Fairer Recommendation Systems through Graph Neural Networks

by

John Hannebery

Thesis for the Masters of Science

School of Mathematics & Statistics
THE UNIVERSITY OF MELBOURNE

May 2023

Contents

Chapter 1

Introduction

Recommendation systems are one of the most prolific applications of machine learning deployed today, powering many of the top internet sites' product offerings including YouTube [1], Spotify [2], Netflix [3] and TikTok [4]. Given this collective reach to billions of users worldwide, it is of great importance that the recommendations generated are fair and unbiased toward all users.

A recommendation system will retrieve and rank relevant items for a given input of either a user or item based on an objective. For a user ID, a recommendation system will return the most relevant items for that user, for example the For You Page on TikTok shown in Figure 1.1 and [5]. Given an item ID in a content library, a recommendation system can also return the most relevant or similar items to that base item, often seen in related item sections on websites such as Amazon shown in Figure 1.2. In this thesis, we will consider the former case, recommendations for a given user as input.

Recommendation systems are an application of machine learning. Without data, recommendation systems cannot exist. Data inputs into a recommendation system are most commonly in the form of a user-item interaction. This interaction could be a user making a purchase, clicking a page, viewing a movie, listening to a song, or rating a restaurant.

There are different recommendation algorithms that learn a users preference to items and output recommendations. A classical method is matrix factorisation which have evolved through time to deep learning methods and more recently gave way to graph

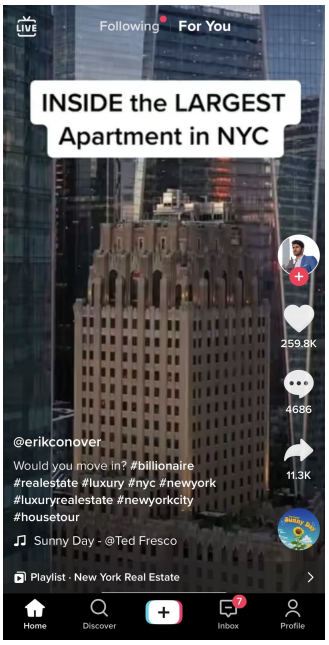

FIGURE 1.1: TikTok For You Page [6], powered by a recommendation system

FIGURE 1.2: Related item recommendation on Amazon

neural networks that can learn higher order information from the given data and provide richer recommendations.

Once a recommendation algorithm has been trained, it can generate recommendations for users. Traditionally, recommendation systems have been analysed using various accuracy metrics, however other paradigms of evaluation have surfaced that allow a more holistic understanding of the system. One of these is beyond-accuracy metrics that attempt to measure other qualities of the recommendation system, such as the diversity of recommendations. The other is fairness. Fairness is a vague term that can have different meaning depending on the application, though in recommendation systems, fairness generally concerns analysing discrete groups of users or items and comparing the discrepancy in the measure of an accuracy or beyond-accuracy metric between the groups.

In the data, some users will have more interactions than others and by the nature of

machine learning algorithm, those users with more data or interactions (high-activity users) generally more accurate predictions than those with lesser data and interactions (low-activity users), an example of unfairness.

1.1 Thesis Outline

A more in-depth background on what recommendation systems are, what data they require, their mathematical representation, algorithms for recommendation, how they work in industry and how they are evaluated are the subject of Chapter 2.

In Chapter 3 we will provide a brief review of fairness in machine learning and bias in recommendation systems, as both provide intuition and pave the way for fairness research in recommendation systems. We will then review the literature on fairness in recommendation systems, with a particular emphasis on user fairness. In doing so, we identify two gaps in the literature:

- Comparison and understanding of what off the shelf algorithms (without explicit fairness optimisation) are fairer for recommendation systems.
- User fairness is almost always computed for accuracy metrics rather than beyond-accuracy, and never together.

Based on these uncovered gaps in the literature, the objective of this thesis is primarily to understand whether graph neural network algorithms result in fairer outcomes for users than matrix factorisation algorithms for recommendation systems. We posit that graph neural networks' ability to learn higher order information in the data may lead to fairer recommendations between user groups, specifically between low and high activity users.

In order to understand the mathematics of matrix factorisation algorithms and graph neural network algorithms and consider technically why graph neural networks may allow fairer outcomes, we will provide a comprehensive mathematical overview of algorithms in each category. In Chapter 4 we will cover the evolution of select matrix factorisation methods for recommendation systems and synthesise a motivation for how graph neural

networks. In Chapter 5, we will go further in depth on a few graph neural network techniques, following the structure of Chapter 4.

In Chapter 6, we will conduct comprehensive experiments using a graph neural network algorithm and a matrix factorisation algorithm for recommendation. We will define our own fairness metric that can be computed across any accuracy or beyond-accuracy metric and compare the fairness for each algorithm between discrete user groups. Specifically, we will split users into high activity and low activity and compute fairness for each selected accuracy and beyond accuracy metric for each algorithm and compare the results. Our experiments aim to answer whether graph neural network approaches lead to fairer recommendations than matrix factorisation in this manner across a suite of accuracy and beyond-accuracy metrics.

The findings of this thesis will be of value to practitioners and researchers alike, providing actionable insights into what off the shelf recommendation algorithms can be chosen to lead to fairer outcomes for users. Given the pervasive nature of recommendation systems, this can lead to safer and fairer user experiences online.

Chapter 2

Recommendation Systems

Almost every online service that provides content to users utilise recommendation systems, having been recognized as one of the most effective methods for personalizing content for users in an age of information overload. Recommendation systems filter and present content (items) to users out of a large corpus, returning a personalised ranked list of such content to a user.

Recommendation systems allow users to find content that is relevant to them as well as promoting discovery of content that a user might like that they may not find on their own. Depending on the objective the system is optimised for, they can keep users on site for long periods of time [1], or increase profits by leading to more sales.

Although it may seem that recommendation systems are more prevalent now than ever before, they have been having impact for a while, generating large value for businesses. In 2016, 80% of movies on Netflix arose from recommendations and they valued their recommendation systems at $1 billion per year [7]. In 2010, 60% of video clicks on YouTube came from home-page recommendations [8].

2.1 Data for Recommendation

The interactions, or feedback given by a user to a website, product or item, (user-item interactions), can be classified into two domains - implicit feedback and explicit feedback. Explicit feedback is where the user expresses their true preference for an item,

for example a rating out of 5 for a movie on Netflix, or a rating given on a Google review for a local restaurant or landmark.

Explicit feedback is not always available, and the data is usually quite sparse. More commonly, implicit feedback is used to train recommendation systems. Implicit feedback data is usually far more readily available. Implicit feedback includes purchases, clicks or searches for an item on site, and it is assumed these interactions are all positive signals for a recommendation system. This can be a downside in the sense that there are no negative signals. As we will see, a way around this for training a recommendation system is to take the absence of a positive signal as a negative signal, if the user has not yet had interaction with a particular item.

Interactions for implicit feedback are usually encoded into an interaction matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$, where M is the number of users and N is the number of items. The interaction matrix will be filled with entries $\mathbf{A}_{ij}$ for user i and item j, equal to 1 if the user has interacted with that item, and either empty or 0 if the user has not interacted with that item (yet). In the case of explicit feedback, the matrix entry $\mathbf{A}_{ij}$ will contain the users rating of that item, typically between 1 and 5.

2.2 Recommendation as Mathematical Problem

In casting a recommendation system as a mathematical problem, there will be a set of users $\mathcal{U} = \{u_1, u_2, \ldots, u_M\}$ and a set of items $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$, where M is the number of users and N is the number of items. Data collected is in the form of user-item interactions, between the user set $\mathcal{U}$ and the item set $\mathcal{V}$, as a set of triples $\{i, j, r\}$ for user i and item j, drawn from an unknown distribution, $r \in Z$ is the feedback label from either ratings, purchases, clicks or views of the item, usually $0, 1$ or an explicit integer rating. The user-item interactions are presented as a feedback matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$, where $\mathbf{A}_{ij} = r$.

The task in the recommendation problem is to predict preference scores for every user to every item $\{\mathbf{S}_{ij}, \forall i \in \mathcal{U}, \forall j \in \mathcal{V}\}$, even if the user has not interacted with that item in the data provided. From here, a ranked list can be produced for each user i, known as the top-K recommendation list of K items, sorted by the predicted preference scores. The top-K list for user i would be $\{v_1, v_2, \ldots, v_K\}$.

2.3 Recommendation Algorithms & Approaches

Methods for generating recommendations can be broadly categorised into two classes: content based and collaborative filtering. Content based approaches focus on inherent item attributes to recommend other items similar to what the user has shown a preference for, whereas collaborative filtering approaches operate under the assumption that similar uses will like similar items, based on the user-item interaction data. In both approaches, items can be sorted by predicted preference score for the user and the top K items can be presented back to the user as a ranked list of recommendations.

2.3.1 Content Based Recommendation

Content based approaches consider each user in isolation, so that other users preferences or profiles will have no impact on the recommendations the user receives. Recommendations are made based on the individual user profile or preferences. The user profile or preferences can be gleaned from the items they have interacted with via implicit or explicit feedback in the past, or the user could suggest their preferences to particular content upon signing up to a website. Items that have similar attributes to the user profile can then be recommended.

As an example, consider some hand engineered features from an example on the Google Play store in Figure 2.1. Item attributes can be represented as columns, including the genre of an app (Education, Casual, Health and Healthcare) or the app developer (TimeWastr and Science R Us). Each item (in this case, an app) can be represented as a row, and there will be an entry in the given (row, column) pair if the app has that attribute. For example, the app in the first row is an Education app, published by Science R Us.

The user can be represented in the same feature space, and the user profile can be represented as a row. For example, the given user has shown preference for Education and Healthcare apps and the app developer Science R Us. This information could have come from prior user interactions (downloads) of the item (app), or explicitly given in a preference survey when signing up for a Google Play account.

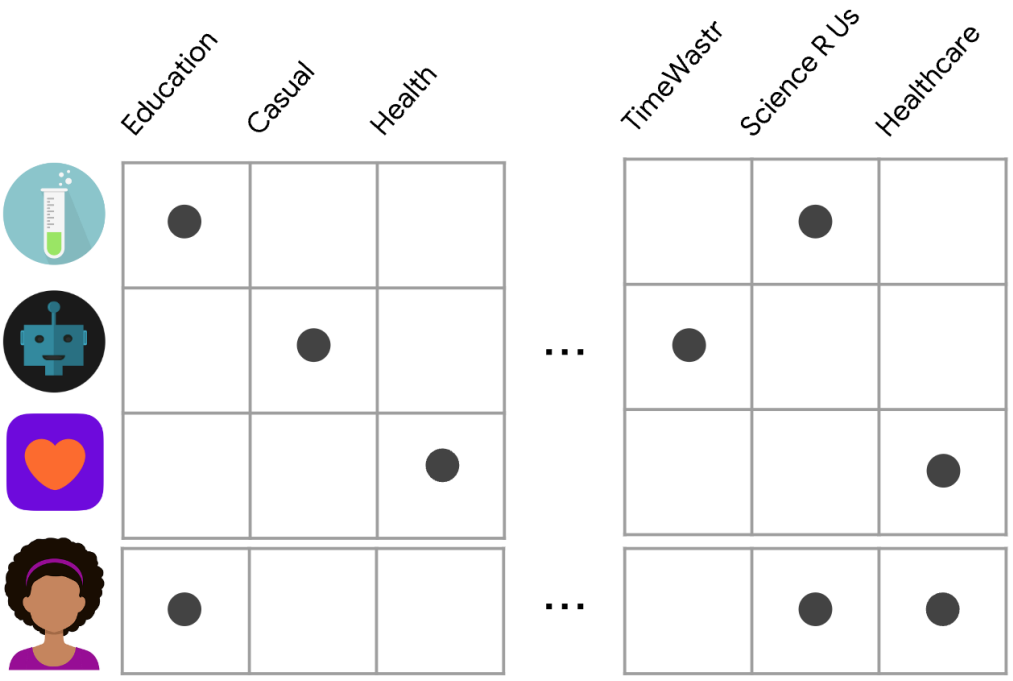

FIGURE 2.1: Content based recommendation toy example from [9]

In order to rank each app for the user, a method for measuring similarity must be used, of which a common approach is using a dot product. Representing the user vector (user embedding) as $\mathbf{u}_i$ and the item vectors as $\mathbf{v}_j$ for each item j, the predicted preference score (measured by the dot product), $\mathbf{S}_{ij}$, for each item, can be calculated:

$$\mathbf{S}_{ij} = \mathbf{u}_i \cdot \mathbf{v}_j^T \tag{2.1}$$

In this example,

$$u_i = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$v_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$v_2 = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$v_3 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

And so,

$$S_{i1} = 2, S_{i2} = 0, S_{i3} = 1$$

In a ranked list with $K = 3$, the recommendations would be app 1, followed by app 3, followed by app 2. In the case where $K = 1$, just the Education app (app 1) would be recommended to this user.

Content based approaches have certain disadvantages, including requiring domain knowledge and struggle to recommend new content to users outside their interests, which can lead to a filter bubble phenomena [10]. In this thesis, we will only consider the collaborative filtering case going forward.

2.3.2 Collaborative Filtering Recommendation

Collaborative filtering approaches are domain independent and operate under the hypothesis that if user i and user k like the same item, they will have shared interests. These interests are usually modelled via specific machine learning algorithms for recommendation, and the algorithmic approaches to learn these interests and generate recommendations can be divided into three areas: shallow methods, deep neural network methods, and graph neural network methods.

Shallow methods were the first proposed approaches for recommendation systems, dating back to 1998 [11]. Matrix factorisation [12] is the most well known shallow collaborative filtering technique, which learns user preference from the feedback matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ by projecting users and items into embedding vectors in a joint factor latent space of a given dimensionality d. User item preference is modelled by the inner product of user and item embedding pairs in the space. This approach does have limitations, including the choice of the inner product as the interaction function, as well as the cold start problem: where users and items receive poorer recommendations, or no recommendations at all, due to inadequate data [13].

Using matrix factorisation based collaborative filtering as a building block, the field has seen rapid growth in algorithmic advancement for recommendation systems. Deep learning based methods were discovered and used successfully for recommendation systems. One of the most famous industrial papers published on recommendation systems [1] used deep learning in the recommendation architecture to generate recommendations on YouTube. This approach was shown to have significant performance improvement over shallow methods.

As discussed in [14], deep neural networks promised a lot of benefit for recommendation, including being able to capture non-linear interactions between users and items and include data beyond just user-item interactions, encoded as side information on top to generate richer recommendations. [15] was proposed, motivated by the idea that the inner product as an interaction function is insufficient to model complex feature interactions between user and items. They address this by proposing learning the interaction function using deep neural networks.

While deep learning methods are more powerful than traditional matrix factorisation techniques, there are still limitations present, including ignoring higher order structure available in the data. In both [15] and [12], the loss function attempts to minimize the difference in predicted vs. actual user item interaction. During training, the users embedding is updated based only on items they interacted with. Thus, these approaches consider interactions in isolation, without taking into account higher order structure explicitly, such as other users interactions.

To address this, graph neural network based approaches have emerged as the dominant approach to tackle the recommendation problem. They have become quite popular due to the vast increase in graph data, including social networks. Graph based approaches allow use of many forms of data, linking them in a graph.

In this paradigm, users and items are treated as nodes on a graph. To take into account higher order structure, the neighbourhood of each node is considered. Essentially, each nodes embedding is generated by exploiting graph neural networks to aggregate information from the local neighbourhood around the node, acting as a way to propagate the embeddings to encode higher order structure.

