Popularity Bias and Fairness in Recommendations

Yuting Gu

Master of Science

School of Informatics
University of Edinburgh

2021

Abstract

If you ever used search engines such as Google or Bing, the only reason why opening these search engines is obvious: you have something to ask, which is a active process, i.e. you know exactly what the query to input in the search engines. However, imagine facing a gigantic movie website, it would be impossible to know the name of certain movie without seen or heard it before, which is the key information the user should obtain to input in the search engine as a query. This indicates the importance of existence of the recommender system which plays an irreplaceable role in nowadays

From user-based, item-based collaborative filtering, to matrix decomposition, and to deep learning technology that has been widely used in recent years, the development of recommendation systems is dramatic. However, due to the wide application of recommender systems in the industry, the sheer amount of user experiences feedback various problems in the recommender system. In this thesis, we mainly explored the influence of popularity bias on the fairness of the recommendation system. To be precise, we narrow down our topic to only focus on the user's perspective. Because previous researches either focused on the performance of prediction and ranking or proposed novel evaluation metrics which has not been widely used in other research. Therefore, in this thesis, we selected a long-tail discovery generative adversarial network model, and used other evaluation methods that focus on popularity bias to measure the performance of this model. In this way, we also explored whether the existing prediction and ranking evaluation metrics and evaluation metrics focused on popularity bias can generate consistent results.

Acknowledgements

This thesis enhances my knowledge in machine learning and artificial intelligence, and offer me the very first time to experience how the recommendation techniques are employed in more realistic problems instead of text-book problem. Also, in the whole process of this thesis, my research skills improved dramatically. All the open-source implementation online and the kindly reply from the author of [21] all helped me significantly during the whole experimental stage. Moreover, these knowledge and experience also helped me in the job hunting and boosted my following internship during the summer, so the multiple meaning of this thesis really makes me feel extremely grateful to have such an opportunity. Therefore, I am very thankful for all the help from my supervisors Roberto Pellegrini and Rafal Karczewski from the Edinburgh Amazon Development Center, as well as professor Iain Murray who emailed me back several times with very directional advice and the author of [21].

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Yuting Gu*)

Table of Contents

1 Introduction **1**

1.1 Motivation and Problem 2

1.2 Objective and Contributions 4

1.3 Structure Organisation 5

2 Background **7**

2.1 Collaborative Filter Recommender System 7

2.2 Matrix Factorisation and Deep Learning 8

2.3 Evaluation in User's Perspective 10

3 Methodology **12**

3.1 Baseline: ALS-MF 12

3.2 Baseline: VAE-CF 14

3.3 Long-tail GAN 15

3.4 Measure Metrics 16

3.4.1 Prediction and Ranking 17

3.4.2 Popularity Bias Measurement 18

4 Experiment Setup **21**

4.1 Dataset and Train/Val/Test Set Formation 21

4.2 Long-tail GAN Dataset Formation 23

4.3 Item Group and User Group Split 24

4.4 Environment Setup 26

5 Experiments and Results **28**

5.1 Training Performance 28

5.1.1 ALS-MF 28

5.1.2 VAE-CF 29

5.1.3 Long-tail GAN 30
5.2 Recommendation Popularity Bias 32

6 Conclusions **36**

Bibliography **38**

Chapter 1

Introduction

One memorable scenario from the famous TV series *Friends*[1]: since Chandler are looking for a new job, the organisation maniac Monica create a huge collection of different job descriptions categorised by industry and across indexed for convenient reference. This is the one typical way that people in 1990s deal with massive information available, which is all manual, time-consuming and information-limited. With the rapid development of internet in 1990s, there are increasing volume of available information online, which hampers human's ability to efficiently retrieve adequate information in website. Such problematic situation is identified as *information overload* by [11], encouraging the researches of information retrieval technologies such as search engines and recommender system (ReSys).

Different from search engines, the interaction between the ReSys and the users is a passive procedure, i.e. the users do not know exactly what they want to search or see, which is the situation requires some suggestions (recommendations) based other's knowledge to make a informed decision. Therefore, this urgent needs expedite the researchers paying more attention and effort on the ReSys. In past decades, many ReSys algorithms can output an acceptable performance, and be widely employed in real-world applications such as the home-page video recommendation on YouTube [2] or [cite, list some famous examples of ReSys application].

However, recent studies show that the long-tail phenomena (shown in Figure 1.1) of the items recommended by the ReSys will amplify the this unbalance[14], which means popular items will be recommended even more frequently than their occurrence ratio in the dataset. This harm to the recommendation results are observed and iden-

[1] One show you should never miss: https://www.imdb.com/title/tt0108778/

[2] https://www.youtube.com/

tified as popularity bias problem by Abdollahpouri et al.[8], which will be defined in detail in the next section.

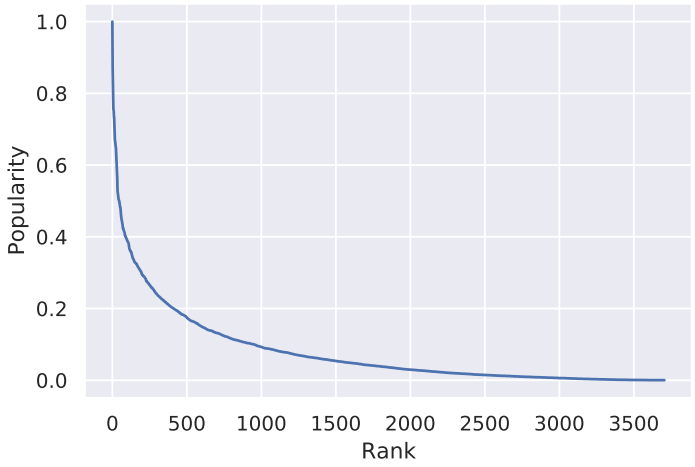

Figure 1.1: This plot presents the long-tail phenomena of MovieLens-1M dataset[16], and items are ranked by their normalised popularity from high(left) to low(right). The popularity of items drops dramatically as the rank increases, and majority attentions only focus on the top 500 movies among 3706 movies in total.

1.1 Motivation and Problem

Before defining the popularity bias, one should be clarified is the basic mechanism of ReSys. By the literal meaning, the goal of a ReSys is to make recommendations to specific users about certain items which can match these users' preference. Therefore, the ultimate goal is to make satisfying recommendations, but generally the whole the black-box of ReSys can be divided into two tasks: prediction and ranking. As shown in Figure 1.2, the goal of the prediction task is to quantify the attitude(like/dislike) of a user to a certain items, by training a regression model on some historical data of users. On the other hand, the ranking task aims to find a subset of all items which are more likely to be interested in by the users, e.g. top-k ranking. Therefore, the whole workflow of ReSys is: the prediction model outputs a predicted score for each user towards each item, and the ranking section sorts these scores by a certain way(e.g. descending) so the top-k items form the recommendation list for each user.

Popularity bias is a term to describe the unbalance distributed popularity of items in the dataset, as well as a series impact of this unbalance. This will cause the uneven exposure problem of items with different popularity in the final recommendation list

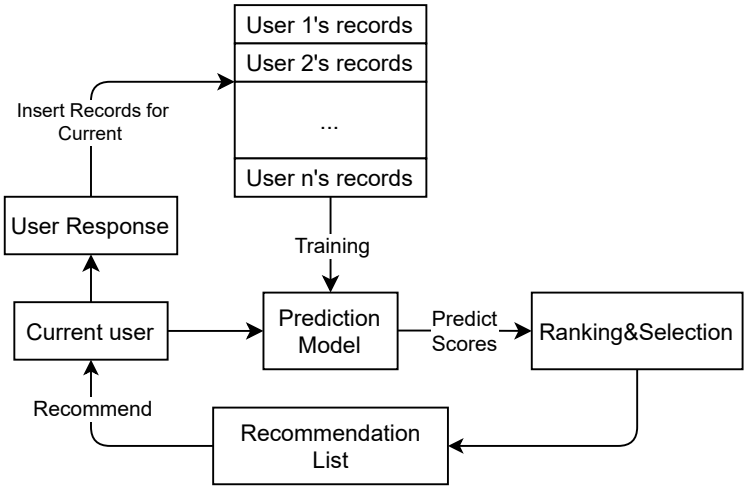

Figure 1.2: Workflow of recommender system. Starting from training the prediction model on users' records, the system can predict current user's preference(scores) of unseen items and then use this scores for the ranking task. In some ReSys, the prediction and ranking tasks are trained together. After the recommendation list is generated, current user will response to this list and this response will be used for update training. In such way, the whole system loops.

to users. It is observed that prediction models will tend to predict higher scores for the popular items, so these items are more likely included in the final recommendation list as they are ranked higher[8, 20]. In some extreme cases, the long-tail(less-popular) items will not be recommended at all, for example, the visualised results from most popular recommendation shown in [10]. Hence, users exposed to such biased recommendation lists will tend to response to the popular items, which will result the famous and harmful *Matthew effect*: the popular items become more popular and the long-tail items receive less attentions. Therefore, as demonstrated in Figure 1.1 and the study of Abdollahpouri et al.[8], popularity bias problem in ReSys is actually an inherent unbalance of dataset which is result from the biased algorithm of ReSys and will continue to be amplified in this cycle.

This popularity bias problem has attracted many researchers to systematically study its causes[8, 20, 4, 9], impacts[7] and algorithmic solutions[3, 5, 10, 18, 24, 23]. However, several problems still exist in their researches, which motivate us to further explore in this area.

