POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Reproducing work for Simple-HGN and proposing a novel algorithm named attention-based matrix factorization

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Zhitao He**

Student ID: 960704
Advisor: Prof. Maurizio Ferrari Dacrema
Academic Year: 2023-24

Abstract

Recommender System is an information filtering system applied to predict the rating that a user would give to an item. With the user's history behaviors and item attributes, the system can generate a personalized recommendation list consisting of the items which a user may like in the future. To seek to improve satisfaction in users' shopping experience and engagement with a product service, the recommender system always plays an important role in the field of electronic commerce. In recent years, due to the rapid development of machine learning algorithms and the continuous expansion of application domains, e.g. virtual assistants and healthcare, the attempt to apply deep learning algorithms to Recommender Systems has also become an increasingly popular research direction, aimed at providing customers with more accurate potential purchase products. However, in some cases, these deep learning models may be over-complex and most of the modeling capacity may come from other simpler parts. Therefore, this paper presents a novel algorithm that replaces the deep-learning part of the Simple-HGN model, a method described in the paper "Are we really making much progress? Revisiting, benchmarking, and refining heterogeneous graph neural networks"[33], with adding an attention mechanism which is implemented by matrix factorization. The thesis initially validates the replicability of the outcomes achieved by Simple-HGN as documented in the paper, subsequently conducts an ablation study to assess the individual contributions of various components within the Simple-HGN model. Following this, the novel model named Attention-based Matrix Factorization is devised and implemented. Additionally, a comparison is made between Simple-HGN and a collection of well-optimized baselines, utilizing several datasets employed within the paper.

Sommario

Il Recommender System è un sistema di filtraggio delle informazioni applicato per prevedere la valutazione che un utente darà a un articolo. Grazie ai comportamenti storici dell'utente e agli attributi dell'articolo, il sistema può generare un elenco di raccomandazioni personalizzate composto dagli articoli che potrebbero piacere all'utente in futuro. Per cercare di migliorare la soddisfazione dell'esperienza di acquisto degli utenti e l'impegno nei confronti di un prodotto, il recommender system svolge sempre un ruolo importante nel campo del commercio elettronico. Negli ultimi anni, grazie al rapido sviluppo degli algoritmi di apprendimento automatico e alla continua espansione dei domini di applicazione, ad esempio gli assistenti virtuali e l'assistenza sanitaria, il tentativo di applicare algoritmi di deep learning ai recommender system è diventato una direzione di ricerca sempre più popolare, con l'obiettivo di fornire ai clienti potenziali prodotti di acquisto più accurati. Tuttavia, in alcuni casi, questi modelli di deep learning possono essere troppo complessi e la maggior parte della capacità di modellazione può derivare da altre parti più semplici. Per questo motivo, il presente documento presenta un nuovo algoritmo che sostituisce la parte di deep learning del modello Simple-HGN, un metodo descritto nel documento "Are we really making much progress? Revisiting, benchmarking, and refining heterogeneous graph neural networks"[33], con l'aggiunta di un meccanismo di attenzione implementato tramite matrix factorization. La tesi inizialmente convalida la replicabilità dei risultati ottenuti da Simple-HGN, come documentato nell'articolo, e successivamente conduce uno studio di ablazione per valutare i singoli contributi dei vari componenti all'interno del modello Simple-HGN. In seguito, viene ideato e implementato il nuovo modello denominato Attention-based Matrix Factorization. Inoltre, viene effettuato un confronto tra Simple-HGN e un insieme di linee di base ben ottimizzate, utilizzando diversi set di dati impiegati nel documento.

Ringraziamenti

During the process of completing this paper, I have received help from many individuals, and I would like to express my gratitude to each of them. First and foremost, I would like to thank my advisor, Maurizio Ferrari Dacrema. Whenever I had questions or encountered difficulties that needed to be resolved, he always provided timely and highly effective advice. He also proactively shared important insights to help me navigate my practical work, saving me time and energy. Throughout it all, he has been dedicated and responsible. Although we have never met in person, our communication has been solely through emails.

Next, I would like to express my gratitude to my family and friends. While they may not be able to provide solutions to my academic problems, they have always been there for me during the lengthy process of completing this paper. Whenever I lacked confidence or felt stuck in my progress, they consoled and encouraged me in various ways. It could be a video call with my mother and grandmother, chatting with my best friend Tang Yihan. It could be enjoying a cocktail with my neighbor Wang Hui, going for a drive with Luca to participate in various activities, or having pizza or a delicious dinner with Roberto. These activities allowed me to relax and provided me with the motivation to seek solutions, overcome challenges, and for that, I am extremely grateful to them.

Lastly, I would like to express my gratitude to myself. I appreciate my continuous efforts, and I am satisfied that I was able to solve all the problems I encountered throughout the entire process. I am also delighted to witness the completion of my master's thesis. At the end, I want to once again thank each and every person mentioned earlier. I am truly grateful to have had your companionship on my journey of growth.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

Recommender System is an information system that can provide users with personalized and relevant suggestions by analyzing their behaviors and product attributes. Nowadays, it plays an important role and there are numerous companies applying it to improve sales, customer satisfaction, and stickiness. There are various algorithms used for building recommender systems, including Collaborative Filtering, Content-Based Filtering, and Hybrid methods. In recent years, the direction of research in recent years started to focus on combining deep learning algorithms with Recommendation algorithms for seeking higher recommendation quality. However, it may bring a problem that the model becomes more complex and most of the modeling capacity may be from the simpler parts.

There is a paper named Are we really making much progress? Revisiting, benchmarking, and refining heterogeneous graph neural networks[33] issued in 2021, in which the authors proposed a new and effective method called Simple Heterogeneous Graph Neural Networks model(Simple-HGN) which consists of two parts, one is the heterogeneous graph neural networks as the deep modeling part, the other one is pre-trained embeddings with Matrix Factorization BPR as the non-deep part. Matrix Factorization is a class in Collaborative Filtering, it decomposes the User-Rating Matrix into the product of several sub-matrices. This research presents a new algorithm which is Attention-based Matrix Factorization(AMF) that replaces the deep graph part of Simple-HGN model with a non-deep part using an attention mechanism that is implemented by Matrix Factorization.

The structure of this thesis is as follows. In Chapter 2, we explain the current state-of-the-art and the necessary references to understand fully this research. In particular, we outline the description of recommender systems and the relevant equation of models. Then, in Chapter 3, Simple-HGN serves as the reference model for AMF model proposed in this paper. The overall structure of Simple-HGN and each component will be introduced in detail. Building upon this, Chapter 4 will provide a detailed description of how AMF evolves from Simple-HGN, as well as the overall structure and each component of AMF. The chapter will conclude with an overview of the subsequent experimental work on constructing, training, and analyzing the results of AMF model. Moving on to Chapter 5, the experimental work will be further described in detail, including the reproduction

work and ablation study of Simple-HGN, the analysis of features in the training dataset, and the description of the methodology and results of its segmentation. Additionally, an outline will be provided for the aspects from which the model's performance can be assessed. Finally, a thorough comparative analysis and discussion of the specific model performance results will be conducted in Chapter 6, followed by a comprehensive summary of AMF performance in Chapter 7.

Chapter 2: State of the art

This chapter will fully present the detailed description of Recommender Systems, mainstream collaborative filtering algorithms, and metrics for evaluating model performance will be fully presented.

First of all, we introduce some basic concepts of Recommender Systems in Section 2.1, which mainly contains the introduction, the main methods, and the type of data. Secondly, in Section 2.2, we focus our narrative on K-nearest neighbors based model which is one sub-class in collaborative filtering, in terms of the idea behind it, the equations of similarities, and its categories. Next, in Section 2.3, we introduce another sub-class which is machine learning models by presenting the principles of different models,such as SLIM, matrix factorization, graph-based models,etc.. Finally, the evaluation techniques and metrics of Recommender Systems are outlined in Section 2.4.

2.1 Basics of Recommender Systems

Recommender Systems(RS) are information filtering systems that tackle the problem of information overload[26] by filtering important information fragments out of large amounts of dynamically generated information according to the user's preferences, interests, or observed behavior about an item[34]. With the exponential growth of data stored in enterprises, and with the goal of increasing profits and improving user retention, RS plays a more and more important role in e-commerce. For example, Netflix is a famous streaming platform, according to McKinsey, 75% of what users watch on it comes from movie recommendations[7], and RS can help them improve the retention rate, which in turn translates to savings on customer acquisition (estimated $1B per year as of 2016)[9]. Behind RS, there are three strategies that should be combined with each other to increase the recommendation quality[12]:

- **Global**: providing users with the most frequently purchased, trending, or popular items.
- **Contextual**: relying on the item attributes, and purchased items together, also can

combine user attributes such as geolocation, and age to divide into different groups.

- **Personalised**: require not only contextual data but also the user's behavior data, such as viewing, clicking, and purchasing history.

2.1.1 Main Methods

There are six different types in RS[4]. However, three main techniques of them are to be described in this section: Collaborative Filtering, Content-based Filtering, and Hybrid.

- **Content-based Filtering(CB)**: recommending similar items to a user based on their past preferences. This kind of algorithms match the attributes of a user's profile, where their preferences and interests are stored, with the attributes of an item to recommend new, interesting items to the user[32]. For example, if a user watched a romantic movie performed by a specific actor, another romantic movie performed by the same actor will be recommended to this user.
- **Collaborative Filtering(CF)**: as opposed to considering item attributes, focuses on filtering or assessing items based on the feedback of other users[39]. It recommends a user an item based on the items purchased or rated by similar users[12]. The underlying concept behind this approach is that the rating given by a user to an unrated item can be predicted based on the observed ratings provided by other similar users or items. For example, if user A likes apples, bananas, and mangoes, and user B likes apples, mangoes, and peaches. Then bananas are recommended to user B, and peaches are recommended to user A as they share the high similar interests.
- **Hybrid**: combining multiple algorithms together to take advantage of the strength of each algorithm. Since each algorithm has its weaknesses and strength, it can achieve synergy between them by fusing together[5], for example, CF is unable to recommend the new items since it based on past user interactions, which is a problem in many domains[58], while CB can fix this problem by considering the similarity between item attributes. With high flexibility, different components in this technique can be developed separately and also can be integrated[12].

2.1.2 Data Type

In this section, we discuss two different matrices as input to RS, the Item-Content Matrix, and the User-Rating Matrix.

- **Item-Content Matrix(ICM)**: a rectangular matrix containing the information of items and attributes, each row represents an item, and each column represents an attribute. In simple form, ICM is filled with binary values (0 and 1), if the value is set to 1, it means that the corresponding item has the specific attribute. In a more practical form, the values in the matrix are positive real, which represents how important an attribute is in an item.
- **User-Rating Matrix(URM)**: a rectangular matrix containing the opinions of users towards items. Each row represents a user, and each column represents an item. The value of the matrix, called rating, can be expressed both in implicit or explicit presentation.
  - **Implicit Rating**: the rating is a binary value, 1 indicates a user performs a positive interaction on an item, while 0 indicates there is no information that can indicate a positive interaction between a user and an item. Implicit data is easy to collect and usually is extracted from users' behaviors[23].
  - **Explicit Rating**: the rating is a range of numbers given by a user to an item. Similarly, 0 indicates we do not have information on the interaction between a user and an item. Explicit data is hard to collect as it needs users to impose additional behaviors, such as ratings purchased items[23].

The focus of this paper lies in CF algorithms. The key advantage of these algorithms is their independence from machine-analyzable content, enabling them to deliver precise recommendations. Therefore, it doesn't require the item's attributes, the main input is URM. K-nearest neighbors (KNN) based methods and machine learning methods as the two main sub-classes of CF will be described in the following part.

2.2 KNN Methods

This kind of recommendation algorithms accesses all the data in URM and defining the similarity between users or items, and then provides recommendations by utilizing the behavior of the K most similar ones which are selected by leveraging the KNN technique[1]. KNN is a simple concept, it is based on a similarity metric between the users or items in URM, and find the K closest items, where K is a hyper-parameter that should be specified, a hyper-parameter is a parameter that is not learned by the model itself but must be specified before the model begins the learning process and it control the learning process of an algorithm and influence the final parameters of the model[2].

When it comes to similarity metric, there are several options. More specifically, given a

set of items I and a set of users U, for any two users $u, v \in U$, the similarity between them can be simply expressed as the vector of ratings that user u gave to each item i, times the vector of ratings that user v gave to these items:

$$s_{uv} = \vec{u} \cdot \vec{v} = \sum_{i \in I} r_{ui} \cdot r_{vi} \tag{1}$$

However, in the more realistic scenario, cosine similarity and Pearson's correlation are popular similarity measures, the former is mainly applied for implicit URM, and the latter is for explicit URM.

- **Cosine Similarity**:

$$s_{uv} = \frac{\sum_{i \in I} r_{ui} \cdot r_{vi}}{\sqrt{\sum_{i \in I} r_{ui}^2} \sqrt{\sum_{i \in I} r_{vi}^2} + C} \tag{2}$$

  Actually, cosine similarity calculates the angle between two vectors, the more similar the two vectors are, the greater it is. It is important to note that when the number of common ratings between user u and v is too small, the calculated similarity is not reflective of the true level, so to solve this problem a shrink term C is added to the denominator, it is a hyper-parameter, and is a constant value.

- **Pearson Correlation Coefficient**:

$$s_{uv} = \frac{\sum_{i \in I} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum_{i \in I} (r_{vi} - \bar{r}_v)^2} + C} \tag{3}$$

  In particular, a correlation coefficient takes a value between -1 and 1, and when it is 1 indicates a perfect positive correlation, whereas one variable increases, the other variable also increases proportionally. When it is -1 indicates a perfect negative correlation, whereas one variable increases and the other variable decreases proportionally. While, when it is 0 indicates no correlation or a very weak correlation between the two variables.

According to the object on which the similarity is calculated, the KNN methods are further divided into two main types: user-based CF and item-based CF[53].

2.2.1 User-based CF

User-based CF makes recommendations based on the opinions on an item from users who are similar to the active user(the user that we want to predict). The set of similar users should be defined as a group of top-N similar users from all users in URM[52]. It is defined that each user's rating for item i is represented as r_{vi}, the similarity to the active user

is represented as s_{uv}. Therefore, the estimated rating a user u gives to an item i can be defined as

$$\widetilde{r}_{ui} = \frac{\sum_{v \in KNN(u)} r_{vi} \cdot s_{uv}}{\sum_{v \in KNN(u)} s_{uv}} \tag{4}$$

2.2.2 Item-based CF

Item-based CF suggestes the items similar to the other items liked by the active user. Accordingly, the set of similar items and the way of calculating the similarity between items apply the same idea. The estimated rating a user u gives to an item i can be defined as

$$\widetilde{r}_{ui} = \frac{\sum_{j \in KNN(i)} r_{vj} \cdot s_{ji}}{\sum_{j \in KNN(i)} s_{ji}} \tag{5}$$

2.3 Machine Learning Methods

KNN-based CF suffers from two problems in terms of sparsity, and scalability[19]. If most users rated on a few items, it means that URM has high sparsity, which would make it hard to learn accurate and diverse results. If the size of URM is large, a lot of computational resources and time are required to make recommendations, in this case called low scalability, which can make the RS slow and inefficient. However, those challenges can be effectively addressed by exploiting machine learning models. SLIM, Matrix Factorization, graph-based, graph neural network based are the four most presentative models of this type, which will be well-presented in the following sections.

2.3.1 SLIM

Sparse LInear Models(SLIM[24]) generates top-N recommendations by aggregating from user rating profiles, is a powerful and efficient algorithm that has demonstrated superior performance compared to other collaborative filtering methods in accuracy and effectiveness. It aims to learn a sparse aggregation coefficient matrix W that estimates the pairwise similarity between items, and the predict rating $\widetilde{r}_{ui}$ that a user u would give to an item i is calculated as a sparse aggregation of items that have been purchased by this user, that is:

$$\widetilde{r}_{ui} = \sum_{j \in I} r_{uj} \cdot w_{ji} \tag{6}$$

the matrix notation presents as:

$$\widetilde{R} = R \cdot W \tag{7}$$

where R is URM, W is the item similarity matrix with the shape of n × n, where n is the number of items in URM. It is important to note that the matrix W should not be an identity matrix, as this would result in the recommendation of the target item itself which should be excluded from R by setting $r_{ui} = 0$ or from W by setting the elements of diagonal is 0. To learn the sparse matrix W, we take the MSE with regularization terms as error function:

$$min_W \|R - RW\|_2 + \alpha\|W\|_2 + \beta\|W\|_1 \tag{8}$$

where values of W are both positive (W $\geq$ 0), Diag(W)=0, the hyper-parameters $\alpha > 0$ and $\beta > 0$.

2.3.2 Matrix Factorization

Borrowing the idea that the rating of a user on an item can be thought of as how much the user likes each attribute of the item, times how much importance each attribute in this item[35]. Matrix Factorization (MF) decomposes URM into two sub-matrices, the user feature matrix, and the item feature matrix. In this case, the attributes of the item are called latent factors. Accordingly, a user can be represented as a vector $\vec{x}_u \in \mathbb{R}^d$ measuring the extent of interest the user has in different features, an item can be represented as a vector $\vec{y}_i \in \mathbb{R}^d$ measuring the extent to which the item possesses this features[27]. The dot-product of two factors, $\vec{x}_u \cdot \vec{y}_i$ is the estimated rating of the user on the item. If we expand to a given set of users and items, the number of users is n_u, and the number of items is n_i, and the number of latent factors is n_k, the estimated rating can expand to express as a matrix representation which is composed by estimated rating matrix R with the shape of $n_u \times n_i$, user feature matrix X with the shape of $n_u \times n_k$, and item feature matrix Y with the shape of $n_k \times n_i$. The relationship between them is depicted as following:

$$\widetilde{R} = X \cdot Y \tag{9}$$

To get the best-estimated value of X, and Y, the following two different methods of defining the loss function can be utilized:

- **Mean Squared Error(MSE)**: uses the square 2-norm to minimize the mean square error between the true rating matrix (URM) and the predicted matrix.

$$min_{X,Y} \|R - \widetilde{R}\|_2 \tag{10}$$

  Since MF is a machine learning technique, thus it's crucial to avoid over-fitting the model. To address this issue, one solution is to add the regulation terms for matrices

X and Y into the loss function, these regulation terms can push matrices X and Y to be sparse. Moreover, the importance of the regulation terms is determined by two parameters λ_1 and λ_2. Therefore, the loss function becomes as follows:

$$min_{U,V}\|R - \widetilde{R}\|_2 + \lambda_1\|X\| + \lambda_2\|Y\| \tag{11}$$

- **Bayesian Personalized Ranking(BPR)**: is a loss function used for pairwise learning. This approach is useful in situations where implicit ratings are provided, such as user clicks, views, and purchases. It firstly attempts to reconstruct the personalized preference ranking for each user u in URM, denoted by $>_u$. Based on the assumption of taking missing values as negative ratings, and if a user u has interacted with an item i, called positive item i, it assumes that the user prefers this item over any other non-interacted item j which is called negative item j, denoted by $i >_u j$. Then, it defines the predicted rating difference $\widetilde{r}_{u,ij}$ with the predicted ratings from user u on item i and j, $\widetilde{r}_{ui}$, $\widetilde{r}_{uj}$ as:

$$\widetilde{r}_{u,ij} = \widetilde{r}_{ui} - \widetilde{r}_{uj} \tag{12}$$

Given these information, the individual probability that user u prefers item i over item j can be described with the logistic sigmoid function:

$$p(i >_u j|\theta) = \frac{1}{1 + e^{-\widetilde{r}_{u,ij}}} \tag{13}$$

where θ is the parameter that determines the personalized ranking. Next, based on the knowledge of Bayesian analysis of the problem using the prior probability for the model parameter $p(\theta)$[37], we can convert the last probability equation to the following one:

$$p(\theta|i >_u j) = p(i >_u j|\theta) \cdot p(\theta) \tag{14}$$

The objective of BPR is to optimize the ranking of items in a pairwise comparison for a given user by maximizing the above probability. Since log function is an incremental function, and we assume that each parameter is subject to a normal distribution:

$$p(\theta) \sim N(0, \Sigma_\theta) \tag{15}$$

Therefore the probability can be defined as:

$$\begin{aligned} p(\theta|i >_u j) &= \sum_{(u,i,j)\in D_s} log\frac{1}{1+e^{-\widetilde{r}_{u,ij}}} + logp(\theta) \\ &= \sum_{(u,i,j)\in D_s} log\frac{1}{1+e^{-\widetilde{r}_{u,ij}}} - \lambda\|\theta\|^2 \end{aligned} \tag{16}$$

where $D_s := \{(u,i,j)|i \in I_u^+ \wedge j \in I \setminus I_u^+\}$, λ is the regularization factor.

Finally, to lean the parameter θ, BPR uses a approach called stochastic gradient descent which is an algorithm calculates the gradients by randomly choosing a few samples instead of the whole training dataset and update the parameter. The updating rule which is defined as:

$$\theta = \theta + \alpha(\frac{1}{1+e^{\widetilde{r}_{u,ij}}} \cdot \frac{\partial \widetilde{r}_{u,ij}}{\partial \theta} + \lambda\theta) \tag{17}$$

where α is the learning rate.

In this case, when MF and BPR are combined together, we refer to this approach as MF BPR.

Apart from the basic model, there are other commonly used techniques, such as PureSVD, FunkSVD, Asymmetric SVD, and Alternate Least Square.