In the aforementioned deep learning and matrix factorization approaches, if we consider them as a graph, each node (user or item) only uses the information of its first order neighbours via user-item interaction. Graph neural network approaches allow nodes to access second, third and higher order information from other nodes in the neighbourhood, learning richer representations that lead to better recommendation.

Graphs also allow side information on top of user-item interaction data from both the user and item side. For example, this can include adding demographic information for

users (age, geography, location, preference) and relevant information for items (genres, metadata, authors).

2.4 Industrial Recommendation Systems

The recommendation algorithms described previously generally only constitute a part of the entire recommendation system architecture in an industry setting. Recommendation systems in industry and recommendation systems in a research setting typically differ. In the research setting, the goal of recommendation is as described above - prediction of user preference to items, based on historical user item interaction data, potentially taking side information into account. Based on this preference, return a ranked list of items to the user. [16] refers to this as offline recommendation.

Industrial recommendation systems typically extend the offline approach by adding an online component that takes in a request for a recommendation in real-time to compute recommendations on the fly. Rather than just using the learned embeddings, companies take in other real-time information, usually through data streams, such as trending items, popular items and recent items a user has interacted with, to prescribe better recommendations in the moment [1].

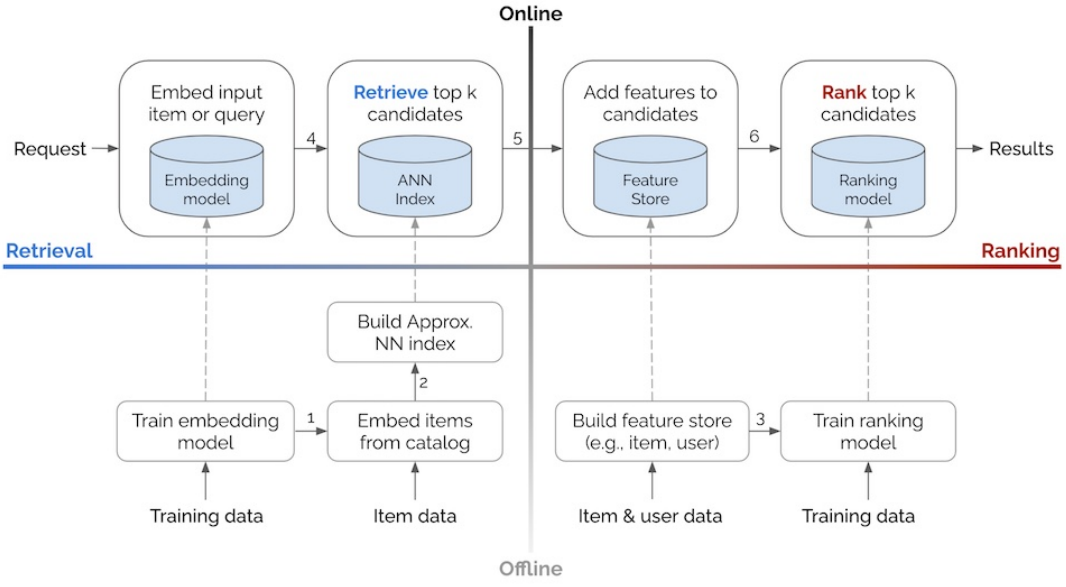

FIGURE 2.2: Industrial recommendation system pipeline from [17]

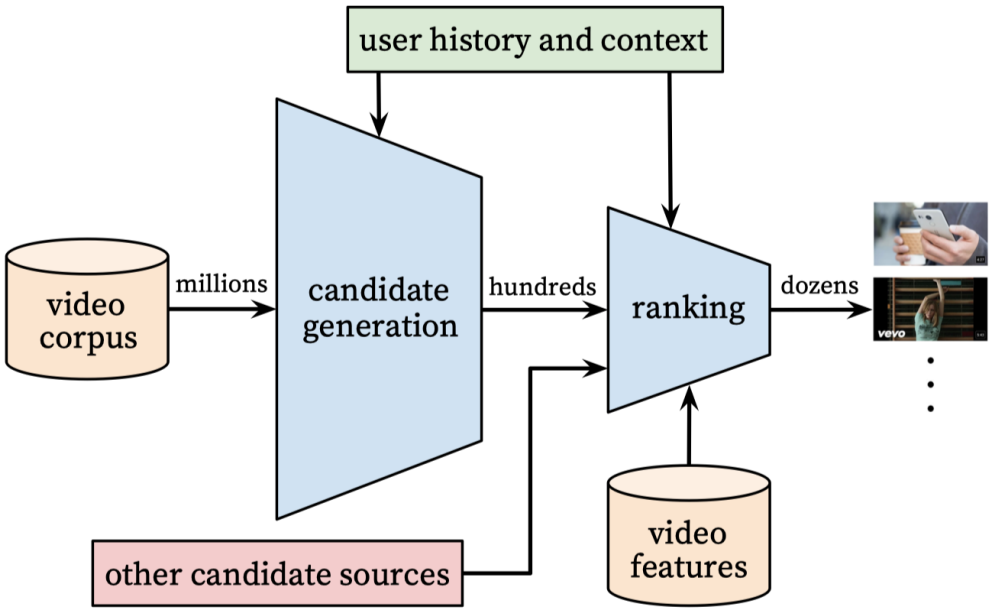

FIGURE 2.3: YouTube's industrial recommendation system pipeline from [1]

Industrial recommendation systems usually are discretely split into a candidate retrieval or candidate generation section and a ranking section [17]. Usually, a re-ranking step for business purposes (promotions, etc.) is implicit in the ranking step too. A general depiction of this architecture is shown in Figure 2.2 and an example from YouTube's recommendation system is shown in Figure 2.3.

These sections also can be cross-sectioned into offline and online parts. The offline parts are usually trained offline as they can take some time, whereas the online parts are computed on the fly (in real-time) when a request comes in and returned to the user, usually within the time it takes a page to load, which means these parts need to be efficient.

Candidate generation or retrieval aims to reduce the huge amount of possible items to recommend, sometimes in the case of billions [18], to a smaller amount, usually in the thousands. The learned embeddings can be used to retrieve the top most similar items to the user. Items can be gathered from other candidate sources, including recent trending items and items the user has recently interacted with, using real-time streaming data. The ranking step is slower but more precise than candidate generation and returns a final ranked list of recommendations, taking into account other features such as user demographics and item metadata.

While it is important to understand how recommendation works in industrial practice as these recommendations affect almost everyone everday, the rest of this research will consider the offline recommendation scenario. The reason for this is that the data and algorithms needed for real-time recommendation are proprietary and not released. In the research setting, we split data into a training and test set, with the objective of achieving the best accuracy in predicting which items the user will interact with in the test set [16]. The recommendation scenario that we consider in this thesis (embedding generation and predicting item preference for a user) falls into the lower left hand quadrant of Figure 2.2, and is used for candidate generation in the online industrial setting.

2.5 Evaluating Recommendation Systems

There is no one approach or metric that is agreed upon for generating a holistic overview of a recommendation system, largely due to the fact that recommendation systems are used in many different scenarios and have different impacts on end users. Historically, the evaluation of recommendation systems has been largely focused on the accuracy of recommending items to users. This is measured using a metric computed on the hold-out test data from the user-item interactions. Some interactions will be held-out from training, then the recommendations for each user in this test-set will be checked against their true interactions.

Beyond-accuracy and fairness measurements have been proposed as both alternative and supplementary methods for evaluating recommendation systems, in an attempt to yield a more holistic view of the system than accuracy metrics can provide alone. There are other metrics that businesses may use in industry to evaluate their recommendation systems such as time spent on a product, click through rates and user engagement, however they are outside the scope of this thesis.

2.5.1 Accuracy Metrics

From a comprehensive study on 85 papers, [19] found that there are no concrete evaluation benchmarks used consistently across the board. However, of the metrics used, they found the most popular accuracy metrics were precision, recall, MAP (mean average precision), hit-rate, MRR (mean reciprocal rank) and NDCG (normalised discounted

cumulative gain). When training recommendation algorithms, hyper-parameters will typically be tuned to optimise one or two metrics, however this may not be optimal for all metrics.

All of these metrics can be computed for a set of top-K recommendations and are usually denoted by metric@K, for example NDCG@K. In defining these metrics, we will use the following notation, adapted from [20].

- i refers to a user $i \in \mathcal{U}$
- j refers to an item $j \in \mathcal{V}$
- $rec_K(i)$ refers to the recommendation ranked list of K items for user i
- $rel(i)$ refers to the list of relevant items (interacted with) in the test set for user i
- $rank(i, j)$ refers to the position of item j in $rec_K(i)$
- $\mathbb{1}$ is an indicator function
- $rating(i, j)$ is the rating of an item j in the test set for user i, 0 or 1 for implicit feedback, 1 if the item was interacted with.

NDCG puts an emphasis on more relevant items being higher on a recommendation list. It does this by discounting the importance of each recommendation position in the list. NDCG@K for a user i is defined as:

$$\text{NDCG@}K(i) = \frac{\text{DCG@}K(i)}{\text{IDCG@}K(i)} \tag{2.2}$$

where

$$\text{DCG@}K(i) = \sum_{l \in rec_K(i)} \frac{2^{rating(i,l)} - 1}{log_2\left(rank\left(i, l\right) + 1\right)}. \tag{2.3}$$

and IDCG@K is the highest possible value of DCG@K, calculated via treating the sorted test set by predicted preference score for a user i as a prediction of length K.

Hit-rate measures whether at least one item in the ranked list is relevant, defined as

$$\text{HR@}K(i) = \mathbb{1}\left[rel(i) \cap rec_K(i) > 0\right]. \tag{2.4}$$

In this way, hit-rate will be equal to 1 if just one item in the recommendation list is in the test set for the user, and 0 otherwise. Hit-rate is simple to calculate and interpret, however doesn't take into account how many relevant recommendations were in the list, only considering if one relevant item is present.

MRR measures where the first relevant item is in the recommendation list. For each user, it is calculated as

$$\text{MRR@}K(i) = \frac{1}{\min_{l \in rel(i) \cap rec_K(i)} rank(i, l)}. \tag{2.5}$$

If no relevant item was predicted, the MRR for a user is 0. MRR is simple and puts particular emphasis on the first relevant recommendation, which does not take into account the strength of other recommendations in the list. For instance, if $K = 5$ and one recommendation list has 4 relevant recommendations in positions $2-5$, and another recommendation list has 1 relevant recommendation in position 1, the latter list would be ranked higher.

Precision and recall are common metrics used in machine learning classification tasks that can be re-purposed for recommendation systems. Precision measures the percentage of items in a recommendation list that were interacted with in the users test set. Generally, precision will decrease as K increases, which is the opposite to hit-rate. Precision is computed as

$$\text{Precision@}K(i) = \frac{|rel(i) \cap rec_K(i)|}{K}. \tag{2.6}$$

Recall measures the ratio of items in a recommendation list that were relevant to the total number of relevant items for a user:

$$\text{Recall@}K(i) = \frac{|rel(i) \cap rec_K(i)|}{|rel(i)|}. \tag{2.7}$$

All of the above accuracy metrics can be averaged across all the users in the test set to get a measure for the recommendation algorithm on a given dataset.

2.5.2 Beyond Accuracy Metrics

Solely assessing recommendation systems based on accuracy metrics may fail to capture essential aspects of their overall performance, especially given recommendation systems

can aim to promote discovery of new content and can sometimes lead to filter bubbles and echo chambers [21, 22]. As a result, alternative metrics beyond accuracy have been suggested to address this issue. Among these metrics are diversity, novelty, and coverage [23].

Diversity aims to assess the range of recommendations provided by a system. It emerged in the context of information retrieval to attempt to account for ambiguity in a user's preferences. By recommending a diverse set of items to a user, the likelihood of satisfying their potentially ambiguous preferences is improved. To measure the diversity of items in a list, it is important to evaluate their similarity. Using a similarity metric between the items achieves this, although this approach has limitations as it relies on learned representations of items rather than the user's perception of diversity. Nonetheless, using a similarity metric remains a viable method and the approach we will use in this thesis.

Diversity is measured for a list of recommendations as the difference in similarity between the items recommended. This is easy to compute from the embeddings of the list of the items recommended and we will use a dot product as the similarity measure:

$$\text{Diversity@}K(i) = \frac{\sum_{l \in rec_K(i)} \sum_{j \in rec_K(i) \setminus l} v_l \cdot v_j}{K(K-1)}. \tag{2.8}$$

While diversity is calculated at a per user basis then aggregated, coverage is a metric that is calculated for the entire system as a whole and not defined at an individual user level. In this thesis, we consider coverage as item coverage, which measures the proportion of items in the item set that are recommended to all users. This is a simple metric but useful, as some recommendation systems with low coverage can mean users receive recommendations for similar items, reducing diversity.

Coverage is defined as the fraction of items that appear in at least one recommendation list out of all possible items:

$$\text{Coverage@}K = \frac{|\cup_{i \in \mathcal{U}} rec_K(i)|}{|\mathcal{V}|}. \tag{2.9}$$

In a somewhat similar vein to diversity, novelty is calculated for each item, and then

summed up for each recommendation list. Novelty aims to provide a metric that measures how unique or original a set of recommendations is to a user. When only user-item interaction data is available, calculating novelty for an item can be ambiguous. Considering any item a user hasn't interacted with as novel would constitute a significant proportion of the item catalog for most users. Just because a user hasn't interacted with an item also does not mean they are not aware of it. To truly calculate novelty, user feedback would be required, which is not feasible in most applications.

Thus, an approximation of novelty is necessary, and using an items popularity is a worthwhile candidate [24]. To calculate the popularity of an item, consider the number of interactions an item has. Novelty per item can then be defined as an opposite of popularity, such as $1 - p(j)$, or a negative logarithm such as $-\log_2 p(j)$ where $p(j)$ is the calculated popularity of item j.

Formally, this can be calculated as the fraction of users who interacted with item j:

$$p(j) = \frac{|\{i \in \mathcal{U}, rating(i, j) = 1\}|}{|\mathcal{U}|}. \tag{2.10}$$

Then, novelty for each a recommendation list can be calculated as:

$$\text{Novelty@}K(i) = \frac{\sum_{l \in rec_K(i)} -\log_2 p(l)}{K}. \tag{2.11}$$

Chapter 3

Fairness in Recommendation Systems

Given the influential role recommendation systems play in society, it is of the utmost importance that they are fair. Fairness in recommendation systems can be understood from different lenses, be it from the user side, item side, or both together. Unfair recommendation systems arise from bias in the overall system, which we will briefly review, then review the current state of fairness in recommendation systems research. Firstly, we will cover general fairness in machine learning as this is a well researched field and is a building block for fairness in recommendation systems.

3.1 Fairness in Machine Learning

In recent years, as machine learning applications play more of a role in society, fairness in machine learning has attracted a lot of attention both in research communities and the general public. Many private companies, government bodies and research institutions have contributed to this topic, including the Australian government [25, 26].

Fairness in machine learning is broad and covers many areas and applications, usually considered from a users point of view. It is generally agreed that a fair and ethical machine learning system should be inclusive and accessible, and should not involved or result in unfair discrimination against individuals, communities or groups [26]. While in

an ideal world we would aim for a machine learning algorithm to be fair and accurate, these two objectives generally conflict with each-other [27].

Machine learning applications are widespread and fairness considerations are important as the machine learning systems can make key decisions that affect human life. [28] considers the notion of algorithmic fairness specifically in the domain of judicial decision making, where algorithms are used to decide whether defendants are suitable or not to be released into the community. The research found that black defendants are more likely to be classified (incorrectly) as high risk.

