (%)		Loss	
		Baseline	Meta-Attack	Baseline	Meta-Attack
0	0%	91.3585	91.3585	0.306598	0.306598
2,000	2%	91.1969	90.4824	0.310836	0.308139
4,000	4%	91.1477	89.7154	0.310462	0.319409
6,000	6%	91.0573	88.9117	0.315557	0.332401
8,000	8%	91.0019	88.1994	0.317225	0.348365
10,000	**10%**	**90.9653**	**87.4894**	**0.319843**	**0.356407**

Table 4.3: Results on Steam Games dataset



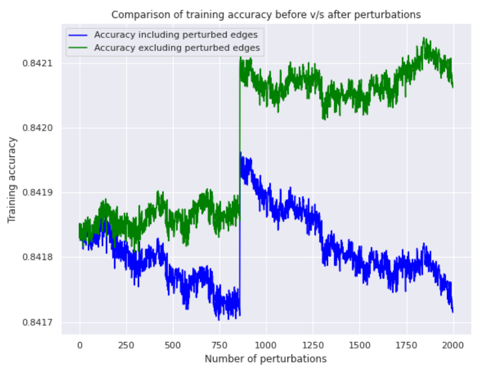
(a) Baseline

(b) Meta-Attack

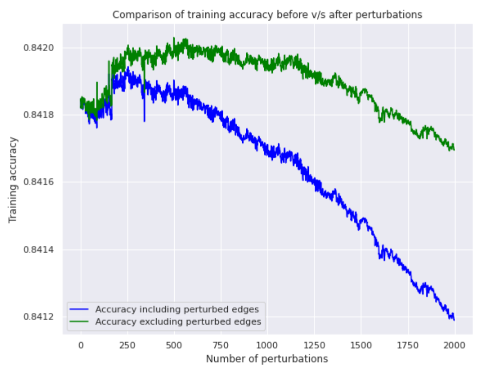
Figure 4.2: Accuracy w.r.t perturbations on Steam Games dataset

N_p	N_p (%)	Accuracy (%)		Loss	
		Baseline	Meta-Attack	Baseline	Meta-Attack
0	0%	84.1843	84.1843	0.354682	0.354682
400	0.0267%	84.1866	84.1962	0.354661	0.354642
800	0.0533%	84.1886	84.1976	0.354642	0.354780
1,200	0.08%	84.2079	84.1941	0.354420	0.355047
1,600	0.1067%	84.2063	84.1781	0.354433	0.355363
2,000	**0.1333%**	**84.2061**	**84.1695**	**0.354460**	**0.355608**

Table 4.4: Results on Netflix dataset



(a) Baseline

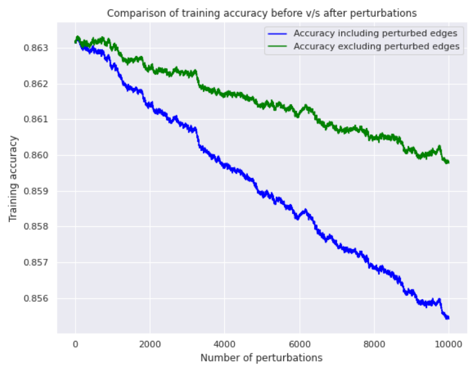


(b) Meta-Attack

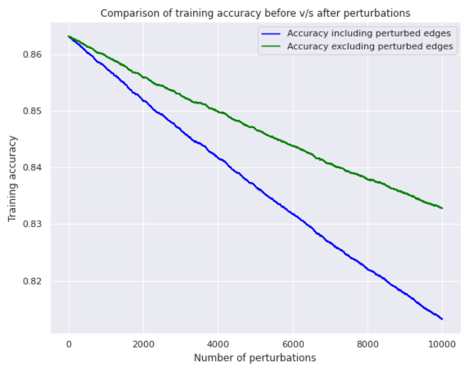
Figure 4.3: Accuracy w.r.t perturbations on Netflix dataset

N_p	N_p (%)	Accuracy (%)		Loss	
		Baseline	Meta-Attack	Baseline	Meta-Attack
0	0%	86.3191	86.3191	0.327948	0.327948
2,000	1%	86.2401	85.5905	0.329356	0.341343
4,000	2%	86.1621	84.9805	0.331362	0.353161
6,000	3%	86.1165	84.3823	0.332903	0.364643
8,000	4%	86.0485	83.7822	0.334481	0.375586
10,000	**5%**	**85.9783**	**83.2774**	**0.335841**	**0.385804**

Table 4.5: Results on Anime Reviews dataset



(a) Baseline



(b) Meta-Attack

Figure 4.4: Accuracy w.r.t perturbations on Anime Reviews dataset

The results have been encouraging on all 4 datasets - our Meta-Attack algorithm consistently performs better than the baseline in terms of drop in performance of target model. The drop in performance on the Netflix dataset might seem less impressive at first glance, but the Netflix dataset is much bigger than the other datasets used here - it contains about 1.6 million user-item interactions even after extensive pre-processing. 5% of 1.6 M interactions would still be 80 K perturbations, whereas we explored only 2K perturbations because of computational constraints.