PureSVD

PureSVD is a dimensionality reduction technique to identify the underlying features in URM. It assumes all missing values are zero and generates recommendations by reconstructing the matrix by using the truncated singular value decomposition(SVD)[10]. SVD decomposes URM into three different matrices: X, Σ, and Y^{T}.

$$\widetilde{R} = X \cdot \Sigma \cdot Y^{\mathrm{T}} \tag{18}$$

X is an orthogonal matrix of $n_u \times n_k$ shape that represents the user features, Σ is a diagonal matrix of $n_k \times n_k$ shape that represents the singular values, and Y^{T} is also an orthogonal matrix of of $n_k \times n_i$ shape that represents the item features. The singular values are non-negative and they correspond to the importance of each feature in characterizing the users and items in R. As for truncated SVD, it only keeps top-k important singular values and the rest are set to zero such that reduces the dimensionality of data and improve the efficiency on computation.

Alternate Least Square(ALS)

The goal of ALS is the same as the basic MF task, to find the optimal values for two sub-matrices: the user feature matrix X, and the item feature matrix Y, see Equation 9 for the prediction function. However, the difference is that instead of using regulation terms to simplify the model, ALS relies on the idea of iteratively optimizing one matrix, while keeping the other one fixed. The prerequisite of it is that the optimization sub-problem can be analytically solved[21].

FunkSVD

FunkSVD is a technique based on the idea of ALS, the prediction function does not change. However, unlike ALS, FunkSVD does not compute all factors at once. It begins with setting the number of latent factors to 1, therefore, the matrices X and Y are initially a user vector and an item vector, respectively, which means the URM is factorized as the scalar product of these two vectors. Then, ALS is applied to find the optimal values of vectors, and after minimizing the error, another feature is added. During the process of optimizing the vectors for the new feature, the previous feature is fixed.

The operation of incrementally increasing features is repeated until either the desired number of latent factors is achieved, or the error is minimized to the desired value. Note that matrix X and Y need to be recalculated whenever a new user appears in R.

Furthermore, SVD++ can be regarded as an enhanced iteration of FunkSVD as it incorporates global effects. These global effects encompass three aspects: the overall average rating given by a user, the average rating assigned by the user to all rated items, and the average rating received by an item from all users who rated it.

Asymmetric SVD

Asymmetric SVD avoids recomputing the estimated ratings from scratch by decomposing matrix U into two sub-matrices: R and Z. The matrix R of $n_u \times n_i$ shape that contains user ratings for items, while the matrix Z of $n_i \times n_k$ shape that captures the extent of the importance of different features in those items. With matrix Y, the estimated ratings are represented as the product of those three matrices, which is shown as following:

$$\widetilde{R} = R \cdot Z \cdot Y \tag{19}$$

2.3.3 Graph-based Models

Unlike content-based and collaborative filtering techniques using metrics of similarity to generate the recommendation, Graph-based RS employs graph theory to represent and model the associations between users, items, and other side information, and has the ability to effectively leverage the heterogeneous information present in networks by extending the reach of neighborhoods, and they can calculate the closeness between users and items[42]. Moreover, graph-based models can effectively address issues of scalability and data scarcity by utilizing a small amount of user preference data, as it is possible to establish connections between items.

With this approach in RS, users and items are represented as nodes, and relationships between them are represented as edges, eventually, the URM is converted as a bipartite user–item interaction graph, which means user nodes can not connect to other user nodes, only connect to item nodes and vice versa. In an unweighted bipartite graph, all edges have the same weight or importance, see Figure 2.1. While in a weighted one, each edge is assigned a weight or a numerical value that represents its importance or strength, such as an explicit rating a user gives to an item.

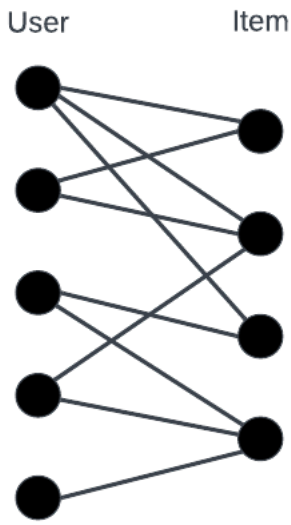


Figure 2.1: Example of an unweighted bipartite graph in which the user nodes are connected to the items they interacted with.

We can define the problem as follows:

Given a set of users U, a set of items I, and a set of ratings L, we can depict them as a

bipartite graph $G = (V, E)$, where $V = U + I$, $E = (u, i) : u \in U, i \in I, u \to i \in L$.

Therefore, the recommendation task transforms into the link prediction task by taking the existing graph(observed ratings) to predict the probability of an edge between unpaired vertices[29, 30]. To solve it, we generate random walk paths starting from a user-item pair and then calculate the probability of jumping from one node to another node based on the paths.

Random walk

In the context of graph theory, a random walk is a mathematical model used to describe a path consisting of a series of random steps taken on a graph. The random walk starts from a specific node within the graph and then randomly selects an adjacent node to travel to at each step, by following the graph's edges. Typically, at each step, the likelihood of moving to any adjacent node is equivalent. However, the probabilities may also be modified based on various factors such as the edge weight, or node degree. Random walks are an example of Markov processes which is a stochastic model that characterizes a series of potential events, where the probability of each event is only determined by the state obtained in the previous event.

In RS, we mainly focus on the scenario of implicit ratings. We denote the number of outgoing links (the number of ratings in RS) from node u as deg(u), and if we take it as the current node at k step, then the probability of arriving at node $v_i \in \{v_i | u \to v\}$ at the step k+ 1 is $\frac{1}{deg(u)}$. If we consider the Markov chain generated by this random walk, the probability of traveling from node i to node j can be inducted as $p_{ij} = \frac{1}{deg(i)}$[41]. Based on this, we can first obtain the probability of arriving at node j at step 1 by summing up the probabilities of arriving at all previous nodes at step 0, times the probability of these nodes moving to j,

$$\Pi_j^{(1)} = \Pi_1^{(0)} p_{1j} + \Pi_2^{(0)} p_{2j} + ... + \Pi_n^{(0)} p_{nj} = \sum_{i:(i,j)\in E} \Pi_i^{(0)} p_{ij} \tag{20}$$

We then extend the scenario to any step k,

$$\Pi_j^{(k+1)} = \sum_{i:(i,j)\in E} \Pi_i^{(k)} p_{ij} \tag{21}$$

P^3

Now, we introduce a simplified version by using this technique, which is P^3. The idea is instead of performing infinite random walks, the algorithm fixes length 3 starting from a target user vertex(Figure 2.2), since from this user u after three steps of jumping, it arrives at a new item node i.

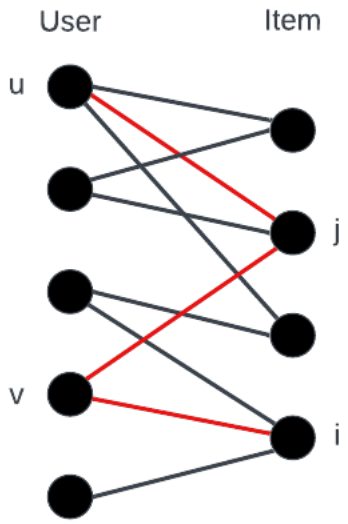


Figure 2.2: Example of three steps of jumping in P^3, which is the route marked in red

In this case, the probability can be represented as follows:

$$\widetilde{r}_{ui} = \Pi_i = \sum_{l \in U} \Pi_l p_{li} = \sum_{j \in I} \sum_{l \in U} \Pi_j p_{jl} p_{li} = \sum_{u \in U} \sum_{j \in I} \sum_{l \in U} \Pi_u p_{uj} p_{jl} p_{li} \tag{22}$$

Since node u is the first node, the probability of arriving at it from previous nodes is equal to 1, thus the equation can be simplified as:

$$\widetilde{r}_{ui} = \sum_{j \in I} \sum_{l \in U} p_{uj} p_{jl} p_{li} \tag{23}$$

If we move summation on l to close to the term $p_{jl}p_{li}$, we can find that it actually calculates the item similarity s_{ij} based on the preferences of user l.

$$\widetilde{r}_{ui} = \sum_{j \in I} p_{uj} s_{ij} = \sum_{j \in I} \frac{r_{uj}}{deg(u)} s_{ij} \tag{24}$$

Moreover, we can disregard the constant term deg(u), this is possible because the denominator does not affect the relative ordering of the scores, which is the aspect we are evaluating (the same would not apply if we were to evaluate the rating prediction) and finally we get the formula as

$$\widetilde{r}_{ui} = \sum_{j \in I} r_{uj} s_{ij} \tag{25}$$

Therefore, it is equivalent to a KNN item-based collaborative filtering algorithm, where the similarity matrix is calculated by taking the dot product of the probability vectors. Based on P^3, $P^3\alpha$ raises the transition probabilities to the power of a fitted parameter α which has been demonstrated to enhance precision in this model[49].The probability p_{ui} to move from user u to item i is computed from the implicit URM as $p_{ui} = (r_{ui}/deg(u))^{\alpha}$ [17]

2.3.4 Heterogeneous Graph Neural Networks

Heterogeneous Graph Neural Networks(HGNNs) are known as applying graph neural networks(GNNs) to heterogeneous graphs(HGs), and in recent years they have become an increasingly popular area of research. The goal is to learn low-dimensional embeddings that capture both the heterogeneous structure and semantics of the data, which can be used in downstream tasks[45]. Many existing HGNNs inherit mechanisms from GNNs for homogeneous graphs, such as the attention mechanism and multi-layer structure and those mechanisms make the model become more complex[54]. In this section, we will briefly discuss the concept of HGs and GNNs, introduce two common networks in GNNs.

Heterogeneous Graphs

HGs are also called heterogeneous information networks. An information network is a conceptualization of the real world that focuses on the objects and the relationships that exist between them[48]. When there is more than one type of object or relationship, the network is a heterogeneous one, i.e. a heterogeneous Graph. For example, in the scenario of movie recommendations, different types of nodes in the graph could be such as users, movies, actors, genres, and directors. The edges could represent different types of relationships, such as "watched", "liked", "acted in", or "directed".

We can define it as follows: Given a set of nodes V and a set of edges E, the sets of types of nodes and edges are denoted as O_V and R_E, respectively, the graph G=$\{V, E, O_V, R_E\}$. When $|O_V| = |R_E| = 1$, the graph is homogeneous one.

Graph Neural Networks

Graph Neural Networks(GNNs) are a type of deep learning method designed to process data in the form of graphs. It can learn representations of nodes and edges in a graph by aggregating and combining feature information from their local neighborhood nodes[55]. In a standard Graph Neural Network, each node in the graph is associated with a vector or a tensor, called a node embedding, which encodes its features and attributes. The GNN operates by iteratively updating these node embeddings by combining information from the node's neighboring nodes and edges, using a set of learnable parameters. In contrast, the method like P_α^3, although graph-based, do not learn embeddings but rather rely on a heuristic approach. Here we detailed discuss two representative networks which are good at modeling homogeneous graphs,

- **Graph Convolution Networks(GCNs)**: employs convolutional aggregations, GCNs architectures usually comprise several graph convolutional layers, which are stacked one after the other, and each layer learns more complex features than the previous layer. To achieve this, GCNs concatenates the inputs derived from node features and graph adjacency matrices[57]. Here is an example of a layer-wise propagation rule introduced by Kipf & Welling, 2017[25]

$$H^{(l)} = \sigma(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}H^{(l-1)}W^{(l)}) \tag{26}$$

  where $H^{(l)}$ is the representation of all nodes in the l-th layer,$W^{(l)}$ is a weight matrix, $\sigma(\cdot)$ is a non-linear activation function, $\widehat{A}$ is an adjacency matrix with self-loops, which is defined as $\widehat{A} = A + I$, A is the adjacency matrix of the user-item graph, I is the identity matrix, and $\widehat{D}$ is the diagonal node degree matrix of $\widehat{A}$.

- **Graph Attention Networks(GATs)[51]**: neural network architectures are specifically designed to learn the significance of nodes and edges in a graph. They achieve this by employing attention mechanisms, which enable models to concentrate on specific aspects of input data during predictions. This is accomplished by assigning weights to different input elements based on their relevance to the given task. As a result, the model can focus on the most relevant information rather than the entire sequence. This is accomplished by allowing each node to focus on its neighbors and assigning different weights based on their importance. GATs implement attention using self-attentional layers, where each node in the graph is represented by a feature vector. The feature vectors of neighboring nodes are combined using attention weights, which are learned during training.

According to the given description, the initial step in computing a weight α_{ij} in the attention mechanism a involves calculating attention coefficients e_{ij}. These coefficients reflect the significance of node j's features in relation to node i.

$$e_{ij} = a(W\vec{h_i}, W\vec{h_j}) \tag{27}$$

where W remains the same representation, a weight matrix, $\vec{h}$ represents a node feature.

In this scenario, the computation of e_{ij} is limited to nodes $j \in \mathcal{N}i$, where $\mathcal{N}i$ represents the neighborhood of node i in the graph. Furthermore, to facilitate comparability across various nodes, we normalize the coefficients by applying the softmax function to all possible choices of j. This process yields the preliminary version of the attention weight, which is described as follows:

$$\alpha_{ij} = softmax_j(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} exp(e_i k)} \tag{28}$$

The attention mechanism a, is actually a single-layer feedforward neural network. It is parameterized by a weight vector $\vec{a}$ and incorporates the LeakyReLU nonlinearity. This leads to the final description of the weight as follows:

$$\alpha_{ij} = \frac{exp(LeakyReLU(\vec{a}^T[W\vec{h_i}||W\vec{h_j}]))}{\sum_{k \in \mathcal{N}_i} exp(LeakyReLU(\vec{a}^T[W\vec{h_i}||W\vec{h_k}]))} \tag{29}$$

where $\cdot^T$ is the symbol of transposition, $||$ is a concatenation operation.

By stacking multiple self-attentional layers, GATs are able to capture increasingly complex relationships between nodes. This method avoids depending on prior knowledge of the graph structure or resorting to expensive matrix operations like inversion.

Moreover, to improve performance, GAT applies the multi-head attention mechanism which is an attention mechanism module that performs multiple attention computations in parallel, each one is called an attention head. In this way, the model is able to simultaneously focus on various aspects of the input, from different feature subspaces and at different positions. The results of these independent computations are then combined by concatenating them and passing them through a linear transformation to obtain a final attention score.[50]

2.4 Evaluation

To determine how well a recommender algorithm is able to predict the preferences of users, and how relevant the recommended items are, we need to apply some methods to evaluate its performance, which is a challenging task in the field of RS[22]. It comes down to three main reasons, the first one is the effectiveness of an algorithm varies depending on the scale of data. Some certain algorithms work well on smaller datasets, but as the scale of data grows, the accuracy decreases. The second one is some indicators have an inherent contradiction between them, such as the accuracy indicator and the diversity indicator. The last one is various indicators need to be measured by different testing and evaluation methods[8]. There are three experiments used to evaluate the performance, online evaluation, user studies, and offline evaluation.

- **Online Evaluation**: executing large-scale experiments on a deployed system. It is achieved by real users carrying out real tasks[8]. For example, to collect users' feedback by asking them to fill in the questionnaires, or by conducting A/B tests, where two different versions of recommender systems are deployed to users and then compare the behavior of users.
- **User Studies**: with offering some kind of compensation, a selected user group is asked to complete a series of tasks by using the system, and then collect their feedback from questionnaires or interviews[44].
- **Offline Evaluation**: different from the two previous methods, instead of involving the interactions from real users, it relies on existing datasets which should be similar to the ones generated from the systems after they are deployed online[8]. As this thesis takes offline evaluation as the evaluation method, therefore detailed descriptions including dataset partitioning, and evaluation metrics will be presented in the following part.

2.4.1 Offline Evaluation

The basic method for conducting Offline evaluation in RS is a standard practice in ML. Dataset partitioning involves splitting the whole dataset into multiple disjoint sets for training, validation, and testing of the algorithms. The objective of partitioning is done to evaluate how well the model generalizes by constructing the model on the training dataset and then evaluating the performance on the testing dataset[20]. In this thesis, for performing hyper-parameter tuning the dataset is split into three sets:

- **Training Set**: as an input to train the model, in general, is the biggest set compared

to the other two sets.

- **Validation Set**: a smaller subset of the original dataset that is held out from the training set and is used to tune the hyper-parameters and estimate the algorithm's performance.
- **Test Set**: a completely independent dataset that has not been used in the training and evaluation phase. It is used to provide a conclusive evaluation of the model, focusing on assessing the ability of generalization.

In RS, there are several methods used to divide the dataset, here the main two of them are introduced:

- **Hold-out**: URM is divided into three parts by randomly selecting certain percentages of ratings or just selecting k ratings of each user, called leave-k-out. This technique is simple to implement and widely used. Moreover, it is also efficient for evaluating the performance on a large dataset.
- **K-Fold Cross-validation**: URM is partitioned into k equally sized sets, and for each iteration of the cross-validation process, the models are trained using k-1 sets and tested using the one remaining set. The model error is calculated based on the observed-simulated pairs from all k iterations of the process. When the dataset is small or imbalanced, this technique can be particularly advantageous because it diminishes the variability of the evaluation metric and provides more dependable estimates of the model's performance.

2.4.2 Evaluation Metrics

Evaluation metrics are quantitative methods used to measure the recommendation quality of recommender algorithms, to determine how well the RS is performing regarding its objective of presenting personalized and relevant recommendations to users. However, for different tasks, we usually choose to use different metrics. For example, in the rating prediction scenario, the performance evaluation relies on error metrics as they measure inaccuracies between predicted ratings and user ratings. While in the top-N recommendation task, our focus changes from predicting a specific rating to arranging the items based on the user's preferences. In this case, ranking metrics measure the accuracy of ranking between the predicted items and actual items.

Error Metrics

In some cases, the goal of RS is to predict the rating a user would give to an item. To measure how close predicted ratings are to the actual user ratings in the test set T which is consisted of user-item pairs, there are two commonly used and easy interpreted metrics:

- **Mean Absolute Error(MAE)**: measures the absolute difference between predicted ratings and actual ratings.

$$MAE = \frac{1}{|T|} \sum_{(u,i) \in T} |r_{ui} - \widetilde{r}_{ui}| \tag{30}$$

- **Root Mean Squared Error(RMSE)**: similar to MAE, but measures the average of squared difference with normalization.

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{(u,i) \in T} (r_{ui} - \widetilde{r}_{ui})^2} \tag{31}$$

Classification Metrics

Classification metrics evaluate the decision-making capacity of RS. They measure the amount of correct and incorrect classifications as relevant or irrelevant items by ignoring the exact ratings and ranking of items[40]. In general, these metrics such as Recall and Precision, are calculated based on the confusion matrix which represents four results from RS, if a recommended item is relevant to a user, it is considered correct, otherwise incorrect[13].

	Recommended	Not Recommended
Preferred	True Positive(TP)	False Negative(FN)
Not Prefered	False Positive(FP)	True Negative(TN)

Table 2.1: Confusion matrix

- **Recall**: the number of relevant items recommended to the user, divided by all items actually relevant to the user.

$$Recall = \frac{\#TP}{\#TP + \#FN} \tag{32}$$

- **Precision**: the number of relevant items recommended to the user, divided by all

items recommended to the user.

$$Precision = \frac{\#TP}{\#TP + \#FP} \tag{33}$$

- **F1-score**: provides a single numerical value that represents the balance between precision and recall. It ranges from 0 to 1, with a score of 1 indicating perfect precision and recall, and a score of 0 indicating poor performance.

$$F1 = \frac{2 * Recall * Precision}{Recall + Precision} \tag{34}$$

Ranking Metrics

Ranking metrics evaluate the performance of RS by assessing its ability to predict the correct order of items based on the user's preferences, usually, the calculation is based on the top-k items instead of all items.

- **Average Precision(AP)**: the mean of the precision scores after each relevant item is retrieved, and it is sensitive to the ranking of recommended items[56]. Based on the following concepts: @i is the length of recommendation list up to the location of the item i, P@i is the precision of the top-i retrieved items, and R is the total number of correct recommended items, then AP can be defined as:

$$AP@k = \frac{\sum_{i=1}^{k} P@i}{R} \tag{35}$$

  A high AP score indicates successful top-ranking of the correct recommendations.

- **Mean Average Precision(MAP)**: While AP is used to evaluate a single user, MAP (Mean Average Precision) is the average of the AP metric calculated for all N users.

$$MAP@k = \frac{1}{|N|} \sum_{i=1}^{N} AP@k \tag{36}$$

- **Discounted Cumulative Gain(DCG)**: an information retrieval metric that logarithmically discounts positions. The idea is that highly relevant items are more useful when appearing earlier in the results list[12]. It measures the "gain" that each user u receives from being recommended an item i. The DCG for a list of items is the average cumulative gain(CG), taking into account the position of each item in the list, and then discounting it logarithmically. CG means to accumulate

the gain of k items, given $rating@i$ is the predicted rating of each item i in the list, the gain of this item i can be represented as $gain@i = 2^{rating@i} - 1$.

$$CG@k = \sum_{i=1}^{k} gain@i \tag{37}$$

$$DCG@k = \sum_{i=1}^{k} \frac{gain@i}{log_2(k+1)} \tag{38}$$

- **Normalized Discounted Cumulative Gain(NDCG)**: one problem with DCG is that it is not straightforward to compare performances across different queries because the scores are not standardized to a range of 0 to 1[12]. To solve it, DCG is divided by ideal DCG(IDCG) to obtain NDCG, where IDCG is the DCG value obtained by sorting all items in the recommendation list by their true relevance scores.

$$IDCG@k = \sum_{i=1}^{|REL_k|} \frac{gain@i}{log_2(k+1)} \tag{39}$$

$$NDCG@k = \frac{DCG@k}{IDCG@k} \tag{40}$$

  where REL_k represents the list of relevant items up to position k.