Fundamentally, fairness issues in machine learning systems arise from bias. Bias in a machine learning system can be attributed to the training data itself being biased, the algorithm being biased, or a combination of both where the algorithm can perpetrate the bias from the training data.

Bias can lie in the data itself, where for instance the data generating process is biased and or the collection of data for training a machine learning algorithm is biased. Statistical bias [29] is an example of this, where data that is used for training an algorithm may not be representative of the entire population. This unrepresentative sample can result from selection bias, the phenomena that data collected comes from a non-random portion of the population - for example, if certain minorities are discriminated against in a hiring process and not selected for a job, and a machine learning algorithm is then trained to assist hiring, these biases will be propagated into the predictions, perpetrating the problem [30].

An essential aspect to consider is the impact of minority groups. In such cases, the data might be adequately sampled and no statistical or selection bias is present. However, pre-existing biases may still exist in the population. By definition of a minority group, even if the data collection is unbiased, due to their limited representation, machine learning algorithms can perform poorly at prediction for these minority groups and the decisions made off the back of the machine learning systems decision will only perpetrate this bias.

These biases can also be interlinked with the feedback loop phenomena which we will see is inherent in recommendation systems. In general machine learning, a feedback loop can arise from training models on biased data, then using these predictions to make

decisions. These decisions then appear in the data when a model is re-trained in the future.

Overall, there are many ways bias can leak into a machine learning pipeline, leading to unfairness. It is important to deal with this, and the field of general machine learning fairness is relatively mature. Generally, methods to address the unfairness in machine learning algorithms fall under three discrete categories: pre-processing methods, in-processing methods and post-processing methods.

Pre-processing methods deal with data transformations or augmentation in an attempt to remove underlying bias in the data. If the model can be trained on this purer data, the model will be more fairer. Methods to achieve this include label augmentation and sampling [31, 32].

In-processing methods intervene on the algorithm itself, intending to augment the training process, recognising that machine learning techniques can become biased due to dominant features [33]. This can be done by regularisation, including fairness terms in the objective or imposing fairness constraints.

Post-processing methods deal with altering the output of the algorithm. Usually these approaches are model agnostic, typically trading off fairness for accuracy, as machine learning systems are optimised for accuracy - by augmenting the output, the accuracy will be diminished.

3.2 Bias in Recommendation Systems

To understand how recommendation systems can exhibit unfairness, just like general machine learning systems, it is critical to understand how bias can enter the system. Bias can enter and be amplified in a self-reinforcing feedback loop pattern at any stage of the recommendation system.

As an application of machine learning, recommendation systems rely on data, and this data can be biased. The data is generated from human users interacting with items in the content library in the form of user-item interactions. Not all users will have equal exposure in the collected dataset, especially if the interactions captured are purchase interactions. Certain groups of users will have lower user-item interactions, which can

stem from demographic problems (lack of money, time and exposure to technology). As we will explore, the nature of collaborative filtering algorithms leads to these groups receiving poorer, more unfair recommendations, shown in our experiments in Chapter 6 and echoed in [34], [35]. As mentioned in Chapter 1, most recommendation algorithms are optimized for an accuracy metric, with no explicit optimisation for bias and fairness correction.

Biased data can then be propagated into a biased recommendation algorithm and biased output can result - recommendation lists themselves can be biased, in the form of popularity bias. The items that the most active users interact with, or just the most interacted items in general, will tend to be over-recommended. As a result, items with less interactions will have a lower chance for further exposure. In this way, the popularity of the higher interaction items is reinforced, known as the Matthew effect.

While human bias can affect the data that is fed into a recommendation algorithm, it can also creep into the consumption of the recommendations in the form of position and conformity bias. Position bias arises when users have a tendency to interact with items that appear higher in recommendation lists. Conformity bias refers to users tending to behave similarly to the other groups of people.

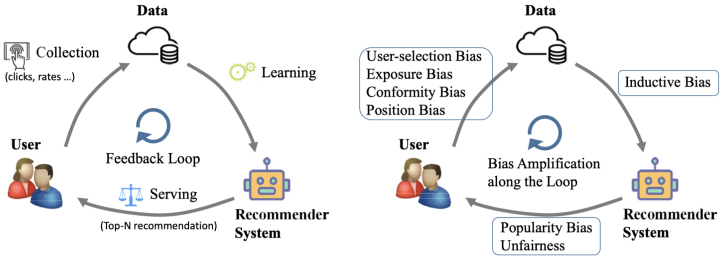

FIGURE 3.1: Recommendation system feedback loop from [36]

Through this loop depicted in Figure 3.1, both users and the recommendation system are in a process of dynamic evolution, where personal interests and behaviors of users can be influenced by recommendation. The recommendation system can lead to a self-reinforced bias propagation by training on new interaction data that could have been affected by the biased recommendations in the first place. Figure 3.2 shows the relations between the biases, ultimately leading to unfair recommendations.

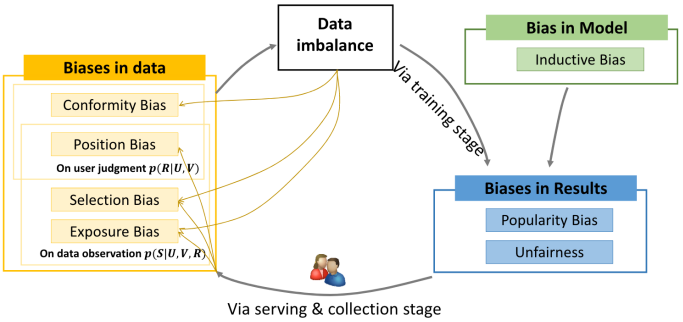

Fig. 7. Relations between seven types of biases.

FIGURE 3.2: Recommendation system bias feedback loop from [36]

3.3 Fairness in Recommendation Systems

While there is a wealth of research in fairness in machine learning and established methods for increasing fairness, tackling fairness in recommendation systems is not as simple as extending current methods for general machine learning. [37] note that the methods for fair machine learning are not easily extendable to recommendation, for multiple reasons including that recommendation systems have multi-sided impacts on the users receiving recommendations and the items that are recommended.

Survey papers and tutorials [37, 38] split fairness in recommendation systems into multiple paradigms: user and item fairness, individual and group fairness, static and dynamic fairness, single-sided and multi-sided, and more. We propose a simpler grouping into three facets of fairness in recommendation systems: user fairness, item fairness and multi-sided fairness.

Under this setting, user fairness concerns fairness amongst the users that receive the recommendations. Fairness in this domain aims for similar users or groups of users to receive recommendations of similar quality. Item fairness concerns the items that are actually being recommended, and fairness here means items receive fair exposure in the recommendations. Multi-sided fairness refers to where a system may want to balance fairness for both users and items simultaneously. While all are important, in this thesis we focus on user fairness, however we will review all three here, albeit briefly for item and multi-sided fairness.

3.3.1 User Fairness

Individual fairness and group fairness are subsets of user fairness. Individual fairness requires that similar users should receive similar treatment, whereas group fairness requires that groups, usually protected or minority groups, should receive similar recommendation treatment to other non-protected groups.

In the recommendation domain, individual fairness lacks richness, however, one approach [39] considered individual fairness in the context of group recommendation, where a recommendation is prescribed to a group of friends or family. They consider fairness in terms of individual utility received between the members for the same recommendations and seek to reconcile this using multi-objective optimization. They measure fairness based on the imbalances of the relevance of items for members in each group.

Group fairness, specifically with two user groups, is more well researched and will be the primary focus of the thesis. Throughout the thesis, we will employ a generic user-group-fairness framework to measure fairness between two groups. This measure is a function of the user groups and given accuracy or beyond accuracy metrics computed on the user groups, denoted as

$$Fairness(G_1, G_2, M), \tag{3.1}$$

where G_1 is one group of users, G_2 is the other group of users such that $G_1 \cap G_2 = \emptyset$ and M will be any of the accuracy and below accuracy metrics that we define in this chapter. In this manner, we can compute a fairness score between user groups for a given accuracy or beyond-accuracy metric.

The paper [35] studies user group fairness specifically in the context of recommendation systems that are generated using a knowledge-graph based algorithm. The paper considers fairness by comparing two groups of users: high activity users and low activity users (split based on the number of user-item interactions per user), and conducts fairness analysis between these groups. They find that while inactive users make up the majority of the user base, they receive poorer recommendations than the active users. They argue this is because the inactive users have less training data and their recommendations will be biased by the users with more user item interactions. The authors propose a post-processing heuristic re-ranking technique to bridge the gap in fairness

between the groups. When applied, they note increases in recommendation performance overall, especially for the inactive group and the fairness metric improving by lowering. The metric they use to define fairness is an absolute difference in the average of a given metric, of which they use NDCG and F1, defined formally under our framework (3.1) as

$$Fairness(G_1, G_2, M) = \left| \frac{1}{G_1} \sum_{i \in G_1} M(i) - \frac{1}{G_2} \sum_{i \in G_2} M(i) \right|. \tag{3.2}$$

[34] considers fairness in recommendation from the user group perspective, similarly grouping users into high activity and low activity users based on their activity levels in the dataset. They do multiple splits based on number of interactions, maximum purchase price and total consumption from purchase datasets. The authors note from experiments that recommendation systems will exhibit unfairness between these groups of users. Active users receive better recommendations than the inactive users, who make up the majority of users. To circumvent this, they propose a similar re-ranking approach, which they validate via experiments and note better overall recommendation performance as well as fairer recommendation between the groups. They use the same fairness metric (3.2) as in [35].

[40] conducted an generalisability extension on [34], repeating the approach for multiple algorithms and multiple datasets. They generally found consistent discrepancy and unfairness between high activity users and low activity users according to the metric (3.2).

Most approaches like the above split users into discrete groups based on activity, age, gender, nationality and others. However, [41] split users into groups based on the personality trait measures opennness, conscientiousness, extraversion, agreeableness, neuroticism. They gathered data from Twitter to mine users music preferences and created a recommendation system to recommend music. They mined tweet data to understand users personality scores, then split users into high or low groups for each personality trait. They compared recall and NDCG for the groups, without an explicit fairness measure, simply commenting on the differences in the metrics, finding that there was a statistically significant difference in these metrics for the high and low groups for neuroticism and openness.

[42] split users into groups based on age and gender and produce recommendations based on two datasets (last-fm and movielens), and compare an accuracy metric (NDCG) between the groups. They found differences between groups, for example, women had better accuracy on a given dataset, however this wasn't consistent amongst the other dataset chosen.

While all of these approaches consider comparing fairness amongst users via accuracy metrics, there isn't any research that looks at fairness between users from beyond-accuracy metrics computed directly from the recommendation output, using the measures we defined in Chapter 2, (2.8), (2.9) and (2.11).

One paper [43] considered user fairness from a beyond-accuracy perspective, drawing the beyond-accuracy measures from surveys, for example asking the user how novel and diverse they *felt* the recommendations they received were, rather than directly computing these metrics from the recommendation results. They split users into multiple groups based on characteristics, including gender, age and the personality traits used in [41]. The data was gathered from a private company (Taobao) and four algorithms were applied. The users were then surveyed on how relevant, diverse, novel, unexpected or serendpitous they thought the recommendations were, then aggregated at a discrete group level within each characterstic. Here, no explicit fairness metric was used, just the scores for each group were compared. They found that serendipity as a survey metric demonstrated unfairness between groups consistently, and that age and curiosity too had distinct unfairness between the user groups.

[44] considers the idea of fairness amongst users after a post-processing algorithm is applied to improve recommendation diversity. The authors measure fairness as the disparity between the optimal recommendations before and after post-processing, comparing the sum of preference scores before and after. They find that as the lists become more diverse, the difference between the score disparity in recommendation lists increases. The metric they use for fairness is defined as score disparity, which can be calculated by the ratio of any metric for recommendation quality after post-processing compared to the same metric computed on the original recommendations.

3.3.2 Item Fairness and Multi-Sided Fairness

Most of the research in fairness in recommendation systems domain has considered fairness from the item side. While fairness on the user side usually compares the quality of recommendations between users based on some metric, item-based fairness is usually measured by how much exposure an item receives from a recommendation algorithm.

Generally, research in this area has found that traditional collaborative filtering algorithms are susceptible to over-represent popular items, which can minimise the exposure of long tail items. This is known as popularity bias and is well researched [45–47]. It is somewhat similar to user fairness, where the high-activity users (items) generally receive better recommendations. In the item case, the popular items have more data points in the interaction data that are then learned into item embeddings, which are propagated by the algorithm and leads to them being recommended more often. Many techniques have been proposed to combat this, aiming to increase coverage of items by having a policy to over-prescribe the less-popular items.

Multi-sided fairness considers both user fairness and item fairness together [48]. [49] researches the trade-off between consumer (users receiving recommendations) fairness and producer (those that produce an item, such as an artist, movie director, business, etc.) fairness, attempting to account for competing interests when prioritizing one side over the other. Consumers want good recommendations, but typically good recommendations can lead to just a few providers getting their items exposed. They seek to trade this off by increasing exposure while keeping consumer utility high. In a similar vein, [50] also addresses the idea of balancing interests and keeping things fair between producers and consumers in a recommendation system.

Chapter 4

Matrix Factorisation for Recommendation Systems

The literature in matrix factorisation is rich and over time a number of methods have been proposed. As mentioned in the introduction, matrix factorisation is a collaborative filtering technique which learns user preference for users $\in \mathcal{U}$ to items $\in \mathcal{V}$ from the user-item interaction matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$, where M is the number of users and N is the number of items. Matrix factorisation methods decompose $\mathbf{A}$ into two lower dimensional matrices $\mathbf{U} \in \mathbb{R}^{M \times d}$ and $\mathbf{V} \in \mathbb{R}^{N \times d}$ where d is the embedding dimension, comprised of rows of user embeddings in $\mathbf{U}$ and item embeddings in $\mathbf{V}$. $\mathbf{U}\mathbf{V}^T$ is usually denoted by the matrix $\mathbf{S}$:

$$\mathbf{A} \approx \mathbf{S} = \mathbf{U}\mathbf{V}^T. \tag{4.1}$$

Each item $j \in \mathcal{V}$ is associated with embedding $\mathbf{v}_j \in \mathbb{R}^d$ and each user $i \in \mathcal{U}$ is associated with embedding $\mathbf{u}_i \in \mathbb{R}^d$. All of the user embeddings form the user embedding matrix, $\mathbf{U} \in \mathbb{R}^{M \times d}$ and all the item embeddings form the item embedding matrix, $\mathbf{V} \in \mathbb{R}^{N \times d}$. User preferences to items are captured via the inner product of the user embedding and item embedding pairs, that is, user i's preference to item j, $\mathbf{S}_{ij}$ is expressed as $\mathbf{u}_i^T \cdot \mathbf{v}_j$, as in (2.1).

Consider a toy example in Figure 4.1 with four users and five items (movies). The feedback matrix can be decomposed into two matrices $\mathbf{U}$, $\mathbf{V}$ where $d = 2$. The resulting matrix of $\mathbf{S} = \mathbf{U}\mathbf{V}^T$ is an approximation of $\mathbf{A}$.