1.2 Objective and Contributions

First of all, such looped and amplified popularity bias preserved in ReSys are believed to hinder the interests of multi-stakeholders of ReSys, including three parties: users, items(and their suppliers) and ReSys algorithm[4], which will be explained in details in Chapter 2.3. Nonetheless, since the idea of 'popularity' is defined based on items' occurrence in recommendation list, the research of popularity bias was affiliated as a sub-topic of long-tail(less-popular items) discovery of ReSys. Many previous study of popularity bias were related to long-tail discovery and evaluated in some common prediction/ranking measure metrics with respect to items'(or their suppliers) interests [2, 13, 3, 24]. Thus, there is few study discussing the performance of model in a user-centred view, and to our best knowledge, only Abdollahpouri et al[10] are explicitly stated their research were evaluating in user-centred measure matrices, which makes existing research results are not comparable to each other. However, their research shows that user group who has niche-taste will be discriminated by the algorithm, i.e their experience of ReSys is worse than the group of people who favor popular items, which reveal a unfairness problem of ReSys[8]. For example, users who prefer block-buster movies are more easier to receive recommendations that satisfying their preference, yet users who tend to watch less popular movie have to spend much more time on exploring the movie they like. Therefore, inspired by [10], this thesis mainly focus on evaluating the algorithm's performance in a user-centred aspect by using some item-wise evaluation matrices and user-wise evaluation matrices. Hence, the first research goal of this thesis is **RG-1: Selecting one other long-tail discovery algorithm and evaluating it in a user-centred perspective to measure whether it can mitigate the impact of popularity bias for users**.

Moreover, since different perspective will result in different approaches to optimise the ReSys model, depending on whose benefit the researchers are concentrating, the performance of models might varies. Also, there are no widely agreed measure matrices are used so many results are not comparable to each other, so by focusing in user's interests, the second research goal of this thesis is **RG-2: Checking whether the measure metrics used by one researcher can be consistent with the study of other researchers**.

Hence, the main contributions of this thesis are in two aspects:

- It is the first time that the long-tail discovery generative adversarial network algorithm(long-tail GAN)[21] are employed in the popularity bias aspect and

evaluated with respect to the impact of popularity bias for different group of users. It is reproduced on the same dataset and same measure metrics with the user-centred research from Abdollahpouri et al.[10], so comparable results can bridge other researchers' results with Abdollahpouri et al. who have conduct a comprehensive study of popularity bias.

- The final results reflect an unexpected inconsistency between the classic predicting/ranking measure metrics and popularity bias measure metrics, which indicate the importance to understand popularity bias by a combination of different measure metric instead of only a single score.

1.3 Structure Organisation

The structure of the following thesis and the brief are as following:

- Chapter 2: This chapter will introduce one important and widely studied method of recommender system, collaborative filter, which is also the principle that all our ReSys are based on. Extended on collaborative filter, the further techniques, matrix factorisation and deep learning approach, to overcome the shortcoming of collaborative filter are also introduced in this chapter. Moreover, we also explained the multi-stakeholder of ReSys in this and the reason why user's perspective are chosen in this thesis.

- Chapter 3: This chapter will describe the algorithmic details of three models(ALS-MF, VAE-CF[22] and long-tail GAN[21]) involved in this thesis, as well as the measure metrics for prediction, ranking and popularity bias.

- Chapter 4: This chapter will include comprehensive details of our experiments setup. Since the open-source implementation from the authors of VAE-CF[22] and long-tail GAN[21] do not include enough details of data processing and environment requirements to fully reproduce their study, this chapter will extend their previous work, and provide a full and executable set-up for further researchers' benefit. Furthermore, the split dataset will also be visualised to present the inherent bias in the dataset.

- Chapter 5: This chapter will present all the settings of experiments, results of experiments as well as the comparison between the long-tail GAN and baseline

methods. By comparing the results among different user groups, the inconsistency problem has surprised us and support evidence to motivate further study.

- Chapter 6: This chapter will review all the results from this thesis, and suggest further research directions.

Chapter 2

Background

Before introducing the ReSys model and popularity bias mitigation algorithm, one important technique, collaborative filter ReSys, should be understood, since all our models are extended based on this technique which is the most fundamental and influential idea for the research of recommendation. Moreover, we will also define in details of the three parties involved in ReSys and what the user-centred evaluation is.

2.1 Collaborative Filter Recommender System

As mentioned in Chapter 1.1, the final goal of ReSys is to generate satisfying recommendations for users. The core mechanism of collaborative filter is to mimic how human obtain recommendations according to known history of ourselves and relatively large group of acquaintances[12]. Collaborative filter is all build on an assumption that users who has similar taste in history will response similarly in the future. For example, as shown in Figure 2.1, there are six users interacting with seven items in binary response. The key idea of collaborative filter is to predict one user's response to one item base on other users who has similar response to other items as this user. In this example, we would like to predict how user 3 will interact with item 3. Among all six users, user 1 has same response on item 4 and item 6 as user 3, so one possible prediction of user 3's attitude to item 3 could be 'dislike'.

This simple example of collaborative filter presents one branch called 'user-based' collaborative filter, which means the prediction is estimated based the similarity between user pairs. However, in real-world application, the number of users often much larger than the number of items, which leads to a heavy computational burden when this technique is employed online.

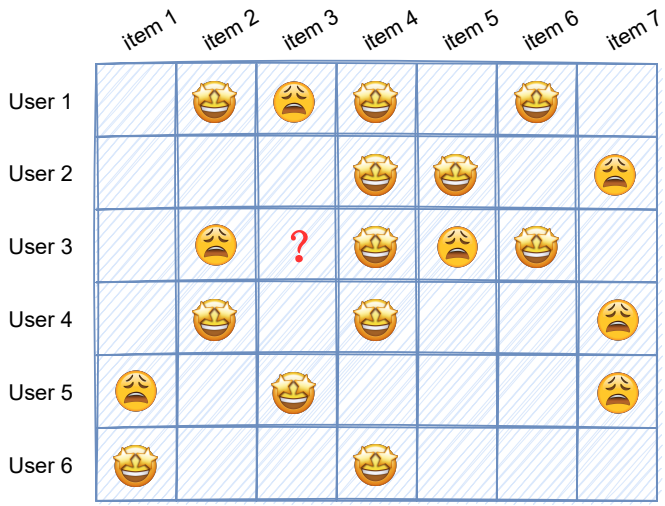

Figure 2.1: This figure presents a binary rating matrix between users and items.

Therefore, another approach, 'item-based' collaborative filter, proposed to use the similarity between items and predict potential interesting items via finding similar items to those already liked by the user.

2.2 Matrix Factorisation and Deep Learning

Even though both user-based and item-based collaborative filter introduce wonderful idea of cooperate with relevant information, in the computational perspective, both computing the similarity between users and between items are computational expensive due to extremely large size of users/items and the sparsity of their ratings. Moreover, both these two collaborative filters are observed tending to recommend the popular items, since these items are viewed by most people. These popular items will have high co-occurrence with other items, which leads to high similarity with most items and be recommended more frequently. Therefore, a further technique called, matrix factorization, was proposed to increase the generalisation performance of the ReSys[19]. This technique views the interaction between users and items as a rating matrix (such as the binary rating matrix shown in Figure 2.1), and expect to predict the score of one user's attitude to an unseen item by matrix factorisation. Since both popular and less popular item are both represent by a rich latent representation, the ability of ReSys to recommend less popular items should be improved.

Mathematically, the $m \times n$ rating matrix $\mathbf{R}$ are decomposed into a $m \times k$ latent matrix $\mathbf{P}^\top$ of users and a $k \times n$ latent matrix $\mathbf{Q}$ of items, where m is the number of users, n

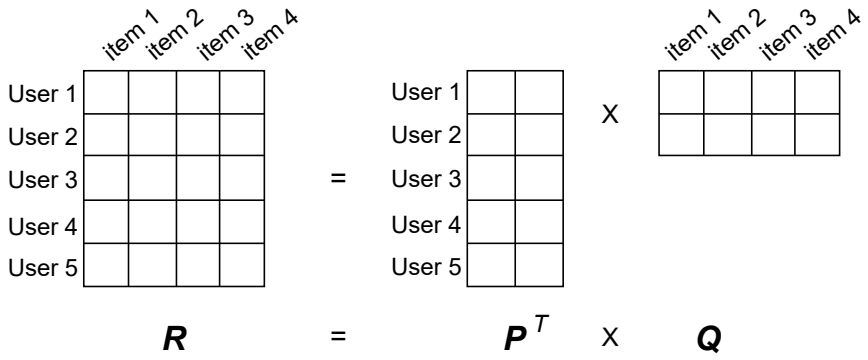

Figure 2.2: This plot shows a 5×4 rating matrix is decomposed into a 5×2 latent matrix of users and 2×4 latent matrix of items

is the number of items and k is the size of latent dimension (such as the example shown in Figure 2.2). By this decomposition, both users' and items' information are reduced to a much lower dimension k, so these latent matrix can infer these information in a dense matrix form. The smaller the k is, the better the generalisation but less information contained; the larger the k is, the worse the generalisation but more information contained. Hence, in application, k is a hyperparameter controlled in experiments to balance the generalising ability of the model and the information contained in the latent matrix.

In ideal mathematical scenario, singular value decomposition(SVD) is the one of the most suitable method to decompose the rating matrix into two lower dimension latent matrix, but the sparsity of the rating matrix causes the SVD to be undefined at these missing elements, so blindly using such technique can easily lead the model to overfit.[19]. Thus, the gradient descent are applied to minimise regularized squared error shown in the following Eq.3.1,

$$\sum_{u,i}(r_{u,i} - p_u^\top q_i)^2 + \lambda(||p_u||^2 + ||q_i||^2), \tag{2.1}$$

where $r_{u,i}$ of $\mathbf{R}$ is user u's true rating to item i, $p_u \in \mathbb{R}^k$ is the u-th column of $\mathbf{P}$ representing the latent vector of user u and $q_i \in \mathbb{R}^k$ is the i-th column of $\mathbf{Q}$ representing the latent vector of item i. This summed error measures how much the predicted ratings are different from the true ratings. Since one of our baseline model ALS-MF is indeed a matrix factorisation model using the alternating least square method to optimise Eq.3.1, the details of the computational iteration will be explained in Chapter 3.1.