- **Single List Normalized Discounted Cumulative Gain(SLNDCG)**: NDCG serves not only to evaluate the quality of recommendations from individual algorithms but also finds utility in scenarios where recommendations from two or more algorithms are combined. In such cases, when measuring NDCG, a single long recommendation list can be created by concatenating all recommendation lists of equal length. During this evaluation, two primary considerations need to be taken into account:

  - Highly relevant items should hold greater value for the user.
  - A relevant item is considered valuable to the user only upon its initial appearance.

  Meanwhile, we defined the length of the recommendation list generated by each algorithm as H, and the order of each algorithm list in the single long list as j.

  Based on these definitions, SLNDCG [14, 18] is introduced, which builds upon traditional NDCG with two significant enhancements.

  - The discounting term is redefined from $\frac{1}{log_2(k+1)}$ to $\frac{1}{log_2((j-1)H+k+1)}$

– In the case of duplicate items appearing on the second and subsequent occurrences, the corresponding ratings $rating@i$ are set to 0.

Recommendation-centric metrics

- **Diversity**: it is the average dissimilarity between all pairs of items in the result list[6], a higher value indicating a more diverse list. The dissimilarity can be measured using a metric that reflects the difference between items, often based on their attributes[28]. Therefore, it is the opposite of similarity. Here, two main metrics are introduced,

  – **Herfindahl Index**: is calculated by aggregating the square of the frequency with which each item has been recommended[15].

  $$Herfindahl = 1 - \frac{1}{rec_t^2} \sum_{i \in I} rec(i)^2 \tag{41}$$

  where rec(i) is the total count of recommendations received by item i from all users, and rec_t represents the total number of recommendations from RS.

  – **Mean Inter-List(MIL)**: is defined as the average pairwise similarity of a list of items and measures the uniqueness of different user recommendation lists[15].

  $$MIL = \frac{1}{|U|^2 - |U|} \sum_{u \in U, v \in U, u \neq v} 1 - \frac{q(u,v)}{c} \tag{42}$$

  where U is the user set, u and v are users, q(u,v) is the number of common recommendation items in their lists, and c is the cut-off.

- **Coverage**: refers to the capability of the recommender system to suggest all items from a given training set to users. In general, it is divided into user-space coverage and item-space coverage.

  The former is the percentage of users to whom an RS can provide item predictions, in some scenarios, the RS may not have a high level of confidence in the accuracy of the prediction for some users. Consequently, user space coverage would measure the proportion of users who receive useful recommendations, while the latter is the percentage of all items that can ever be recommended.

- **Novelty**: refers to the ability to suggest new and surprising items that are unlikely to be familiar to the user. The average popularity(AP) of the recommendation results is usually used to measure. The popularity of an item i $\phi(i)$ is defined as

the number of users who have rated this item after being normalized by setting the item with the highest popularity as 1. Then AP is defined as follows:

$$AP = \frac{\sum_{u \in U} \frac{\sum_{i \in r_u} \phi(i)}{|r_u|}}{|U|} \tag{43}$$

where r_u is the list of recommended items for user u. A high AP value means that the algorithm is more biased to recommend items with high popularity, thus the novelty of the recommendation results is lower and the user's satisfaction with the recommendation results decreases.

Chapter 3: Simple-HGN

In this chapter, we will provide a comprehensive introduction to the structure of the Simple-HGN model and delve into the workings of each individual component. It is worth noting that our work involves the reproduction for this model and deriving ideas for our new model from it. Additionally, we will conclude by providing a brief overview of our forthcoming work in the end.

3.1 Overview of Simple-HGN

Simple-HGN[33] is derived from GATs and incorporates improvements by modifying three established methodologies: the first one is learnable edge-type embedding where edge-type refers to different types of edges in a graph and they can represent different relationships between nodes, such as friendships, family relationships in a social network. The second one is residual connections, and the last one is applying L_2 normalization on the output embeddings. To improve the performance, apart from calculating embeddings from HGNN, see Section 2.3.4, Simple-HGN also adds the pre-trained MF BPR embeddings $\vec{e}_u, \vec{e}_v$, where u and v represent two nodes. As a result, the similarity score between these nodes can be defined as follows:

$$f(u, v) = HGNN(u)^T HGNN(v) + \vec{e}_u \cdot \vec{e}_v \tag{44}$$

using the BPR loss function to train it.

$$Loss(u, v^+, v^-) = -logsigmoid(f(u, v^+), f(u, v^-)) \tag{45}$$

The pipeline of HGNN in the Simple-HGN model is shown below, the purple parts are the improvements over GATs.

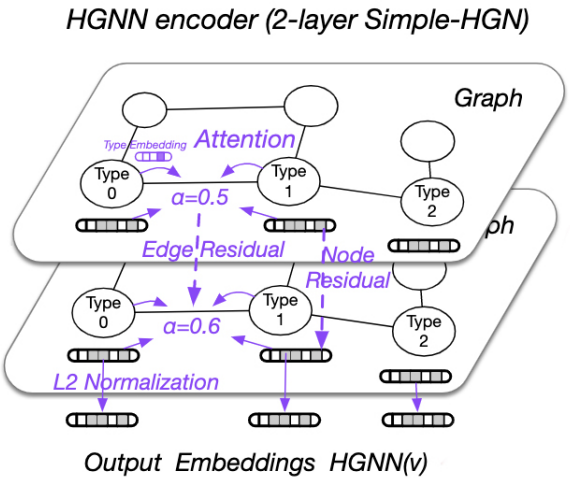


Figure 3.1: The architecture of HGNN in the Simple-HGN model

3.1.1 Learnable edge-type embedding

GATs can also model heterogeneous graphs by simply ignoring the types of node and edge, however, the capacity is not powerful enough. Therefore, by adding edge-type information to attention calculation, this problem can be solved. In detail, considering R_E as the set of edge types, each edge type is encoded into an embedding $r_{O(e)}$ where $O(e) \in R_E$. Consequently, the weight $\widehat{\alpha}_{ij}$ undergoes the following changes:

$$\widehat{\alpha}_{ij} = \frac{exp(LeakyReLU(\vec{a}^T[W\vec{h_i}||W\vec{h_j}||W_r r_{O(<ij>)}]))}{\sum_{k\in\mathcal{N}_i} exp(LeakyReLU(\vec{a}^T[W\vec{h_i}||W\vec{h_k}||W_r r_{O(<ik>)}]))} \tag{46}$$

where $O(< ij >)$ is the type of edge between node i and j, W_r is a weight matrix to transform type embeddings.

3.1.2 Residual Connection

In traditional, data in a feedforward neural network follows a sequential flow through each layer, where the output of one layer becomes the input for the next layer. The residual connection provides another path for data by connecting the output of one earlier layer to the input of another later layer by skipping some intermediate layers. It can help us train deep networks by solving the issues such as exploding gradients, and vanishing gradients.

- **Edge Residual**: is based on Equation 31, the new attention weight on edge i and

j at the l^{th} layer is defined as

$$\alpha_{ij}^{(l)} = (1-\beta)\widehat{\alpha}_{ij}^{(l)} + \beta\alpha_{ij}^{(l-1)} \tag{47}$$

where β is a scaling factor ranging from 0 to 1, and the initial attention weight α_{ij}^{0} is calculated from the initial node feature vectors of the nodes i and j.

- **Node Residual**: The aggregation at the l^{th} layer is defined as

$$\vec{h}_i^{(l)} = \sigma(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^{(l)} W^{(l)} \vec{h}_j^{(l-1)} + \vec{h}_i^{(l-1)}) \tag{48}$$

where σ is the activation function, and $\vec{h}^0 \in \mathbb{R}^{d_0}$ is the initial node feature vector.

- **Multi-head Attention**: To increase the model's ability, similar to GATs, K separate attention mechanisms are applied to nodes, and then the final representation at layer L can be obtained by averaging.

$$\alpha_{ijk}^{(l)} = (1-\beta)\widehat{\alpha}_{ijk}^{(l)} + \beta\alpha_{ijk}^{(l-1)} \tag{49}$$

$$\widehat{h}_{ik}^{(l)} = \sigma(\sum_{j\in\mathcal{N}_i} \alpha_{ijk}^{(l)} W_k^{(l)} \vec{h}_j^{(l-1)}) \tag{50}$$

$$\vec{h}_i^{(L)} = \frac{1}{K}\sum_{k=1}^{K} \widehat{h}_{ik}^{(L)} \tag{51}$$

where α_{ijk} is the attention weight computed by the k^{th} linear transformation matrix $W_k^{(l)}$.

3.1.3 L_2 normalization

L_2 normalization divides each element of a vector by the L_2 norm of the vector itself, and L_2 is the square root of the sum of the squares of the vector's elements. It can help improve the performance and stability of models by reducing the impact of varying magnitudes of inputs.

$$o_i = \frac{\vec{h}_i^{(L)}}{\left|\left|\vec{h}_i^{(L)}\right|\right|} \tag{52}$$

where o_i is the output embedding of node i after normalization, and $\vec{h}_i^{(L)}$ is the output after residual connection.

3.2 Our work

Since the previous evaluation work in the field of recommendation systems have shown that shallow methods can be superior to deep ones for the traditional CF task [11, 16, 17, 36, 38, 47], our goal is to assess whether this is the case also for Simple-HGN. In particular we want to see if the pre-trained MF BPR embeddings are primarily responsible. To achieve this, we introduce Attention-based Matrix Factorization (AMF) which is a modified version of the Simple-HGN model. The Simple-HGN model utilizes HGNN to model different entity types and relationships. In contrast, AMF replaces this deep structure with a novel non-deep structure that combines the concept of graph-based models and the technique of MF.

Our upcoming work consists of three main parts. Firstly, we need to reproduce the results of the Simple-HGN model on four different datasets from the original paper to verify the reliability of those results. This step is crucial because there are often inconsistencies between the source code and the paper, or important details that are not adequately described or explained.

Next, we will conduct an ablation study on the Simple-HGN model. The purpose of this study is to analyze the individual performance of each component in the model structure and provide a visual representation of whether the pre-trained MF BPR embeddings are primarily responsible for the performance.

Finally, after implementing the AMF model, we will train it along with state-of-the-art models. We will compare the best performances achieved by each model and draw conclusions based on the analysis.

Chapter 4: Model

Our model Attention-based Matrix Factorization is derived from Simple-HGN by replacing the most complex deep HGNN component in the Simple-HGN model. We did this based on two assumptions: one is that the HGNN model is too complex for recommendation system scenarios and we have known that shallow methods have been shown to be superior to deep ones for the CF tasks as described in previous evaluation work[11, 16, 17, 36, 38, 47], the other is that the main predictive power of Simple-HGN comes from the simpler part compared to HGNN. Therefore, to simplify the model and further improve its predictive capabilities, we modified Simple-HGN. Regarding assumption one, as discussed in the previous chapter, we have learned that HGNN is a deep learning model based on techniques such as GATs combined with learnable edge-type embedding, residual connections, and others to enhance its predictive capabilities. Deep learning models are generally more complex compared to other machine learning algorithms and are typically applied to tasks with complex training data, such as speech recognition, image recognition, natural language processing, etc. However, for recommendation systems where the main training data is URM, the predictive power of deep learning models often exceeds the required predictive capabilities. As for assumption two, we conducted an ablation study to individually quantify and compare the performance of each component of Simple-HGNN. This allowed us to determine the main source of predictive power in the model among its various parts.

Having acknowledged that the foundation of the AMF model relies heavily on the two assumptions mentioned earlier, this chapter aims to deliver a thorough introduction to the AMF model itself. The content is presented in a structured manner, starting with the definition of AMF and an extensive description of each component in Section 4.1. Following that, Section 4.2 outlines the incorporation of Bayesian Personalized Ranking (BPR) for parameter learning within the model. Lastly, a comprehensive explanation of the model's implementation and training specifics is provided in Section 4.3.

4.1 Our model

The Simple-HGN model is composed of two main parts as introduced in Chapter 3: the HGNN and the pre-trained MF BPR embeddings. Building upon the assumption presented at the beginning of this chapter, we believe that the main predictive power of the model stems from the pre-trained MF BPR embeddings. As mentioned before, we will perform an ablation study to assess its performance.

To create the AMF model, we implemented enhancements to the two main components of Simple-HGNN. The first improvement focused on recognizing the significance of the MF BPR embeddings. We decided to retain this component while introducing a modification: substituting the pre-trained embeddings with learnable embeddings that are trained by the model during the training process. This adjustment was motivated by our understanding of the crucial role played by the MF BPR embeddings in enhancing the model's predictive capabilities. What remains unchanged is that these learnable embeddings are still obtained through MF decomposition. As described in Section 2.3.2, these embeddings correspond to the user embedding and item embedding, which are multiplied together to form a weight matrix composed of various weight coefficients. For ease of reference, we refer to this matrix as the user-item weight matrix.

Next, we will provide a detailed explanation of the user-item weight matrix in terms of vector dimensions. A vector of user embedding $\vec{x}_u$ represents how much the user u likes each feature k, and the set of features is denoted as K. The size of K is the latent factor and also a hyper-parameter needed to be tuned, and a vector of item embedding $\vec{y}_i$ represents how much importance of each feature k in item i. By multiplying these two vectors, we can obtain the predicted rating α_{ui} of user u for item i, which can be expressed as:

$$\alpha_{ui} = \vec{x}_u \cdot \vec{y}_i \tag{53}$$

By unpacking vector dot product we can re-write as:

$$\alpha_{ui} = \sum_{k \in K} x_{uk} \cdot y_{ki} \tag{54}$$

The second improvement involved removing the HGNN component from the Simple-HGN model and replacing it with a simpler non-deep learning model. More specifically, this non-deep learning model comprises two identical structured sub-components: one is based on the user-user similarity and the other on the item-item similarity. Leveraging the previously mentioned user-item weight matrix, these two sub-components utilize the

user embedding and item embedding from the weight matrix, respectively. They combine these embeddings with the idea of graph-based algorithms to calculate two prediction scores.

In Section 2.3.3 we have known that P^3 as a graph-based algorithm, it determines the predicted rating of user u on item i as the likelihood of arriving at item node i from user node u after three random steps, which is equal to the product of the probability p of reaching each node, see equation 23.

In AMF, in addition to the two types of nodes mentioned above, namely item nodes and user nodes, there is an additional type of node called feature nodes. The feature nodes encompass the features that play a role in shaping both user embedding and item embedding within the user-item weight matrix. These feature nodes are introduced to create new paths that can be traversed during the three-step random walk. Based on this, we consider two scenarios. One is from a user node u to firstly reach a feature node m then finally reach an item node i, in this scenario, the set of feature nodes is M. The other is from a user node u to firstly reach an item node j then finally reach another item node i by walking by the feature node n, and correspondingly, the set of feature nodes is N.

At the beginning, we consider the former scenario, as shown in the figure 4.1, where we place the feature nodes between the user and item nodes for a simple and easy-to-read composition.

Correspondingly, if we denote the probability of traveling from node i to node j as p_{ij}, the predicted rating can be expressed as:

$$\widetilde{r}_{ui} = \sum_{m \in M} \sum_{v \in U} p_{um} p_{mv} p_{vi} \tag{55}$$

After moving the summation of v to the front of p_{vi}, it becomes as:

$$\widetilde{r}_{ui} = \sum_{m \in M} p_{um} p_{mv} \sum_{v \in U} p_{vi} \tag{56}$$

the former part $\sum_{m \in M} p_{um} p_{mv}$ calculates the similarity between user u and v, denoted by s_{uv}. In this scenario, the user similarity is calculated based on the user embedding present in the user-item weight matrix. Since the weights in the user embedding are continuous values rather than binary (0 or 1), we utilize the Pearson coefficient to calculate the similarity between users. Meanwhile, in the latter part p_{vi} represents the probability of a

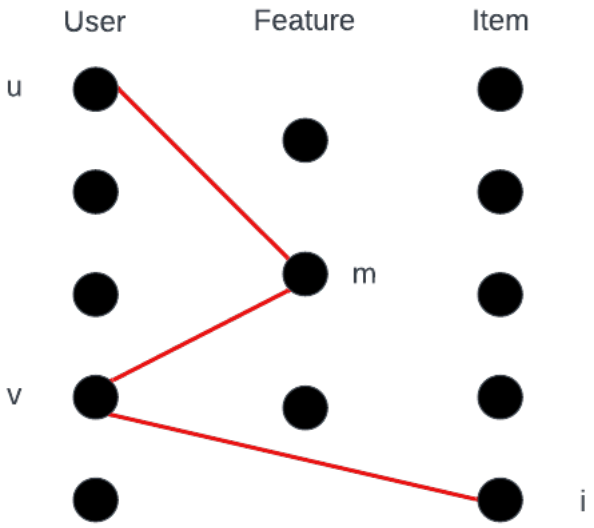


Figure 4.1: The three-step jumping starting from a user node u to reach an item node i by firstly walking by a feature node m

user v reaching an item node i. However, unlike P^3 this probability can be computed from URM, here we try to estimate it by using MF as well and it represents the importance of user v's rating for item i, which is denoted by α_{vi}. This MF is also composed of a set of embeddings. Since it is applied in a graph-based context, we refer to this set of embeddings as graph-based embeddings.

So far, the predicted score of user u for item i is obtained by multiplying the similarity and corresponding importance between user u and each other user v, and then summing them up. The importance is derived from the model training based on the user similarity, therefore we consider it as an attention mechanism. We refer to these importance values as attention weights ranging between 0 to 1. A weight of zero indicates that a particular similarity rating is meaningless and should be disregarded. This approach is akin to the User-based CF technique.

If we use the vectors $\vec{x}_u$, $\vec{x}_v$ to describe how much users u, v like each feature respectively, therefore we can further obtain:

$$\widetilde{r}_{ui} = \vec{x}_u \cdot \vec{x}_v \sum_{v \in U} \alpha_{vi} = \sum_{v \in U} \vec{x}_u \cdot \vec{x}_v \cdot \alpha_{vi} = \sum_{v \in U} s_{uv} \cdot \alpha_{vi} \tag{57}$$

$$\alpha_{vi} = \sum_{m \in M} x_{vm} \cdot y_{mi} \tag{58}$$

where the size of M is the latent factor and hyper-parameter, and s_{uv} is equal to 0 when user u and user v are the same.

In the second scenario, where we start from a user node u and aim to reach item node j and then another item node, the random walk path is illustrated in Figure 4.2.

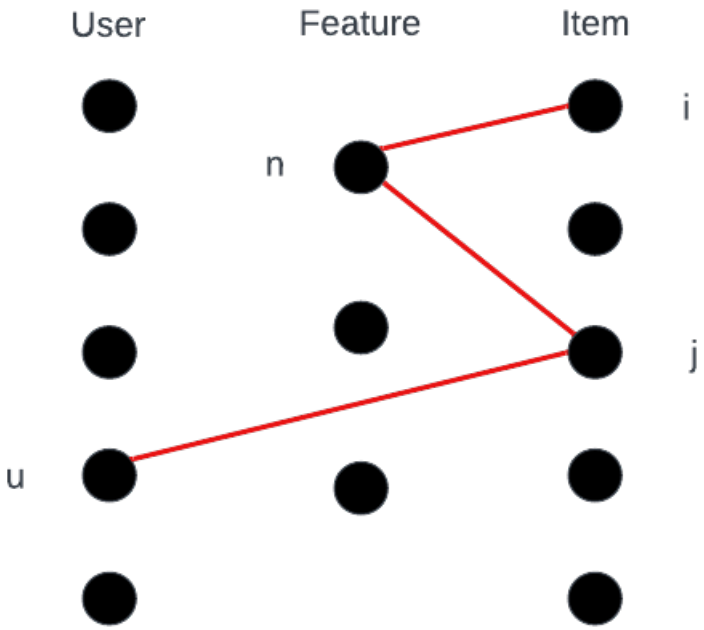


Figure 4.2: The three-step jumping starting from a user code u to reach an item node i by firstly walking by another item node j

Similarly, the predicted rating is defined as:

$$\widetilde{r}_{ui} = \sum_{j \in I} \sum_{n \in N} p_{uj} p_{jn} p_{ni} \tag{59}$$

After moving the summation of n to the front of p_{jn}, it becomes as:

$$\widetilde{r}_{ui} = \sum_{j \in I} p_{uj} \sum_{n \in N} p_{jn} p_{ni} \tag{60}$$

The subsequent part involves calculating the similarity between item i and j, denoted by

s_{ij}, which is obtained from the item embedding in the user-item weight matrix. Furthermore, similar to the previous case, the Pearson coefficient is employed to compute the item similarities, owing to the floating-point nature of the item embedding weights.

Additionally, p_{uj} represents the probability of reaching item node j from user node u. Similarly, we replace this probability with a predicted value α_{uj}, which is obtained using another set of graph-based embeddings. We consider α_{uj} as an indicator of the importance of user u to the ratings of other item j. Like before, this importance is derived from the model training based on item similarity and follows the same attention mechanism. Therefore, we refer to these importance values as attention weights, ranging between 0 and 1. This approach is similar to the one employed in Item-based CF technique. We use the vectors $\vec{y}_i$,$\vec{y}_j$ to describe how much importance of each feature in items respectively, the number of features is N,then we can obtained:

$$\widetilde{r}_{ui} = \sum_{j \in I} \alpha_{uj} \vec{y}_j \cdot \vec{y}_i = \sum_{j \in I} \alpha_{uj} \cdot s_{ji} = \sum_{j \in I} s_{ji} \cdot \alpha_{uj} \tag{61}$$

$$\alpha_{uj} = \sum_{n \in N} x_{un} \cdot y_{nj} \tag{62}$$

where the size of N is the latent factor and hyper-parameter, and s_{ji} is equal to 0 when item i and item j are the same.