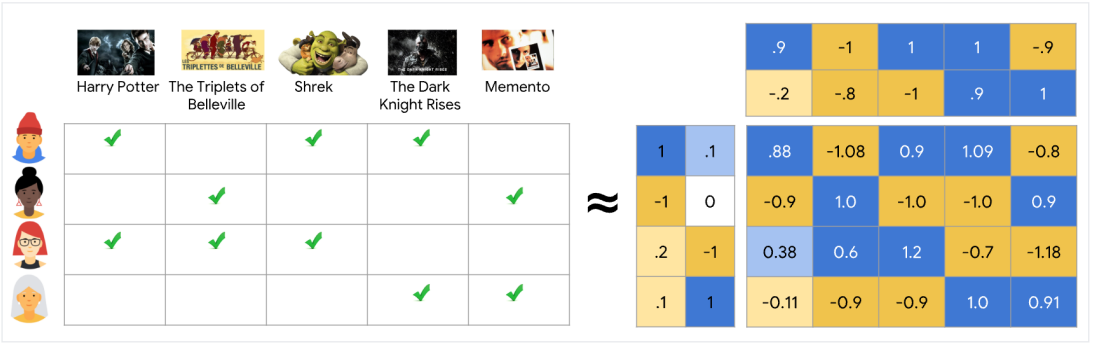

FIGURE 4.1: Matrix factorisation collaborative filtering toy example from [51]

In this example, the user item interactions are comprised of movies viewed by users and the matrix $\mathbf{A}$ is encoded as

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

The embedding matrices $\mathbf{U}, \mathbf{V}$ are learned via matrix factorisation as

$$\mathbf{U} = \begin{bmatrix} 1 & 0.1 \\ -1 & 0 \\ 0.2 & -1 \\ 0.1 & 1 \end{bmatrix}, \mathbf{V} = \begin{bmatrix} 0.9 & -0.2 \\ -1 & -0.8 \\ 1 & -1 \\ 1 & 0.9 \\ -0.9 & 1 \end{bmatrix}.$$

The predicted preference matrix $\mathbf{S}$ is then

$$\mathbf{S} = \mathbf{UV} = \begin{bmatrix} 0.88 & -1.08 & 0.9 & 1.09 & -0.8 \\ -0.9 & 1 & -1 & -1 & 0.9 \\ 0.38 & 0.6 & 1.2 & -0.7 & -1.18 \\ -0.11 & -0.9 & -0.9 & 1 & 0.91 \end{bmatrix}.$$

User 1's preference to item 1 is as in (2.1), $S_{11} = u_1 \cdot v_1^T = \begin{bmatrix} 1 & 0.1 \end{bmatrix} \cdot \begin{bmatrix} 0.9 \\ -0.2 \end{bmatrix} = 0.88$ and the recommendations for user 1 in order would be items $4, 3, 1, 5, 2$.

We will now review a few methods of matrix factorisation that explicitly learn the embedding matrices.

4.1 Singular Value Decomposition

The first matrix factorisation method proposed for recommendation systems was an application of singular value decomposition [11]. Singular value decomposition decomposes the interaction matrix $\mathbf{A}$ into three matrices:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T, \tag{4.2}$$

where $\mathbf{U} \in \mathbb{R}^{M\times M}$ and $\mathbf{V} \in \mathbb{R}^{N\times N}$ are orthogonal matrices with columns composed of left singular vectors and right singular vectors respectively, and $\Sigma \in \mathbb{R}^{M\times N}$ is a non-negative diagonal matrix, with entries as the singular values of the original matrix $\mathbf{A}$.

While computationally intensive to solve, singular value decomposition can also be used to find a low rank approximation of $\mathbf{A}$, given by

$$\mathbf{A} = \mathbf{U}_k\Sigma_k\mathbf{V}_k^T, \tag{4.3}$$

where $\mathbf{U}_k \in \mathbb{R}^{M\times k}$, $\mathbf{V}_k \in \mathbb{R}^{N\times k}$ are matrices that use the first k columns from matrix $\mathbf{U}$ and the first k columns of matrix $\mathbf{V}$. Σ_k represents the $k \times k$ principle diagonal sub-matrix of Σ. In this manner, we can represent $\mathbf{U}_k\Sigma_k^{1/2}$ as embedding vectors for users and $\mathbf{V}_k\Sigma_k^{1/2}$ as embedding vectors for items. To calculate the predicted rating $\mathbf{S}_{ij}$ for user i on item j, a dot product is computed between the i row and j row of the respective embedding vectors:

$$\mathbf{S}_{ij} = \left(\mathbf{U}_k\Sigma_k^{1/2}\right)_i \left(\mathbf{V}_k\Sigma_k^{1/2}\right)_j. \tag{4.4}$$

Singular value decomposition has its drawbacks, struggling when the interaction matrix is sparse.

4.2 Classical Matrix Factorisation

Classical matrix factorisation that decomposes the interaction matrix into two matrices instead of the three in the singular value decomposition approach rose to prominence during the Netflix Prize [52]. The idea is to minimise the sum of squared errors across all

user, item pairs i, j from the user-item interaction matrix $\mathbf{A}$ and the embedding matrices $\mathbf{U}, \mathbf{V}$. The embedding for user i is denoted by the row vector $\mathbf{u}_i$ and the embedding for item j is denoted by the row vector $\mathbf{v}_j$:

$$\min \sum_{i,j \in P} \left(\mathbf{A}_{ij} - \mathbf{u}_i \mathbf{v}_j^T \right)^2, \tag{4.5}$$

where P is the set of i, j pairs where $A_{ij} \neq 0$.

To solve this, optimisation methods such as stochastic gradient descent can be used, however, a problem that can arise is overfitting to the observed data. In matrix factorisation, this is a problem as it can easily occur, given that the algorithm is learning only from observed ratings in the dataset. To circumvent this, regularisation can be added with a regularisation parameter λ.

The loss function then becomes

$$\min \sum_{i,j \in K} \left(\mathbf{A}_{ij} - \mathbf{u}_i \mathbf{v}_j^T \right)^2 + \lambda \left| |\mathbf{u}_i| \right|^2 + \left| |\mathbf{v}_j| \right|^2 \tag{4.6}$$

and to update the parameters, stochastic gradient descent can be used, which will perform an update step with α as a learning rate:

$$\mathbf{u}_i = \mathbf{u}_i + \alpha \left(e_{ij} \mathbf{v}_j - \lambda \mathbf{u}_i \right), \tag{4.7}$$

$$\mathbf{v}_j = \mathbf{v}_j + \alpha \left(e_{ij} \mathbf{u}_i - \lambda \mathbf{v}_j \right), \tag{4.8}$$

with e_{ij} as the error from prediction:

$$e_{ij} = \mathbf{A}_{ij} - \mathbf{u}_i \mathbf{v}_j^T. \tag{4.9}$$

The stochastic gradient descent approach takes one data point $\mathbf{A}_{ij}$ at a time, updating the embedding vectors for user i and item j. An issue here is that the objective function is non-convex, which becomes an NP-hard optimisation problem to solve. As a result, stochastic gradient descent can be slow and require a lot of iterations for a decent result.

4.3 Alternating Least Squares

By fixing either of the matrices $\mathbf{U}$ and $\mathbf{V}$, a quadratic form can be obtained which is able to be solved directly. This leads to monotonically decreasing loss function. From these ideas, the method of alternating least squares was proposed.

If $\mathbf{v}_j$ is taken to be constant, the derivative of the loss function (4.6) becomes

$$-2\sum_j \left(\mathbf{A}_{ij} - \mathbf{u}_i\mathbf{v}_j^T\right)\mathbf{v}_j + 2\lambda\mathbf{u}_i, \tag{4.10}$$

which can be solved via setting this equal to zero and re-arranging:

$$-\left(\mathbf{A}_i - \mathbf{u}_i\mathbf{V}^T\right)V + \lambda\mathbf{u}_i = 0, \tag{4.11}$$

which leads to a solution of

$$\mathbf{u}_i = \mathbf{A}_i\mathbf{V}\left(\mathbf{V}^T\mathbf{V} + \lambda\mathbf{I}\right)^{-1}. \tag{4.12}$$

In a similar fashion, taking u_i constant, solving for $\mathbf{v}_j$ yields

$$\mathbf{v}_j = \mathbf{A}_j\mathbf{U}\left(\mathbf{U}^T\mathbf{U} + \lambda\mathbf{I}\right)^{-1}. \tag{4.13}$$

This can be repeated as a two-step process: updating the $\mathbf{u}_i$ vectors and holding $\mathbf{v}_j$ constant, and updating the $\mathbf{v}_j$ vectors while holding $\mathbf{u}_i$ constant, until convergence, scaling linearly with the data.

4.4 Bayesian Personalised Ranking

Another popular matrix factorisation approach is Bayesian personalised ranking [53]. It uses a different loss function to classical matrix factorisation, instead using a pairwise ranking loss, which is tailored to learn from implicit feedback. The previous approaches we reviewed only considered positive user item interactions as implicit feedback, treating the unknown interactions as negative, by representing them with a 0 in the matrix $\mathbf{A}$. Matrix factorisation algorithms are then fitted to this matrix, optimised to predict 1 for a positive user item interactions and 0 otherwise.

The authors of Bayesian personalised ranking argue that matrix factorisation doesn't lead to a proper ranking of items by not considering items that a user has not yet interacted with, which is what the recommendation problem wants to solve and then provide recommendations for. Introducing a regularisation parameter can improve this to a degree, avoiding over-fitting to the user item interaction data.

In an attempt to completely circumvent this issue, Bayesian personalised ranking takes samples of item pairs from training data and the objective is to correctly rank pairs relatively. This isn't possible for two pairs of items that a user has interacted with, or two pairs of items that a user hasn't interacted with. As a result, one positive and one negative sample are taken at a time.

To formalise this mathematically, the pairwise user-item interaction data can be written as

$$\mathcal{O} = \{(i, j, k) | (i, j) \in \mathcal{R}^+, (i, k) \in \mathcal{R}^-\}, \tag{4.14}$$

where $\mathcal{R}^+$ are true observed interactions and $\mathcal{R}^-$ are unobserved interactions. By presenting the data in this manner, we assume that user i has a preference to item i over item j.

Formulating ranking recommendations in a Bayesian manner for items $i \in \mathcal{V}$ is the same as maximising a posterior density for model parameters Θ:

$$p(\Theta| >_i) \propto p(>_i |\Theta) p(\Theta), \tag{4.15}$$

where $>_i$ is the preference order for user i, assuming users act independently of each other. It is also assumed that the likelihood function for a user can be written as a product of single densities and combined for all users, as the ordering of item preference for a pair of items (i, j) for a user is independent of the ordering of any other pair of items:

$$\prod_{i \in \mathcal{U}} p(>_i |\Theta) = \prod_{(i,j,k) \in \mathcal{U} \times \mathcal{V} \times \mathcal{V}} p(j >_i k|\Theta)^{\delta((i,j,k) \in O)} (1 - p(j >_i k|\Theta))^{\delta((u,i,j) \notin \mathcal{O})}, \tag{4.16}$$

where δ is the indicator function.

Continuing, using totality and antisymmetry of a pairwise ordering scheme, this simplifies to

$$\prod_{i \in \mathcal{U}} p(>_i |\Theta) = \prod_{(i,j,k) \in \mathcal{O}} p(j >_i k|\Theta). \tag{4.17}$$

To establish a personalised order, define the individual probability a user prefers item i to item j:

$$p(j >_i k|\Theta) = \sigma(\hat{x}_{ijk}(\Theta)), \tag{4.18}$$

where σ is the sigmoid function and $\hat{x}_{ijk}$ is an arbitrary real-valued function of the parameter vector Θ, capturing the interaction between user i and items j, k. This delegates the interaction function modelling to matrix factorisation.

To obtain the posterior, a normal prior distribution can be used:

$$p(\Theta) \sim N(0, \lambda_\Theta I), \tag{4.19}$$

then the maximum posterior estimate can be found:

$$\begin{aligned} L_{BPR} &= \ln(p(\Theta| >_u) \\ &= \ln(p(>_u |\Theta)p(\Theta) \\ &= \ln \prod_{(i,j,k) \in \mathcal{O}} \sigma(\hat{x}_{ijk})p(\Theta) \\ &= \sum_{(i,j,k) \in \mathcal{O}} \ln \sigma(\hat{x}_{ijk}) + \ln p(\Theta) \\ &= \sum_{(i,j,k) \in \mathcal{O}} \ln \sigma(\hat{x}_{ijk}) - \lambda_\Theta ||\Theta||^2. \end{aligned}$$

Then, when conducting matrix factorisation, the loss function to minimise is (for a given pair of items j, k and user i:

$$L_{BPR} = \sum_{(i,j,k) \in \mathcal{O}} - \ln \sigma \left(\mathbf{S}_{ij} - \mathbf{S}_{ik}\right) + \lambda \, ||\Theta||_2^2 \tag{4.20}$$

4.5 Deep Learning Matrix Factorisation

As mentioned in the introduction, using an inner product as an interaction function may not be enough capture sufficient complexities in the user item interaction data. To this end, deep learning based matrix factorisation (neural collaborative filtering) was proposed [15], which still decomposes the interaction matrix $\mathbf{A}$ into embedding matrices $\mathbf{U}, \mathbf{V}$ with embedding $\mathbf{u}_i$ for user i and embedding $\mathbf{v}_j$ for item j, however the approach to learn these uses deep learning with a nonlinear interaction function.

Specifically, neural collaborative filtering naturally extends matrix factorisation into a dual neural network, combining the decomposition of user and item matrices with a multi-layer perceptron. The multi-layer perceptron approximates the interaction function, taking into account non-linearity in the data. The approach has added benefits, being able to take into account side information, such as user profiles and item features, denoted by $\mathbf{s}_i$ and $\mathbf{s}_j$ for user i and item j.

The prediction $\mathbf{S}_{ij}$ can be formulated as

$$\mathbf{S}_{ij} = f\left(\mathbf{U}^T\mathbf{s}_i, \mathbf{V}^T\mathbf{s}_j | \mathbf{U}, \mathbf{V}, \Theta_f\right), \tag{4.21}$$

where f is a multi-layer neural network, $\mathbf{U} \in R^{m \times d}$ is the embedding matrix for the users, $\mathbf{V} \in R^{n \times d}$ is the embedding matrix for the items and Θ_f are the neural network parameters. Considering (4.21) as a neural network, this is also equivalent to

$$f\left(\mathbf{U}^T\mathbf{s}_i, \mathbf{V}^T\mathbf{s}_j\right) = \phi_{out}\left(\phi_x\left(...\phi_1\left(\mathbf{U}^T\mathbf{s}_i, \mathbf{V}^T\mathbf{s}_j\right)\ldots\right)\right), \tag{4.22}$$

where ϕ_{out} and ϕ_x represent activation functions for the outermost layer and an inner layer x respectively.

Using this framework, it can be shown that matrix factorisation is a special case of neural collaborative filtering. In the case of matrix factorisation, $\mathbf{s}_u$ and $\mathbf{s}_i$ are representing the identities of the user and item, so they are binarised one-hot encoded vectors. As neural matrix factorisation consists of many neural network layers, consider the output of the first neural layer as

$$\phi_1\left(\mathbf{u}_i, \mathbf{v}_j\right) = \mathbf{u}_i \odot \mathbf{v}_j, \tag{4.23}$$

where $\odot$ represents the element-wise product. If this output is projected to the output layer:

$$\mathbf{S}_{ui} = a_{out}\left(\mathbf{h}^T\left(\mathbf{u}_i \odot \mathbf{v}_j\right)\right), \tag{4.24}$$

where a_{out} and $\mathbf{h}$ are the activation function and output layer parameters. Take a as an identity function and $\mathbf{h}$ as a row vector of 1s, then the classical matrix factorisation prediction is reached as in (2.1).