However, one problem with matrix factorisation is that, after finding the latent vector, the prediction is essentially just a linear operation between the user latent vector and item latent vector, which is a simple and naive assumption. Not only that, with the widespread application of deep learning in the fields of natural language processing and computer vision, many researchers in the field of recommendation have begun to apply deep learning in this field[17, 15, 26, 25]. Therefore, many further works employed the nonlinear modeling ability of neural network to obtain better performance, such as NeuralCF[17] which uses a multilayer perceptron to replace the inner product of the latent vectors. Another method called VAE-CF[22] introduces variational auto-encoders into collaborative filter ReSys and presents a promising result. This method(VAE-CF) is also one of our baseline method which will be explained in Chapter 3.2.

Although there exists such an incapability in nonlinear modeling of matrix factorisation, the idea of reducing user/item information into a latent lower dimension vector with more dense representation indeed founded the embedding technique which are used closely with the future deep learning model in next era of ReSys.

2.3 Evaluation in User's Perspective

As we mentioned in Chapter 1.2, three parties (users, items and ReSys algorithms) are identified in the ReSys, and with respect to different parties' interests, many previous evaluation measures of a ReSys algorithm cannot capture some unfair treatment of the ReSys[4]. Before explaining the evaluation in user's perspective, the three parties of ReSys can be more specifically defined as following:

- *Users*: the users of ReSys are the group of people who has needs to retrieve information from gigantic database and will receive recommendation list of items generated by the algorithm eventually. Therefore, the ultimate goal of users are effectively finding satisfying recommendation results.

- *Items(and their suppliers)*: the items involved in ReSys can be physical or virtual things (such as actual merchandises sold on Amazon[1] or videos on the homepage of Youtube[2]) which are often not owned by the same identity who provide the recommendation algorithm. Therefore, these items and the suppliers of them are

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

viewed as one party of ReSys by [4], who aim to obtain as much exposure to users as possible, so the revenue is maximised.

- *ReSys Algorithm*: the ReSys algorithm itself are often can be viewed as one stakeholder, since it is usually developed by the platform who offer recommendation services but not own the items that they are recommending. Therefore, the ReSys algorithm's interests will focus on both motivating more users to be active and including more items(and their suppliers) in their platform.

Therefore, the unfairness for users are defined as that different users receive different levels of satisfaction, i.e. certain users may not like the recommendation results as much as some other users. The difficulty is that users' satisfaction is a completely subjective feeling which cannot be easily and objectively measured by only the users' feedback. For examples, a small and casual group interview among our friends reflect that even though they feel the recommendation results from commercial ReSys has tendency to certain advertising items or extremely concentrated only on the popular items, they still will interact with the ReSys since there is no other better choice. However, this behavior causes the number of their responses either less than the true expectation or concentrated on popular items since there is not many other choice. This will further implies that the items which could attract these users received less ratings and popular items receive more than it deserved, which will amplify the popularity bias.

Hence, we believe that a better understanding of users' experience is very important for re-balancing this biased interaction. A more balanced dataset should cause less biased impact on users' experience which further motivates users to feedback more healthy record data for continued training. In order to do so, we will include not only the items-centred evaluations and analyses in a user's perspective, but also measure ReSys's performance using a measurement to reflect how the recommendation results deviate from users' taste. The details of these measure metrics will be described in Chapter 3.4.

Chapter 3

Methodology

In this chapter, we will introduce all the models included in this thesis. First of all, because our final goal is to evaluate whether the chosen long-tail discovery algorithm can be adequate solution for the popularity bias mitigation task, there are only the chosen long-tail discovery algorithm(long-tail GAN) and relevant baselines included.

3.1 Baseline: ALS-MF

As mentioned in Chapter 2.2, the user/item latent vectors can be generated by minimising the regularised squared error (Eq.3.1) using gradient descent method, so the matrix decomposition problem is equivalent to an optimisation problem whose objective is Eq.3.1 subject to p_u and q_u. However, since two parameters are optimised simultaneously, this optimisation problem is not convex, so [19] proposed using alternating least squares(ALS) method to update p_u and q_i until converge. The main idea of ALS is to switching between fixing p_u while optimising q_i and fixing q_i while optimising p_u. In this way, at each sub-problem, this objective will become quadratic as well as convex, so theoretically there is a globule minima which can be solved optimally. The mathematical derivations are shown as following.
First of all, let us recall the objective function:

$$C = \sum_{u,i} (r_{u,i} - p_u^\top q_i)^2 + \lambda(||p_u||^2 + ||q_i||^2) \tag{3.1}$$

Then fix q_i by viewing it as a constant, the partial derivative with respect to p_u is

$$\begin{aligned}\frac{\partial C}{\partial p_u} &= \frac{\partial}{\partial p_u}\left[\sum_{i=1}^{n}(r_{u,i}-p_u^\top q_i)^2+\lambda(||p_u||^2+||q_i||^2)\right] \\ &= \sum_{i=1}^{n}\left[2(r_{u,i}-p_u^\top q_i)(-q_i)+2\lambda p_u\right] \\ &= 2\sum_{i=1}^{n}\left[(q_i^\top q_i+\lambda)p_u - r_{u,i}q_i\right]\end{aligned} \tag{3.2}$$

By setting the derivative equal to zero, the new update for p_u is

$$\begin{aligned}\frac{\partial C}{\partial p_u}=0 &\implies \sum_{i=1}^{n}\left[(q_i^\top q_i+\lambda)p_u\right]=\sum_{i=1}^{n}r_{u,i}q_i \\ &\implies (QQ^\top+\lambda I)p_u = Qr_{u,:}^\top \\ &\implies p_u \leftarrow p_u = (QQ^\top+\lambda I)^{-1}Qr_{u,:}^\top\end{aligned} \tag{3.3}$$

where $r_{u,:}$ is the u-th row of the rating matrix $\mathbf{R}$ representing user u's ratings to all items. Next, differentiate the objective with respect to q_i in a similar way

$$\begin{aligned}\frac{\partial C}{\partial q_i} &= \frac{\partial}{\partial q_i}\left[\sum_{u=1}^{m}(r_{ui}-p_u^\top q_i)^2+\lambda(||p_u||^2+||q_i||^2)\right] \\ &= 2\sum_{u=1}^{m}\left[(p_u^\top p_u+\lambda)q_i - r_{u,i}p_u\right]\end{aligned} \tag{3.4}$$

Therefore, the new update for q_i is

$$\begin{aligned}\frac{\partial C}{\partial p_u}=0 &\implies \sum_{u=1}^{m}\left[((p_u^\top p_u+\lambda)q_i\right]=\sum_{u=1}^{m}r_{u,i}p_u \\ &\implies (PP^\top+\lambda I)q_i = Qr_{:,i}^\top \\ &\implies q_i \leftarrow q_i = (PP^\top+\lambda I)^{-1}Qr_{:,i}^\top\end{aligned} \tag{3.5}$$

where $r_{:,i}$ is the i-th column of the rating matrix $\mathbf{R}$ representing items i's ratings from all users.

For computational iterations, firstly, q_i is initialised as a random matrix, then the above two steps are repeated alternatively, until the iterations reaching limit or the objective function converging. The root mean square error (RMSE) is used to evaluate the prediction accuracy which will be described in section 3.4. In our application, since we only want a simple baseline to observe the impact of popularity bias and compare its performance with popularity bias mitigation algorithm, the regularisation hyperparameter is set to zero.

3.2 Baseline: VAE-CF

We have talked about the linear limitation of matrix factorisation and in our experiments such limitation is actually reflected by its poor inclusion of long-tail items (which will be presented in Chapter 5). Therefore, deep learning approach is introduced to collaborative filter ReSys, and variational auto-encoder (VAE-CF[22]) is one such application. Moreover, the further algorithm long-tail GAN also use VAE-CF as their base recommender, so we include this VAE-CF as well for a more comprehensive reproducibility study. All equations below are cited from the original paper[22]. The key contribution of VAE-CF is to proposed a model that assumes entries of the rating matrix is sampled from a multinomial distribution.

More specifically, the whole generating procedure can be viewed in three parts show in the below equations. First, it generates a k-dimensional latent representation from a standard Gaussian prior. Then a nonlinear transformation $f_\theta(\cdot)$ is learnt and the output of this transformation is normalised by a softmax function. Next, the predicted binary rating will be sampled from the multinomial distribution with probability $\pi(z_u)$ where N_u is the number of items for user u.

$$\begin{aligned} z_u &\sim \mathrm{N}(0, \mathbf{I}_k) \\ \pi(z_u) &\propto \exp\{f_\theta(z_u)\} \\ r_{u,:} &\sim Multi(N_u, \pi(z_u)) \end{aligned} \tag{3.6}$$

This generative model is modeled by a VAE which requires the knowledge of the intractable posterior of z_u. In order to obtain the intractable posterior distribution $p(z_u|r_{u,:})$, variational inference is introduced to approximate this posterior by a simpler variational distribution $q(z_u)$, where $q(z_u) = \mathrm{N}(\mu_u, diag\{\sigma_u^2\})$. Moreover, the author introduces a parameterisation of μ_u and σ_u dependent on $r_{u,:}$ to overcome the bottleneck of optimisation when the number of users increases. Thus, the variational distribution are set as follow:

$$q(z_u|r_{u,:}) = \mathrm{N}(\mu_\phi(r_{u,:}), diag\{\sigma_\phi^2(r_{u,:})\}), \tag{3.7}$$

Since, by variational inference, we want to minimise the Kullback-Leiber(KL) divergence of $p(z_u|r_{u,:})$ and $q(z_u|r_{u,:})$ and maximise the multinomial log-likelihood, the final objective is defined as the following Eq.3.8

$$\mathcal{L}_u(\theta, \phi) = \mathbb{E}_{q_\phi(z_u|x_u)}[\log p_\theta(x_u|z_u)] - \beta \cdot KL(q_\phi(z_u|x_u)||p(z_u)), \tag{3.8}$$

which is known as evidence lower-bound(ELBO) and $\beta < 1$ is the regularisation parameter. Eventually, the output of the whole generative model is the predicted ratings

from all users to all items, then all items for each user are ranked in a descending way so top-k items for each user will be recommended. The network structure of the VAE-CF is relatively simple. The author only uses a fully connected network with one hidden layer for both encoder and decoder and dropout is also applied to avoid overfitting. The details of how these parameters are set will be described in Chapter 5.