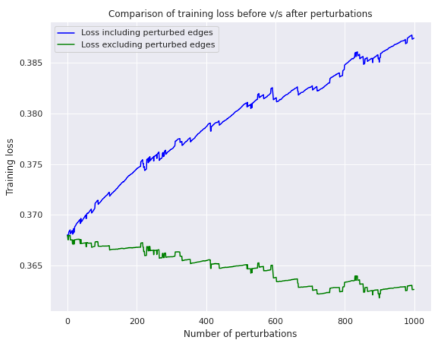
In the above figures, we have plotted both cases - both including and excluding perturbed edges when computing accuracy, to better understand whether these attacks generalize well or not. Moving forward, by default we exclude the perturbed edges when computing any metrics.

4.3 Meta-Approx Algorithm

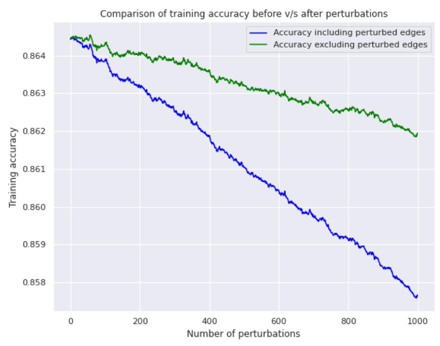
We also experimented with our Meta-Approx algorithm (Algorithm 3), which uses an approximation of the meta-gradient to reduce the memory requirements of the meta-gradient computation, by approximating the gradients at the end of each iteration of the training loop. It is a tradeoff between space-complexity and time-complexity of the first Meta-Attack algorithm.

N_p	Accuracy			Loss		
	Baseline	Meta-Attack	Meta-Approx	Baseline	Meta-Attack	Meta-Approx
0	86.4430	86.4430	86.4430	0.367983	0.367983	0.367983
200	86.4570	86.3679	86.4034	0.367199	0.365422	0.366818
400	86.4544	86.3111	86.3617	0.367556	0.365359	0.365512
600	86.3977	86.2432	86.3019	0.368961	0.365086	0.363863
800	86.3817	86.1677	86.2560	0.369355	0.366105	0.363393
1000	**86.3756**	**86.0854**	**86.1950**	**0.369761**	**0.367071**	**0.362686**

Table 4.6: Meta-Approx algorithm compared with Meta-Attack and Baseline on Movielens dataset



(a) Change in loss



(b) Change in accuracy

Figure 4.5: Accuracy w.r.t perturbations for Meta-Approx algorithm

Though the GPU memory requirements of Meta-Approx algorithm is lower than that of Meta-Attack, the time complexity is significantly higher. And in this case (for recomendation systems) the benefits of Meta-Approx algorithm does not seem to outweigh that of Meta-Attack, in terms of both complexity and performance. Hence, we decided to stick to using Meta-Attack algorithm for the recommendation systems problem, moving forward.

4.4 Surrogate Meta-Attack Algorithm

After promising results on the Meta-Approx algorithm, where meta-gradient based attacks were able to reduce the performance of the collaborative filtering model, we experimented extensively with the *Surrogate Meta-Attack* algorithm (Algorithm 4), where the surrogate model (the model performing the attacks) can have different architecture and parameter initialisations as compared to the target model (the model being attacked).

The results on unseen data are promising for some classes of recommendation systems architectures, but not all. We got good results when the surrogate and target models were based on architectures which had few number of hidden layers, such as standard collaborative filtering models and GMF model [15] with few hidden layers. However, the impact of these meta-attacks are not very effective for more complex models like the NeuMF architecture from [15].

Notably, the difference in performance between Meta-Attack and baseline algorithms depends on a lot of factors - choice of underlying attacker/target models, choice of embedding sizes, difference in weight initialisations, activation functions used inside the underlying models (use of *tanh* activation in NCF often leads to exploding meta-gradients), and even the choice of optimization algorithm used like SGD or ADAM. And we also started monitoring the AUROC metric along with accuracy for the user-item edge-prediction. The results for experiments on the Movielens dataset are given in the following tables. In this dataset, 10,000 perturbations is 10% of the total number of edges which are eligible to be perturbed.