To combine user and item embeddings to make meaningful recommendations, raw concatenation is an option, however, this would disregard latent interactions between user and item embeddings. To circumvent this, neural matrix factorisation proposes using a multi-layer perceptron to add hidden layers to concatenate. The benefit of this is to learn a flexible interaction function rather than just the inner product. To do so, the multi-layer perceptron part of the model is defined as:

$$\mathbf{z}_1 = \phi_1\left(\mathbf{u}_i, \mathbf{v}_j\right) = \begin{bmatrix}\mathbf{u}_i \\ \mathbf{v}_j\end{bmatrix}, \tag{4.25}$$

$$\mathbf{z}_2 = \phi_2\left(\mathbf{z}_1\right) = a_2\left(\mathbf{W}_2^T\mathbf{z}_1 + \mathbf{b}_2\right), \tag{4.26}$$

$$\ldots,$$

$$\mathbf{z}_L = \phi_L\left(\mathbf{z}_{L-1}\right) = a_L\left(\mathbf{W}_L^T\mathbf{z}_{L-1} + \mathbf{b}_L\right), \tag{4.27}$$

$$\mathbf{S}_{ij} = \sigma\left(\mathbf{h}^T\mathbf{z}_L\right), \tag{4.28}$$

where ϕ_l represents a general activation function of layer l. The activation function for the intermediate layers l, is represented by a_l and $\mathbf{W}_l$, $\mathbf{b}_l$ and σ refer to the weight matrix, bias vector and activaction functions respectively for layer l. a_l is chosen to be the ReLU activation function, as from empirical results it performed better than other candidates, such as tanh and the sigmoid function.

The final output of the MLP component, $\mathbf{S}_{ij} = \sigma\left(\mathbf{h}^T\mathbf{z}_L\right)$ uses a sigmoid activation function. To create the full neural collaborative filtering model, the authors propose a fusion of generalised matrix factorisation and the multi-layer perceptron. Generalised matrix factorisation and multi-layer perceptron components learn separate embeddings

that are concatenated at the final hidden layer:

$$\phi^{GMF} = \mathbf{u}_i^G \odot \mathbf{v}_j^G, \tag{4.29}$$

$$\phi^{MLP} = a_L \left(\mathbf{W}_L^T \left(a_{L-1} \left(... \left(a_2 \mathbf{W}_2^T \begin{bmatrix} \mathbf{u}_i^M \\ \mathbf{v}_j^M \end{bmatrix} + \mathbf{b}_2 \right) ... \right) \right) + \mathbf{b}_L \right), \tag{4.30}$$

$$\mathbf{S}_{ij} = \sigma \left(\mathbf{h}^T \begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix} \right), \tag{4.31}$$

where $\mathbf{u}_i^G$ and $\mathbf{u}_i^M$ represent the user embedding for generalised matrix factorisation and the user embedding for multi-layer perceptron respectively.

In order to efficiently learn model parameters and therefore make better and faster recommendations, initialisation of parameters is important. As the neural collaborative filtering objective function is non-convex, it leads to local minima as solutions. To overcome this, it is proposed to first train generalised matrix factorisation and multi-layer perceptron separately and use their weights for initialising the full neural collaborative filtering model. The paper performs benchmark comparisons comapred to Bayesian personalised ranking and alternating least squares, compared using the metrics HR@10 and NDCG@10 on two datasets, improving relatively over alternating least squares and Bayesian personalised ranking by 4.5% and 4.9% respectively.

Chapter 5

Graph Neural Networks for Recommendation Systems

Graph neural networks have recently become the state of the art for recommendation systems, outperforming many other matrix factorisation techniques methods that we previously introduced in Chapter 4. Graph neural networks allow for the use of higher order information present in the user-item interaction data used as input for recommendation systems that the previous methods cannot. They also allow the utilisation of other structural data, such as user profile information and item metadata. By exploiting graph structure using nodes and edges, effectively any piece of information can be included in the graph which leads to richer recommendations.

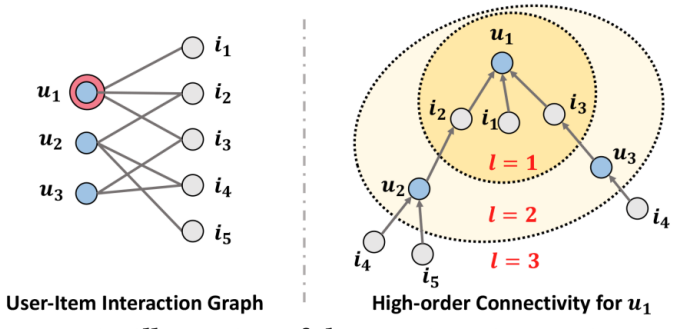

FIGURE 5.1: User item interactions represented visually as a graph with higher order connections from [54]

Figure 5.1 illustrates how user-item interactions can be represented as a graph and how high order connectivity can be deduced from this structure, usually lost when

just encoding the user-item interaction data into the matrix A. Consider a user u_1. The information given from the interactions when treated as a matrix just provides information about the direct interactions with items i_1, i_2 and i_3. The graph structure allows for the definition and discovery of paths that have length greater than one. The path $u_1 \rightarrow i_2 \rightarrow u_2$ contains information about the relationship between u_1 and u_2 that can be learned into the embeddings with an appropriate graph learning technique. Additionally, u_1 has two paths connecting to i_4 ($i_4 \rightarrow u_2 \rightarrow i_2 \rightarrow u_1$ and $i_4 \rightarrow u_3 \rightarrow i_3 \rightarrow u_1$) with which an appropriate graph learning technique can weight a higher relationship between u_1 and i_4 than u_1 and i_5 for which there is only one path connecting them.

5.1 Graph Neural Networks

Graph neural networks are a general framework for defining deep learning algorithms over graph-structured data. Traditional deep learning approaches bode well for linear sequence-based data however they can't generalise to or make use of complex graph structured data.

Graph neural networks generate embeddings by defining a computation graph for each node in a graph based on the nodes it is locally connected to. Local functions are learned to aggregate feature information for all nodes in the graph, resulting in embedding vectors. This process is known as neural message passing, where the node features are treated as messages.

At each layer in a graph neural network, the embedding of a node is refined based on the embeddings of its neighbors and graph neural networks generally employ the following three steps:

- Messaging: For a given node, each neighbor passes its current embedding to the central node.
- Aggregation: The messages from the neighbors are aggregated using an aggregation function, for example min, max, average or summation to produce a final aggregated embedding for the layer.

- Update: The embedding of the node at the previous layer is combined with the embedding from the aggregation of the neighbours to yield a final updated embedding for the current layer.

There are two main graph neural network architectures that allow deep learning to be defined on graph data: spectral and spatial. Spectral approaches embody signal processing and spatial models apply graph convolutions. Spatial models apply graph convolutions on graph structures to aggregate feature information into embeddings.

GraphSage [55] is a specific spatial architecture, allowing efficient embedding updating, meaning that it can accomodate new nodes into a graph to be learned. This is known as an inductive approach, because not all data is required for training and it won't need to be re-trained from scratch in the event of new data coming through. This is achieved by learning functions for embedding generation, rather than learning raw embeddings themselves. GraphSage works by selecting a target node, samples neighbours and aggregates their embeddings and merges them with the original target embedding as an update step. Mathematically, this can be represented as

$$\mathbf{h}_{\mathcal{N}_i}^l = f_l\left(\{\mathbf{h}_j^l, \forall j \in \mathcal{N}_i\}\right), \tag{5.1}$$

$$\mathbf{h}_i^{l+1} = \delta\left(\mathbf{W}^l\left[\mathbf{h}_i^l || \mathbf{h}_{\mathcal{N}_i}^l\right]\right), \tag{5.2}$$

where $\mathbf{h}_i^l$ is node i's embedding in layer l, f_l is an aggregate function, $\mathbf{W}^l$ is a learnable transformation matrix in layer l, $\mathcal{N}_i$ are the sampled neighbours for node i, $\delta(\cdot)$ is a non-linear activation function and $||$ is a concatenation operation.

Spectral models treat graphs as signals and process the signals using graph convolution. Graph signals can be transferred to the spectral space using Fourier transforms, filtered and processed, then finally transformed back to the spatial domain. Graph Convolutional Network [56] is an example of a spectral graph neural network architecture. It performs embedding updates using

$$\mathbf{H}^{l+1} = \delta\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^l\mathbf{W}^l\right), \tag{5.3}$$

where $\mathbf{H}^l \in \mathbf{R}^{|V|\times D}$ is an embedding matrix of graph nodes V with embedding dimension D in layer l, $\tilde{A} \in \mathbf{R}^{|V|\times|V|}$ is the graphs adjacency matrix and $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$. It

learns hidden layer representations that encompass node features ($\mathbf{H}^l$) and encodes local graph structure in $\tilde{\mathbf{D}}, \tilde{\mathbf{A}}$. Graph convolutional network can be used for semi-supervised classification as highlighted in the paper, and has benefits including computational speed as it scales linearly with the number of edges.

While it makes motivational sense to apply graph neural networks to recommendation systems, there exists a few challenges simply extending the spectral and spatial methodologies for graph neural networks to the recommendation problem [57]. To apply a graph neural network, there must first be a graph and the recommendation problem must be cast as a task on the given graph. In the collaborative filtering case, the user-item interaction data can be represented as a bipartite graph as in Figure 5.1, and the problem is to predict preference for each user and each item which directly translates to predicting the likelihood of an edge existing between a given user and item.

5.2 Collaborative Filtering on Graphs

The flexibility and ability to aggregate information make graph neural networks excellent candidates for collaborative filtering recommendation systems. A few approaches have been proposed for recommendation systems that have demonstrated state of the art accuracy and performance.

An early landmark paper in the field of graph neural networks for recommendation systems is Neural Graph Collaborative Filtering [54]. The key idea is to model the higher order connectivity depicted in Figure 5.1 into an embedding function. This is achieved by architecting a neural network approach that allows the propagation of embeddings on a graph, encoding the higher order structural information. Specifically, an embedding propagation layer in the neural network aggregates embeddings for users or items that have been interacted with. Multiple embedding propagation layers are stacked together, to access the higher order structural information on paths longer than length one.

The graph encoding is visible in Figure 5.1, where a stack of two layers will capture the similarity between u_1 and u_2 via i_2, and a stack of three layers will capture u_1 and i_4. The final learned embeddings are combined in a prediction layer that can predict user u_i's preference to item v_j.

Similarly to matrix factorisation methods, users are represented by embeddings u_i and items by embeddings v_j. The embeddings in traditional methods as we saw are directly inputted into the interaction function and learned via a loss function. In this graph neural network approach, the embeddings are refined via embedding propagation to capture both the collaborative signal and higher order information, before the interaction function.

To perform embedding propagation, messaging within the graph is crucial. This requires messaging construction and message aggregation, where messages are represented with their own message embeddings. A message from j to i between a user-item pair (i, j) is defined as

$$\mathbf{m}_{i \leftarrow j} = f\left(\mathbf{v}_j, \mathbf{u}_i, p_{ij}\right), \tag{5.4}$$

where $\mathbf{u}_i, \mathbf{v}_j \in R^d$ refer to embeddings for user i and item j, $\mathbf{m}_{i \leftarrow j}$ is the message embedding, f is a message encoding function and p_{ij} is a coefficient that controls decay on the propagation edge (i, j) where $p_{ij} = \frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}}$.

The function f is defined as

$$f(\mathbf{v}_j, \mathbf{u}_i, p_{ij}) = \frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}}\left(\mathbf{W}_1\mathbf{v}_j + \mathbf{W}_2\left(\mathbf{v}_j \odot \mathbf{u}_i\right)\right), \tag{5.5}$$

where $\mathbf{W}_1, \mathbf{W}_2 \in R^{d' \times d}$ are learnable weight matrices that extract information for the propagation, d' is an appropriate transformation size, $\mathcal{N}_i$ and $\mathcal{N}_j$ represent the first-hop neighbours of user i and item j. Thus, $\mathcal{N}_i$ are the items that the user i has interacted with and $\mathcal{N}_j$ are the users that have interacted with item j.

Once the messages have been computed, they need to be aggregated to aggregate local information and refine the embedding. For user i, the aggregation function is defined as

$$\mathbf{u}_i^{(1)} = f\left(\mathbf{m}_{i \leftarrow i} + \sum_{k \in \mathcal{N}_i} \mathbf{m}_{i \leftarrow k}\right), \tag{5.6}$$

and $\mathbf{u}_i^{(1)}$ represents the refined embedding for using u after one layer of message passing and aggregation. The same can be repeated to obtain $\mathbf{v}_j^{(1)}$. As well as taking into account the messages propagated from the neighbours $\mathcal{N}_i$, the self-connection is also included ($\mathbf{m}_{i \leftarrow i} = \mathbf{W}_1\mathbf{u}_i$), retaining original feature information. f is a non-linear activation function, chosen in the paper to be LeakyReLU.

To unlock the power of higher order structure in the graph and improve collaborative filtering, higher order propagation for paths longer than length one in the graph is desirable. This builds directly on message passing and aggregation for the first layer. After l layers, based on the propagation in the previous layers, the embeddings can be recursively refined and defined. The l layer refinement for embedding u_i is formulated as

$$\mathbf{u}_i^{(l)} = f\left(\mathbf{m}_{i\leftarrow i}^{(l)} + \sum_{k\in\mathcal{N}_i} \mathbf{m}_{i\leftarrow k}^{(l)}\right), \tag{5.7}$$

with the message propagation functions defined as

$$\mathbf{m}_{i\leftarrow j}^{(l)} = p_{ij}\left(\mathbf{W}_1^{(l)}\mathbf{v}_j^{(l-1)} + \mathbf{W}_2^{(l)}\left(\mathbf{v}_j^{(l-1)} \odot \mathbf{u}_i^{(l-1)}\right)\right), \tag{5.8}$$

$$\mathbf{m}_{i\leftarrow i}^{(l)} = \mathbf{W}_1^{(l)}\mathbf{u}_i^{(l-1)}, \tag{5.9}$$

again where $\mathbf{W}_1, \mathbf{W}_2 \in R^{d_l \times d_{l-1}}$ are learn-able weight matrices of dimension d_l, and to aggregate message passing from previous layers, $\mathbf{u}_i^{(l-1)}$ and $\mathbf{v}_j^{(l-1)}$ are aggregated embedding representations of the user i and item j.

Thus, in full form, the propagated embeddings at layer l (after $l-1$ layers of propagation) are

$$\mathbf{u}_i^{(l)} = f\left(\mathbf{W}_1^{(l)}\mathbf{u}_i^{(l-1)} + \sum_{p\in\mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_p|}}\left(\mathbf{W}_1^{(l)}\mathbf{v}_p^{(l-1)} + \mathbf{W}_2^{(l)}\left(\mathbf{v}_p^{(l-1)} \odot \mathbf{u}_i^{(l-1)}\right)\right)\right), \tag{5.10}$$

$$\mathbf{v}_j^{(l)} = f\left(\mathbf{W}_1^{(l)}\mathbf{v}_j^{(l-1)} + \sum_{p\in\mathcal{N}_j} \frac{1}{\sqrt{|\mathcal{N}_p||\mathcal{N}_j|}}\left(\mathbf{W}_1^{(l)}\mathbf{u}_p^{(l-1)} + \mathbf{W}_2^{(l)}\left(\mathbf{u}_p^{(l-1)} \odot \mathbf{v}_j^{(l-1)}\right)\right)\right). \tag{5.11}$$

This allows the collaborative signal we saw in Figure 5.1 to be captured between u_1 and i_4 for instance across multiple hops in the graph.