Therefore, in total, the whole formula of predicted ratings is depicted as follows:

$$\begin{aligned} \widetilde{r}_{ui} &= \vec{x}_u \cdot \vec{y}_i + \sum_v \vec{x}_u \cdot \vec{x}_v \alpha_{vi} + \sum_j \vec{y}_j \cdot \vec{y}_i \alpha_{uj} \\ &= \alpha_{ui} + \sum_{v \in U} s_{uv} \cdot \alpha_{vi} + \sum_{j \in I} s_{ji} \cdot \alpha_{uj} \end{aligned} \tag{63}$$

where

$$\alpha_{ui} = \sum_{k \in K} x_{uk} \cdot y_{ki} \tag{64}$$

$$\alpha_{vi} = \sum_{m \in M} x_{vm} \cdot y_{mi} \tag{65}$$

$$\alpha_{uj} = \sum_{n \in N} x_{un} \cdot y_{nj} \tag{66}$$

4.2 Learning the solution

Based on the description above, AMF further incorporates additional predicted scores by applying an attention mechanism based on the similarity calculated for the two embeddings in the user-item weight matrix, the weights obtained by multiplying the two sets of graphed-based embeddings separately. Since all the parameters required for learning the solution in the entire algorithm are both derived from these embeddings. Therefore, similar to MF BPR mentioned in Section 2.3.2, we decided to also use BPR technique to learn the optimal solution.

In order to train the model, we first need to perform BPR sampling from URM in a uniform and random way, and each sample is composed of three elements <u,a,b>:

- **u**: an user who have at least an interaction in their user profile.
- **a**: a positive sample which is an item the user u interacted with.
- **b**: a negative sample which is an item the user u did not interact with.

During the training process, BPR algorithms uses these samples to continuously optimize the parameters by stochastic gradient descent following the rule below:

$$\theta = \theta + \alpha(\frac{1}{1 + e^{\widetilde{r}_{u,ab}}} \cdot \frac{\partial \widetilde{r}_{u,ab}}{\partial \theta} + \lambda\theta) \tag{67}$$

In the scenario of AMF, the predicted rating difference between positive item a and negative item b from user u can be described as:

$$\begin{aligned} \widetilde{r}_{u,ab} &= \alpha_{ua} + \sum_{v \in U} s_{uv} \cdot \alpha_{va} + \sum_{j \in I} s_{ja} \cdot \alpha_{uj} - (\alpha_{ub} + \sum_{v \in U} s_{uv} \cdot \alpha_{vb} + \sum_{j \in I} s_{jb} \cdot \alpha_{uj}) \\ &= (\alpha_{ua} - \alpha_{ub}) + \sum_{v \in U} s_{uv} \cdot (\alpha_{va} - \alpha_{vb}) + \sum_{j \in I} (s_{ja} - s_{jb}) \cdot \alpha_{uj} \end{aligned} \tag{68}$$

For ease of reading, we use the same nomenclature as $\widetilde{r}_{u,ab}$ for the each pair of differences between the same type of parameters in the above equation and also for subsequent equations, then $\widetilde{r}_{u,ab}$ in turn can be simplified to express as:

$$\widetilde{r}_{u,ab} = \alpha_{u,ab} + \sum_{v \in U} s_{uv} \cdot \alpha_{v,ab} + \sum_{j \in I} s_{j,ab} \cdot \alpha_{uj} \tag{69}$$

Once we obtain the $\widetilde{r}_{u,ab}$, following the updating rule, we need to calculate the derivative of it with respect to the parameter θ. Since the similarity terms in the latter two parts are calculated based on the embeddings in the user-item weight matrix, we proceed with a

comprehensive simplification by decomposing each of the three terms into scalar products:

The first term:

$$\alpha_{u,ab} = \sum_{k \in K} x_{uk} \cdot y_{ka} - \sum_{k \in K} x_{uk} \cdot y_{kb} = \sum_{k \in K} x_{uk} \cdot (y_{ka} - y_{ka}) = \sum_{k \in K} x_{uk} \cdot y_{k,ab} \tag{70}$$

The second term:

$$\begin{aligned} \sum_{v \in U} s_{uv} \cdot \alpha_{v,ab} &= \sum_{v \in U} \sum_{k \in K} x_{uk} \cdot x_{vk} \cdot (\sum_{m \in M} x_{vm} \cdot y_{ma} - \sum_{m \in M} x_{vm} \cdot y_{mb}) \\ &= \sum_{v \in U} \sum_{k \in K} x_{uk} \cdot x_{vk} \cdot \sum_{m \in M} x_{vm} \cdot (y_{ma} - y_{mb}) \\ &= \sum_{v \in U} \sum_{k \in K} x_{uk} \cdot x_{vk} \cdot \sum_{m \in M} x_{vm} \cdot y_{m,ab} \\ &= \sum_{v \in U} \sum_{k \in K} \sum_{m \in M} x_{uk} \cdot x_{vk} \cdot x_{vm} \cdot y_{m,ab} \end{aligned} \tag{71}$$

The third term:

$$\begin{aligned} \sum_{j \in I} (s_{ja} - s_{jb}) \cdot \alpha_{uj} &= \sum_{j \in I} (\sum_{k \in K} y_{kj} \cdot y_{ka} - \sum_{k \in K} y_{kj} \cdot y_{kb}) \cdot \sum_{n \in N} x_{un} \cdot y_{nj} \\ &= \sum_{j \in I} (\sum_{k \in K} y_{kj} \cdot (y_{ka} - y_{kb}) \cdot \sum_{n \in N} x_{un} \cdot y_{nj} \\ &= \sum_{j \in I} \sum_{k \in K} y_{kj} \cdot y_{k,ab} \cdot \sum_{n \in N} x_{un} \cdot y_{nj} \\ &= \sum_{j \in I} \sum_{k \in K} \sum_{n \in N} y_{kj} \cdot y_{k,ab} \cdot x_{un} \cdot y_{nj} \end{aligned} \tag{72}$$

Based on the above equations, we have to differentiate with respect to the parameters used in embeddings of three different MFs. It should be noted that we have two parameter sets for each sample, the weights of the positive item and those of the negative one, we have to update both. Moreover, when finding the partial derivative, all the terms except θ itself are constant. Therefore, the derivative of $\widetilde{r}_{u,ab}$ with respect to the each parameter

θ in AMF can be specified as:

$$\frac{\partial \widetilde{r}_{u,ij}}{\partial \theta} = \begin{cases} y_{k,ab} + \sum_{v \in U} x_{vk} \cdot \alpha_{v,ab} & \text{if } \theta = x_{uk}, \\ x_{uk} + \sum_{j \in I} y_{kj} \cdot \alpha_{uj} & \text{if } \theta = x_{uk}, \\ -x_{uk} - \sum_{j \in I} y_{kj} \cdot \alpha_{uj} & \text{if } \theta = y_{kb}, \\ \sum_{v \in U} s_{uv} \cdot y_{m,ab} & \text{if } \theta = x_{vm}, \\ \sum_{v \in U} s_{uv} \cdot x_{vm} & \text{if } \theta = y_{ma}, \\ \sum_{j \in I} s_{j,ab} \cdot y_{nj} & \text{if } \theta = y_{mb}, \\ \sum_{j \in I} s_{j,ab} \cdot y_{nj} & \text{if } \theta = x_{vn}, \\ \sum_{j \in I} s_{j,ab} \cdot x_{un} & \text{if } \theta = y_{nj}, \\ 0 & \text{else} \end{cases}$$

4.3 Implementation details

4.3.1 Technologies adopted

Python[31] was chosen as the programming language for developing AMF model due to its simple syntax and readable code. Additionally, it has a large number of useful third-party libraries and frameworks. For instance, the two main libraries that were relied upon in this development, Numpy library is used to store and support large matrices operations, while Scipy library is used for scientific computations such as statistics and linear algebra. Additionally, PyTorch is chosen as the development framework because of its effectiveness in training machine learning models. Its primary advantage is the provision of two core functions, which are listed below:

- **Support for GPU-accelerated tensor calculations**: tensors are similar to NumPy's ndarrays, except that tensors can run on GPUs. They are used to encode the inputs and outputs of a model, as well as the model's parameter. In fact, tensors and NumPy arrays can often share the same underlying memory, eliminating the need to copy data.
- **Automatic differentiation mechanism that facilitates optimization of the model**:as mentioned before, parameters of model are adjusted according to the gradient of the loss function with respect to the given parameter. To compute those gradients, PyTorch has a built-in differentiation engine called torch.autograd. It supports automatic computation of gradient for any computational graph, and

tensors are optimized for automatic differentiation.

During the preparation stage of data training, it becomes challenging to perform complicated operations on URM after reading it into memory due to the large size. In addition, computing the loss function requires calculating on each data sample, which can drastically reduce the iteration speed. To tackle this issue, the dataset is divided into several batches, and the size of each batch is a hyper-parameter, which is usually a multiple of 32 or a factor. Therefore, the update of model parameters is no longer based on individual samples but on individual batches. After conducting practical experiments, we found that the PyTorch's DataLoader object which returns a batch of training data on each iteration while iterating through the dataset, is not efficient enough for our recommender system dataset scenario. Consequently, we solve this problem by exploiting a new DataLoader implemented in Cython[3] and the return value of it is a batch of BPR triple samples.

4.3.2 Model implementation

The model implementation involves three steps: defining the model, defining the loss, and selecting an optimization technique. Since the last step is mostly built into PyTorch, we just need to work on the former two steps, let's start with the most challenging first. All models in PyTorch inherit from torch.nn.Module, which requires only the class and a forward method to be defined. Upon initializing the class, We define three sets of MF embeddings, corresponding to the user-item weight matrix and two pairs of graph-based embeddings used to compute the attention weights. In order to make the model easier to converge, we modify the parameter distribution of the PyTorch initialization embedding from the default normal distribution with a mean of 0 and a variance of 1 to a normal distribution with a mean of 0 and a variance of 0.1. In the forward function, we translate it into code logic following the predicted rating formula. Furthermore, as the prediction formula requires numerous matrix multiplication operations and all matrices are transformed into the tensors in PyTorch, therefore we use the Einstein summation method to enhance operation efficiency and simplify the code. This method can reduce the calculation involved in tensor multiplication and express the tensor operation in a concise formula according to the repeated index convention, avoiding element expansion. We take the first part of prediction formula as an example to explain it visually:

$$\alpha_{ui} = \sum_{k \in K} x_{uk} \cdot y_{ki} \tag{73}$$

In this case, the index of α is ui, the index of x and y is uk, kj respectively. It is evident that the value of k determines the values of x and y, which in turn determines the value

of α. Therefore, it serves as a repeated index. Based on this, this part can be expressed using the Einstein summation convention as follows:

$$uk, ki \rightarrow ui \tag{74}$$

We begin by expressing the product of the first pair of MF embeddings, which results in the user-item weight matrix, using the Einstein summation convention. Then, based on their user embedding and item embedding, we calculate the similarity matrices using the Pearson coefficient method. These similarity coefficients are individually L_1-normalized and stored in two tensors for ease of computation in the subsequent steps.

Next, we utilize the remaining two pairs of MF embeddings, as defined earlier, to calculate two attention weight matrices using the Einstein summation convention. To ensure that each weight falls within the range of 0 to 1, we perform Min-Max scaling on these two matrices.

Up to this point, we have obtained the user similarity matrix along with its corresponding attention weight matrix, as well as the item similarity matrix along with its corresponding attention weight matrix. The final step is to multiply these two pairs of matrices element-wise. Since the resulting matrices have the same shape as the user-item weight matrix, which is $|U| \times |I|$, we sum them together to obtain the final predicted rating matrix.

It is worth noting that since we train the model on a batch-wise basis, during the calculation of the predicted rating, we do not compute the complete three MF matrices at once. Instead, we slice and extract the corresponding sub-matrices based on the user and item information within the batch. This allows us to obtain the predicted rating matrix specific to the current batch.

Once the prediction ratings matrix is computed, we proceed to specify the loss function. Since the PyTorch framework does not have a built-in function for BPR loss, we developed one that uses the implemented forward function to obtain the predicted ratings for both negative and positive items of users and calculate the BPR loss for each batch. To express the loss, we need to assign a minus sign in front of Equation 16 which represents the probability for the model parameter. Then, the BPR loss function can be expressed as:

$$Loss_{BPR} = -\sum_{(u,i,j)\in D_s} log\frac{1}{1+e^{-\widetilde{r}_{u,ij}}} + \lambda\|\theta\|^2 \tag{75}$$

In fact, during the implementation process, we removed the regularization part from the loss function because it is already included in the optimizer, as described later. Therefore,

the loss function is simplified to:

$$Loss_{BPR} = -\sum_{(u,i,j)\in D_s} log\frac{1}{1+e^{-\widetilde{r}_{u,ij}}} \tag{76}$$

In the PyTorch framework, there are multiple optimizers to choose from. When using them, we simply need to specify the model parameters to be optimized, the learning rate, and the weight decay (also known as L_2 regularization). When weight decay is specified, a penalty term, which is the sum of the squares of the parameters, is added to the loss function. This penalty term corresponds to the omitted regularization part mentioned earlier in the loss function. In our experiment, we select four of optimizers as options, SGD, Adagrad, RMSprop, and Adam. We set the name of optimizer as a hyper-parameter to specify the one used for stochastic gradient descent before the model training phase.

After the above three steps are completed, the model training process for each batch is executed as follows:

- The optimizer is cleared of any previous computed gradients.
- The loss for the current batch is computed.
- Backpropagation is performed to generate the gradients.
- The optimizer updates the parameters using the computed gradients.

Due to the large number of matrix operations involved, in order to shorten the training time, we put the model and the data used in the calculation process on the GPU.

4.3.3 Hyper-parameter tuning

The predictive performance of recommender algorithms is heavily influenced by the hyper-parameters used during their training process. Hyper-parameter tuning aims to optimize the model for achieving the best possible performance on unseen data. There are some essential approaches, here we exploit one of the most efficient approaches, which is Bayesian Optimization(BO), an automatic approach that models the generalization performance of a learning algorithm as a sample from a Gaussian process (GP). The GP's tractable posterior distribution enables efficient use of information from previous experiments, leading to optimal choices about which parameters to try next[46]. BO has outperformed other state-of-the-art optimization algorithms in many problems, such as reducing the number of evaluations required for searching the next set of parameters to test. In our implementation, the Scikit-Optimize library in Python is utilized as a tool of BO.

Given that AMF model is trained based on stochastic gradient descent technique, early stopping is essential for efficiently identifying the optimal model performance and terminating training for each set of hyper-parameters generated by BO. Early stopping entails monitoring the model's performance on a validation dataset throughout the training process, and stopping the training process early when the performance on the validation dataset starts to decline, using a stopping criterion such as the condition that the metric of interest ceases to improve for five consecutive evaluations.

Chapter 5: Experiments

We divide this chapter into four sections, we first in Section 5.1 describe the datasets used to do experiments by showing the detailed information, such as the distribution of interactions, the density,etc.. Then two experiments were carried out on Simple-HGN: one is to test the reproducibility of the model results of the original article in Section 5.2, and the other is to conduct an ablation study to figure out how the performance changes without the pre-trained MF BPR embeddings in Section 5.3. Finally, we describe what and how experiments were performed to evaluate the performance of AMF in Section 5.4.

5.1 Datasets

As AMF is based on Simple-HGN model, we use the same four datasets which are Movielens, Yelp-2008, LastFM, and Amazon-book to train our new algorithm. This ensures that the training results can be compared directly. The collaborative information in these datasets comprises of implicit ratings, and the content-based information is not involved. All the benchmark algorithms' results are also obtained on these same datasets. In this section, we will explain the characteristics of these datasets and the method used to split the data for training, validation, and testing.

5.1.1 MovieLens

The GroupLens research group at the University of Minnesota developed the MovieLens [1] dataset, which is widely used in the field of RS. This dataset includes ratings on a scale of 1 to 5 for various movies, and there are four versions available with varying numbers of ratings: 100K, 1M, 10M, and 20M. The authors of Simple-HGN model used a subset of the 20M version, where they transformed explicit ratings into implicit ones by only retaining the interaction records between users and items, while discarding the specific rating data. As Simple-HGN model is trained using both the URM and the information of graph nodes and their relationships. Hence, this dataset is composed of two distinct parts of data. One is for building URM and the other is for describing the graph. Since

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

the ablation study involves re-training Simple-HGN model on all of four datasets, all the information about this subset is retained, as described in Table 5.1.

#Users	#Items	#Ratings	#Entities	#Relations	#Triplets
37,385	6,182	539,300	24,536	20	237,155

Table 5.1: The statistics of MovieLens dataset

In this dataset, the number of items rated by users is between 10 and 20, the user frequency of the number of item ratings is depicted in Figure 5.1. It is easy to note that as the number of ratings increases, the number of users decreases linearly. The number of users with a rating count of 20 is approximately half of the number of users with a rating count of 10.

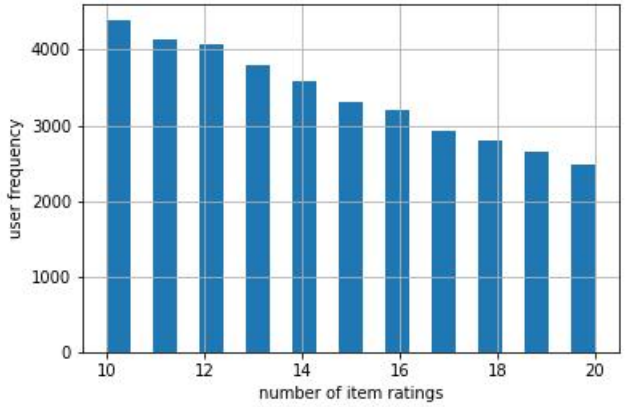
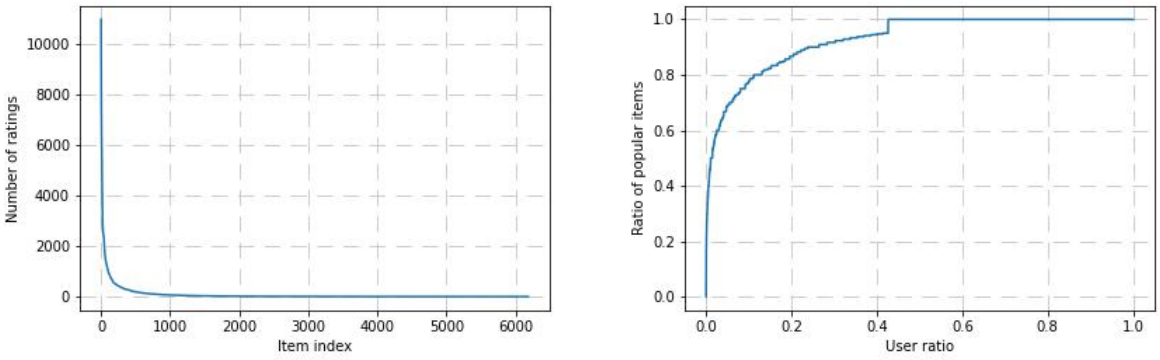


Figure 5.1: The user frequency of the number of items ratings on dataset MovieLens

Figure 5.2(a) displays the popularity of the item, which from the origin and descends quickly from the highest point, showing that only a few items get a lot of ratings. Most of the items in the tail of the curve have only a few ratings, which is called the long tail phenomenon. We take the items corresponding to the first 20% of the curve as popular items, and the rest as non-popular items, and then count the proportion of popular items in each user's profile, and the results are presented in Figure 5.2(b). It is easy to find that the vast majority of users have a high degree of love for popular items, more than 80% of users in their historical rating lists, the number of popular items accounted for about 85% of the total number of rated items, more than 60% of users rated all of the items are popular ones.



(a) Item popularity (b) Ratio of popular items for different users

Figure 5.2: Item popularity and user interest in popular items on dataset MovieLens

5.1.2 Yelp-2018

The Yelp-2018 [2] dataset is a large-scale dataset of user ratings from the Yelp platform which is an online platform that allows users to discover and review local businesses which are viewed as items, such as restaurants, cafes, bars. Yelp-2018 is adopted from the 2018 edition of the Yelp challenge. The detailed information of this dataset is shown below.

#Users	#Items	#Ratings	#Entities	#Relations	#Triplets
45,919	45,538	1,183,610	136,499	42	1,853,704

Table 5.2: The statistics of Yelp-2018 dataset

It is much bigger than the last dataset MovieLens in both the number of items and interactions. Since there is less than one percent of users have the number of ratings higher than 200, and the values are very scattered, therefore in order to make the distribution more intuitive, this part is not shown in Figure 5.3 which describes the distribution of user frequency of the number of ratings. As we can see, there is no user without ratings, and the number of ratings given by most users is concentrated in the range of less than 25. According to the item popularity shown in Figure 5.4(a), the most popular item receives around 1600 ratings from users, which is not as significant as the one on MovieLens. Compared with MovieLens, the decline of the curve from the maximum value of the y-axis to close to 0 is slightly slower, but more than half of the items are still very close to the X-axis, which means that they receive very few ratings and form a long tail. From Figure 5.4(b), the user's dependence on popular items is not as high as that of MovieLens users, only about 20% of users, and the number of popular items rated by them accounts

[2]https://www.yelp.com/dataset

for more than 80% of the total number of rated items, while only a very small number of users, less than 5%, have this percentage of 100%.