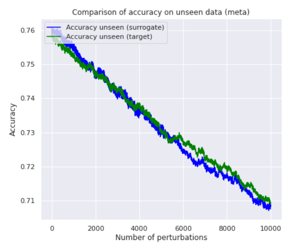
Case 1: Same surrogate and target architectures, but different weight initializations

N_p	Meta-Attack (Accuracy %)				Baseline (Accuracy %)			
	Surrogate		Target		Surrogate		Target	
	Training	Unseen	Training	Unseen	Training	Unseen	Training	Unseen
0	90.8212	76.0555	90.9615	75.8822	90.8212	76.0555	90.9615	75.8822
2000	89.9967	74.7826	90.3653	74.7331	90.5136	75.2779	90.6024	74.7505
4000	89.3020	73.6905	89.9048	73.7599	90.2522	74.3592	90.3748	74.0348
6000	88.7735	72.4548	89.4485	72.7099	89.9239	73.4974	90.1891	73.2497
8000	88.3121	71.6896	89.1603	71.9670	89.8277	72.7867	89.8073	72.4944
10000	87.7830	70.8229	88.7588	70.9318	89.5708	71.8382	89.6107	71.8654
Drop %	3.0382	**5.2326**	2.2027	**4.9503**	1.2504	**4.2173**	1.3508	**4.0167**

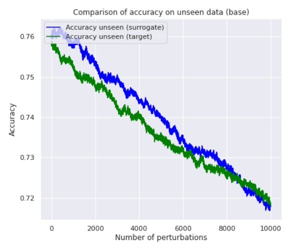
Table 4.7: Change in accuracy using Surrogate Meta-Attack in Case-1

N_p	Meta-Attack (AUROC)				Baseline (AUROC)			
	Surrogate		Target		Surrogate		Target	
	Training	Unseen	Training	Unseen	Training	Unseen	Training	Unseen
0	0.9721	0.8371	0.9728	0.8351	0.9721	0.8371	0.9728	0.8351
2000	0.9668	0.8203	0.9692	0.8193	0.9703	0.8260	0.9708	0.8233
4000	0.9623	0.8065	0.9661	0.8070	0.9687	0.8178	0.9694	0.8144
6000	0.9587	0.7938	0.9634	0.7971	0.9672	0.8085	0.9684	0.8047
8000	0.9550	0.7857	0.9610	0.7877	0.9660	0.8014	0.9666	0.7979
10000	0.9519	0.7756	0.9586	0.7778	0.9648	0.7922	0.9653	0.7905
Drop %	2.0195	**6.1499**	1.4268	**5.7276**	0.7304	**4.4873**	0.7565	**4.4593**

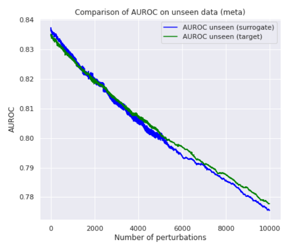
Table 4.8: Change in AUROC using Surrogate Meta-Attack in Case-1



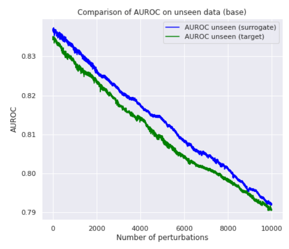
(a) Meta-Attack (accuracy)



(b) Baseline (accuracy)



(c) Meta-Attack (AUROC)



(d) Baseline (AUROC)

Figure 4.6: Accuracy and AUROC plots for the unseen data in Case-1

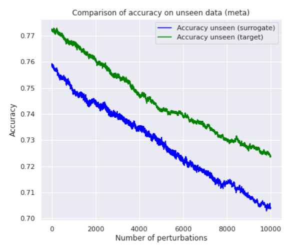
Case 2: Same overall architectures, but different embedding sizes and layers

N_p	Meta-Attack (Accuracy %)				Baseline (Accuracy %)			
	Surrogate		Target		Surrogate		Target	
	Training	Unseen	Training	Unseen	Training	Unseen	Training	Unseen
0	90.9616	75.8822	86.8856	77.2343	90.9616	75.8822	86.8856	77.2343
2000	90.1179	74.4855	86.1603	76.1398	90.6024	74.7505	86.4434	76.2165
4000	89.3908	73.4974	85.4536	74.7134	90.3749	74.0348	86.1348	75.7386
6000	88.8582	72.1552	85.0240	73.9209	90.1891	73.2498	85.8304	75.0155
8000	88.3115	71.3529	84.5165	73.0393	89.8074	72.4945	85.7152	74.1314
10000	87.8465	70.4094	84.1305	72.3880	89.6108	71.8655	85.4112	73.3216
Drop %	3.1151	**5.4728**	2.7551	**4.8463**	1.3508	**4.0167**	1.4744	**3.9127**

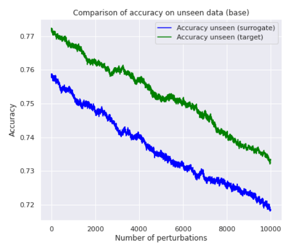
Table 4.9: Change in accuracy using Surrogate Meta-Attack in Case-2