Rather than consider the embedding propagations for single users i and items j at a time, representing the above equations in matrix form allows for updating all users and items at once. The l layer aggregation can be defined as

$$\mathbf{E}^{(l)} = f\left((\mathcal{L} + I)\,\mathbf{E}^{(l-1)}\mathbf{W}_1^{(l)} + \mathcal{L}\mathbf{E}^{(l-1)} \odot \mathbf{E}^{(l-1)}\mathbf{W}_2^{(l)}\right), \tag{5.12}$$

where $\mathbf{E}^{(l)} \in \mathcal{R}^{(M+N)\times d_l}$ are the combined representations of user and item (M users,

N items) after l message propagation steps. f is a non-linear activation function (LeakyReLU) and $\mathcal{L}$ is the Laplacian for the user item graph, defined as

$$\mathcal{L} = \mathbf{D}^{-\frac{1}{2}} \mathcal{A} \mathbf{D}^{-\frac{1}{2}}, \tag{5.13}$$

with

$$\mathcal{A} = \begin{bmatrix} 0 & \mathbf{A} \\ \mathbf{A}^T & 0 \end{bmatrix}, \tag{5.14}$$

where $\mathbf{A} \in \mathcal{R}^{M \times N}$ is the user item interaction matrix, 0 is a zero matrix, $\mathcal{A}$ is the adjacency matrix and $\mathbf{D}$ is the diagonal degree matrix, $\mathbf{D} \in R^{(M+N)\times(M+N)}$, each entry D_{ii} is the number of nonzero entries in the i-th row vector of the adjacency matrix $\mathbf{A}$. As a result, the non-zero off-diagonal entries in $\mathcal{L} = \frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}}$.

After l layers of propagation there will be l representations for each user i and item j, all encoding various levels of information. The user representations will be $\{\mathbf{u}_i^{(1)}, \ldots, \mathbf{u}_i^{(L)}\}$ and similar for the items. To combine these, the embeddings are concatenated for each layer to return a final embedding for both users and items:

$$\mathbf{u}_i^* = \mathbf{u}_i^{(0)} || \ldots || \mathbf{u}_i^{(L)}, \tag{5.15}$$

and for items

$$\mathbf{v}_j^* = \mathbf{v}_j^{(0)} || \ldots || \mathbf{v}_j^{(L)}. \tag{5.16}$$

Due to the complex nature of learning the embeddings, the inner product is used to calculate similarity between a user i and item j as in (2.1),

$$\mathbf{S}_{ij} = \mathbf{u}_i^* \cdot \mathbf{v}_j^{*T}. \tag{5.17}$$

In order to learn parameters, Bayesian personalised ranking loss is used:

$$L_{NGCF} = \sum_{(i,j,k)\in\mathcal{O}} - \ln \sigma \left(\mathbf{S}_{ij} - \mathbf{S}_{ik}\right) + \lambda \left|\left|\Theta\right|\right|_2^2, \tag{5.18}$$

where $\mathcal{O} = (i,j,k)|(i,j) \in \mathcal{R}^+, (i,k) \in \mathcal{R}^-$ is the pairwise user-item interaction data, $\mathcal{R}^+$ are the observed interactions, $\mathcal{R}^-$ are the unobserved interactions, σ is the sigmoid

function, $\Theta = \{\mathbf{E}, \{\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)}\}_{l=1}^{L}\}$ are the trainable parameters and λ is a regularisation co-efficient.

5.3 Simplifying Graph Collaborative Filtering

LightGCN [58] is another landmark paper that applies a graph neural network for recommendation systems, building on neural graph collaborative filtering and simplifying it. The motivation for this approach came from the fact that the feature transformation and nonlinear activation parts of neural graph collaborative filtering adds a lot of computational strain.

The authors of LightGCN hypothesise that these operations are not useful for collaborative filtering, as the nodes in the user-item graph only encode ID as an input, so there is no point in attempting to extract value from non-linear transformations on this. The value is in the embedding propagation step. They performed some experiments by removing the feature transformation and nonlinear activation from NGCF and discovered faster and surprisingly better recommendation results from an accuracy perspective.

LightGCN still retains embedding propagation from neighbourhood aggregation, however this is done linearly (no nonlinear activation function) and then combines the embeddings into an interaction function. Specifically, LightGCN eliminates the weight matrices and nonlinear activation function that were defined above for neural graph collaborative filtering in (5.10), (5.11). As a result, the only parameters to be learned are the embeddings at layer 0 for each node. Then, similar to NGCF, message passing and aggregation is used to propagate embeddings across multiple layers.

In LightGCN, removing the nonlinear activation function f and the feature transformation matrices $\mathbf{W}_1, \mathbf{W}_2$, the embedding propagation step just becomes (for user embedding i and item embedding j at layer l):

$$\mathbf{u}_i^{(l)} = \sum_{p \in N_i} \frac{1}{\sqrt{|N_i||N_p|}} \mathbf{u}_i^{(l-1)}, \tag{5.19}$$

$$\mathbf{v}_j^{(l)} = \sum_{p \in N_j} \frac{1}{\sqrt{|N_p||N_j|}} \mathbf{v}_j^{(l-1)}. \tag{5.20}$$

The final embeddings are then aggregated across the l layers as:

$$\mathbf{u}_i^* = \sum_{m=0}^{l} \alpha_m \mathbf{u}_i^{(m)}, \tag{5.21}$$

$$\mathbf{v}_j^* = \sum_{m=0}^{l} \alpha_m \mathbf{v}_j^{(m)}, \tag{5.22}$$

where α_k is an importance given to the weights of the embedding at each layer. The prediction for user i and item j is the same as in (5.17), following (2.1),

$$\mathbf{S}_{ij} = \mathbf{u}_i^* \cdot \mathbf{v}_j^{*T} \tag{5.23}$$

LightGCN can also be represented in matrix form. Given the user-item interaction matrix $\mathbf{A} \in \mathcal{R}^{M \times N}$ for M users and N items, the adjacency matrix of the graph is the same as in NGCF (5.14),

$$\mathcal{A} = \begin{bmatrix} 0 & \mathbf{A} \\ \mathbf{A}^T & 0 \end{bmatrix} \tag{5.24}$$

The 0-th layer embeddings can also be represented in a matrix matrix $\mathbf{E}^{(0)} \in \mathcal{R}^{(M+N) \times T}$ where T is the embedding dimension.

Then the matrix equivalent of the user and item embeddings after k layers of propagation is

$$\mathbf{E}^{(k+1)} = \mathcal{L}\mathbf{E}^{(k)}, \tag{5.25}$$

where as in NGCF, $\mathcal{L} = \mathbf{D}^{-\frac{1}{2}} \mathcal{A} \mathbf{D}^{-\frac{1}{2}}$.

The final embedding matrix is then recursively calculated as

$$\mathbf{E} = \alpha_0 \mathbf{E}^{(0)} + \alpha_1 \mathbf{E}^{(1)} + \ldots + \alpha_K \mathbf{E}^{(K)} = \alpha_0 \mathbf{E}^{(0)} + \alpha_1 \mathcal{L}\mathbf{E}^{(1)} + \ldots + \alpha_K \mathcal{L}\mathbf{E}^{(K)}. \tag{5.26}$$

For LightGCN, the only trainable model parameters are the initial embeddings, $\mathbf{E}^{(0)}$, which means it has the same amount of model parameters as Matrix Factorisation.

BPR loss is used, the same as in NGCF (5.18):

$$L_{GCN} = \sum_{(i,j,k) \in \mathcal{O}} -\ln \sigma \left(\mathbf{S}_{ij} - \mathbf{S}_{ik} \right) + \lambda \left|\left| \Theta \right|\right|_2^2, \tag{5.27}$$

where $\Theta = \mathbf{E}^{(0)}$, the only learnable parameters.

5.4 Expanding to Knowledge Graphs

As mentioned, graph neural networks can take in other side information on top of just uiser-item interaction data. This side information can be from both the user and item side. For example, this can include adding demographic information for users (age, geography, location, preference) and relevant information for items (genres, metadata, authors). Knowledge graphs are a way of encoding this extra information on top of user-item interaction data in the same graph. The extra information can be encoded as nodes and edges to increase recommendation quality. Combining this information and encoding it in a graph leads to a collaborative knowledge graph [59].

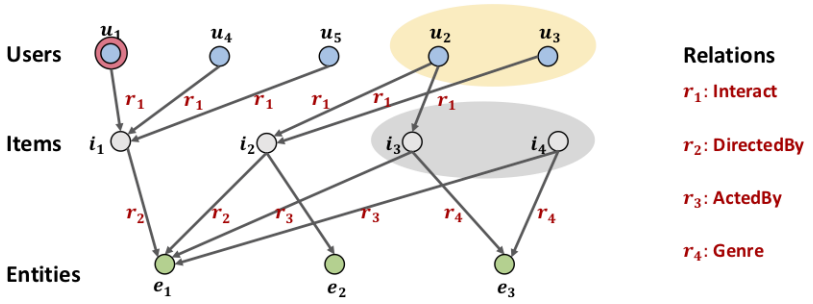

FIGURE 5.2: User item interactions with side information represented as a graph from [59]

As shown in Figure 5.2 from the KGAT paper, there is an interaction between user u_1 and movie i_1, which is directed by the person e_1. CF methods focus on the histories of similar users who also watched i_1 , i.e., u_4 and u_5. This fails to take into account high order relations amongst side information, such as the users in the yellow circle who watched other movies directed by the same person e_1, or the items in the grey circle that share other common relations with e_1.

KGAT [59] is a method proposed to learn user, item and entity (side-information) embeddings from the collaborative knowledge graph as input.

The collaborative knowledge graph is defined as the unified graph of user-item interactions and item knowledge (knowledge graph). User item interactions are represented as triplets $(i, Interact, j)$ (user i interacting with item j) and item knowledge graph information are also represented as triplets (h, r, t) where h, t are entities and r is the

relationship. In this manner, user item interactions can fit into the knowledge graph where r is an interaction.

In order to go from a collaborative knowledge graph to a prediction of a user i's preference to item j, there are three steps used: initial embedding layer, attentive embedding propagation layers and finally prediction. Firstly, the collaborative knowledge graph must be translated to embeddings. We denote an general embedding for either head, relation and tail as $\mathbf{e} : e_h, e_r, e_t$. To translate the graph to embeddings, TransR [60] is used, which attempts to optimise for $\mathbf{e}_h^r + \mathbf{e}_r = \mathbf{e}_t^r$ if the triplet exists. $\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t$ are the embeddings for h, r and t, and $\mathbf{e}_h^r, \mathbf{e}_t^r$ are the projected embeddings of h and t in the relation space.

This can be formalised to a triplet score as

$$g(h, r, t) = ||\mathbf{W}_r \mathbf{e}_h + \mathbf{e}_r - \mathbf{W}_r \mathbf{e}_t||_2^2 , \tag{5.28}$$

where $\mathbf{W}_r \in R^{(k \times d)}$ is a transformation matrix for relation r that projects embeddings from the d-dimensional entity space to the k-dimensional relation space. This is then trained with a pairwise ranking loss to yield L_{KG} which will be part of the final KGAT model:

$$L_{KG} = \sum_{(h,r,t,t') \in \mathcal{T}} - \ln \sigma \left(g \left(h, r, t' \right) - g \left(h, r, t \right) \right) , \tag{5.29}$$

where $\mathcal{T} = \{(h, r, t, t') | (h, r, t) \in \mathcal{G}, (h, r, t') \notin \mathcal{G}\}$, (h, r, t') is a broken triplet which is constructed by taking a valid triplet and replacing one entity. σ is the sigmoid function.

Similar to NGCF and Graph Convolutional Network, embeddings are aggregated and propagated to refine them. Considering a single entity h, $\mathcal{N}_h = (h, r, t)$ is the set of triplets where h is the head. The first order connectivity can be approximated by

$$\mathbf{e}_{\mathcal{N}_h} = \sum_{(h,r,t) \in \mathcal{N}_h} \pi \left(h, r, t \right) \mathbf{e}_t, \tag{5.30}$$

where π is a decay factor that uses knowledge-aware attention, defined as:

$$\pi(h, r, t) = \left(\mathbf{W}_r \mathbf{e}_t \right)^T \arctan \left(\mathbf{W}_r \mathbf{e}_h + \mathbf{e}_r \right) . \tag{5.31}$$

A softmax can then be used for normalisation:

$$\pi(h,r,t) = \frac{\exp\left(\pi\left(h,r,t\right)\right)}{\sum_{(h,r',t')\in N_h}\exp\left(\pi\left(h,r',t'\right)\right)}. \tag{5.32}$$

Then, to aggregate the entity representation e_h and its head representations $\mathbf{e}_{N_h}$ as $\mathbf{e}_h^{(1)} = f(\mathbf{e}_h, \mathbf{e}_{N_h})$, an aggregator such as GCN aggregator [61] can be used, which wraps a nonlinear activation function around a weight matrix adding the entity and head representation.

$$f_{GCN} = LeakyReLU\left(\mathbf{W}\left(\mathbf{e}_h + \mathbf{e}_{N_h}\right)\right), \tag{5.33}$$

where $\mathbf{W} \in \mathcal{R}^{d'\times d}$ are trainable weight matrices with d' transformation size.

This then gives first-order connectivity information for user, item and knowledge graph entities. This can be then propagated among higher orders by stacking more layers, recursively calculated as

$$\mathbf{e}_h^{(l)} = f\left(\mathbf{e}_h^{(l-1)}, \mathbf{e}_{N_h}^{(l-1)}\right), \tag{5.34}$$

where

$$\mathbf{e}_{N_h}^{(l-1)} = \sum_{(h,r,t)\in N_h} \pi\left(h,r,t\right)\mathbf{e}_t^{(l-1)} \tag{5.35}$$

and $\mathbf{e}_t^{(l-1)}$ is the representation of entity t from the previous propagation steps, $\mathbf{e}_h^{(0)}$ is $\mathbf{e}_h$ at the intital step.

After propagation, for each user i, item j and knowledge graph entity, there will be multiple representations from the multiple stacked layers, as in NGCF and LightGCN. From here, the representations are concatenated to a final embedding. Let the embedding for user i be u_i and the embedding for item j as v_j as consistent with our previous notation.

Then, following (5.15), (5.16):

$$\mathbf{u}_i^* = \mathbf{u}_i^{(0)}||\ldots||\mathbf{u}_i^{(L)}, \tag{5.36}$$

$$\mathbf{v}_j^* = \mathbf{v}_j^{(0)}||\ldots||\mathbf{v}_j^{(L)}. \tag{5.37}$$

User i's preference to item j is calculated as the inner product between their embeddings, same as NGCF and LightGCN in (5.17), (5.23),

$$\mathbf{S}_{ij} = \mathbf{u}_i^* \cdot \mathbf{v}_j^{*T}.$$

The loss function used in training combines L_{KG} defined earlier in (5.29) as well as L_{CF}. These two losses are added with a regularisation term and then optimised for.

L_{CF} is similar to (5.18) and (5.27), except removing the regularisation term,

$$L_{CF} = \sum_{(i,j,k)\in\mathcal{O}} -\ln \sigma\left(\mathbf{S}_{ij} - \mathbf{S}_{ik}\right). \tag{5.38}$$

Then, the overall L_{KGAT} is then formulated and optimised for in model training:

$$L_{KGAT} = L_{KG} + L_{CF} + \lambda||\Theta||_2^2 \tag{5.39}$$

Chapter 6

Evaluation and Experiments

In this chapter, we present our own original research by conducting and analysing experiments that aim to investigate whether graph neural networks lead to fairer outcomes in recommendation systems out of the box. We do this by comparing the performance of a graph neural network algorithm for recommendation to a matrix factorisation algorithm as a benchmark on different recommendation datasets. We choose a fairness framework and evaluate each algorithms performance with respect to this fairness as well as select accuracy and beyond-accuracy metrics to determine if the use of graph neural networks generally lead to fairer outcomes for users in recommendation systems.