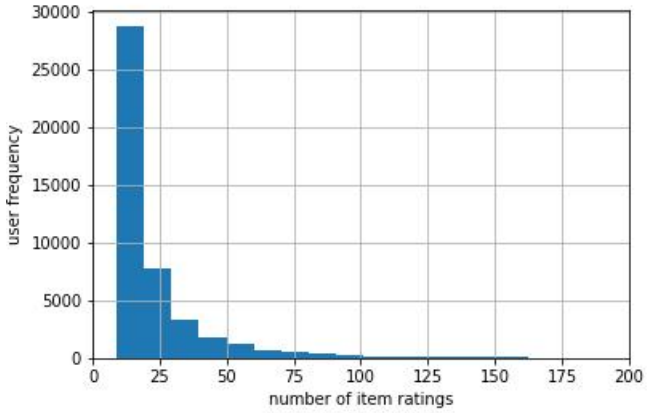
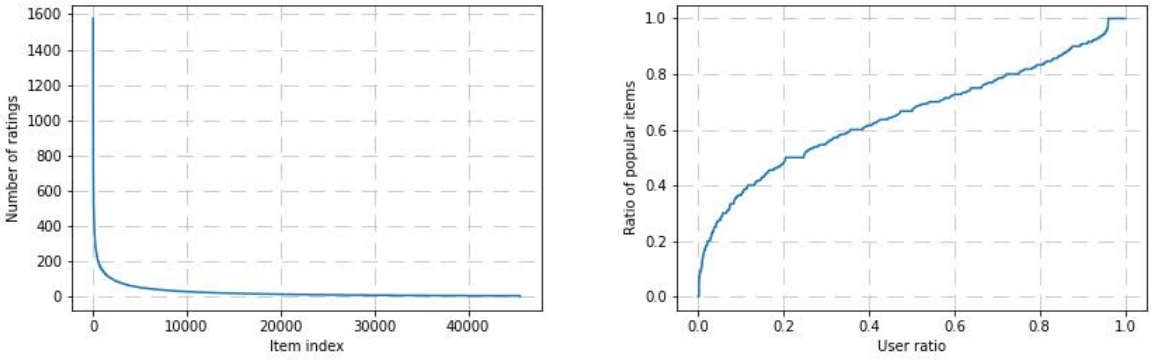


Figure 5.3: The user frequency of the number of items ratings on dataset Yelp-2018



(a) Item popularity (b) Ratio of popular items for different users

Figure 5.4: Item popularity and user interest in popular items on dataset Yelp-2018

5.1.3 LastFM

The LastFM dataset records user listening sequences collected and published by Last.fm which is a music website, founded in the United Kingdom in 2002. It is also often used in the field of RS. The LastFM dataset contains information about users, and the songs they have listened to, therefore in this scenario, a rating is obtained when a user listens to a song. We extract a subset with the timestamp from January,2015 to June,2015. The statistics of this subset is summarized in Table 5.3.

Since the ratings represents the action of the song playing, which means that the duplicate items exist on dataset because one song can be played several times, thus the number of

#Users	#Items	#Ratings	#Entities	#Relations	#Triplets
23,566	48,123	3,034,763	106,389	9	464,567

Table 5.3: The statistics of LastFM dataset

ratings is much more than previous two datasets. However, our models are all trained on the implicit URM, therefore, when this dataset is converted into URM, all the duplicate items are only counted once.

The histogram in Figure 5.5 shows the frequency of users with respect to the number of item ratings, and for the same reason as the previous dataset, we omitted the frequency of users with more than 1200 ratings from the figure. For each user's historical score list, the item popularity before and after removing duplicate values is shown in Figure 5.6(a), and 5.6(b) respectively, it can be seen that the descending slope of the curve slows down after de-duplication, indicating that there are a few popular songs with a high number of repetitions, and the gap between the ratings of non-popular items becomes smaller after de-duplication. From the trend of ratio of popular items for different users in Figure 5.7 is similar to that of Yelp-2018, with the slight difference that the slope of the LastFM curve is relatively slow in the first 40% of users.

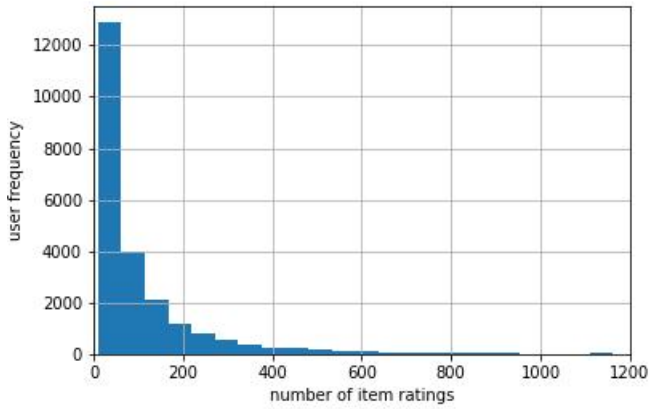
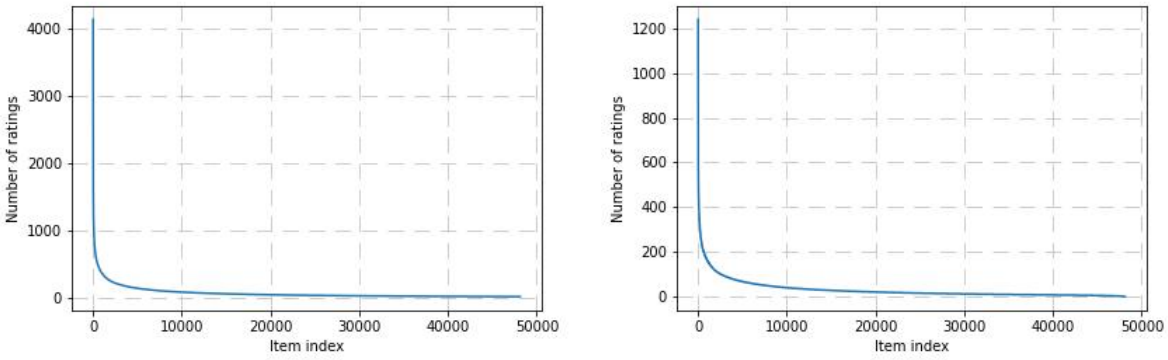


Figure 5.5: The user frequency of the number of items ratings on dataset LastFM

5.1.4 Amazon-book

Amazon-book [3] is another popular dataset that is commonly used to train the models in RS. It is a collection of book ratings and reviews obtained from Amazon.com. The original dataset comprises a vast collection of over 22 million ratings for nearly 2.8 million

[3]https://jmcauley.ucsd.edu/data/amazon/



(a) Original distribution (b) The distribution after de-duplication of users

Figure 5.6: Item popularity on dataset LastFM

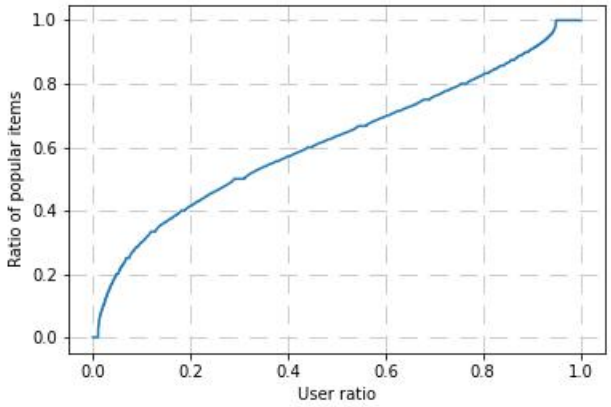


Figure 5.7: Ratio of popular items for different users on dataset LastFM

books from more than 900,000 users. Here we use the subset of it, and it is comprised by the following data:

#Users	#Items	#Ratings	#Entities	#Relations	#Triplets
70,679	24,915	846,434	113,487	39	2,557,746

Table 5.4: The statistics of Amazon-book dataset

Figure 5.8 displays the user frequency of the number of item ratings. As it can be observed that about eighty percent of the users have less than twenty ratings, and there are no users with no ratings. The item popularity and user interest on popular items are displayed in Figure 5.9(a),(b) respectively. Different from the previous datasets, there are a small number of Amazon-book users who do not have any rating records of popular items, the curve is relatively not smooth, and the proportion of users whose rated items are all

popular one is relatively large, about 20%.

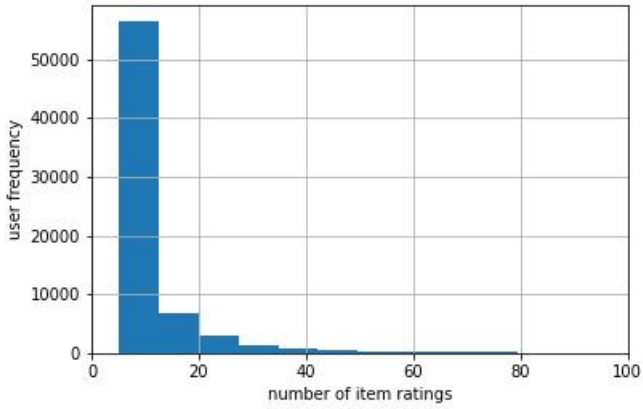
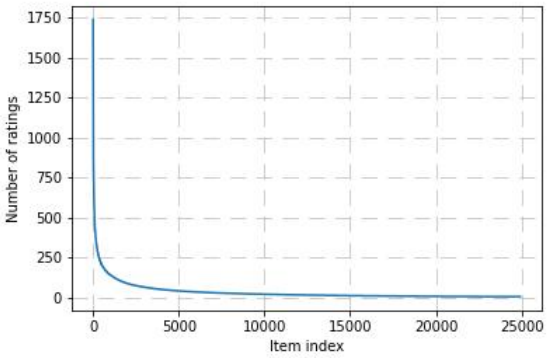
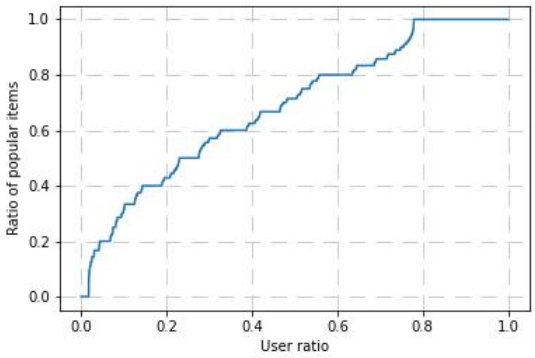


Figure 5.8: The user frequency of the number of items ratings on dataset Amazon-book



(a) Item popularity



(b) Ratio of popular items for different users

Figure 5.9: Item popularity and user interest in popular items on dataset Amazon-book

5.1.5 Dataset splitting

As introduced in Section 2.4.1, in order to evaluate model performance and guarantee the generalization ability of the model, we need to slice the complete dataset into three parts: the training set, the validation set, and the test set. the authors of Simple-HGN slice the four datasets introduced above into training set and test set, by randomly splitting 20% of ratings for each user as the test set , and the remaining 80% of ratings as the training set. On this basis, we split the training set again by exploiting the hold-out method to randomly extract 80% of the all ratings as the final training set and the rest 20% of the ratings as the validation. These two parts of dataset are used to train the model and search the best configuration of hyper-parameters, then they are integrated

and used to train and obtain the final models which will be evaluated the performance on test set. The table below shows the results of the four datasets after splitting(Since the graph knowledge-based information is not required by training our new algorithm and benchmark algorithms, therefore it is not presented here):

Dataset	#Users	#Items	#Ratings for training	#Ratings for validation	#Ratings for test
MovieLens	37,385	6,182	333,973	83,493	121,834
Yelp-2018	45,919	45,538	744,026	186,006	253,578
LastFM	23,566	42,123	1,031,202	257,801	423,635
Amazon-book	70,679	24,915	522,011	130,503	193,920

Table 5.5: The statistics of four datasets after splitting

5.2 Reproduction work

In order to compare the results of both AMF and Simple-HGN, it is necessary to check whether they are reliable by reproducing Simple-HGN results shown by the authors in the original paper. The work consists of two parts.

After reviewing the source codes given by the authors, we first conducted two checks. The first was to verify that the authors' implementation of the algorithm aligned with the paper's description. The second was to examine if the hyper-parameter configurations used during training, along with the stopping criteria, were in accordance with the paper's description. Both of these checks yielded positive results.

Following that, To ensure a fair comparison of results with other algorithms and to evaluate all algorithms under identical conditions, we proceeded with the second task of porting the model to our standard RS framework which is used in the paper "A Troubling Analysis of Reproducibility and Progress in Recommender Systems Research"[11]. This adaptation was based on the source code provided by the authors, while ensuring that the core algorithm implementations remained unaltered. In more detail, this task is mainly involved implementing three classes:

- **simpleHGN_RecommenderWrapper**: mainly involves changing the original logical framework of Simple-HGN model to a version compatible with our framework which mainly includes the model fitting function for initializing the model, the function of model training for each iteration, the function of computing user-item scores for the validation process, the function of saving current model state, and the function of loading previously saved model.

- **test_simpleHGNRecommender**: is mainly to write test functions for the main functional modules in the first class to check whether it works correctly and whether the output is consistent with the expected results.
- **run_simpleHGN_RecommenderWrapper**: to run the experiment pipeline for Simple-HGN model, begin by loading the necessary data, including the URM and pre-trained MF BPR embeddings. Then, proceed to initialize and start the hyper-parameter tuning process which involves utilizing a train-test-validation data split, in contrast to the train-test split used in the original paper. Consequently, the results obtained after the split will be used.

The results of reproduction work on four different datasets are shown in Table 5.6.

Dataset	RECALL	NDCG
MovieLens	0.4618±0.0007	0.3090±0.0007
	0.4612	**0.3078**
Yelp-2018	0.0732±0.0003	0.0466±0.0003
	0.0729	**0.0464**
LastFm	0.0917±0.0006	0.0797±0.0003
	0.0914	**0.0795**
Amazon-book	0.1587±0.0011	0.0854±0.0005
	0.1593	**0.0855**

Table 5.6: The result of Recall and NDCG for reproducing work in four datasets with cutoff at 20, the reproduction results are shown in bold, the one marked in red color means that it deviates from the result range provided by the author

From the comparison of results, we can find that except that the result of NDCG on MovieLens is slightly deviated from the result range provided by the author, other results are consistent, thus concluding that the results in the original article are reliable.

5.3 Ablation study

As the introduced in Chapter 3, Simple-HGN is composed by two components, one is the deep graph method and the other is pre-trained embeddings with MF BPR. To compare the model effectiveness when only one of them is used with the original model, there are two tasks to conduct.

One task involves training and evaluating the Simple-HGN model after completely removing the pre-trained MF BPR embeddings. In this case, only the HGNN part remains, and the focus is on learning node embeddings. Since Simple-HGN has already been integrated into the standard framework, which allows for convenient ablation experiments,

conducting this task simply requires setting a flag to control the usage of pre-trained embeddings.

The other task involves assessing the performance of the pre-trained embeddings. In this regard, the authors have generously made their pre-trained embeddings data available in their source code repository. As these embeddings are derived from the MF BPR model which is already included in the standard framework, we can easily obtain the performance results by replacing the initial embeddings of the MF BPR model with the pre-trained embeddings, eliminating the necessity of conducting the training process.

Tables 5.7-5.10 show the results of the complete ablation study on the four data sets:

Model Component	RECALL	NDCG
Complete model	0.4612	0.3078
Only HGNN remaining	0.3681	0.2285
Pre-trained embeddings	0.3992	0.2559

Table 5.7: The results of Recall and NDCG for ablation study in MovieLens dataset with cutoff at 20

Model Component	RECALL	NDCG
Complete model	0.0729	0.0464
Only HGNN remaining	0.0519	0.0323
Pre-trained embeddings	0.0627	0.0393

Table 5.8: The results of Recall and NDCG for ablation study in Yelp-2018 dataset with cutoff at 20

Model Component	RECALL	NDCG
Complete model	0.0914	0.0795
Only HGNN remaining	0.0572	0.0491
Pre-trained embeddings	0.0724	0.0617

Table 5.9: The results of Recall and NDCG for ablation study in LastFM dataset with cutoff at 20

Model Component	RECALL	NDCG
Complete model	0.1593	0.0855
Only HGNN remaining	0.1178	0.0603
Pre-trained embeddings	0.1300	0.0679

Table 5.10: The results of Recall and NDCG for ablation study in Amazon-book dataset with cutoff at 20

Based on the outcomes above, the performance of the pre-trained MF BPR model is slightly superior to that of Simple-HGN with only the HGNN component remained. However, when the two models are combined, the overall performance is significantly enhanced across various datasets.This combination results in a minimum improvement of 20% in Recall and at least a 35% improvement in NDCG.

Hence, it can be concluded that the HGNN alone is not particularly effective, the pre-trained MF BPR model performs relatively well, and the hybrid model successfully leverages the strengths of both approaches.

5.4 Performance analysis of AMF

In this section, we will first compare and analyze the performance of AMF model on four datasets with some state-of-the-art models in two aspects: one is the recommendation situation of the top-N tasks, and the other is the overall control ability of the recommended item popularity. Then, taking the MovieLens dataset as an example, the analysis of the sensitivity of AMF model to changes in the size of the three latent factors will be performed.

5.4.1 Performance analysis of top-N task

In this part, we measure the performance of AMF and other models in terms of accuracy, coverage, and diversity based on the recommendation results obtained from the final test set. The specific indicators used are listed below:

- **Classification accuracy**: Recall
- **Ranking accuracy**: NDCG
- **Coverage**: Item-space Coverage(IC)
- **Diversity**: Mean Inter-List(MIL)

The performance of AMF is then presented and compared with that of Simple-HGN model and the following state-of-art CF models, and analyzed in depth.

- **User KNN CF**: a user-based collaborative filtering algorithm introduced in Section 2.2.1.
- **Item KNN CF**: an item-based collaborative filtering algorithm introduced in Section 2.2.2.
- **SLIM BPR**: a collaborative filtering algorithm with BPR loss, which is described in Section 2.3.1.
- **PureSVD**: an algorithm of matrix factorization, using the truncated singular value decomposition technique, introduced in Section 2.3.2.
- **MF BPR**: a basic model of matrix factorization trained with BPR loss, introduced in Section 2.3.2.
- **IALS**: an algorithm of matrix factorization iterative optimizing one feature matrix, introduced in Section 2.3.2.

- **MF SVD++**: an algorithm of matrix factorization computing one latent factor at a time while considering global effects, introduced in Section 2.3.2.
- **MF ASYSVD**: an algorithm of matrix factorization decomposing URM into three matrices, the product of two of them produces an asymmetric similarity matrix, introduced in Section 2.3.2.
- $\boldsymbol{P}_{\alpha}^{\mathbf{3}}$: an graph-based algorithm introduced in Section 2.3.3.

5.4.2 Carousel analysis

In real-life scenarios, popular video and music platforms such as Netflix and Spotify often utilize multiple carousels for providing recommendations to users. Each carousel corresponds to an independent recommendation algorithm, and when evaluating the recommendation quality of these algorithms, it is important to consider the combined results of these carousels rather than focusing solely on individual recommendations[14, 18]. In this scenario the goal is not to find the single model with the best recommendation quality, but to develop a model that is good at complementing the recommendations already present and generated by the other ones. In particular, we are interested in the ability of AMF to provide recommendations that are not trivial or "easy". For this reason we evaluate it in a scenario where the user is provided first with a list of recommendations from a TopPopular(TopPop) recommender which recommends popular items based on global popularity statistic, followed by the recommendations of the algorithm we want to evaluate [14, 18] (AFM, Simple-HGN or other algorithms). The model that is better able to complement the TopPop will have the highest accuracy. It's important to note that the user interface comprises two separate lists stacked on top of each other. However, in this scenario, we do not take into account characteristics of user navigation behavior on the user interface. Consequently, we can utilize SLNDCG as the metric to directly assess the performance.

We will compare the individual models' performance with the performance of the carousel layout, which is formed by combining any one of them with TopPop. It is worth noting that the evaluation for individual models entails a single recommendation list, while our evaluation of the carousel involves two recommendation lists. As a result, direct comparison of the absolute values of NDCG is not feasible. Therefore, the analysis will primarily focus on comparing the rankings of all models.

5.4.3 Popularity analysis

Average Popularity(AP) comparison

For a recommender system, in addition to optimizing the four important indicators mentioned above, the overall popularity of the recommended items is also an important metric to consider. It is desirable for the overall popularity level of the recommended results to be as close as possible to the one of true results. To measure this aspect, we use the AP as a benchmark metric.

Moreover, its comparative analysis on all algorithms is performed by using the notation AP_p to denote the AP value in actual user profiles of test set, while AP_r represents the AP value for recommendations generated by a particular algorithm. To evaluate the performance of each recommendation algorithm, we calculate the change in AP, which reflects the degree of undesired influence that the algorithm imposes on the popularity of recommended items. A value of $\Delta AP = 0$ indicates that the recommendations align well with the users' preferences towards item popularity.

Popularity bias

A common issue with present recommendation algorithms is the popularity bias dilemma which refers to the situation in recommender systems where the probability of items being recommended is inconsistent with their actual popularity. It reflects the fact that non-popular items have no or few chances to be recommended while the system tends to recommend popular items frequently. This results in the popular items becoming even more popular, while the less popular ones remain neglected.