N_p	Meta-Attack (AUROC)				Baseline (AUROC)			
	Surrogate		Target		Surrogate		Target	
	Training	Unseen	Training	Unseen	Training	Unseen	Training	Unseen
0	0.9728	0.8351	0.9458	0.8519	0.9728	0.8351	0.9458	0.8519
2000	0.9677	0.8190	0.9402	0.8366	0.9708	0.8233	0.9432	0.8419
4000	0.9635	0.8065	0.9352	0.8243	0.9694	0.8144	0.9407	0.8340
6000	0.9592	0.7927	0.9310	0.8141	0.9684	0.8047	0.9388	0.8271
8000	0.9553	0.7822	0.9275	0.8064	0.9666	0.7979	0.9372	0.8207
10000	0.9523	0.7729	0.9243	0.7978	0.9653	0.7905	0.9355	0.8134
Drop %	2.0516	**6.2226**	2.1541	**5.4076**	0.7565	**4.4593**	1.0301	**3.8507**

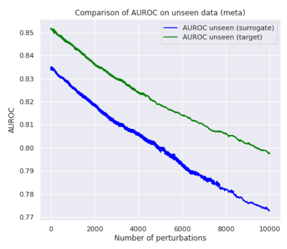
Table 4.10: Change in AUROC using Surrogate Meta-Attack in Case-2



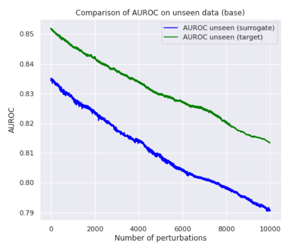
(a) Meta-Attack (accuracy)



(b) Baseline (accuracy)



(c) Meta-Attack (AUROC)



(d) Baseline (AUROC)

Figure 4.7: Accuracy and AUROC plots for the unseen data in Case-2

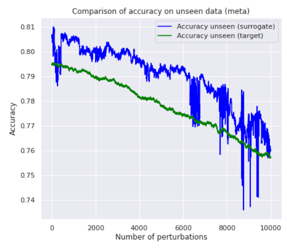
Case 3: Different architectures (Surrogate: GMF model, Target: CF model)

N_p	Meta-Attack (Accuracy %)				Baseline (Accuracy %)			
	Surrogate		Target		Surrogate		Target	
	Training	Unseen	Training	Unseen	Training	Unseen	Training	Unseen
0	85.0870	80.6790	83.1812	79.4904	85.0870	80.6790	83.1812	79.4904
2000	83.9613	79.6687	82.7546	78.9455	84.8273	80.6419	83.0855	79.0446
4000	83.6280	79.6984	82.2502	78.1878	85.0027	80.4487	83.0313	78.7648
6000	82.3661	79.2898	81.7536	77.4671	83.4729	78.9034	82.8872	78.4280
8000	81.0261	77.6999	81.0327	76.7737	84.3312	78.7301	82.7117	77.7742
10000	79.6946	76.0531	80.3302	75.6990	83.9763	78.4503	82.5147	77.3532
Drop %	5.3924	**4.6259**	2.8510	**3.7914**	1.1107	**2.2287**	0.6665	**2.1372**

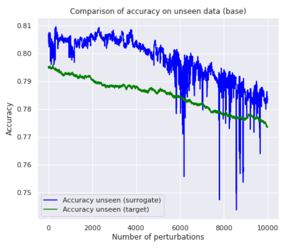
Table 4.11: Change in accuracy using Surrogate Meta-Attack in Case-3

N_p	Meta-Attack (AUROC)				Baseline (AUROC)			
	Surrogate		Target		Surrogate		Target	
	Training	Unseen	Training	Unseen	Training	Unseen	Training	Unseen
0	0.9298	0.8844	0.9135	0.8744	0.9298	0.8844	0.9135	0.8744
2000	0.9189	0.8752	0.9081	0.8684	0.9274	0.8847	0.9118	0.8696
4000	0.9151	0.8745	0.9026	0.8615	0.9276	0.8818	0.9110	0.8667
6000	0.9027	0.8701	0.8977	0.8529	0.9152	0.8673	0.9098	0.8636
8000	0.8895	0.8556	0.8924	0.8453	0.9230	0.8689	0.9084	0.8593
10000	0.8819	0.8466	0.8867	0.8392	0.9204	0.8682	0.9065	0.8551
Drop %	4.7864	**3.7781**	2.6867	**3.5136**	0.9435	**1.6265**	0.6998	**1.9253**

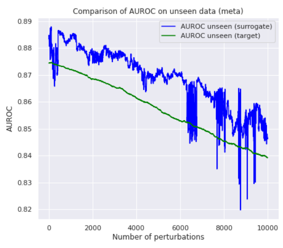
Table 4.12: Change in AUROC using Surrogate Meta-Attack in Case-3



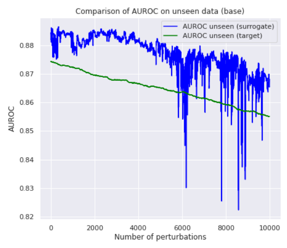
(a) Meta-Attack (accuracy)