6.1 Overview

This thesis began with an overview of recommendation systems and the importance of fairness in this domain. Through a review of existing literature on fairness in recommendation systems, we identified two gaps in the research. These gaps include a lack of explicit investigation into which off-the-shelf algorithms are most effective at promoting fairness between users, and a lack of research that considers both accuracy and beyond-accuracy metrics in assessing fairness.

Most previous research on user group fairness in recommendation systems has focused solely on traditional accuracy metrics. While these approaches have generally found some degree of unfairness between user groups, the findings are not always consistent across multiple datasets. One paper [43] considered user fairness from a beyond-accuracy

perspective, however, these metrics were drawn from surveys, such as asking the user how novel and diverse they *felt* the recommendations were, rather than objectively computing these metrics from the recommendation results.

On the other hand, the unique capability of graph neural networks to capture higher-order structure in the interaction data suggests that they may be better suited to improving fairness in recommendation systems than traditional matrix factorisation methods. Chapters 4 and 5 of this thesis provide a comprehensive review of graph neural networks and matrix factorisation techniques. In this chapter, we draw upon the insights synthesised in the previous chapters to make our own original research contributions. Specifically, we seek to answer two key questions:

RQ1 (Primary Research Question): Can graph neural networks lead to fairer outcomes in recommendation systems compared to matrix factorisation algorithms?

RQ2 (Secondary Research Question): Does a different picture of fairness in recommendation systems emerge when we go beyond traditional accuracy metrics?

In order to answer these questions, we will need to partition users into distinct groups based on an attribute. Following this, we will need to select a graph neural network algorithm and a matrix factorisation algorithm to generate recommendations applied to multiple datasets. We will compute relevant accuracy and beyond-accuracy metrics to get a holistic understanding of the recommendation system, and define a fairness framework that is a function of these metrics to understand fairness between the user groups. By conducting experiments according to a fairness framework, we aim to provide valuable insights into these questions and advance the current understanding of fairness in recommendation systems.

6.2 Methods

A natural split of users into groups is based on their activity level in the interaction data, following [34, 35, 40]:

User grouping

Users are split in an 80:20 ratio. We select the users in the lowest 80% quantile of

activity (number of interactions in the training data set) as the low activity (G_1) group, and the rest (highest 20% quantile) as high activity users (G_2), such that $G_1 \cap G_2 = \emptyset$.

Datasets

Last-FM (from [59]): Music listening dataset from Last.fm, data subset from Jan-June 2015. 10-core setting used (retained users and items with at least 10 interactions) to ensure data quality.

MovieLens 100K [62]: Movie review dataset containing 100000 interactions between users and items and their ratings of movies.

We split the datasets into a single split of a train set and a test set. Algorithms will be trained on the training set and evaluated on the test set. The datasets chosen for our study are diverse in terms of their industrial representation (movies and music) and have been widely researched in other recommendation systems papers. The selection of these datasets enables us to conduct a robust evaluation of the performance of graph neural networks and matrix factorisation algorithms in different contexts, contributing to a more comprehensive understanding of fairness.

Further statistical information on the datasets:

Dataset	LastFM	Movielens
Users	23,566	1,000
Items	48,123	1,700
Interactions	3,034,796	100,000

Algorithms

Alternating Least Squares (ALS) [12]: Alternating Least Squares, a collaborative filtering matrix factorisation algorithm. Implemented using Apache Spark MLlib library in Python.

LightGCN [58]: Lightweight graph neural network that maps user-item interaction data into a bipartite graph. Propagates user and item embeddings to learn richer representations. Implemented using TensorFlow in Python.

We have opted to focus on two algorithms to simplify the analysis, one matrix factorisation and one graph neural network. We have selected Alternating Least Squares as our matrix factorisation algorithm due to its scalability to large datasets, as discussed in

Chapter 2, and believe it is a good general candidate to represent matrix factorisation methods.

For our graph neural network algorithm, we have chosen LightGCN due to its superior performance and scalability compared to neural graph collaborative filtering. We also considered KGAT, but its slower computation speed and the unavailability of a knowledge graph for the MovieLens dataset led us to select LightGCN instead.

Evaluation Metrics

To compute fairness, we define our fairness score under the generic user-group fairness framework (3.1)

$$Fairness(G_1, G_2, M) = \frac{1}{|G_2|} \sum_{i \in G_2} M(i) - \frac{1}{|G_1|} \sum_{i \in G_1} M(i) \tag{6.1}$$

M refers to any metric, thus we can use the equation to compute fairness for any of the accuracy and beyond-accuracy metrics we will choose. G_1 and G_2 refer to discrete user groups such that $G_1 \cap G_2 = \emptyset$. In this case G_1 are the low activity users and G_2 are the high activity users.

While the fairness metrics in [34, 35, 40] employ an absolute value, we opt not to, in order to not lose information by suppressing the sign of the fairness score which suggests which group receives a better raw accuracy or beyond-accuracy metric. For this fairness score (6.1), a lower absolute value is desirable, as it indicates more fairness - the closer M between the two groups G_1, G_2 is.

For all metrics, we set $K = 10$. The accuracy metrics we select are NDCG@K (2.2) and HR@K (2.4) and the beyond-accuracy metrics we use are Novelty@K (2.11) and Diversity@K (2.8). We choose diversity and novelty as beyond accuracy metrics rather than coverage because coverage is computed at a population level, not at a user level, thus it loses some interpretability to compare across groups of users with different cardinality. We choose NDCG and hit-rate over precision, recall, F1 and MRR based on the discussion in chapter 2. Together, we believe NDCG and hit-rate offer a comprehensive view of accuracy.

TABLE 6.1: Results on the MovieLens dataset, for both algorithms (Alternating Least Squares (ALS) and LightGCN (LGCN)). Metrics (NDCG@10, HR@10, Diversity@10 and Novelty@10) computed for all users, high activity users and low activity users. Fairness score (6.1) computed between the two user groups. Bold face numbers indicate which algorithm is more fair, according to the absolute fairness score.

Algorithm		**NDCG**@10	**Hitrate**@10	**Diversity**@10	**Novelty**@10
ALS	**All**	0.18	0.38	1.25	2.99
	Low	0.16	0.33	1.23	2.83
	High	0.26	0.56	1.33	3.58
	Fairness	0.10	0.23	**0.10**	0.75
LGCN	**All**	0.18	0.42	2.19	2.27
	Low	0.18	0.40	2.23	2.32
	High	0.21	0.51	2.03	2.11
	Fairness	**0.03**	**0.11**	-0.20	**-0.21**

TABLE 6.2: Results on the Last-FM dataset, for both algorithms (Alternating Least Squares (ALS) and LightGCN (LGCN)). Metrics (NDCG@10, HR@10, Diversity@10 and Novelty@10) computed for all users, high activity users and low activity users. Fairness score (6.1) computed between the two user groups. Bold face numbers indicate which algorithm is more fair, according to the absolute fairness score.

Algorithm		**NDCG**@10	**Hitrate**@10	**Diversity**@10	**Novelty**@10
ALS	**All**	0.11	0.21	0.39	6.66
	Low	0.09	0.18	0.39	6.64
	High	0.18	0.36	0.36	6.73
	Fairness	0.09	0.18	**-0.03**	**-0.09**
LGCN	**All**	0.20	0.42	3.51	7.54
	Low	0.20	0.40	3.50	7.58
	High	0.22	0.50	3.55	7.38
	Fairness	**0.02**	**0.10**	0.05	0.20

6.3 Experimental Results & Analysis

Table 6.1 contains the results for all of the metrics, including the fairness score, on the MovieLens dataset. Similarly, Table 6.2 contains the results for the Last-FM dataset for the same metrics.

Low refers to the low activity group users, High refers to the high activity group users, All refers to all users, and Fairness refers to the fairness metric computed in (6.1). Fairness is computed on each of the accuracy and beyond-accuracy metrics (NDCG, hit-rate, diversity and novelty).

The metrics are calculated on a hold-out test set. For all users, we use the entire test set. For the low and high activity user groups respectively, we use the data in the test set with those users interactions only.

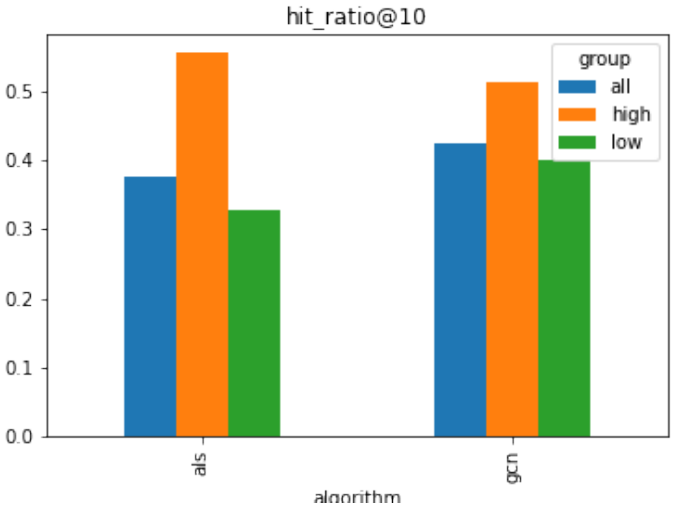

FIGURE 6.1: Graphical representation of hit ratio (hit-rate or HR@10) for the MovieLens dataset, split into all users (blue), high-activity users (orange) and low-activity users (green). The grouped bar chart on the left represents recommendation from the Alternating Least Squares algorithm and the grouped bar chart on the right from LightGCN.

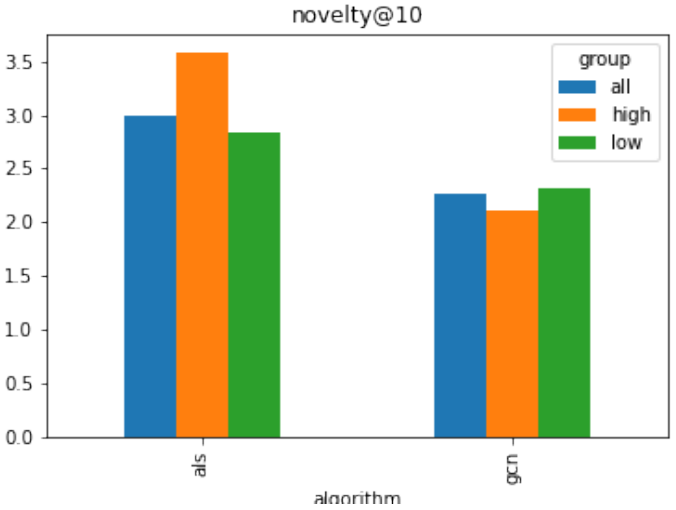

FIGURE 6.2: Graphical representation of novelty (Novelty@10) for the MovieLens dataset, split into all users (blue), high-activity users (orange) and low-activity users (green). The grouped bar chart on the left represents recommendation from the Alternating Least Squares algorithm and the grouped bar chart on the right from LightGCN.

Figure 6.1 shows a graphical representation of hit-rate for the low-activity users, high-activity users and all users together for both Alternating Least Squares (als) and LightGCN (gcn) on the MovieLens dataset. Figure 6.2 shows the same for novelty. Figures 6.3 and 6.4 show the results for hit-rate and novelty on the Last-FM dataset.

The results indicate that LightGCN is fairer on both datasets for the accuracy metrics,

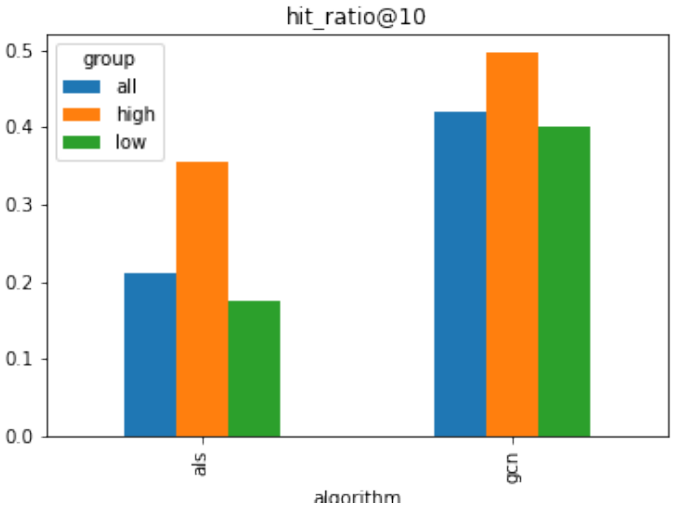

FIGURE 6.3: Graphical representation of hit ratio (hit-rate or HR@10) for the Last-FM dataset, split into all users (blue), high-activity users (orange) and low-activity users (green). The grouped bar chart on the left represents recommendation from the Alternating Least Squares algorithm and the grouped bar chart on the right from LightGCN.

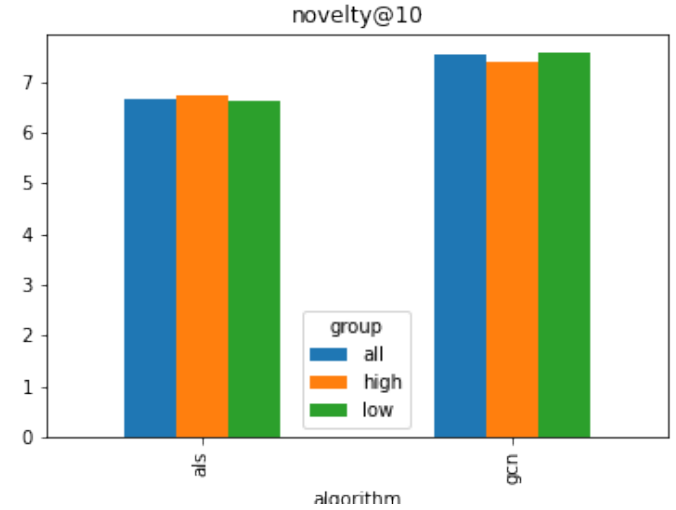

FIGURE 6.4: Graphical representation of novelty (Novelty@10) for the Last-FM dataset, split into all users (blue), high-activity users (orange) and low-activity users (green). The grouped bar chart on the left represents recommendation from the Alternating Least Squares algorithm and the grouped bar chart on the right from LightGCN.

and for the beyond-accuracy metrics, Alternating Least Squares is fairer for both beyond-accuracy metrics for Last-FM, and it is split for MovieLens. It is worth noting that while Alternating Least Squares appears fairer for the diversity metric on Last-FM (-0.03 vs. 0.05) if the fairness score was normalised, LightGCN would be fairer. We also note that when LightGCN is fairer (novelty metric on MovieLens) it is fairer by a larger margin than when Alternating Least Squares is fairer.

We speculate that LightGCN leads to outright fairer recommendation for accuracy metrics due to graph neural networks ability to learn higher-order information from the data. As the higher activity users have more interactions in the dataset they should thus enjoy better recommendation than the low activity users due to the fundamental nature of collaborative filtering. However, the ability to learn higher order structure by propagating the embeddings in (5.15), (5.16) allows more to be learnt from what information is available in the raw interaction data for the low activity users, improving the recommendation quality and thus bridging the gap of fairness between the user groups.