3.3 Long-tail GAN

Based on the VAE-CF, Krishnan et al.[21] proposed a new framework using an adversary learning approach to improve the long-tail discover ability of the base ReSys. This whole process is actually quite natural, since the recommendation task can can be viewed as a generating interaction behaviors between users and items based on users' historical records. Therefore, inspired by the recent performance booming of generative adversarial network(GAN) in computer vision and natural language process, the author proposed this idea that the ReSys is viewed as the generator of GAN and the discriminator is build to learn the association between the popular and niche items. The overall model structure is shown in Figure 3.1.

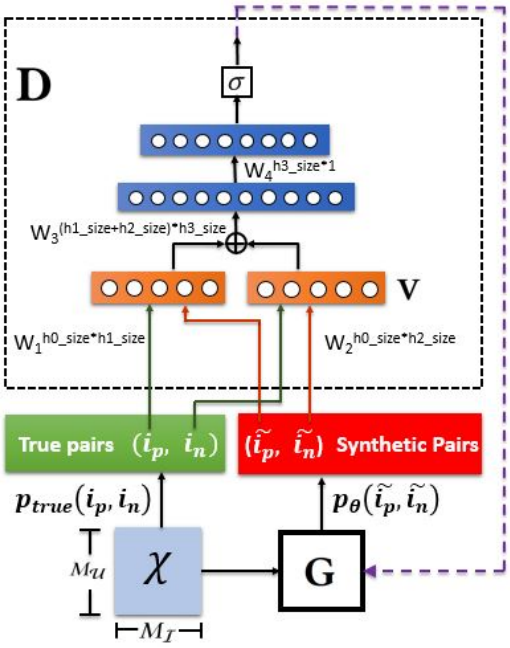

Figure 3.1: This figure present the general model structure of long-tail GAN.(This figure is cited from [21])

The most special part of this model is the using of the pairs of popular-niche items (i^p, i^n), where i^p is the popular item and i^n is the niche item. These pairs are sampled from the global co-occurrence counts in either each user's records or recommendation list. The pairs sampled from users' historical records in training set are referred as true pairs which should follow the the true popular-niche association distribution, $P_{true}(i^p, i^n)$. Moreover, since the ReSys will generate recommendations and the popular-niche item pairs can also be sampled from all recommendation lists, these popular-niche item pairs generated from recommendations are referred as synthetic pairs which follows the synthetic popular-niche association distribution, $P_{syn}(\hat{i}^p, \hat{i}^n)$. The author concatenates each of these pairs and use them as the input to train the discriminator identifying whether the synthetic popular-niche item pairs are 'fake' and learn the association between popular and niche items. Though the author mention no words of GAN in [21], we will still refer the model as long-tail GAN, since in the whole framework, there also exists two models competing with each other to form the two-player mini-max game.

Mathematically, the objective of the generator solely is written as O_G for sanity reason and the probabilistic model of generator is referred as $p_\theta(u)$ where θ is the parameter of the generator model. If the discriminator identify the fake item pairs, a penalty $\mathbf{D}_\phi(\hat{i}^p, \hat{i}^n)$(shown in Eq.3.9) will be imposed to the objective of the generator.

$$\mathbf{D}_\phi(i^p, i^n) = \sigma(f_\phi(i^p, i^n)) = \frac{1}{1 + exp(-f_\phi(\mathbf{V}_{i^p}, \mathbf{V}_{i^n}))}, \tag{3.9}$$

where ϕ is the parameter of the discriminator function $f_\phi(i^p, i^n)$ and $\mathbf{V}_{i^p}, \mathbf{V}_{i^n}$ are the latent representation of i^p and i^n learnt by the discriminator. The overall objective O is given as Eq. 3.10. Since we will not focus on modify the model itself, readers who is interested in the full deviation could refer[21]. The λ here is the hyperparameter to control the strength of penalty, which is set to four levels of value in our experiments.

$$\begin{aligned} O \quad = \quad \min_\theta \max_\phi O_G \quad + \quad \lambda \sum_{u \in \mathcal{U}} & \mathbb{E}_{(i^p, i^n) \sim p_{true}(i^p, i^n)} \left[\log \mathbf{D}_\phi(i^p, i^n) \right] + \\ & \mu \cdot \mathbb{E}_{(\hat{i}^p, \hat{i}^n) \sim p_\theta(\hat{i}^p, \hat{i}^n | u)} \left[\log(1 - \mathbf{D}_\phi(\hat{i}^p, \hat{i}^n)) \right] \end{aligned} \tag{3.10}$$

3.4 Measure Metrics

Evaluation of ReSys has always been a debate since many traditional measure metrics only focus on evaluate how accurate the recommendation list is. However, as we discussed in Chapter 2.3, users' response to items are actually polluted since recommendation results are biased towards the popular items, which makes the so-called

accuracy cannot actually represent users' satisfaction. Therefore, in this thesis, we include not only those traditional measure metrics but also several measurement in popularity bias perspective.

Since in real-world application, the key task of recommendation is to predict what the users might like in the future, and during the experimental stage, the engineers will mask certain proportion of dataset to mimic this unseen future. Therefore, for all the measurement listed in following sections, the evaluations are conducted on a test/validation set which contains ratings are assumed not have been seen in the training set. All our measure metrics are using same terminology as in [1] so it can has consistent results with works from Abdollahpouri et al.[10, 8, 4] Note, the notation p and q used in this section is different from section 3.1.

3.4.1 Prediction and Ranking

Before measuring the models' performance in a user's perspective, it is important to evaluate the prediction and ranking performance, since these are widely-agreed measurement to evaluate basic performance ReSys.

First of all, as mentioned in section 3.1, in order to monitor and evaluate the training performance, the **Root Mean Square Error (RMSE)** is used to measure the difference between predicted ratings and the true ratings. The equation is shown below, where τ is binary records of all interactions between users and items. This measurement is used for ALS-MF during training procedure.

$$RMSE = \sqrt{\frac{1}{|\tau|} \sum_{(u,i)\in\tau} (r_{u,i} - \hat{r}_{u,i})^2} \tag{3.11}$$

For the ranking stage, two measure metrics are applied. One is the **Recall** which measures the ratio of how many items are correctly recommended to users among all items in the test user's previous rating histories. Usually, this measurement is computed on recommendation list with a given length K. This measure metric will reflect that, within the given length K of recommendation list, how the ReSys perform in including items that will be clicked by users. The equation of recall is shown in Eq.3.12, where L_u is the recommendation list for user u and ρ_u is user u's true click records in the test set. The higher this score is, the better the performance of ReSys. In the following thesis, the term Recall@K will be used to indicate which length of recommendation list this score is computed on.

$$Recall(L_u) = \frac{\sum_{i\in L_u} \mathbb{1}(i \in \rho_u)}{|\rho_u|} \tag{3.12}$$

The other ranking measure metric is the **Normalized Discounted Cumulative Gain(NDCG)**(shown in Eq.3.13 and Eq.3.14) which is used to measure the ranking position performance. In expectation, the items ranked higher in user's previous records should also be ranked in the higher position in the recommendation list. Therefore, NDCG will reflect the ranking performance and the higher this score is, the better the ranking performance is. In application, NDCG is also usually computed at a given length K of recommendation list which will be noted as NDCG@K in the following thesis.

$$NDCG(L_u) = \frac{1}{iDCG} \sum_{i \in L_u} \frac{2^{\mathbb{1}(i \in \rho_u)} - 1}{\log_2(K_i + 1)} \tag{3.13}$$

$$iDCG(L_u) = \sum_{k=1}^{min(|L_u|,|\rho_u|)} \frac{1}{\log_2(k+1)} \tag{3.14}$$

3.4.2 Popularity Bias Measurement

Besides the notation used in previous section, there are other the terminologies used in the following description: L represents the combined list of all recommendation lists given to different users; I is the set of all items in the catalog; and U is the set of all users[10].