(b) Baseline (accuracy)



(c) Meta-Attack (AUROC)



(d) Baseline (AUROC)

Figure 4.8: Accuracy and AUROC plots for the unseen data in Case-3

More cases

Drop in accuracy (%) after attacks											
N_p	Meta-Attack				Baseline				Surrogate model	Target model	Different layers
	Surrogate		Target		Surrogate		Target				
	Training	Unseen	Training	Unseen	Training	Unseen	Training	Unseen			
10K	3.0382	5.2326	2.2027	4.9503	1.2504	4.2173	1.3508	4.0167	CF	CF	No
10K	3.0382	5.2387	1.2558	4.9652	1.2504	4.2174	0.8039	4.0737	CF	CF	Yes
10K	3.1151	5.4728	2.7551	4.8463	1.3508	4.0167	1.4744	3.9127	CF	CF	Yes
10K	3.6289	4.6508	2.4157	5.4382	1.4504	3.8608	1.2053	4.2272	CF	CF	Yes
5K	2.1043	2.7736	0.8318	3.0584	0.6979	1.8127	0.5021	2.2560	CF	CF	Yes
5K	0.7823	2.0926	1.5729	2.1842	0.5309	2.0975	0.7661	1.8301	CF	CF	Yes
10K	5.3924	4.6259	2.0686	6.8473	1.1107	2.2287	0.4507	3.4372	GMF	GMF	No
10K	2.6553	2.6893	1.3354	3.3481	0.8673	2.2312	0.4507	3.4372	CF	GMF	Yes
10K	5.3924	4.6259	2.8510	3.7914	1.1107	2.2287	0.6665	2.1372	GMF	CF	Yes
10K	10.0657	10.6337	1.4868	1.0747	-0.0700	1.1516	0.0474	1.1218	NeuMF	NeuMF	No
10K	1.6037	0.7851	0.0897	-0.0941	0.4227	1.2358	-0.5823	0.8593	CF	NeuMF	Yes

Table 4.13: Change in accuracy during more Surrogate Meta-Attack experiments

Drop in AUROC (%) after attacks											
N_p	Meta-Attack				Baseline				Surrogate model	Target model	Different layers
	Surrogate		Target		Surrogate		Target				
	Training	Unseen	Training	Unseen	Training	Unseen	Training	Unseen			
10K	2.0195	6.1499	1.4268	5.7276	0.7304	4.4873	0.7565	4.4593	CF	CF	No
10K	2.0195	6.1499	0.5282	5.6011	0.7304	4.4873	0.3161	4.3205	CF	CF	Yes
10K	2.0516	6.2226	2.1541	5.4076	0.7565	4.4593	1.0301	3.8507	CF	CF	Yes
10K	3.0820	5.2641	1.4894	5.9508	1.0877	4.0100	0.7105	4.3315	CF	CF	Yes
5K	1.8615	3.1690	0.3449	3.6249	0.5928	2.0022	0.1658	2.4467	CF	CF	Yes
5K	0.3017	2.8459	1.1813	2.6899	0.2021	2.3252	0.6447	2.1424	CF	CF	Yes
10K	4.7864	3.7781	1.7574	5.9428	0.9435	1.6265	0.4657	2.9401	GMF	GMF	No
10K	2.5066	2.5019	1.3962	2.9510	0.8805	1.9424	0.4657	2.9401	CF	GMF	Yes
10K	4.7864	3.7781	2.6867	3.5136	0.9435	1.6265	0.6998	1.9253	GMF	CF	Yes
10K	2.6505	2.7195	2.4587	2.4853	0.0758	2.4304	-0.0356	1.0751	NeuMF	NeuMF	No
10K	1.7319	0.7779	1.9667	0.5256	0.3940	0.8121	0.0750	-0.2558	CF	NeuMF	Yes

Table 4.14: Change in AUROC during more Surrogate Meta-Attack experiments

Key observations

- We can observe that our Meta-Attack algorithm usually outperforms the baseline algorithm, when it comes to drop in performance of target model on unseen data.
- The above point is especially true when both surrogate and target models are pure collaborative filtering architectures, even though the surrogate and target may have different weight initializations, different embedding sizes for users and items, or different layers in general.
- Meta-Attack also visibly outperformed the baseline when both surrogate and target models were based on GMF architecture with different initializations.

- But the Meta-Attack algorithm does not perform significantly better than baseline when either surrogate or target was GMF-based and the other model was CF-based. This shows that Meta-Attack does not perform very well when the two models (surrogate and target) are based on very different architectures.
- When it comes to NeuMF architecture, the Meta-Attack algorithm struggles even when the two models are based on the same NeuMF architecture. When we analysed the accuracy and AUROC plots for experiments with the NeuMF architecture, we found that the decrease in performance is very inconsistent with the number of perturbations.
  In general, we observed that the Meta-Attack algorithm has little impact when the surrogate/target model has large number of hidden layers with non-linear activations, like *sigmoid* or *tanh*. In particular, the use of *tanh* activation in NeuMF often causes the meta-gradients to explode, which was another observation from these experiments with more complex architectures.