The goal of the top-N recommendation task is to enhance the fairness of recommendations by promoting the exposure of less popular items while ensuring a satisfactory level of accuracy. To evaluate the performance of each algorithm in achieving this objective, the comparison of the Gini Index values of item popularity for all algorithms will be conducted. This Gini Index measures the inequality in the selection of different items by users when a specific recommender system is utilized[43], and then evaluate the degree to which the algorithm alleviates the problem of popularity bias. If the fraction of total ratings received by that item i after training the model is denoted as $x(i)$. Gini Index can be given by:

$$GiniIndex = \frac{1}{n-1}\sum_{j=1}^{n}(2j - n - 1)x(i_j) \tag{77}$$

where n is the number of recommended items, i_1,$\cdots i_n$ is the list of items ordered according to increasing $x(i)$. The index is 0 when all items are chosen equally often, and 1 when a single item is always chosen. Therefore, a lower value of this metric is preferred.

Long tail exploration ability

In general, when the popularity bias of a recommender system decreases, the ability to explore long-tail items often improves, but the former factor is not a sufficient condition for the latter factor. Even a recommender system with low popularity bias may not be able to explore long-tail items well. Therefore, to better discover long-tail items, recommender systems need to adopt appropriate algorithms and techniques to improve the ability to explore long-tail items while reducing popularity bias. In Section 5.1.1, we have defined popular items as those receiving ratings in the top 20% of all items, while the remaining 80% are categorized as non-popular or long-tail items. Therefore, we focus on the recommendation of non-popular items by each algorithm, and the evaluation will be performed from the following two perspectives:

- **Compare the proportion of popular items and long-tail items in all items**: by counting the unique values of recommended popular and non-popular items separately and adding them together to obtain the total number of recommended items, we will calculate the proportion of each category, and then compare the distribution of these two categories among different algorithms.
- **Compare the recommendation frequency between popular and long-tail items**: by directly counting the total number of recommended items for each category in the recommendation list, we will calculate the proportion of each category's total recommendations, and then compare the distribution of these two categories among different algorithms.

5.4.4 Model sensitivity analysis

As introduced in Section 4.1, AMF stands out because it incorporates three different MF techniques, with each MF decomposing the URM into a pair of embeddings. The performance of MF models is often influenced by the size of these embeddings, which is directly determined by the size of the latent factors in the MF. Therefore, for the sensitivity analysis of the models in this section, we will take the MovieLens dataset as an example and primarily focus on the impact of different size of three latent factors in AMF on model performance. As a comparison, we selected four algorithms from the MF family that also rely on latent factors decomposition of the URM. These algorithms are MF ASYSVD, MF BPR, MF SVD++, and IALS.

Chapter 6: Results

In this chapter, we will first present the training results of AMF, Simple-HGN, and state-of-the-art algorithms on the test sets of four datasets from Sections 6.1 to 6.4. Then, for each dataset, we will conduct performance analysis for individual models, carousel analysis, and popularity analysis. In Section 6.5, we will summarize and provide an overview of all the aforementioned analyses. Finally, in Section 6.6, we will discuss the sensitivity of the performance of the AMF model to three different dimensions of latent factors.

6.1 MovieLens

6.1.1 Performance analysis

Algorithm	Recall	NDCG	IC	MIL
Item KNN CF	**0.4377**	**0.3020**	**0.3661**	0.7489
User KNN CF	**0.4700**	**0.3301**	**0.1839**	**0.8429**
SLIM BPR	**0.4320**	0.2475	0.1568	0.8077
PureSVD	0.3527	0.2351	0.0113	0.6550
MF BPR	0.4092	0.2564	0.1393	0.8070
IALS	**0.4544**	**0.3023**	0.0759	**0.8937**
MF SVD++	**0.4224**	**0.2715**	0.0829	**0.8673**
MF ASYSVD	**0.4534**	**0.3017**	0.0834	0.7782
P^3_α	**0.4252**	**0.2891**	**0.2643**	0.7164
Simple-HGN	**0.4612**	**0.3078**	**0.2131**	**0.8299**
AMF	0.4125	0.2657	0.1664	0.8218

Table 6.1: The performance evaluation of algorithms on dataset MovieLens with cutoff at 20. The red font highlights the result of AMF, and the bold font indicates the better performance than AMF.

The performance evaluation results with cutoff at 20 are shown in the table above. From the overall point of view, the optimal results appear in the state of art algorithms. Among them, the KNN algorithms is particularly noteworthy. User KNN CF has achieved the highest accuracy in classification and ranking, while Item KNN CF has the most extensive

item coverage since the recommendations are sorted based on item similarity. However, it is worth noting that according to the analysis of the dataset in Section 5.1.1, we know that MovieLens users have a strong willingness to interact with popular items, so in this case, excessive coverage will reduce the accuracy of model recommendation. Therefore, even though the IALS algorithm is lower than that of Item KNN CF in coverage, this is not a negative result and it outperforms other algorithms in terms of diversity of recommendations for different users, meanwhile it still retains excellent performance in terms of accuracy, ranking second only to User KNN CF with 3.32% lower in Recall and 8.42% lower in NDCG.

Next, we focus on the performance of AMF. First, compared with Simple-HGN model, they have obvious differences in both accuracy and item coverage, AMF is with a decrease of 10.56% in Recall, 13.68% in NDCG, and 21.91% in IC. However, they are similar in terms of the diversity of recommended item lists for different users.

To objectively evaluate the performance of AMF, which is based on the MF BPR, adding an attention mechanism based on graph algorithm, the comparison between the MF BPR algorithm is performed. In terms of accuracy, AMF under the help of graph-based attention mechanism performs slightly better than the traditional MF BPR, with an increase of 1.00% in Recall and 3.63% in NDCG. The attention mechanism also plays a role in improving the diversity, resulting in AMF's better performance. The MIF of AMF is higher than MF BPR by 2.39%.

6.1.2 Carousel analysis

From Table 6.2, it is easy to see that TopPop has the lowest recommendation quality among all models in both individual and carousel evaluations. This is not surprising because in the carousel evaluation, TopPop serves as the first carousel, while any other model serves as the second carousel. Their recommendation results are concatenated into a single list, and even if the recommendations of the second carousel are identical to TopPop's, the final NDCG value will still be the same as TopPop's. Moreover, based on the individual evaluation performance, all models perform better than TopPop. Therefore, in the carousel evaluation, the performance of each algorithm remains higher than TopPop. Additionally, based on the previous explanation, the more overlap exists between the recommendations of the models in the second carousel and TopPop's recommendations in the first carousel, the worse the performance will be in the carousel evaluation.

According to the results of the carousel evaluation, the top three performing algorithms are SLIM BPR, IALS, and Item KNN CF. On the other hand, in the individual evaluation,

	Individual		Carousel-TopPop	
	NDCG	**Rank**	**SLNDCG**	**ΔRank**
TopPop	0.1630	12	0.1630	-
Item KNN CF	0.3020	4	0.2133	1
User KNN CF	0.3301	1	0.2096	-5
SLIM BPR	0.2475	10	0.2233	9
PureSVD	0.2351	11	0.1907	0
MF BPR	0.2564	9	0.1978	-1
IALS	0.3023	3	0.2192	1
MF SVD++	0.2715	7	0.2107	2
MF ASYSVD	0.3017	5	0.2125	1
P_α^3	0.2891	6	0.2052	-1
Simple-HGN	0.3078	2	0.2019	-7
AMF	0.2657	8	0.2033	0

Table 6.2: The comparison and ranking of NDCG with cutoff at 20 for all models on dataset MoiveLens, excluding the rank of TopPop as it is already included in the carousel. ΔRank represents the difference between the rank when evaluated individually and the rank when evaluated in the carousel layout. A negative ΔRank indicates that the model is in a lower ranking position.

the top three algorithms are User KNN CF, Simple-HGN, and IALS. When comparing the rankings of both evaluations, the majority algorithms showed varying degrees of changes. Among those with minor changes of 1 to 2 ranks are Item KNN CF, IALS, and MF algorithms other than PureSVD. On the other hand, three algorithms showed significant changes: SLIM BPR had the largest improvement, rising 9 ranks to claim the first position in the carousel evaluation; User KNN CF experienced a decline of 5 ranks, whereas Simple-HGN demonstrated the most significant drop with 7 ranks. This suggests that Simple-HGN's superior performance in offline evaluation due to the popularity bias, resulting in a higher number of "trivial" recommendations that are already present in the TopPop recommendation list. In addition, PureSVD and AMF maintained their ranks, indicating that their recommendations are significantly less trivial.

Based on the comparison of the carousel evaluation performance of MF BPR, Simple-HGN, and AMF, it differs from the conclusion of the individual evaluation. In the individual evaluation, Simple-HGN performs the best, followed by AMF. However, in the carousel evaluation, AMF outperforms Simple-HGN and becomes the best performer among the three.

6.1.3 Popularity analysis

AP comparison

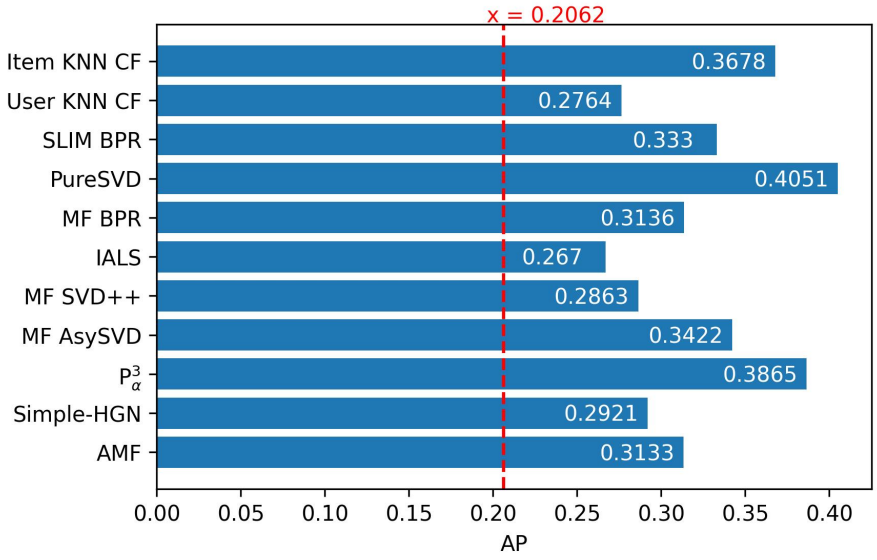


Figure 6.1: Comparison between the recommended results of each algorithm and the actual user profiles in AP on dataset MovieLens, the AP value of user profiles is marked by the red dotted line and the right part of it shows the ΔAP.

From Figure 6.1, We can note that the average item popularity of all algorithms exceeds that of the actual user profiles. Among them, IALS exhibits the smallest difference, with a 29.49% increase above the actual level. This indicates that all recommendation algorithms reinforce the popularity bias by recommending more popular items and exposing users to a greater number of them. PureSVD has the largest difference, greatly exceeding the actual level by 96.46%. This is because PureSVD has a low number of latent factors, only 5. This means that the model tends to rely on global patterns for recommendations and overlook individual user preferences. When the number of latent factors is reduced to 1, the model becomes equivalent to a top popular recommender. In comparison, Simple-HGN and AMF both have AP values in the middle range among all algorithms. However, the AP level of Simple-HGN model is closer to the actual level. Compared to MF BPR with AMF, their AP values are almost identical, which means that they have the same ability to manage the popularity of items.

Popularity bias

Based on the Gini index of item popularity of all algorithms shown in Figure 6.2, the value of PureSVD is the lowest, indicating that these items are recommended with the most uniform frequency distribution. On the contrary, P^3_α has the highest Gini index value,

closely followed by Item KNN CF, with a difference of less than 0.01, indicating the most severe uneven distribution of item recommendation. Looking at their AP values, all three algorithms have similar levels and are inclined to recommend high-popularity items. However, overall, it can be concluded that P_α^3 and Item KNN CF are more likely to further promote popular items to be even more popular, as their gini index values are much higher than that of PureSVD, exacerbating the problem of popularity bias. In comparison between these two algorithms, the popularity bias problem of the former(P_α^3) is more severe, as both its AP and Gini index are slightly worse than those of the latter.

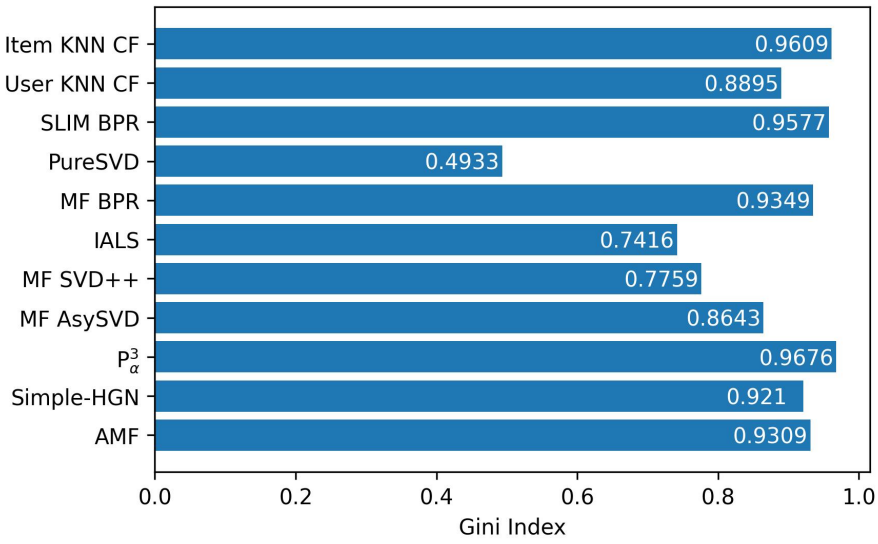


Figure 6.2: Comparison of the Gini index of the item popularity in the recommendation results of each algorithm on dataset MovieLens

In comparison, AMF, Simple-HGN, and MF BPR have similar Gini index levels around 0.93, which belongs to the situation where the frequency distribution of recommended items is relatively uneven. AMF's performance of mitigating the popularity bias is intermediate between the other two. Simple-HGN appears to be the best at mitigating this issue, with the best AP and Gini index among the three. While, MF BPR is the worst, having the worst AP and Gini index, indicating a relatively highest recommendation frequency for highly popular items.

Long tail exploration ability

Figure 6.3 shows the proportion of popular and non-popular items in all items in all algorithms recommendation results. Item KNN CF has the highest number of non-popular items recommended, accounting for more than 75% of the total number, which means it gives the highest fair chance of exposure to non-popular items. Followed by SLIM BPR

and P_{α}^{3}, accounting for 70%. It is important to highlight that there are four algorithms that recommend mostly or only popular items, namely PureSVD, IALS, MF SVD++, and MF ASYSVD.

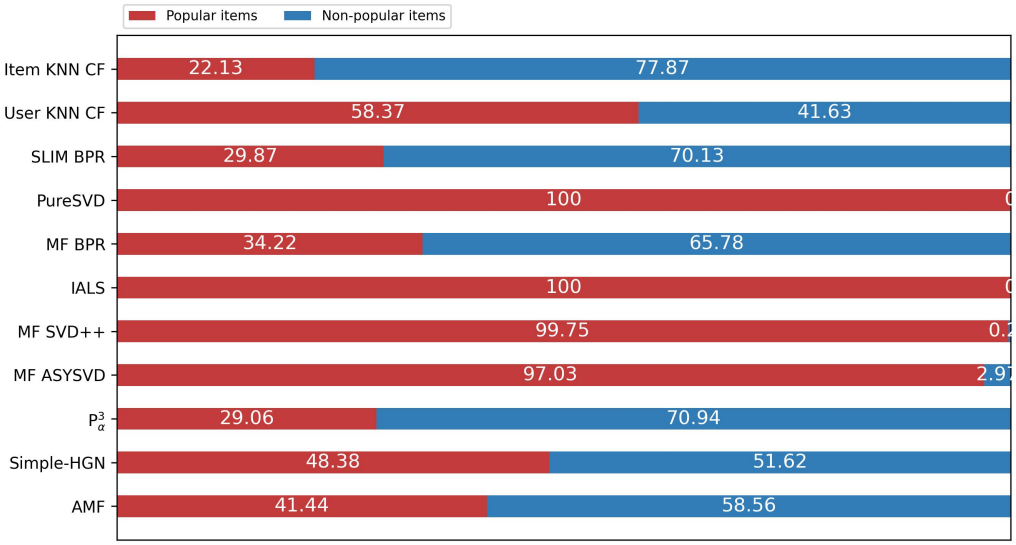


Figure 6.3: Comparison of the proportion distribution of popular and non-popular items in each algorithm's recommendation results on dataset MovieLens

In regards to AMF, when compared to Simple-HGN, the proportion of non-popular items recommended by both algorithms is similar and falls in the middle range among all algorithms. However, the proportion of non-popular items recommended by AMF is slightly higher than that of Simple-HGN by 6.94%. On the other hand, when compared to MF BPR, AMF has a lower proportion of non-popular items, which is 7.19% lower than MF BPR.

From the recommended frequency distribution of this two types of items, shown in Figure 6.4, there is an obvious phenomenon that the recommended results of all algorithms are almost all popular items, and the proportion is close to 1. Item KNN CF is in first place with a narrow margin: 2.56% of exposure for non-popular items. Moreover, the frequency of recommending non-popular items between Simple-HGN, MF BPR, and AMF is almost the same.

Therefore, in terms of the ability to explore long-tail items, we can conclude that from the comparison of all algorithms, Item KNN CF not only recommends the largest number of non-popular items, but also the highest frequency. Thus, it has the best performance.

Focusing on Simple-HGN and AMF, the latter gives more exposure for non-popular items when both of them recommend non-popular items at the same frequency, therefore, it

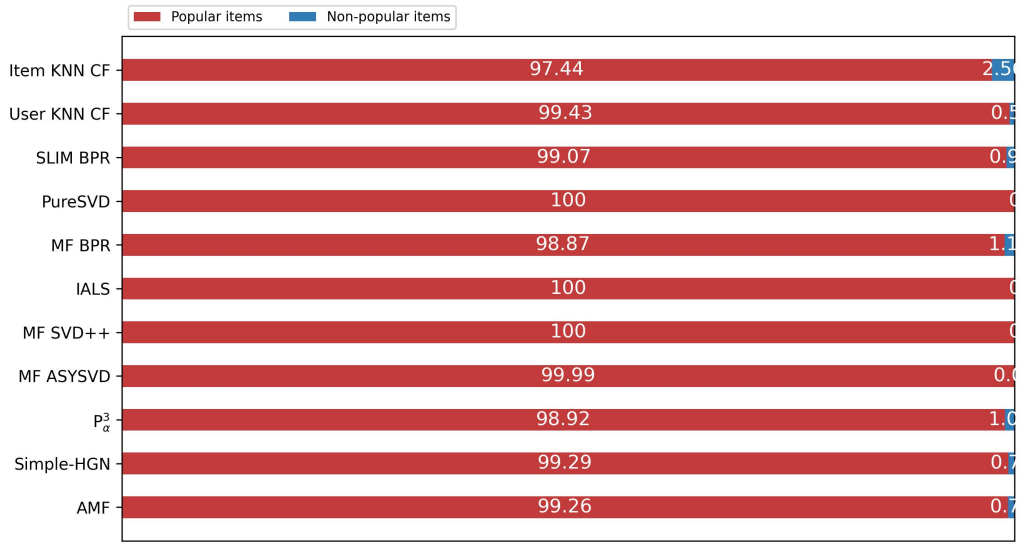


Figure 6.4: Comparison of the frequency distribution of popular and non-popular items in each algorithm's recommendation results on dataset MovieLens

can be concluded that AMF has the better performance. When it turn to the comparison between MF BPR and AMF, the former has the stronger ability to mining the long tail items since it has a slightly better performance in terms of the exposure for non-popular items when the recommendation frequencies of them among the two algorithms are the same.

6.2 Yelp-2018

6.2.1 Performance analysis

Algorithm	Recall	NDCG	IC	MIL
Item KNN CF	**0.0747**	**0.0489**	**0.4001**	**0.9748**
User KNN CF	**0.0715**	**0.0477**	**0.2802**	**0.9724**
SLIM BPR	**0.0708**	**0.0466**	**0.3642**	**0.9561**
PureSVD	0.0552	0.0365	0.0643	**0.9725**
MF BPR	0.0497	0.0316	**0.3394**	**0.9736**
IALS	**0.0772**	**0.0503**	**0.2047**	**0.9905**
MF SVD++	**0.0625**	**0.0397**	**0.2523**	**0.9903**
MF ASYSVD	**0.0636**	**0.0409**	**0.1951**	**0.9871**
P^3_α	**0.0711**	**0.0467**	**0.2969**	**0.9521**
Simple-HGN	**0.0729**	**0.0464**	**0.3756**	**0.9884**
AMF	0.0559	0.0365	0.1254	0.9520

Table 6.3: The performance evaluation of algorithms on dataset Yelp-2018 with cutoff at 20. The red font highlights the result of AMF, and the bold font indicates the better performance than AMF.

Table 6.3 shows the performance of all algorithms for the dataset Yelp-2018. the IALS algorithm performs the best in terms of accuracy, variety of items recommended. Item KNN CF leads the other algorithms in terms of the item coverage, with around 40% of items being recommended. It is worth noting that compared to the fact that MovieLens users have a high reliance on popular items (about 85% of users have more than 80% of their overall item rating number for popular items), users in Yelp 2018 are at a lower level of reliance on popular items, with only about 20% of users having more than 80% of their overall item rating number for popular items, which means that users are more inclusive of non-popular items, thus it makes the Item KNN CF algorithm not only ensure high item coverage but also perform well in terms of accuracy, which is only 3.23% lower than IALS in Recall and 2.78% lower in NDCG. In contrast, PureSVD and AMF, which have a low level of item coverage, also perform poorly in terms of accuracy, with Recall around 0.055, 27.60% lower than that of IALS, and with NDCG around 0.0365, 27.44% lower than that of IALS.