The first measure metrics is **Average Recommendation Popularity (ARP)** which present the average of popularity of items included in the recommendation list for users who are in the given user set. If this score is lower, then the ReSys is believed better to recommend more long-tail items. However, one problem with this score is, once the recommendation include a extremely niche item, the average score might be deviate significantly, so other measure metrics should be used together to obtain a overall evaluation of the ReSys's performance. The equation for ARP is shown in Eq.3.15, where $\phi(i)$ is the popularity of item i. In our experiments, we compute the number of rates for item i and normalise all popularity by dividing the largest count of ratings among all items, so all popularity will range between [0,1].

$$ARP = \frac{1}{|U|} \sum_{u \in U} \frac{\sum_{i \in L_u} \phi(i)}{|L_u|} \tag{3.15}$$

The second measure metric is **Aggregate Diversity(Agg-Div)**(shown in Eq.3.16) which mainly the diversity and coverage of popular items in the recommendation results[1]. It will compute overall size of items set of given user group and divide it by the total number of items. This measure metric can cooperate with the ARP, since if the recommendation list concentrated on the popular items and only recommend

very few extremely unpopular items, this score will be lower. This means even though ARP might present good results, as long as Agg-Div is lower, the model are possible in only recommending a few item at the tail of item set.

$$Agg\text{-}Div = \frac{|\cup_{u \in U} L_u|}{|I|} \tag{3.16}$$

The last measure metric is the only one which explicitly state to used for user-centred evaluation. This **User Popularity Deviation(UPD)**[10] measure how the recommended items deviate from the user's historical interest in items belonging to different popularity groups. The computation of this measure metric is more complicated than previous measures. Before computing this, we need to affiliate all items according to their popularity. First, the each users' propensity ($p(c|u)$) towards items in different item groups $c \in C$ need to be computed. In this thesis, three are only three item groups $c \in H, M, T$ which will be defined in Chapter 4.3. So this propensity is actually the ratio of items belonging each group in the user's historical records, which will also be referred as user's taste distribution in the following chapters. Then once we get the user's true taste distribution, we also need to compute the ratio of each item group occupied in the recommendation list for user u ($q(c|u)$). The formal equations are defined as Eq.3.17 and Eq.3.18 in [10].

$$p(c|u) = \frac{\sum_{i \in \rho_u} r_{u,i} \mathbb{1}(i \in c)}{\sum_{c' \in C} \sum_{i \in \rho_u} r_{u,i} \mathbb{1}(i \in c')} \tag{3.17}$$

$$q(c|u) = \frac{\sum_{i \in L_u} \mathbb{1}(i \in c)}{\sum_{c' \in C} \sum_{i \in L_u} \mathbb{1}(i \in c')} \tag{3.18}$$

The next step is to calculate the how diverse this two distribution are different from each other. Since the KL-Divergence has orientation requirement, i.e. $KL(p||q) \neq KL(q||p)$, another divergence, Jensen–Shannon divergence, is used to measure deviation. The Jensen-Shannon divergence has two properties are especially adequate for our situation. First, it is symmetric so $\mathcal{J}(P,Q) = \mathcal{J}(Q,P)$; also, it is always finite even if one of P and Q has zero elements, which is a highly possible such as the situation when the recommendation list only contain the popular items. Therefore, the Jensen-Shannon divergence(defined in Eq.3.19) is used to compute how deviate the recommendation is from the users' preference.

$$\mathcal{J}(P,Q) = \frac{1}{2}KL(P,M) + \frac{1}{2}KL(Q,M), M = \frac{1}{2}(P+Q) \tag{3.19}$$

Thus, the final equation to compute UPD is

$$UPD = \frac{\sum_{g \in G} \frac{\sum_{u \in g} \mathcal{J}(p(c|u), q(c|u))}{|g|}}{|G|}, \tag{3.20}$$

where $|G|$ is the number of user groups and $|g|$ is the size each user group. The smaller the score is, the less deviation these distributions are. Hence, UPD is very suitable to compare whether one user group receive fair treatment as other user group, and we will present the UPD performance of each model on each user group in Chapter 5.

Therefore, combination of these three measure metrics will not only make sure the recommendation cover enough evenly distributed niche items, but also measure how even this recommendation is.

Chapter 4

Experiment Setup

This chapter include all the details of what the dataset is and how the dataset, Movie-Lens 1M[16], is processed and split, as well as the exact number of ratings included in each set. Moreover, since the both VAE-CF[22] and Long-tail GAN[21] are using binary rating matrix but the original data in Movie-Lens 1M[16] integers ranged in [1,5], a mapping procedure is also conducted. In addition, several empirical problems has rise when we were actually running the experiments, so details of environment setup are also included to help further researchers overcome these problems.

The implementation of this project is uploaded on the GitHub repository[1] which can be used for further study.

4.1 Dataset and Train/Val/Test Set Formation

The dataset used in this thesis is the Movie-Lens 1M[16]. Although Movie-Lens also offer a much larger dataset with 25 million ratings, due to computation resource limitation, we stay with this dataset. This dataset can be directly downloaded from the MovieLens website[2], but it is stored in `.dat` files, so a dataset converting script is written to convert the dataset in a `.csv` format, including four columns `[userId', 'movieId','rating','timestamp']`, to fully store the interaction between users and items. It contains 1,000,209 ratings of 6,040 users to 3,706 items. The dataset is actually very sparse with only 4.468% non-zero entries. Generally speaking, 20% of the ratings are holdout as the test set and the rest 80% are used for the training procedure.

[1] https://github.com/YutingGu/MSc_Thesis
[2] https://grouplens.org/datasets/movielens/

Figure 4.1: (left)This plot illustrate the first dataset split approach which holdout 20% ratings of all users as test set. (right)This plot illustrate the second dataset split approach which holdout ratings of 20% users as test set.

However, in our experiments, there are actually two approaches to split the training data from the dataset. The reason is that, for our ALS-MF, if a user has not been seen in the training set, it will not be included in the rating matrix, which means there is no latent vector of this user and we cannot predict this user's preference by dot product of this user's latent vector and the latent matrix of all items. These two approaches are:

1. First, We holdout around 20% ratings of each user's as the test set, and the rest are used as training set (as shown in the left subplot of Figure 4.1). Moreover, since we use a filter to exclude ratings of users who have rated less than 5 movies, holdout 20% of each user's ratings is possible.

2. The second split approach is to holdout 20% users as test set and use the ratings of rest 80% users as training set(as shown in the right subplot of Figure 4.1). This is the same data-processing approach as used in VAE-CF[22].

The first approach is used in ALS-MF, but for VAE-CF[22] and the first experiments of long-tail GAN[21], the model's performance is significantly worse than training on the dataset split by the second approach, so in further experiments, we continue with the dataset split by the second approach.

After obtain the training set, since for both VAE-CF[22] and long-tail GAN[21] are neural network model, we also include a validation set as to compute the NDCG scores mentioned in Chapter 3.4 to monitor the training performance. 25% of training set are divided as validation set, and 80% ratings of all users in both validation and test set are

	Users	Movies	Ratings	Percentage
Training Stage	3624	3646	603535	60.34%
Validation Stage				20.48%
Validation(Train)	1208	3421	164308	
Validation(Test)	1208	3044	40491	
Test Stage				19.17%
Test(Train)	1208	3370	153914	
Test(Train)	1208	2964	37876	

Table 4.1: This table presents the exact number of ratings included in training/validation/test set.

used for validation-stage/test-stage training. The overall training, validation and test set proportion are shown in Table 4.2:

4.2 Long-tail GAN Dataset Formation

Because our experiments of long-tail GAN are build on the open-source implementation published by the author, we need to re-format our dataset so the packed script can successfully execute. This new version of explanation of all required files are our understanding of the description on the open-source `README.md`, since in our experiments, the original description actually leads to many confusion and after emailing the author, we settle down with this new version of explanation. All files in the used are listed below, which can also be found in `long-tail_GAN/Dataset`.

- `item_counts.csv`: This `.csv` file is the training set in split from MovieLens 1M. Each row contain `[userId, movieId, rating]`.
- `item_list.txt`: List of all true movie ids without duplication.
- `unique_item_id.txt`: This is the mapped movie ids stored in the order corresponding to `item_list.txt`.
- `item2id.txt`: This is the map between true movie ids in `item_list.txt` to the mapped movie id in `unique_item_id.txt`.
- `profile2id.txt`: This is the map between true user ids to the mapped user id.

- `niche_items.txt`: This is the true movie ids of niche items. Here, we select the M items since the size of all T items is much larger and the model need to compute a Cartesian pairs between niche and popular items, using T items is a computational burden that cannot be conducted in this thesis.

- `train_GAN.csv`: This `.csv` contains (mapped userId,mapped movieId) pairs in the training set.

- `train_GAN_popular.csv`: This `.csv` contains (mapped userId,mapped movieId) pairs in the training set, where the true movie ids are affiliated to H items.

- `train_GAN_niche.csv`: This `.csv` contains (mapped userId,mapped movieId) pairs in the training set, where the true movie ids are affiliated to M items.

- `validation_tr.csv`: This `.csv` contains (mapped userId,mapped movieId) pairs in the training set in the validation stage.

- `validation_te.csv`: This `.csv` contains (mapped userId,mapped movieId) pairs in the test set in the validation stage.

- `test_tr.csv`: This `.csv` contains (mapped userId,mapped movieId) pairs in the training set in the testing stage.

- `test_te.csv`: This `.csv` contains (mapped userId,mapped movieId) pairs in the test set in the testing stage.

4.3 Item Group and User Group Split

In order to categorise users into different user groups by their taste distribution of items of different popularity, firstly, we need to compute the popularity of items. Since the popularity is defined as that the more people interact with one item, the more popular this item is, we compute the count of ratings of each movie to represent its popularity. Moreover, we normalise this count to range [0,1] by dividing the number of counts of the most popular item.

Thus, We affiliate all items into three groups:

- **H items:** items whose total number of ratings occupy 20% total number of ratings and ranked at top of the sorted list of movie id.

- **M items:** items whose total number of ratings occupy the 60% total number of ratings and ranked after head items.
- **T items:** items whose total number of ratings occupy the 20% total number of ratings and ranked at the bottom of the sorted list of movie id.