4.5 Attacking Recommendations using New Users

As explained in the problem formulation, we also extensively studied the problem of introducing new and similar malicous users into the training dataset, instead of just introducing new user-item edges, and then performing meta-gradient based attacks exclusively on the edges corresponding to new users. The results for these set of experiments are not very promising though, indicating that this algorithm doesn't generalize well when trained on a different set of users and tested on a different set of users.
Some of these results are given below. Here, N_p denotes number of perturbations, M_S and M_T denote surrogate and target models respectively, D denotes whether dropout was used in either of the models and N_new denotes number of new users introduced into the training dataset.

Drop in accuracy (%) after attacks												
N_p	Meta-Attack				Baseline				M_S	M_T	D	N_new
	Surrogate		Target		Surrogate		Target					
	Training	Unseen	Training	Unseen	Training	Unseen	Training	Unseen				
5K	-0.2292	-0.9608	-0.0797	-0.6909	-0.1090	-1.1143	-0.1002	-1.0178	CF	CF	Yes	50
10K	1.1427	0.0346	2.3875	-0.4309	2.2455	-0.6959	2.2281	-0.3566	CF	CF	No	20
5K	1.7982	-1.1342	0.3289	-0.1065	0.3896	-0.2452	-0.0158	0.8643	GMF	GMF	Yes	20
5K	0.2930	-0.2550	0.4936	-0.2898	0.5192	-0.4036	0.4906	0.1065	CF	GMF	No	30
5K	0.6112	0.6018	1.0199	0.2352	0.8742	0.4136	0.7820	0.2377	CF	GMF	Yes	15
10K	0.0564	-1.3150	0.3393	-0.6340	0.4141	-0.3567	0.4287	-0.2724	GMF	CF	No	100
10K	3.9582	-0.7280	2.9732	-0.1461	3.6439	-1.0178	3.9219	-0.2947	CF	CF	No	10

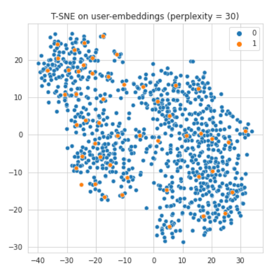
Table 4.15: Change in accuracy from perturbations of new users

Drop in AUROC (%) after attacks												
N_p	Meta-Attack				Baseline				M_S	M_T	D	N_{new}
	Surrogate		Target		Surrogate		Target					
	Training	Unseen	Training	Unseen	Training	Unseen	Training	Unseen				
5K	0.1066	-0.8208	0.2912	-0.6677	0.2128	-0.7569	0.3217	-0.9167	CF	CF	Yes	50
10K	2.1893	0.0955	2.3236	-0.2691	2.2493	-0.5884	2.2302	-0.1958	CF	CF	No	20
5K	3.9824	-0.3191	0.9461	-0.1799	1.4309	0.2285	0.5731	0.5295	GMF	GMF	Yes	20
5K	1.0018	-0.2718	0.7780	-0.4328	1.0038	-0.2830	0.6215	0.0148	CF	GMF	No	30
5K	1.3205	0.4446	1.1326	0.1069	1.2421	0.4642	0.8911	0.3999	CF	GMF	Yes	15
10K	1.0178	-1.2765	0.8204	-0.4794	0.9604	-0.1586	0.8211	-0.1628	GMF	CF	No	100
10K	2.2750	-0.2668	2.1431	0.0322	2.1928	-0.5307	2.3376	-0.0367	CF	CF	No	10

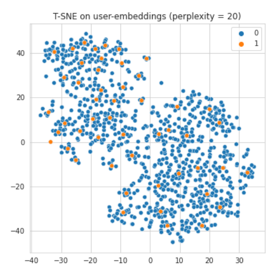
Table 4.16: Change in AUROC from perturbations of new users

4.6 Attacking Recommendations using Similar Users

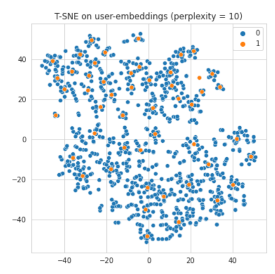
Similar to the new-users attack algorithm, we have also implemented a *similar-users* attack algorithm, where we ensure that the embeddings corresponding to the new-users are not entirely *new*, but have some similarity to the embeddings of orignial users in the training set.
Some T-SNE based visualisations of trained user-embeddings are shown in Figure 4.9. In these visualisations, it is hard to determine any noticeable clusters among users. Only when the perplexity of T-SNE is very low (which decreases the stability of the dimensionality reduction process), we can see some semblance of clusters among the user-embeddings. After applying K-means clustering, the means of these corresponding clusters are also shown.