Compared to AMF, Simple-HGN outperforms it on all four metrics. On the other hand, when compared to MF BPR, AMF sacrifices some item coverage and diversity, but guarantees accurate recommendations. As a result, it improves Recall by 12.47% and NDCG by 15.51%.

6.2.2 Carousel analysis

	Individual		Carousel-TopPop	
	NDCG	**Rank**	**SLNDCG**	**ΔRank**
TopPop	0.0110	12	0.0109	-
Item KNN CF	0.0489	2	0.0236	-5
User KNN CF	0.0477	3	0.0229	-6
SLIM BPR	0.0466	5	0.0240	0
PureSVD	0.0365	10	0.0259	8
MF BPR	0.0316	11	0.0194	0
IALS	0.0503	1	0.0273	0
MF SVD++	0.0397	8	0.0242	5
MF ASYSVD	0.0409	7	0.0239	1
P^3_α	0.0467	4	0.0242	0
Simple-HGN	0.0464	6	0.0236	-2
AMF	0.0365	9	0.0227	-1

Table 6.4: The comparison and ranking of NDCG with cutoff at 20 for all models on dataset Yelp-2018, excluding the rank of TopPop as it is already included in the carousel. ΔRank represents the difference between the rank when evaluated individually and the rank when evaluated in the carousel layout. A negative ΔRank indicates that the model is in a lower ranking position.

According to the results in Table 6.4, the top three performing algorithms in the carousel evaluation are IALS, PureSVD, and MF SVD++, with P^3_α also ranked third. In the individual evaluation, the top three algorithms are IALS, ITEM KNN CF, and USER KNN CF. When comparing the rankings between the two evaluations, the algorithms with minor changes of 1 to 2 ranks are MF ASYSVD, Simple-HGN, and AMF. On the other hand, the two KNN CF algorithms, PureSVD, and MF SVD++, showed significant changes: PureSVD had the largest improvement, rising 8 ranks to claim the second position in the carousel evaluation, while User KNN CF experienced the largest decline, dropping 6 ranks, and Item KNN CF dropped 5 ranks. Furthermore, in this dataset, there is a reduced presence of popularity bias, and we can see Simple-HGN has learned to provide less trivial recommendations because its quality does not drop a lot under a carousel evaluation.

Comparing the performance of MF BPR, Simple-HGN, and AMF in the carousel evaluation, the conclusion is consistent with the individual evaluation. Simple-HGN exhibited the best performance among the three algorithms, while AMF performed between the other two.

6.2.3 Popularity analysis

AP comparison

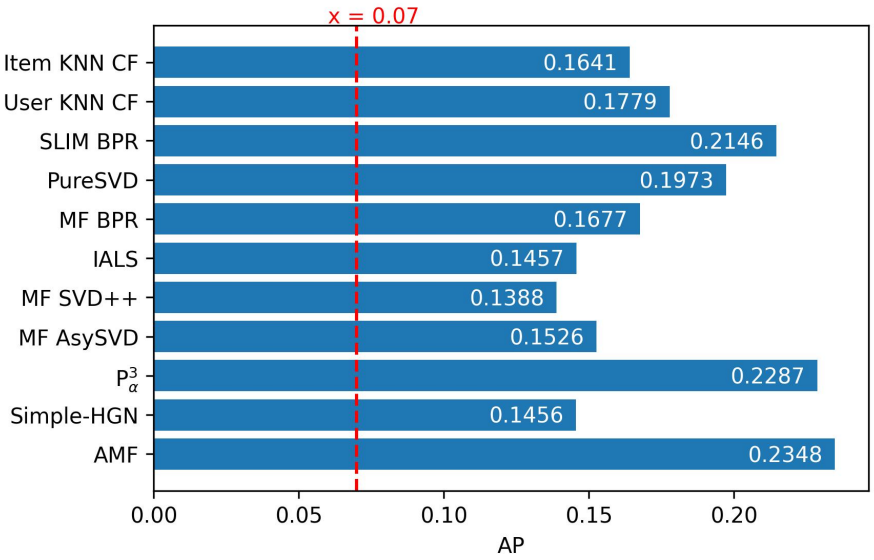


Figure 6.5: The comparison between the recommended results of each algorithm and the actual user profiles in AP on dataset Yelp-2018, the AP value of user profiles is marked by the red dotted line and the right part of it shows the ΔAP.

In terms of the AP level of the recommended results of each algorithm, which is shown in Figure 6.5, all algorithms also significantly exceed the actual level. However, the traditional MF family algorithms have a lower degree of deviation compared to other algorithms. Among them, MF SVD++ performs the best with an average popularity of 0.1388, which is about 1 times higher than the actual level. Focusing on the deviation degrees of AMF, Simple-HGN and MF BPR, the AP of the recommended items by AMF is the worst fit with the actual level, which is about 2.5 times higher. The performance of Simple-HGN in this regard is the best of the three, second only to the performance of MF SVD++.

Popularity bias

Based on the results in the figure above, the best and worst Gini Index results are consistent with those of MovieLens. PureSVD has the most uniform item recommendation frequency among all algorithms. Compared with other algorithms having the similar and higher AP values, such as KNN CFs, SLIM BPR, and P^3_α, it has the best performance on mitigating the popularity bias problem. While the frequency distribution of AMF is the most uneven, since it also has the worst AP level, which means it tends to recommend

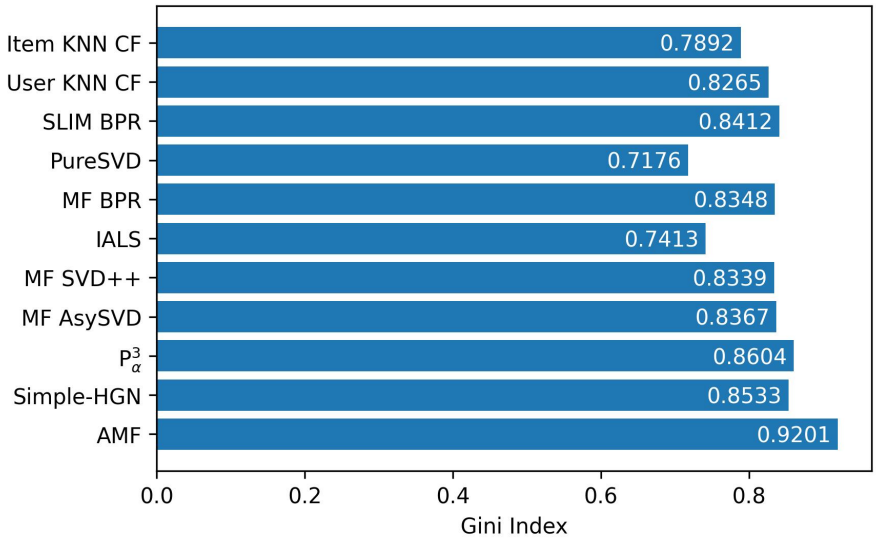


Figure 6.6: Comparison of the Gini index of the item popularity in the recommendation results of each algorithm on dataset Yelp-2018

high popularity items multiple times, exacerbating the situation where popular items become even more popular. When comparing Simple-HGN with MF BPR, both the Gini index values and the AP values of the two are very close, differing only around 0.02, thus their ability to mitigate the popularity bias problem is similar.

Long tail exploration ability

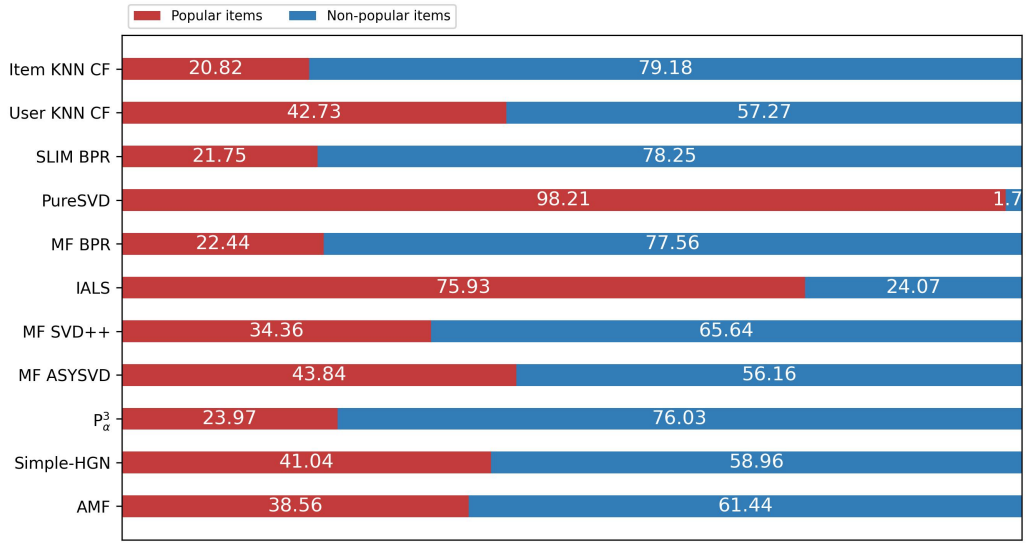


Figure 6.7: The comparison of the proportion distribution of popular and non-popular items in each algorithm's recommendation results on dataset Yelp-2018

In Figure 6.7, it is displayed how the various algorithms allocate their recommendations between popular and non-popular items when using 20 as the cutoff to retrieve recommendations. There are four algorithms that excel in providing the exposure for non-popular items, accounting for at least 75% of all recommended items. When arranged in descending order of proportion, they are Item KNN CF, SLIM BPR, MF BPR, and P_{α}^{3}. In contrast, PureSVD performs poorly in this regard, with only 1.7% of its recommended items being non-popular, followed by IALS, with about a quarter of its recommendations being non-popular.

Regarding AMF, it only shows a slight 2.5% improvement on the exposure for non-popular items respect to Simple-HGN. In contrast, compared to the MF BPR, AMF performs significantly worse, with a 16.12% lower proportion of non-popular items recommended.

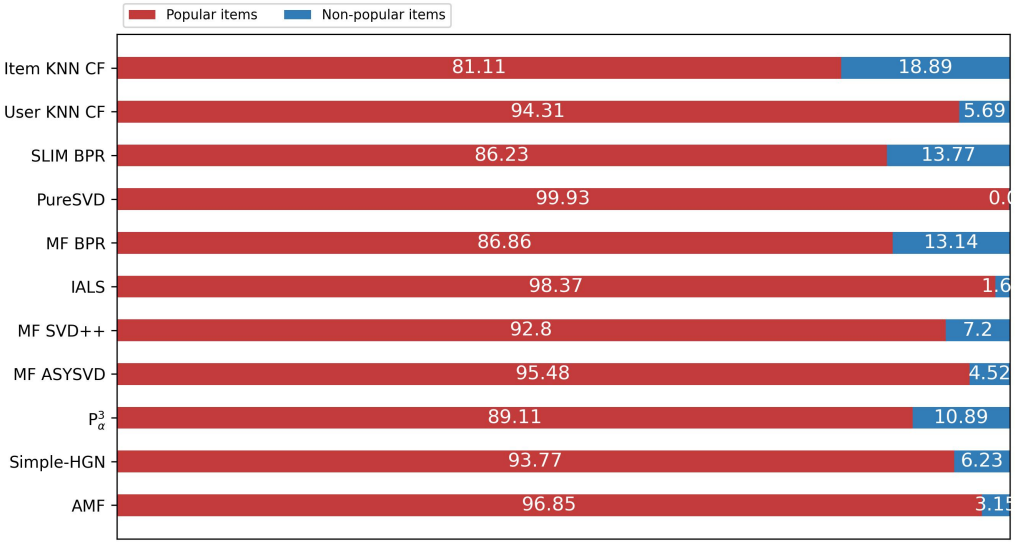


Figure 6.8: The comparison of the frequency distribution of popular and non-popular items in each algorithm's recommendation results on dataset Yelp-2018

Looking at the frequency distribution of recommended popular and non-popular items by each algorithm, the conclusions on the best and worst performance are consistent with Figure 6.7. Among them, Item KNN CF ranks first with an absolute advantage in the proportion of recommendations frequency for non-popular items, accounting for 18.89%. The worst performer is PureSVD, with very few recommendations frequency for non-popular items, which can be ignored.

Combining the results of two figures, we can note that as Item KNN CF not only performs outstandingly in ensuring the fairness of exposure opportunities for non-popular items, but also maintains a high level of exposure frequency. Therefore, it performs the best

among all algorithms in excavating long tail items. As for AMF, compared with Simple-HGN, AMF recommends more non-popular items than Simple-HGN. However, from the perspective of the recommended frequency of non-popular items, Simple-HGN obtains the better result. Therefore, they have different advantages. When it comes to MF BPR, it has a significant advantage over AMF on the number of exposure opportunities and frequency of non-popular items, thereby MF BPR has the better performance in this regard.

6.3 LastFM

6.3.1 Performance analysis

Algorithm	Recall	NDCG	IC	MIL
Item KNN CF	**0.1164**	**0.1090**	**0.9246**	**0.9926**
User KNN CF	**0.1047**	**0.0967**	**0.8842**	**0.9990**
SLIM BPR	**0.1095**	**0.0992**	**0.6933**	**0.9947**
PureSVD	**0.0781**	**0.0735**	0.2932	**0.9926**
MF BPR	**0.0790**	**0.0713**	**0.5758**	**0.9862**
IALS	**0.0999**	**0.0842**	**0.6083**	**0.9943**
MF SVD++	**0.0960**	**0.0816**	**0.5760**	**0.9930**
MF ASYSVD	**0.0741**	**0.0623**	**0.5388**	**0.9933**
P^3_α	**0.1193**	**0.1080**	**0.7638**	**0.9970**
Simple-HGN	**0.0914**	**0.0795**	**0.6046**	**0.9923**
AMF	0.0664	0.0567	0.4046	0.9599

Table 6.5: The performance evaluation of algorithms on dataset LastFM with cutoff at 20. The red font highlights the result of AMF, and the bold font indicates the better performance than AMF.

The results obtained from the top-N performance analysis are shown in Table 6.5. P^3_α and Item KNN CF performed impressively and similarly in accuracy, with the best results achieved in Recall and NDCG respectively by a slight advantage over each other. The two KNNCF algorithms performed outstandingly in indicators other than accuracy. They not only covered a vast majority of recommended items, accounting for around 90%, but also ensured high diversity among the recommended lists for users.

Focusing on AMF, its performance is low in all aspects. Compared with Simple-HGN, it has a significant gap in accuracy and item coverage. It is 27.35%, 28.68%, and 33.08% lower than Simple-HGN in Recall, NDCG, and IC, respectively. Compared with MF BPR, it also lags behind in these three indicators, but the gap is smaller, at 15.95%, 20.48%, and 29.73%, respectively.

6.3.2 Carousel analysis

According to the analysis of the results in Table 6.6, the top three performing algorithms in the carousel evaluation are Item KNN CF, P^3_α, and SLIM BPR, ranked from highest to lowest. In the individual evaluation, these three algorithms are still the top performers, but their rankings have changed to P^3_α, SLIM BPR, and Item KNN CF. Comparing the two evaluation results, it can be observed that the rankings of MF SVD++ and AMF

have not changed. SLIM BPR, P^3_α, and all MF family algorithms except MF SVD++ have all moved up one position in the rankings, while Simple-HGN has been most affected with a decrease of three positions. The two KNN CF algorithms also experienced slight decreases in rankings, dropping by 1 or 2 positions.

In terms of the carousel evaluation performance of MF BPR, Simple-HGN, and AMF, it is inconsistent with the findings from the individual evaluation. In the carousel evaluation, MF BPR outperforms Simple-HGN and is the best-performing algorithm among the three. However, the performance of AMF aligns with the conclusion from the individual evaluation, as it is the poorest performer among the three algorithms.

	Individual		**Carousel-TopPop**	
	NDCG	**Rank**	**SLNDCG**	**ΔRank**
TopPop	0.0106	12	0.0095	-
ItemKNNCF	0.1090	1	0.0449	-2
UserKNNCF	0.0967	4	0.0428	-1
SLIM BPR	0.0992	3	0.0489	1
PureSVD	0.0735	8	0.0382	1
MF BPR	0.0713	9	0.0355	1
IALS	0.0842	5	0.0448	1
MF SVD++	0.0816	6	0.0427	0
MF ASYSVD	0.0623	10	0.0350	1
P^3_α	0.1080	2	0.0539	1
Simple-HGN	0.0795	7	0.0325	-3
AMF	0.0567	11	0.0319	0

Table 6.6: The comparison and ranking of NDCG with cutoff at 20 for all models on dataset LastFM, excluding the rank of TopPop as it is already included in the carousel. ΔRank represents the difference between the rank when evaluated individually and the rank when evaluated in the carousel layout. A negative ΔRank indicates that the model is in a lower ranking position.

6.3.3 Popularity analysis

AP comparison

According to the AP results shown in Figure 6.9, there is an interesting fact that unlike the results of the first two datasets, the Item KNN CF's AP of recommendation results is lower than the actual AP level, indicating an underestimation of the user's preference for item popularity, while the recommendation result of P^3_α perfect match the user's real preference, with an AP value that is very close to the real AP value, differing only by 0.005. However, it is necessary to notice that the AP value of the recommended results

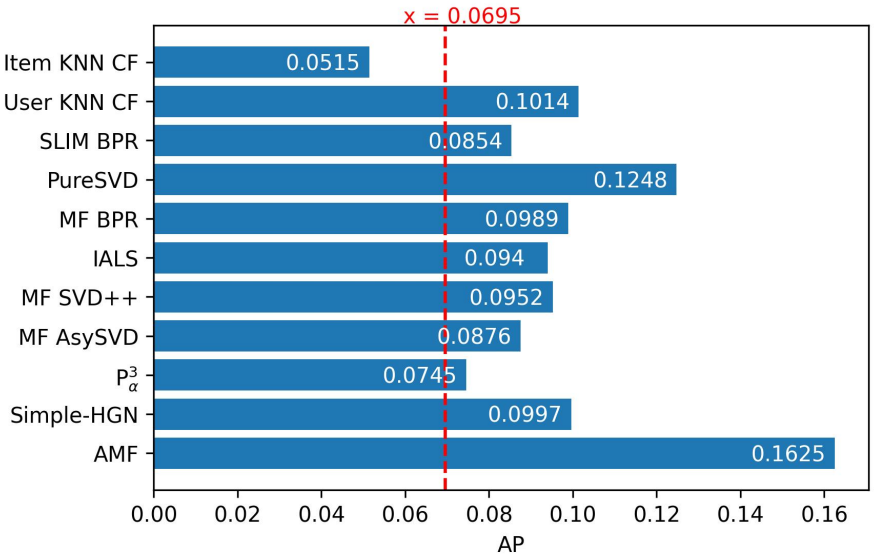


Figure 6.9: The comparison between the recommended results of each algorithm and the actual user profiles in AP on dataset LastFM, the AP value of user profiles is marked by the red dotted line and the right part of it shows the ΔAP.

of AMF is the highest respect to others, significantly exceeds the real value by 1.34 times and greatly overestimate the user's real preference. Simple-HGN and MF BPR have the similar performance in this regard with two very close values, around 0.099.

Popularity bias

Item KNN CF and AMF achieved the minimum and maximum values of the Gini index, respectively. It is important to note that the AP value of Item KNN CF is also the lowest among all algorithms, thus we can conclude that it is most inclined to recommend items with lower popularity and performs best at mitigating popularity bias issue. For AMF, its values in both AP and Gini index are the worst, therefore it has the worst performance in alleviating this problem.

In the comparison between Simple-HGN and MF BPR, their AP levels are almost identical. However, the latter has a lower Gini index, indicating that the recommended items are more uniformly distributed in terms of frequency, with a lower likelihood of frequently recommending highly popular items. This results in a lower likelihood of exacerbating popularity bias.

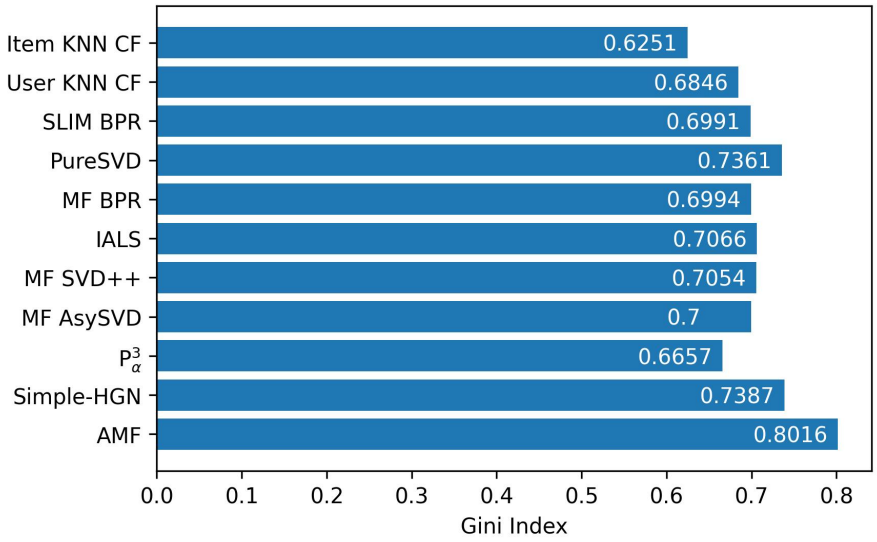


Figure 6.10: Comparison of the Gini index of the item popularity in the recommendation results of each algorithm on dataset LastFM

Long tail exploration ability

In Figure 6.11, the ratio of recommended quantities of popular and non-popular items in all algorithms are presented. With the exception of PureSVD, which covers a relatively low proportion of non-popular items, the other algorithms have a concentration of around 70%, with Item KNN CF performing the best in this regard. However, when AMF and Simple-HGN are both taken into consideration, the exposure for non-popular items are not exceptional compared to other algorithms. In comparison, AMF recommends a slightly lower proportion of non-popular items than Simple-HGN, with a difference of 3.84%. This gap widens when compared to MF BPR, with AMF recommending a proportion that is 8.40% lower than MF BPR.