After this affiliation, we have found that there are only 113 H items (3% of total items) but these items occupy 20% of ratings. Moreover, there are 1069 M items and 2525 T items, which present a significant unbalance that much fewer H and M items account for 80% ratings, and over half of total items (T items) only account for 20% ratings.

Then, we create a new column to store the label of item groups for each ratings in the dataset. The column `'movies_pop'` has values 1 or 2 or 3 which represent H or M or T type of items respectively. By computing the ration of each labels existing in each user's rating records, we obtain a taste distribution for each user (i.e. for each user, we have three float numbers to represent the ratio of H/M/T items occurred in this user's records). The average taste distribution of all users are shown in the most left column of Figure 4.2. We sort users by the ratio of H items in their records, and divide all users into three groups:

- **G1 users:** group of users who has strongest preference to popular items. (1208 users)
- **G2 users:** group of users who more prefer M and T items. (3624 users)
- **G3 users:** group of users who mainly focus on M and T items and has around 30% T items in their ratings. (1208 users)

It is clearly to observe that the G1 users' taste distribution focuses more on H items and only contain 7.14% T items, while G2 users having similar distribution as the overall users since their are the majority of all users. The ratio of H items contain in G3 users' records is only half of the H ratio of overall users. Therefore, we can conclude that different users indeed have considerable different from each other. The whole item groups and user groups split is reproduced by the approach described in [10, 6, 5].

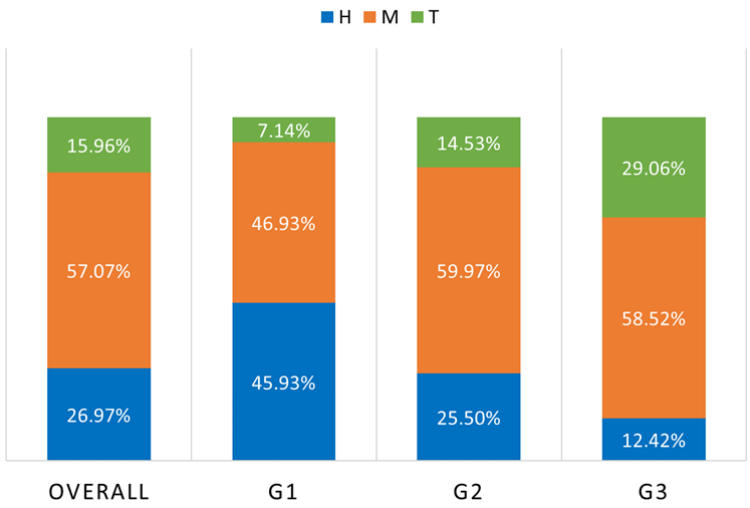

Figure 4.2: This bar plot present the users' average taste distribution of over all users, G1 users, G2 users and G3 users.

4.4 Environment Setup

First of all, as mentioned in previous section, the main implementation of all our methods, ALS-MF[3], VAE-CF[4] and long-tail GAN [5], are based on GitHub repository published by the author of corresponding methods(except ALS-MF which is just a simple version online). Therefore, some packages used are in some old version and not clearly stated, which takes us significant time to cooperate the whole environment. So we have tested several different environments and conclude a version that are consistent. With the required packages installed, all the experiments has been successfully run. The required packages are:

numpy=1.16.0	tensorflow-gpu=1.14.0	scikit-learn=0.23.1
seaborn=0.11.1	matplotlib=3.3.2	pandas=1.1.3
bottleneck=1.3.2	psutil=5.8.0	scipy=1.5.2
configparser=5.0.2		

Table 4.2: This table present all the required packages with corresponding version.

The implementation for first dataset split approach as well as user groups split can be found in the file `process_rawdata.ipynb`. The second dataset split approach is

[3]https://github.com/tushushu/imylu/tree/master/imylu/recommend
[4]https://github.com/dawenl/vae_cf
[5]https://github.com/CrowdDynamicsLab/NCF-GAN

written together with VAE-CF's experiments in `VAE_CF/VAE_CF(holdout users).ipynb`, and the dataset for long-tail GAN is also generated in this file.

All the experiments are conducted on a cloud service with Intel(R) Xeon(R) Gold 6130 CPU (31G RAM, 32 cores) and Tesla V100-32GB GPU (31G memory).

Chapter 5

Experiments and Results

In this chapter, we will present all the information and decision making during training procedure, as well as all the results obtained. Moreover, the measure scores will also be explained to help readers to understand the meaning of each measure metrics.

5.1 Training Performance

5.1.1 ALS-MF

First of all, for our simplest baseline, ALS-MF, we adapt a open-resource implementation and conduct the experiments on the dataset with 20% ratings holdout. RMSE is used to evaluation the performance during training and we use $K = 5$ as the size of dimension of latent size. Since the model is fairly simple, only 5 iterations are used for training(presented in Table 5.1). There is no fine-tuning or hyper-parameter searching for this model as we only want a simple model for comparison.

Iterations: 1	RMSE: 3.515243
Iterations: 2	RMSE: 0.503536
Iterations: 3	RMSE: 0.488565
Iterations: 4	RMSE: 0.480616
Iterations: 5	RMSE: 0.477763

Table 5.1: This table present the RMSE at each iteration of training of ALS-MF model.

5.1.2 VAE-CF

The training of VAE-CF is much more complicated than ALS-MF. There are two dataset split approaches used in training VAE-CF and the network architecture of the VAE are (N, 600, 200, 600, N) where N is the number of items. All hyper-parameters are set as same as the best performance from the original paper[22](i.e 200 epochs, batch size of 500, 0.5 probability of dropout and 0.2 as largest annealing parameter). In the following experiments of VAE-CF, both loss values and NDCG@100 are used to evaluate model's performance during training.

At beginning, due to the dataset consistency reason, we tried to conduct the training on the same dataset as used in ALS-MF. However, in comparison with the dataset split approach used in the implementation from the author[22], the dataset split by the first approach is significantly outperformed by the second dataset. As shown in Figure 5.1, though the model converges, the highest value that NDCG@100 ever reaches is only around 0.13 while the model's performance stay more stably at around NDCG@100 = 0.43 for the model trained on the dataset split by second approach (shown in Figure 5.2). Because VAE-CF do not require the users that has to be seen in training stage, we decide to use the dataset split by the second approach for following popularity bias evaluation and also for the training of long-tail GAN.

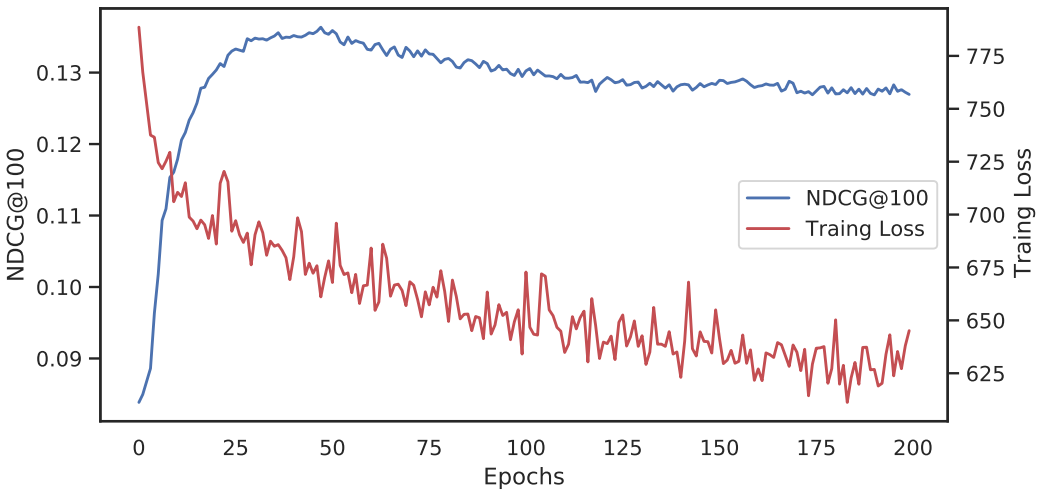

Figure 5.1: This plot present the training performance of VAE-CF on the dataset with holdout ratings as test set. Two evaluations, NDCG@100 and the training loss, are both presented in this plot.

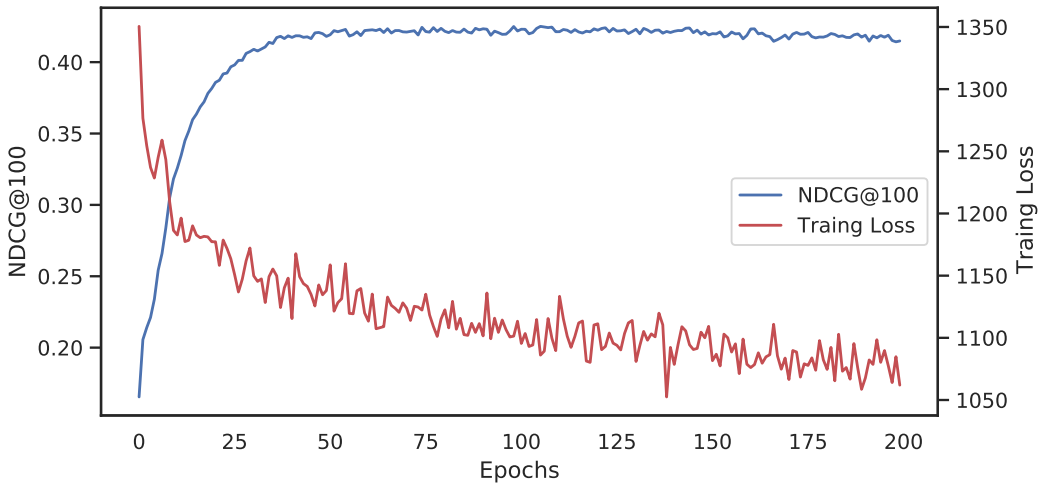

Figure 5.2: This plot present the training performance of VAE-CF on the dataset with holdout users as test set. Two evaluations, NDCG@100 and the training loss, are both presented in this plot.