It is worth noting that for the accuracy metrics, for both algorithms and both datasets, the high activity users enjoy better recommendations according to these metrics in all cases. For the beyond-accuracy metrics, it is not the case, where for MovieLens, Alternating Least Squares had better diversity and novelty for the high activity users, whereas LightGCN led to better outcomes for the low activity users. For the Last-FM dataset, Alternating Least Squares had better diversity and worse novelty for the low activity users, and for LightGCN it was the opposite.

6.4 Limitations

While we glean relatively promising results from our experiments and analysis that suggest graph neural networks can lead to fairer outcomes for users for at least accuracy metrics, the study has a few limitations present that can be addressed in future research:

- Sensitivity to K: we conducted experiments for $K = 10$ only. It is possible results may differ depending on different choices of K. Possible other choices include $K = 1, K = 5, K = 20, K = 100$.
- Sensitivity to user group split: we split users based on activity at an 80:20 split. This ratio can be adjusted to 50:50 for instance, or the attribute users are split on can be changed, such as demographic attributes (age, gender, nationality, etc).
- Train/test split: we performed one split into train and test for each data set. In order to get more robust results, one can make different splits into train and test then average out the results.

- Fairness metric: The fairness metric can be improved to add normalisation, to avoid the issue with the diversity metric for Last-FM.
- Generalisation: More datasets and algorithms can be used to get a bigger picture understanding of generalisation.

6.5 Conclusion

In these experiments, we split users into groups based on their activity levels in the dataset and sought to understand fairness in terms of accuracy and beyond-accuracy metrics. We applied a graph neural network recommendation algorithm and a matrix factorisation recommendation algorithm.

Graph neural networks were clearly fairer for the accuracy metrics chosen (NDCG and hit-rate) for both algorithms. The results show that the matrix factorisation approach was fairer in more cases for the beyond-accuracy metrics, although when the graph neural network was fairer, it was fairer by a larger margin.

This shows promise for using graph neural network algorithms out of the box for recommendation systems for those interested in fairer approaches, at the very least for accuracy metrics. In future work, these experiments can be extended to address the limitations in 6.4 and understand generalisation.

Bibliography

[1] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 191–198, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340359. doi: 10.1145/2959100.2959190. URL https://doi.org/10.1145/2959100.2959190.

[2] Rishabh Mehrotra and Benjamin Carterette. Recommendations in a marketplace. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, page 580–581, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362436. doi: 10.1145/3298689.3346952. URL https://doi.org/10.1145/3298689.3346952.

[3] Mohammad Saberian and Justin Basilico. Recsysops: Best practices for operating a large-scale recommender system. In *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, page 590–591, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384582. doi: 10.1145/3460231.3474620. URL https://doi.org/10.1145/3460231.3474620.

[4] Zhuoran Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Da Tang, Bolin Zhu, Yijie Zhu, Peng Wu, Ke Wang, and Youlong Cheng. Monolith: Real time recommendation system with collisionless embedding table, 2022. URL https://arxiv.org/abs/2209.07663.

[5] TikTok Newsroom. How tiktok recommends videos for you. https://newsroom.tiktok.com/en-us/how-tiktok-recommends-videos-for-you, 2021.

[6] Hootsuite. How the tiktok algorithm works: For you page guide. https://blog.hootsuite.com/tiktok-for-you-page/, 2021.

[7] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4), dec 2016. ISSN 2158-656X. doi: 10.1145/2843948. URL https://doi.org/10.1145/2843948.

[8] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The youtube video recommendation system. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, page 293–296, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589060. doi: 10.1145/1864708.1864770. URL https://doi.org/10.1145/1864708.1864770.

[9] Google Developers. Content-based recommendation systems. https://developers.google.com/machine-learning/recommendation/content-based/basics, 2021.

[10] Eli. Pariser. *The filter bubble : what the Internet is hiding from you.* Penguin Press, New York, 2011. ISBN 9781594203008 1594203008. URL http://www.worldcat.org/search?qt=worldcat_org_all&q=1594203008.

[11] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, page 46–54, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1558605568.

[12] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, aug 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.263. URL https://doi.org/10.1109/MC.2009.263.

[13] Jyotirmoy Gope and Sanjay Kumar Jain. A survey on solving cold start problem in recommender systems. In *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pages 133–138, 2017. doi: 10.1109/CCAA.2017.8229786.

[14] Shuai Zhang, Lina Yao, and Aixin Sun. Deep learning based recommender system: A survey and new perspectives. *CoRR*, abs/1707.07435, 2017. URL http://arxiv.org/abs/1707.07435.

[15] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. *CoRR*, abs/1708.05031, 2017. URL http://arxiv.org/abs/1708.05031.

[16] Harald Steck. Calibrated recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, page 154–162, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450359016. doi: 10.1145/3240323.3240372. URL https://doi.org/10.1145/3240323.3240372.

[17] Ziyou Yan. Real-time machine learning for recommendations. *eugeneyan.com*, Jan 2021. URL https://eugeneyan.com/writing/real-time-recommendations/.

[18] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. Billion-scale commodity embedding for e-commerce recommendation in alibaba, 2018.

[19] Zhu Sun, Di Yu, Hui Fang, Jie Yang, Xinghua Qu, Jie Zhang, and Cong Geng. Are we evaluating rigorously? benchmarking recommendation for reproducible evaluation and fair comparison. In *Proceedings of the 14th ACM Conference on Recommender Systems*, RecSys '20, page 23–32, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375832. doi: 10.1145/3383313.3412489. URL https://doi.org/10.1145/3383313.3412489.

[20] Yan-Martin Tamm, Rinchin Damdinov, and Alexey Vasilev. Quality metrics in recommender systems: Do we calculate metrics consistently? In *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, page 708–713, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384582. doi: 10.1145/3460231.3478848. URL https://doi.org/10.1145/3460231.3478848.

[21] Matus Tomlein, Branislav Pecher, Jakub Simko, Ivan Srba, Robert Moro, Elena Stefancova, Michal Kompan, Andrea Hrckova, Juraj Podrouzek, and Maria Bielikova. An audit of misinformation filter bubbles on youtube: Bubble bursting and recent behavior changes. In *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, page 1–11, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384582. doi: 10.1145/3460231.3474241. URL https://doi.org/10.1145/3460231.3474241.

[22] Tim Donkers and Jürgen Ziegler. The dual echo chamber: Modeling social media polarization for interventional recommending. In *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, page 12–22, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384582. doi: 10.1145/3460231.3474261. URL https://doi.org/10.1145/3460231.3474261.

[23] Marius Kaminskas and Derek Bridge. Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems. *ACM Trans. Interact. Intell. Syst.*, 7(1), dec 2016. ISSN 2160-6455. doi: 10.1145/2926720. URL https://doi.org/10.1145/2926720.

[24] Tao Zhou, Jian-Guo Liu, Zoltan Kuscsik, Matús Medo, and Yi-Cheng Zhang. Being accurate is not enough: measuring and optimizing the diversity of recommendations. *CoRR*, abs/0808.2670, 2008. URL http://arxiv.org/abs/0808.2670.

[25] Australian Human Rights Commission. Using artificial intelligence to make decisions: Addressing the problem of algorithmic bias. Technical report, Australian Human Rights Commission, 2020. URL https://humanrights.gov.au/sites/default/files/document/publication/ahrc_technical_paper_algorithmic_bias_2020.pdf.

[26] Department of Industry, Science, Energy and Resources. *Australia's Artificial Intelligence Ethics Framework*. Australian Government, 2020. URL https://www.industry.gov.au/sites/default/files/2020-11/australias-artificial-intelligence-ethics-framework.pdf.

[27] Susan Wei and Marc Niethammer. The fairness-accuracy pareto front. *Stat. Anal. Data Min.*, 15(3):287–302, may 2022. ISSN 1932-1864. doi: 10.1002/sam.11560. URL https://doi.org/10.1002/sam.11560.

[28] Sam Corbett-Davies, Emma Pierson, Avi Feller, Sharad Goel, and Aziz Huq. Algorithmic decision making and the cost of fairness. *CoRR*, abs/1701.08230, 2017. URL http://arxiv.org/abs/1701.08230.

[29] Alessandro Castelnovo, Riccardo Crupi, Greta Greco, and Daniele Regoli. The zoo of fairness metrics in machine learning. *CoRR*, abs/2106.00467, 2021. URL https://arxiv.org/abs/2106.00467.

[30] Harini Suresh and John Guttag. A framework for understanding sources of harm throughout the machine learning life cycle. In *Equity and Access in Algorithms, Mechanisms, and Optimization*. ACM, oct 2021. doi: 10.1145/3465416.3483305. URL https://doi.org/10.1145%2F3465416.3483305.

[31] Ioannis Pastaltzidis, Nikolaos Dimitriou, Katherine Quezada-Tavarez, Stergios Aidinlis, Thomas Marquenie, Agata Gurzawska, and Dimitrios Tzovaras. Data augmentation for fairness-aware machine learning: Preventing algorithmic bias in law enforcement systems. In *2022 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '22, page 2302–2314, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393522. doi: 10.1145/3531146.3534644. URL https://doi.org/10.1145/3531146.3534644.

[32] Yuji Roh, Kangwook Lee, Steven Euijong Whang, and Changho Suh. Sample selection for fair and robust training, 2021.

[33] Simon Caton and Christian Haas. Fairness in machine learning: A survey. *CoRR*, abs/2010.04053, 2020. URL https://arxiv.org/abs/2010.04053.

[34] Yunqi Li, Hanxiong Chen, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. User-oriented fairness in recommendation. *CoRR*, abs/2104.10671, 2021. URL https://arxiv.org/abs/2104.10671.

[35] Zuohui Fu, Yikun Xian, Ruoyuan Gao, Jieyu Zhao, Qiaoying Huang, Yingqiang Ge, Shuyuan Xu, Shijie Geng, Chirag Shah, Yongfeng Zhang, and Gerard de Melo. Fairness-aware explainable recommendation over knowledge graphs. *CoRR*, abs/2006.02046, 2020. URL https://arxiv.org/abs/2006.02046.

[36] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. Bias and debias in recommender system: A survey and future directions. *CoRR*, abs/2010.03240, 2020. URL https://arxiv.org/abs/2010.03240.

[37] Yunqi Li, Yingqiang Ge, and Yongfeng Zhang. Tutorial on fairness of machine learning in recommender systems. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 2654–2657, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380379. doi: 10.1145/3404835.3462814. URL https://doi.org/10.1145/3404835.3462814.

[38] Yunqi Li, Hanxiong Chen, Shuyuan Xu, Yingqiang Ge, Juntao Tan, Shuchang Liu, and Yongfeng Zhang. Fairness in recommendation: A survey, 2022. URL https://arxiv.org/abs/2205.13619.

[39] Lin Xiao, Zhang Min, Zhang Yongfeng, Gu Zhaoquan, Liu Yiqun, and Ma Shaoping. Fairness-aware group recommendation with pareto-efficiency. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 107–115, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346528. doi: 10.1145/3109859.3109887. URL https://doi.org/10.1145/3109859.3109887.

[40] Hossein A. Rahmani, Mohammadmehdi Naghiaei, Mahdi Dehghan, and Mohammad Aliannejadi. Experiments on generalizability of user-oriented fairness in recommender systems, 2022. URL https://arxiv.org/abs/2205.08289.

[41] Alessandro B. Melchiorre, Eva Zangerle, and Markus Schedl. Personality bias of music recommendation algorithms. In *Proceedings of the 14th ACM Conference on Recommender Systems*, RecSys '20, page 533–538, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375832. doi: 10.1145/3383313.3412223. URL https://doi.org/10.1145/3383313.3412223.

[42] Michael D. Ekstrand, Mucun Tian, Ion Madrazo Azpiazu, Jennifer D. Ekstrand, Oghenemaro Anuyah, David McNeill, and Maria Soledad Pera. All the cool kids, how do they fit in?: Popularity and demographic biases in recommender evaluation and effectiveness. In Sorelle A. Friedler and Christo Wilson, editors, *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*, volume 81 of *Proceedings of Machine Learning Research*, pages 172–186. PMLR, 23–24 Feb 2018. URL https://proceedings.mlr.press/v81/ekstrand18b.html.

[43] Ningxia Wang and Li Chen. User bias in beyond-accuracy measurement of recommendation algorithms. In *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, page 133–142, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384582. doi: 10.1145/3460231.3474244. URL https://doi.org/10.1145/3460231.3474244.

[44] Jurek Leonhardt, Avishek Anand, and Megha Khosla. User fairness in recommender systems. In *Companion of the The Web Conference 2018 on The Web Conference*

2018 - WWW '18. ACM Press, 2018. doi: 10.1145/3184558.3186949. URL https://doi.org/10.1145%2F3184558.3186949.

[45] Alex Beutel, Jilin Chen, Tulsee Doshi, Hai Qian, Li Wei, Yi Wu, Lukasz Heldt, Zhe Zhao, Lichan Hong, Ed H. Chi, and Cristos Goodrow. Fairness in recommendation ranking through pairwise comparisons. *CoRR*, abs/1903.00780, 2019. URL http://arxiv.org/abs/1903.00780.

[46] Himan Abdollahpouri, Masoud Mansoury, Robin Burke, and Bamshad Mobasher. The unfairness of popularity bias in recommendation. *CoRR*, abs/1907.13286, 2019. URL http://arxiv.org/abs/1907.13286.

[47] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. Managing popularity bias in recommender systems with personalized re-ranking. *CoRR*, abs/1901.07555, 2019. URL http://arxiv.org/abs/1901.07555.

[48] Robin Burke. Multisided fairness for recommendation. *CoRR*, abs/1707.00093, 2017. URL http://arxiv.org/abs/1707.00093.

[49] Gourab K. Patro, Arpita Biswas, Niloy Ganguly, Krishna P. Gummadi, and Abhijnan Chakraborty. Fairrec: Two-sided fairness for personalized recommendations in two-sided platforms. *CoRR*, abs/2002.10764, 2020. URL https://arxiv.org/abs/2002.10764.

[50] Himan Abdollahpouri and Robin Burke. Multi-stakeholder recommendation and its connection to multi-sided fairness. *CoRR*, abs/1907.13158, 2019. URL http://arxiv.org/abs/1907.13158.

[51] Google Developers. Matrix factorization for collaborative filtering. https://developers.google.com/machine-learning/recommendation/collaborative/matrix, 2021.

[52] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of the KDD Cup Workshop 2007*, pages 3–6, New York, August 2007. ACM. URL http://www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf.

[53] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. *CoRR*, abs/1205.2618, 2012. URL http://arxiv.org/abs/1205.2618.

[54] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. *CoRR*, abs/1905.08108, 2019. URL http://arxiv.org/abs/1905.08108.

[55] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017. URL http://arxiv.org/abs/1706.02216.

[56] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL http://arxiv.org/abs/1609.02907.

[57] Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, and Yong Li. A survey of graph neural networks for recommender systems: Challenges, methods, and directions, 2023.

[58] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. *CoRR*, abs/2002.02126, 2020. URL https://arxiv.org/abs/2002.02126.

[59] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. KGAT: knowledge graph attention network for recommendation. *CoRR*, abs/1905.07854, 2019. URL http://arxiv.org/abs/1905.07854.

[60] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 2181–2187. AAAI Press, 2015. ISBN 0262511290.

[61] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[62] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), dec 2015. ISSN 2160-6455. doi: 10.1145/2827872. URL https://doi.org/10.1145/2827872.