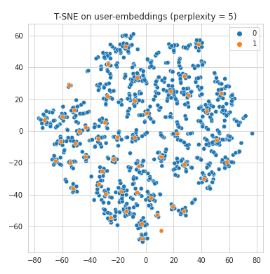
(a) Perplexity = 30



(b) Perplexity = 20



(c) Perplexity = 10



(d) Perplexity = 5

Figure 4.9: T-SNE visualizations of user-embeddings

The results for these sets of experiments are also not very promising, and are shown in tables 4.17 and 4.18. Here, C denotes whether K-means was used to generate similar user-embeddings (see section 3.12) and N_{sim} denotes number of new similar users introduced into the training set.

Drop in accuracy (%) after attacks												
N_{p}	Meta-Attack				Baseline				M_S	M_T	C	N_{sim}
	Surrogate		Target		Surrogate		Target					
	Training	Unseen	Training	Unseen	Training	Unseen	Training	Unseen				
5K	1.4781	0.0223	1.2570	-0.0942	0.5257	0.3219	0.4472	0.1386	CF	CF	No	200
10K	2.9785	0.3566	1.6605	0.4507	1.0726	0.1981	1.0913	0.3021	CF	CF	Yes	200
10K	3.0867	0.4978	1.9790	0.1287	1.0614	0.1610	1.2274	0.1882	CF	CF	No	300
10K	4.1189	0.9559	2.0839	0.5077	1.0501	1.0401	0.9340	0.6612	CF	CF	Yes	300
10K	5.4951	-0.1015	1.8668	-0.1114	1.9068	0.0792	1.2130	-0.4185	GMF	GMF	No	300
10K	2.6475	0.8841	1.4832	0.8469	1.2593	1.0178	1.2881	0.5176	CF	CF	No	300
10K	0.7582	0.1115	0.6226	0.4062	0.0652	0.2353	0.1262	0.1858	CF	CF	No	943
10K	1.3206	0.3690	1.0052	0.3293	0.3743	0.3591	0.3453	0.2724	CF	CF	No	500
10K	1.1103	-0.7330	1.0666	0.3269	0.1541	-0.1337	0.2132	-0.0049	GMF	CF	No	943
10K	1.3092	0.5498	0.6837	0.7380	0.7436	0.4210	0.7161	0.5498	CF	CF	Yes	943

Table 4.17: Change in accuracy from perturbations of new similar users

Drop in AUROC (%) after attacks												
N_{p}	Meta-Attack				Baseline				M_S	M_T	C	N_{sim}
	Surrogate		Target		Surrogate		Target					
	Training	Unseen	Training	Unseen	Training	Unseen	Training	Unseen				
5K	1.4581	-0.0183	1.2115	-0.1156	0.5971	0.1712	0.5622	0.1029	CF	CF	No	200
10K	1.8421	0.0312	1.0705	0.2030	0.8230	0.0742	0.7150	-0.1098	CF	CF	Yes	200
10K	2.4938	0.5217	1.4321	0.3471	0.9602	0.4188	1.0141	0.2469	CF	CF	No	300
10K	3.1366	1.0881	1.8048	0.4742	0.9281	1.1665	0.9250	0.5174	CF	CF	Yes	300
10K	3.6560	-0.3716	1.2098	-0.4885	1.1447	-0.3474	0.8196	-0.7257	GMF	GMF	No	300
10K	1.4129	0.8980	0.7281	0.8300	0.6224	1.1437	0.6223	0.3045	CF	CF	No	300
10K	0.9508	0.0323	0.8039	0.2950	0.1767	0.1552	0.2334	0.1872	CF	CF	No	943
10K	1.5708	0.2897	1.1906	0.1895	0.5461	0.3127	0.5391	0.1499	CF	CF	No	500
10K	1.3437	-0.6487	1.2985	0.4234	0.3076	-0.1833	0.3695	0.2008	GMF	CF	No	943
10K	0.5829	0.6238	0.3188	0.7314	0.2984	0.4967	0.2920	0.4680	CF	CF	Yes	943

Table 4.18: Change in accuracy from perturbations of new similar users

Key observations

- In both 'new users' attack (tables 4.15, 4.16) and 'similar users' attack (tables 4.17, 4.18) results, there is no significant difference between Meta-Attack and baseline algorithms in terms of drop in performance of target model, unlike our previous Meta-Attack algorithms.
- Moreover, it can be observed that even the baseline algorithm fails to reduce the performance of the target model, unlike our earlier experiments.
- These experiments indicate that our meta-gradient based adversarial attack approaches are not very effective when the attacks are carried out on a different set of users/items than the original training and unseen datasets. We suspect this behaviour could be due to the collaborative-filtering nature of recommendation algorithms, which learns the interdependence between specific users and items. Hence, attacking the dataset with perturbations on new users/items (which have different embeddings altogether) may be ineffective when it comes to collaborative filtering models.