5.1.3 Long-tail GAN

As stated above, since the VAE-CF trained on the dataset split by the second approach outperform the first approach significantly, the dataset reformation procedure for long-tail GAN is also conducted on the dataset split by the second approach. For the experiment settings, we also stay with the setting used in original paper[21]: 80 global epochs with 10 epochs for both generator and discriminator at each global epoch. Learning rate are fixed at 0.0001 and batch size of 50. The Figure 3.1 from Chapter 3.3 presents the overall workflow of the long-tail GAN model, where other parameters setting information such as the hidden layers dimension can be found in the `long-tail_GAN/Codes/config.ini` file.

As the training of a generative adversarial network is computational expensive, though it rises our attention that the models have not completely converge (as shown in Figure 5.3), the model's performance is already considerably better than VAE-CF. The best VAE-CF can reaches is approximately 0.43, but long-tail GAN can approaches about NDCG@100 = 0.62. In fact, we have tried a larger number of training epochs and there is no significant increase in NDCG@100 so there is no much effort spent on fine-tuning the model's performance by grid-search on the hyperparameters.

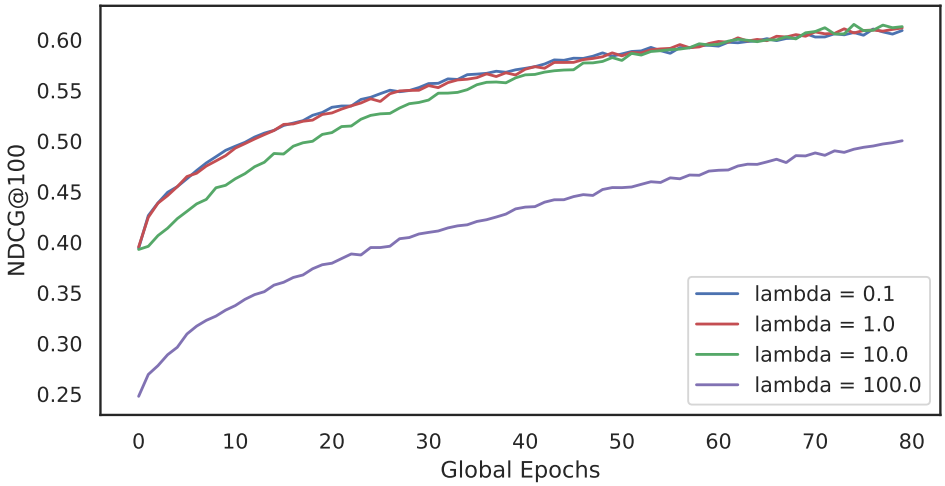

Figure 5.3: This plot present the training performance of long-tail GAN with penalty parameter $\lambda = 0.1, 1.0, 10.0, 100.0$ on the dataset with holdout users as test set.

Model	NDCG@100↑	Recall@20↑	Recall@50↑
VAE-CF(holdout ratings)	0.1431	0.1406	0.2669
VAE-CF(holdout users)	0.4299	0.3589	0.4542
long-tail GAN($\lambda = 0.1$)	0.6148	0.5419	0.6384
long-tail GAN($\lambda = 1.0$)	**0.6218**	0.5444	0.6420
long-tail GAN($\lambda = 10.0$)	0.6169	**0.5498**	**0.6423**
long-tail GAN($\lambda = 100.0$)	0.4959	0.4228	0.5458

Table 5.2: This table presents the ranking evaluation, NDCG@10 and Recall@20, for all VAE-CF models and long-tail GAN models.

Therefore, we repeat this experiments for different value of the key hyperparameter of long-tail GAN, λ. We trained our model on the four levels of λ as same as in [21], i.e. $\lambda = 0.1, \lambda = 1, \lambda = 10$ and $\lambda = 100$. This hyperparameter will control the strength of punishment, which means if the generator is producing results which are recognised as more negative by discriminator, the overall objective will be larger. We can observe that, for $\lambda = \{0.1, 1, 10\}$, the model's performance are almost same with only $\lambda = 10$ presenting a slightly slower converging; but for $\lambda = 100$, the model demonstrate a much worse performance. Furthermore, as displayed in Table 5.3, both Recall@20 and Recall@50 also shows similar good results for $\lambda = 0.1, 1, 10$ and significant drop when λ increase to 100. Even though the highest score are not always presented at $\lambda = 1$ which is the best results shown in the original paper[21], the different is minor in

contrast with the obvious decline at $\lambda = 100$. Therefore, our experiments successfully reproduce the results from [21] and generate consistent conclusion.

5.2 Recommendation Popularity Bias

Speaking of measure metric in popularity bias perspective, we first compute the overall ARP and Agg-Div scores on the recommendation list for all users in the test set. The results are presented in Table 5.3. Surprisingly, the results cannot consistent with the training performance evaluations. In previous section and conclusion from[21], the best performed model is the long-tail GAN with $\lambda = 0.1$ and $\lambda = 1.0$. However, in the ARP@10 evaluated for four long-tail GAN models, none of the long-tail GAN model cannot outperform the VAE-CF(with holdout users as test set) and the value of ARP@10 is monotonically decrease(better) as λ increases. On the other hand,

Model	ARP@10 ↓	Agg-Div@10 ↑
ALS-MF	0.6893	0.0696
VAE-CF(holdout users)	**0.2005**	0.3613
long-tail GAN($\lambda = 0.1$)	0.3816	**0.4679**
long-tail GAN($\lambda = 1.0$)	0.3630	0.4609
long-tail GAN($\lambda = 10.0$)	0.3084	0.4355
long-tail GAN($\lambda = 100.0$)	**0.2394**	0.3160

Table 5.3: This table presents the popularity bias evaluation, ARP@10 and Agg-Div@10, for all models on the whole test set.

the Agg-Div@10 present highest score at $\lambda = 0.1$ which matches our expectation. As we talked in Chapter 3.4, this ARP@10 cannot completely reflect the overall popularity distribution solely if the score is long but include certain few extremely unpopular items, and the higheset Agg-Div@10 is highest(best) when ARP@10 is also highest(worst), excluding ALS-MF since both VAE-CF and long-tail GAN models produce better performance than ALS-MF. Therefore, only considering the general performance of a ReSys for all test users may not be able to conclude which is the best model.

Hence, in next stage of our experiments, we evaluate both ARP@10 and Agg-Div@10 of all models on three group of users. As shown in Table 5.4, the result for all users is consistent with the observations above. None of long-tail GAN can out-

ARP@10↓			
Model	G1	G2	G3
ALS-MF	0.7348	0.6952	0.6262
VAE-CF(holdout users)	**0.1921**	**0.2060**	**0.1913**
long-tail GAN($\lambda = 0.1$)	0.3737	0.3803	0.3941
long-tail GAN($\lambda = 1.0$)	0.3509	0.3716	0.3512
long-tail GAN($\lambda = 10.0$)	0.2999	0.3140	0.3012
long-tail GAN($\lambda = 100.0$)	**0.2358**	**0.2413**	**0.2380**

Table 5.4: This table displays the ARP@10 for all models' recommendation to three group of users. G1 is the users who mainly prefer popular items; G2 is the users who have more diverse taste distribution; G3 is the users who have least interest to popular items.

perform VAE-CF(holdout users), and only for the 4 long-tail GAN models, ARP@10 decreases(better) as λ increases(stronger punishment). Horizontally view(comparing between G1,G2 and G3), in contrast with ALS-MF which the ARP@10 decrease by only 0.1 when it is recommending for niche-taste users, both VAE-CF(holdout users) and long-tail GAN($\lambda = 100$) oscillate within only around 0.01. So we considered both two models having more stable performance in anti-discrimination to different group of users.

Next, the Agg-Div@10 are also computed for recommendation results from all models toward three user groups. First of all, both VAE-CF and long-tail GAN is significantly better than ALS-MF on all three groups, which means only VAE-CF can already generate a much diverse recommendation list. Along with the adversarial learning, the performance further improves. However, if we consider horizontally,for G1 and G2 users, the best performance presents at long-tail GAN($\lambda = 0.1$), but for G3 users, long-tail GAN($\lambda = 10.0$) shows better final results, though the difference is still minor. In the view of balance among groups, long-tail GAN($\lambda = 0.1$) is obviously favoring G2 users. But since the number of users in G2 is triple the size in either G1 or G3, such higher score could caused by the larger number of users in this group.

The last measure metric is the UPD@10. We only compute this score on each user groups not for all users, since the meaning of this score is to measure if different group of people are receive less satisfying results. We can observe that there is no global best model, and for G1 users, long-tail GAN can still perform well but for rest two user

Agg-Div@10↑			
Model	G1	G2	G3
ALS-MF	0.0370	0.0594	0.0666
VAE-CF(holdout users)	0.1989	0.3060	0.1913
long-tail GAN($\lambda = 0.1$)	0.2580	**0.3821**	0.2288
long-tail GAN($\lambda = 1.0$)	0.2582	0.3743	0.2299
long-tail GAN($\lambda = 10.0$)	**0.2601**	0.3724	**0.2423**
long-tail GAN($\lambda = 100.0$)	0.2129	0.2858	0.2016

Table 5.5: This table displays the Agg-Div@10 for all models' recommendation to three group of users.

UPD@10↓				
Model	G1	G2	G3	Average
ALS-MF	0.2294	0.3299	0.3700	0.3098
VAE-CF(holdout users)	0.1205	**0.0591**	**0.0550**	0.0782
long-tail GAN($\lambda = 0.1$)	**0.0624**	0.0820	0.1506	0.0983
long-tail GAN($\lambda = 1.0$)	0.0631	**0.0811**	0.1399	0.0947
long-tail GAN($\lambda = 10.0$)	0.0811	0.0660	**0.1194**	0.0883
long-tail GAN($\lambda = 100.0$)	0.1964	0.1356	0.14837	0.1601

Table 5.6: This table displays the UPD@10 for all models' recommendation to three group of users.

groups, it is the VAE-CF that generate recommendations least deviating from users' true taste distribution.

Moreover, if we compare the variation tendencies of different models in different groups (Figure 5.4), there is only VAE-CF presenting a decreasing trend, which means the preference of ReSys is reversed from favoring popular-taste users to favoring niche-taste users. However, in general, the ultimate hope is to make a fair recommendation for all user groups, so both VAE-CF and long-tail GAN present a more flat tendency of variation and these two models also have the two smallest average UPD@10

To conclusion, different popularity bias measure metrics are actually not consistent with each other, and since different measure metrics are focusing on different aspect with different shortcoming, there are no global agreed one single measure metric and a fair and meaningful judgement should based on cooperative consideration of all mea-

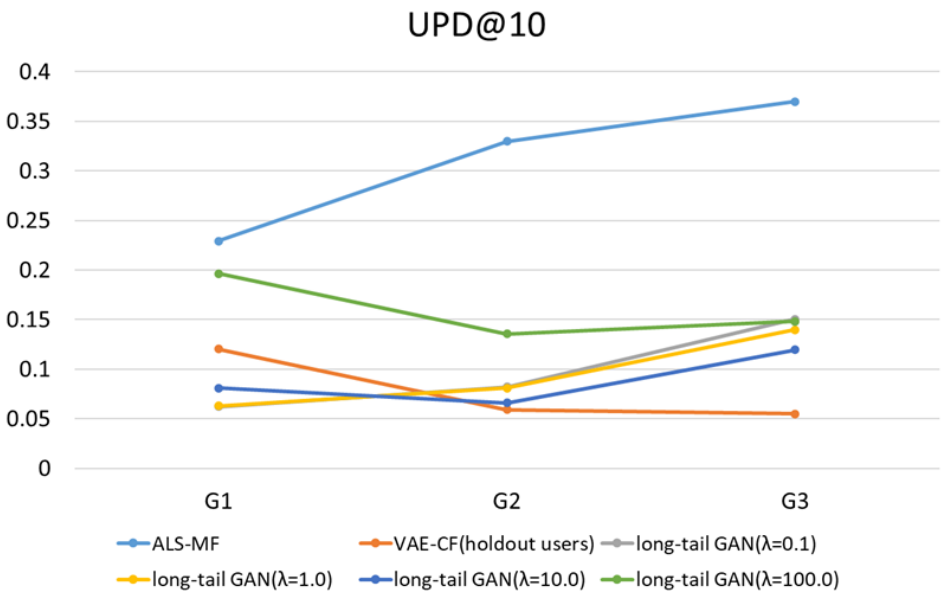

Figure 5.4: This plot demonstrates the how different models' UPD@10 vary in different groups.

sure metrics. Hence, we consider long-tail GAN($\lambda = 10$) is generally the best model in our experiments. First of all, it has the second largest NDCG and best Recall@20 and Recall@50 scores. Moreover, the ARP@10 for either all users and different user groups is also relatively low and Agg-Div@10 of this model also present either best results or at least top 2. Finally, it produce relatively stable treatment to all user groups which slightly favours users who like less popular items. Therefore, in general performance, long-tail GAN($\lambda = 10$) outperforms other models in the task of popularity mitigation.

Chapter 6

Conclusions

Popularity bias in ReSys has been widely studied in recent years, and there are some effort has done in this area. In this thesis, we reviewed the definition of popularity bias as well as basic introduction of ReSys. Moreover, we demonstrate the necessity of studying popularity bias in a user's perspective and conduct experiments concentrated in this sub-topic. In addition, because most previous studies only evaluate ReSys in classic measure metrics for predicting and ranking, such as NDCG and Recall, but recent studies from Abdollahpouri et al.[1, 5, 9, 3, 8, 4] demonstrate a series of measurements and present a novel understanding of popularity bias. Therefore, inspired by this pioneering understanding approach, we would like to explore how other researchers' work perform in these measure metrics. A generative adversarial network model, long-tail GAN[21], for long-tail discovery is selected since there no other method for popularity bias mitigation using the idea of GAN, and this model is only evaluated in NDCG and Recall in the published work. Also, a very basic baseline model is also included as the bottom performance of ReSys.

In our experiments, three different models are compared with different measure metrics, and the results discussed in previous chapter reflect that models with better classic ranking measure scores and training performance might be outperformed by other models. Three popularity bias measurements are presenting three important features of the recommendation list: 1)if the recommendation list less focus on the head(popular) items; 2)if the recommendation list contains a diversified set of items; 3)if different group of users are treated fairly by the ReSys. Evaluation results in popularity bias perspective shows that the model which is less accurate in ranking performance can present a overall better popularity bias mitigation. Unfortunately, there is no widely agreed single measure metrics, but combining different popularity bias

measure metrics and evaluating overall could be a framework to measure popularity bias in ReSys. Although, the overall performance of the long-tail GAN model is not significantly better than the baselines, our experiments present evidence that only the traditional predicting and ranking measure metrics are not enough to evaluate a state-of-art model in recommendation result.

In the future, the first work could be a more exhaustive review of all measure metrics existed in recommendation researches area, and evaluating a more stable performed model on different measure metrics to help summarising of the framework for ReSys evaluation. It seems impossible to have a single scores to reflect how the impact of popularity bias in ReSys, because different stakeholders have different interests. Therefore, building a evaluation framework with combination of different measure metrics is a more realistic way generally measure the popularity bias problem.

Furthermore, due to the limitation of one Master thesis, we spent no time on evaluating the impact of popularity bias on items' supplier who also plays a important role in the whole eco-system of recommendation. The platform offering ReSys algorithm is aiming to make profit and usually believe a more fair treatment towards suppliers will result in a higher quality of items available on the platform, so these higher quality items will further attract users using this platform. Therefore, supplier's satisfaction is also an interesting point of view to understand popularity bias in ReSys.

Bibliography

[1] Himan Abdollahpouri. *(PhD Dissertation)Popularity Bias in Recommendation: A Multi-stakeholder Perspective*. University of Colorado Boulder, 2020.

[2] Himan Abdollahpouri, Robin Burke, and Masoud Mansoury. Unfair exposure of artists in music recommendation, 2020.

[3] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. Controlling popularity bias in learning-to-rank recommendation. RecSys '17, page 42–46, New York, NY, USA, 2017. Association for Computing Machinery.

[4] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. Recommender systems as multistakeholder environments. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, UMAP '17, page 347–348, New York, NY, USA, 2017. Association for Computing Machinery.

[5] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. Managing popularity bias in recommender systems with personalized re-ranking, 2019.

[6] Himan Abdollahpouri and Masoud Mansoury. Multi-sided exposure bias in recommendation. *arXiv preprint arXiv:2006.15772*, 2020.

[7] Himan Abdollahpouri, Masoud Mansoury, Robin Burke, and Bamshad Mobasher. The impact of popularity bias on fairness and calibration in recommendation, 2019.

[8] Himan Abdollahpouri, Masoud Mansoury, Robin Burke, and Bamshad Mobasher. The unfairness of popularity bias in recommendation, 2019.

[9] Himan Abdollahpouri, Masoud Mansoury, Robin Burke, and Bamshad Mobasher. The connection between popularity bias, calibration, and fairness in recommendation. In *Fourteenth ACM Conference on Recommender Systems*,

RecSys '20, page 726–731, New York, NY, USA, 2020. Association for Computing Machinery.

[10] Himan Abdollahpouri, Masoud Mansoury, Robin Burke, Bamshad Mobasher, and Edward C. Malthouse. User-centered evaluation of popularity bias in recommender systems, 2021.

[11] Sarabjot Singh Anand and Bamshad Mobasher. Intelligent techniques for web personalization. In *IJCAI Workshop on Intelligent Techniques for Web Personalization*, pages 1–36. Springer, 2003.

[12] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-based systems*, 46:109–132, 2013.

[13] Òscar Celma Herrada et al. *(PhD Thesis)Music recommendation and discovery in the long tail*. Universitat Pompeu Fabra, 2009.

[14] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. Bias and debias in recommender system: A survey and future directions, 2020.

[15] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[16] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[17] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering, 2017.

[18] Toshihiro Kamishima, Shotaro Akaho, Hideki Asoh, and Jun Sakuma. Correcting popularity bias by enhancing recommendation neutrality. In *RecSys Posters*, 2014.

[19] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[20] Dominik Kowald, Markus Schedl, and Elisabeth Lex. The unfairness of popularity bias in music recommendation: a reproducibility study. *Advances in Information Retrieval*, 12036:35, 2020.

[21] Adit Krishnan, Ashish Sharma, Aravind Sankar, and Hari Sundaram. An adversarial approach to improve long-tail performance in neural collaborative filtering. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 1491–1494, New York, NY, USA, 2018. Association for Computing Machinery.

[22] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 world wide web conference*, pages 689–698, 2018.

[23] Weiwen Liu and Robin Burke. Personalizing fairness-aware re-ranking. *arXiv preprint arXiv:1809.02921*, 2018.

[24] Harald Steck. Item popularity and recommendation accuracy. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 125–132, 2011.

[25] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 5941–5948, 2019.

[26] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1059–1068, 2018.