4.7 Datasets

The following datasets were used throughout the course of this project.

- **Movielens Dataset:** This is a standard dataset [16] which has been used in this project without any further processing. It consists of 100k ratings from 943 users on 1682 movies, with each user interacting with atleast 20 movies. The ratings are between 1 and 5. For any user-item rating, the edge between the corresponding user and item is marked 1, since the user has interacted with the item.
- **Steam Games Dataset:** Before preprocessing, the dataset [17] contained 200K interactions from 12,393 users and 5,155 games. To make the dataset more manageable for our attack algorithm, we put constraints such that each users plays 20 or more games, and each game is played by 20 or more users. The resulting dataset contains a total of 132,831 interactions between 1,698 users and 1,397 games.
- **Netflix Dataset:** Before preprocessing, the dataset [18] contained 24,053,764 interactions between 470,758 users and 4,499 movies. We filtered the data such that only 4-star and 5-star ratings are considered for positive edges, each user has watched 150 or more movies, and each movie has been watched by 150 or more users. The resulting data contains 1,660,495 interactions between 8,138 users and 1,503 users.
- **Anime Reviews Dataset**: The original dataset [19] contained 7,813,737 interactions between 73,515 users and 11,200 anime series. After filtering the data such that each user has watched 30 or more series, and each anime is watched by 50 or more users, and randomly sampling 2000 users, the resulting dataset consists of 207,662 interactions between 2,000 users and 1,197 anime series.

These details are summarized in the following table.

	Movielens	Steam	Netflix	Anime
Number of users	943	1,698	8,138	2,000
Number of items	1,682	1,397	1,503	1,197
Number of user-item interactions	100,000	132,831	1,660,495	207,662
Number of unobserved interactions sampled (negative sampling)	100,000	100,000	1,500,000	200,000

Table 4.19: Summary of datasets

Chapter 5

Conclusion

We have proposed a novel approach to perform poisoning attacks on collaborative-filtering based recommendation system models, using the concept of meta-gradients with respect to the data itself. Our experiments show that attacks created using this meta-gradient approach consistently lead to decrease in the edge-prediction performance of collaborative-filtering based architectures in many different settings, even when the attacker model does not have information about the target model's exact architecture or parameters. But the impact of these attacks did not seem very significant when new users are introduced exclusively in the training set – even the baseline model is not able to significantly reduce the performance of target model. There is scope for more research in the future in this direction, regarding how we can attack a recomendation model using meta-gradients on new users alone. These studies could help design more robust recommendation systems, which are immune to these kinds of gradient-based attacks.

Bibliography

[1] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bylnx209YX. 1, 2, 3, 10, 16, 17

[2] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJU4ayYgl. 2

[3] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, Sep. 2008. doi: 10.1609/aimag.v29i3.2157. URL https://ojs.aaai.org/index.php/aimagazine/article/view/2157. 2

[4] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2): 127–163, 2000. 2, 16

[5] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6572. 2

[6] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017. 2

[7] Xiangnan He, Zhankui He, Xiaoyu Du, and Tat-Seng Chua. Adversarial personalized ranking for recommendation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 355–364, 2018. 2, 3

[8] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth*

Conference on Uncertainty in Artificial Intelligence, UAI '09, page 452–461, Arlington, Virginia, USA, 2009. AUAI Press. ISBN 9780974903958. 3

[9] Vito Walter Anelli, Alejandro Bellogín, Yashar Deldjoo, Tommaso Di Noia, and Felice Antonio Merra. Msap: Multi-step adversarial perturbations on recommender systems embeddings. *The International FLAIRS Conference Proceedings*, 34, Apr. 2021. doi: 10.32473/flairs.v34i1.128443. URL https://journals.flvc.org/FLAIRS/article/view/128443. 3

[10] Konstantina Christakopoulou and Arindam Banerjee. Adversarial attacks on an oblivious recommender. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 322–330, 2019. 3

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 3

[12] Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer, 1998. 3

[13] Devang K Naik and Richard J Mammone. Meta-neural networks that learn by learning. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 1, pages 437–442. IEEE, 1992. 3

[14] Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12 (8):1889–1900, 2000. 3

[15] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017. 7, 17, 21

[16] Grouplens Research. Movielens 100k dataset. URL https://grouplens.org/datasets/movielens/100k/. 17, 29

[17] Tamber. Steam video games dataset. URL https://www.kaggle.com/datasets/tamber/steam-video-games. 17, 29

[18] Netflix Inc. Netflix prize dataset. URL https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data. 17, 29

[19] Cooper Union. Anime recommendations database. URL https://www.kaggle.com/datasets/CooperUnion/anime-recommendations-database. 17, 29