Looking at the distribution of recommended frequencies for popular and non-popular items in Figure 6.12, the strengths and weaknesses of each algorithm are consistent with Figure 6.11.

Specifically, Item KNN CF has a significant advantage over other algorithms in the distribution of recommended frequencies for non-popular items, already reaching the 40% level, while others are below 30%. Another noteworthy algorithm is P_α^3, it has the highest recommended frequency except Item KNN CF when it does not have a clear advantage over other algorithms in the distribution of recommended quantities for non-popular items.

To be more specific, Item KNN CF outperforms other algorithms in terms of the frequency distribution of recommended non-popular items, with a proportion reaching as high as

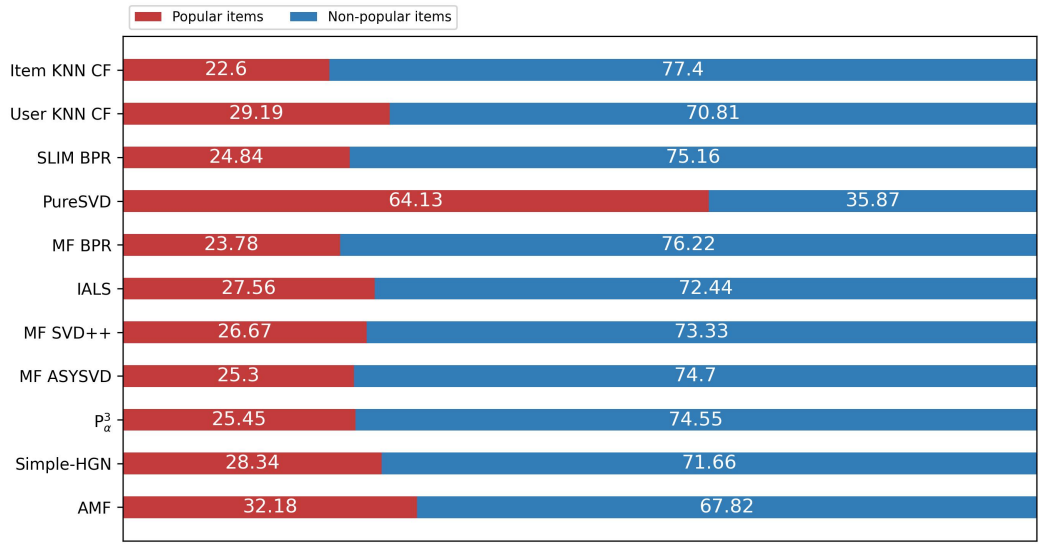


Figure 6.11: The comparison of the proportion distribution of popular and non-popular items in each algorithm's recommendation results on dataset LastFM

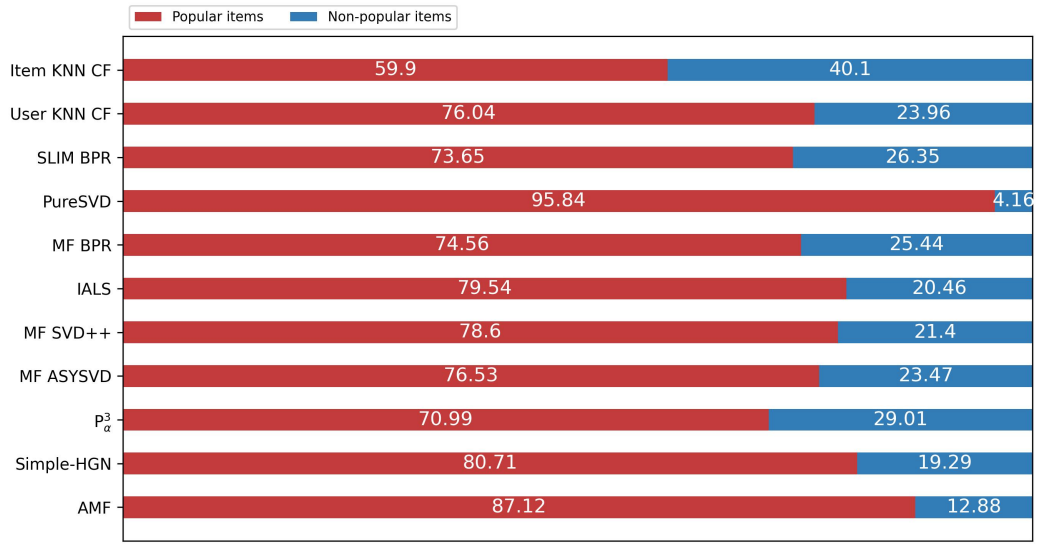


Figure 6.12: The comparison of the frequency distribution of popular and non-popular items in each algorithm's recommendation results on dataset LastFM

40%, while the others are below 30%. Another noteworthy algorithm is P^3_α, which has the second highest frequency of recommendations after Item KNN CF, despite lacking a distinct advantage over other algorithms in terms of recommending non-popular items.

In summary, we draw the conclusion in the aspect of Long tail exploration ability based on the results of the above two figures: Item KNN CF and P^3_α perform well among

all algorithms. Both have provided a good balance in terms of the fairness of exposure opportunities and frequency for non-popular items. However, Item KNN CF has a greater advantage. Let's take a look at AMF again. Whether compared with Simple-HGN or MF BPR, it does not perform well in this regard: it recommends the least number of popular items with the lowest frequency. In contrast, the MF BPR has the biggest ability.

6.4 Amazon-book

6.4.1 Performance analysis

Algorithm	Recall	NDCG	IC	MIL
Item KNN CF	**0.1686**	**0.0977**	**0.8239**	**0.9811**
User KNN CF	**0.1662**	**0.0943**	**0.6696**	**0.9778**
SLIM BPR	**0.1688**	**0.0973**	**0.7782**	0.9660
PureSVD	0.1052	0.0619	0.1948	**0.9809**
MF BPR	0.1215	0.0668	**0.6454**	**0.9830**
IALS	**0.1632**	**0.0869**	**0.5402**	**0.9889**
MF SVD++	**0.1409**	**0.0766**	**0.5781**	**0.9887**
MF ASYSVD	**0.1406**	**0.0744**	0.4352	**0.9803**
P_α^3	**0.1712**	**0.0979**	**0.8493**	**0.9847**
Simple-HGN	**0.1593**	**0.0855**	**0.7572**	**0.9861**
AMF	0.1348	0.0691	0.4666	0.9697

Table 6.7: The performance evaluation of algorithms on dataset Amazon-book with cutoff at 20. The red font highlights the result of AMF, and the bold font indicates the better performance than AMF.

It can be seen in Table 6.7 that shows that the highest accuracy for both classification and ranking is achieved by P_α^3, followed by Item KNN CF and SLIM BPR. The performance of the latter two is almost the same, close to 0.1690 in Recall, and around 0.0975 in NDCG. In terms of non-accuracy metrics, P_α^3 also has the highest item coverage in its recommendation results, while IALS achieves the best result in recommendation list diversity.

After comparing AMF with Simple-HGN, it is evident that AMF performs poorly in both accuracy and non-accuracy metrics. Specifically, it falls short of Simple-HGN by 15.38% in Recall and 19.18% in NDCG, with an even more significant difference in terms of IC, where AMF is 38.38% lower. In terms of MIL, AMF is 1.67% lower than Simple-HGN. When compared to the performance of MF BPR, AMF shows an improvement in accuracy, with a 10.95% higher Recall and a 3.44% higher NDCG. However, this advantage is not reflected in non-accuracy, as AMF shows a 27.70% lower IC and a 1.35% lower MIL.

6.4.2 Carousel analysis

Based on the results in Table 6.8, the top three algorithms with the best performance remained consistent between the carousel evaluation and the individual evaluation. These algorithms are P_α^3, Item KNN CF, and SLIM BPR. When comparing the rankings be-

	Individual		Carousel-TopPop	
	NDCG	**Rank**	**SLNDCG**	**ΔRank**
TopPop	0.0123	12	0.0122	-
Item KNN CF	0.0977	2	0.0457	0
User KNN CF	0.0943	4	0.0422	0
SLIM BPR	0.0973	3	0.0456	0
PureSVD	0.0619	11	0.0353	2
MF BPR	0.0668	10	0.0333	-1
IALS	0.0869	5	0.0420	0
MF SVD++	0.0766	7	0.0379	-1
MF ASYSVD	0.0744	8	0.0393	1
P^3_α	0.0979	1	0.0471	0
Simple-HGN	0.0855	6	0.0400	0
AMF	0.0691	9	0.0349	-1

Table 6.8: The comparison and ranking of NDCG with cutoff at 20 for all models on dataset Amazon-book, excluding the rank of TopPop as it is already included in the carousel. ΔRank represents the difference between the rank when evaluated individually and the rank when evaluated in the carousel layout. A negative ΔRank indicates that the model is in a lower ranking position.

tween the two evaluations, there were no significant changes observed. About half of the algorithms showed minor changes, with only a shift of 1 to 2 ranks. This includes the all MF algorithms except for IALS, and AMF. The remaining half of the algorithms maintained their original rankings, such as the two KNN CF algorithms, SLIM BPR, and IALS.

Regarding the comparison of MF BPR, Simple-HGN, and AMF in the carousel evaluation, the findings were consistent with the individual evaluation. Simple-HGN exhibited the best performance among the three algorithms, while AMF ranked between the other two in terms of performance.

6.4.3 Popularity analysis

AP comparison

Looking at the AP results of all algorithms depicted in Figure 6.13, there are three algorithms that have relatively lower values, around 0.14, which exceed the true level by 28.57%, ranked in ascending order as IALS, MF SVD++, and PureSVD. On the other hand, AMF shows the worst performance among all algorithms with an AP value that is about twice the true level. The results of Simple-HGN and MF BPR are almost identical,

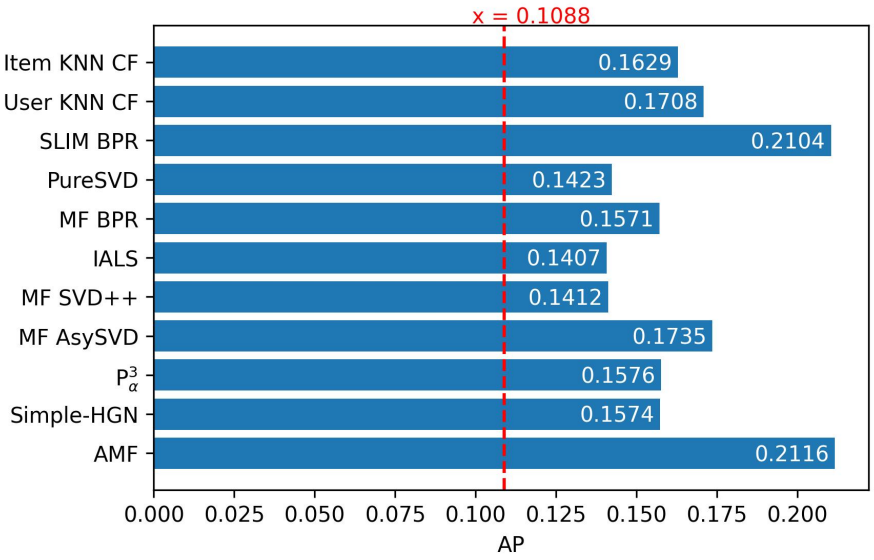


Figure 6.13: The comparison between the recommended results of each algorithm and the actual user profiles in AP on dataset Amazon-book, the AP value of user profiles is marked by the red dotted line and the right part of it shows the ΔAP.

around 0.157 exceeds the true level by 44.30%.

Popularity bias

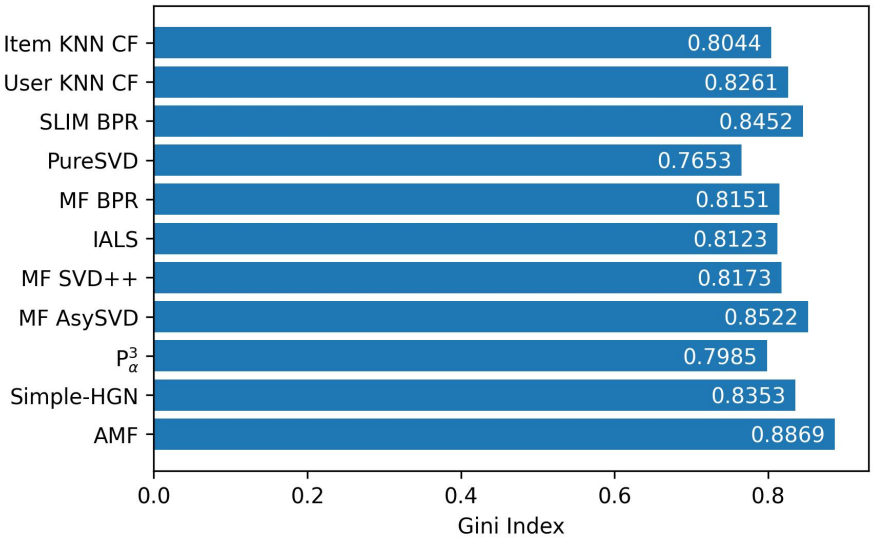


Figure 6.14: Comparison of the Gini index of the item popularity in the recommendation results of each algorithm on dataset Amazon-book

According to the results shown in Figure 6.14, PureSVD and AMF have the lowest and highest value for the Gini index, respectively. Given that the AP level of PureSVD

is also relatively low compared to other algorithms, with a difference of less than 0.01 from the lowest value, it can be inferred that PureSVD performs almost the best among all algorithms in mitigating popularity bias issue. Moreover, AMF exhibits the poorest performance on this issue as it also has the highest AP level among all the algorithms.

When comparing Simple-HGN with MF BPR, it can be observed that the latter has a lower Gini index, which indicates a more uniform frequency distribution of recommended items. Additionally, since the AP levels of both algorithms are equal, it can be inferred that MF BPR has a stronger capability to reduce popularity bias.

Long tail exploration ability

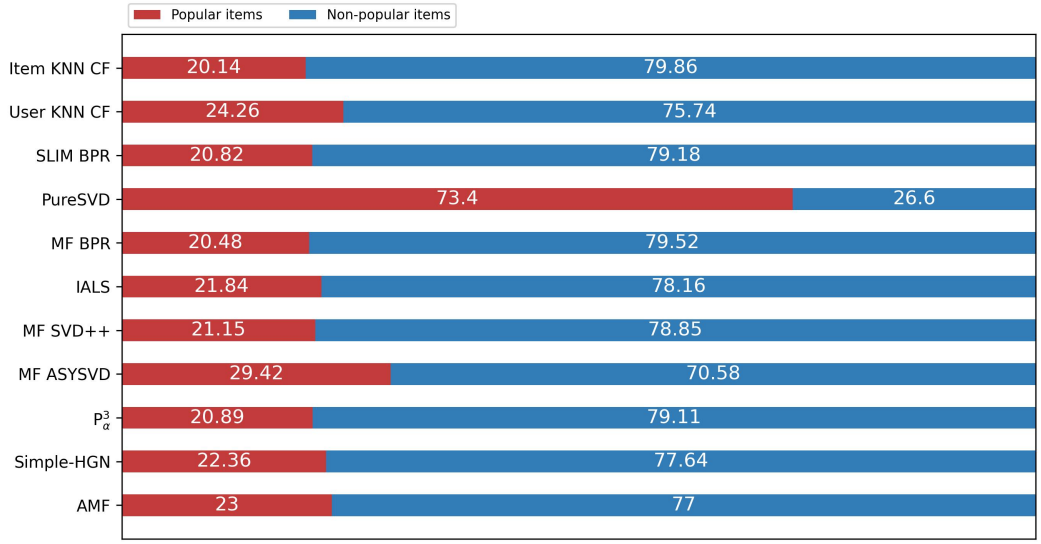


Figure 6.15: The comparison of the proportion distribution of popular and non-popular items in each algorithm's recommendation results on dataset Amazon-book

In Figure 6.15, the ratio of recommended quantities of popular and non-popular items for each algorithm is presented. The majority of algorithms have a percentage of non-popular items close to 80%, with Item KNN CF slightly surpassing the others. However, PureSVD has the lowest proportion, which is less than 30%. The recommended proportion of non-popular items by AMF and Simple-HGN are similar, accounting for approximately 77% of the total proportion of non-popular items. Compared to MF BPR, the latter has a higher proportion, which is 2.52% more.

Figure 6.16 presents the frequency of recommendation for non-popular items. P^3_α has the best performance in recommending non-popular items, surpassing Item KNN CF by a small margin. In comparison between AMF, Simple-HGN and MF BPR, AMF

recommends the lowest frequency of non-popular items, while MF BPR has the highest frequency.

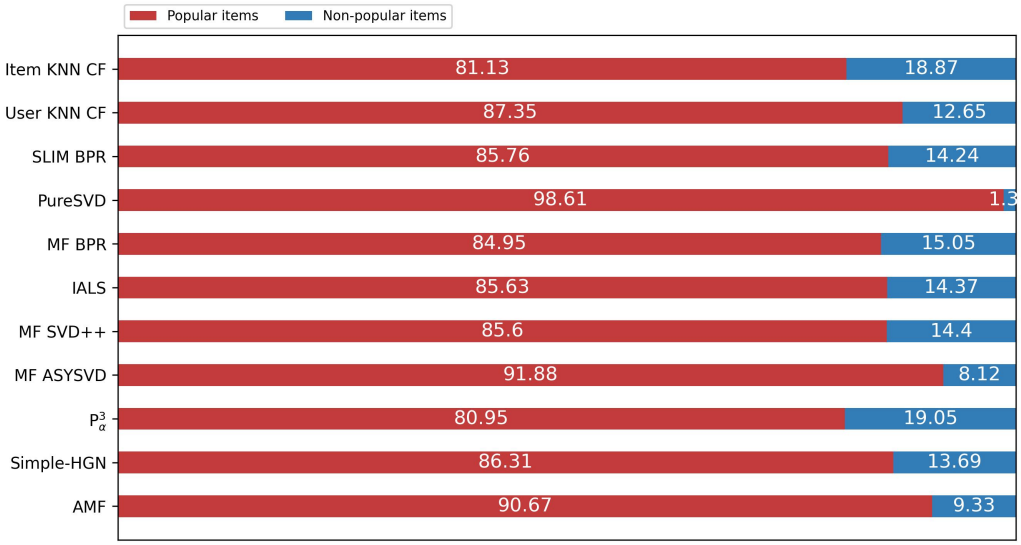


Figure 6.16: The comparison of the frequency distribution of popular and non-popular items in each algorithm's recommendation results on dataset Amazon-book

In summary, from the comparison between Item KNN CF and P_α^3, the latter's recommendation frequency of non-popular items is better than the former's. Hence, when the recommended quantities of both types of items are distributed similarly, P_α^3 is found to be effective on excavation of long tail items.

When analyzing AMF, it becomes evident that, under similar circumstances of recommending a nearly equal number of non-popular items, AMF has a lower frequency of recommending such items compared to Simple-HGN. Consequently, its capability of recommending long-tail items is weaker. Conversely, when compared to MF BPR, the latter has clear advantages in both the number and frequency of non-popular items recommended, thus demonstrating a stronger ability in this regard.

6.5 Summary

In this section, we provide a concise summary of the analysis conducted on the four aforementioned datasets. We will present the findings from three different analytical perspectives: model performance analysis, carousel analysis, and popularity analysis.

Starting with the model performance analysis, the recommendation quality of the AMF model is not competitive compared to KNN algorithms, which are known for their strong performance and consistent leading position among all algorithms. However, in most cases, its recommendation quality is comparable to other MF algorithms like MF BPR, albeit at the expense of sacrificing item coverage and diversity. In Chapter 4, we described one of the motivations for transforming Simple-HGN into the AMF model, as we believed that the HGNN component performed less useful compared to the pre-trained MF BPR embeddings component. However, the results indicate otherwise: even after enhancing the learning of the remaining MF BPR embeddings component through the application of attention mechanisms, AMF consistently underperformed Simple-HGN in all cases. This implies that the deep GNN was valuable, and its removal adversely impacted the model quality.

Moving on to the analysis of carousel recommendations, considering the performance of all algorithms, both KNN algorithms and Simple-HGN show a relative decrease in the quality of recommended results compared to their individual performance when using the TopPop algorithm's recommendations as the preset carousel layout. This indicates that they recommend more popular items compared to other algorithms. On the other hand, for AMF, in the carousel scenario, its recommendation quality slightly lags behind other MF algorithms and their individual performance. This is because other MF algorithms generally experience a slight improvement in their rankings, while AMF remains unchanged.

Finally, let's discuss the analysis of popularity. This part can be summarized into two aspects: addressing the popularity bias issue and the ability to explore long-tail items.

Regarding the first aspect, in most cases, the algorithms involved in the experiments tend to recommend popular items to varying degrees, thereby exacerbating the popularity bias. The algorithms that exhibit a higher degree of this tendency include Item KNN CF and P_α^3. In addition, because AMF incorporates the idea of P_α^3 in its design to enhance the predictive ability of the model, it also significantly amplifies the popularity bias issue. As a result, AMF often falls short in comparison to Simple-HGN and other MF algorithms in this regard.

Moving on to the analysis of the ability to explore long-tail items, although Item KNN CF tends to heavily recommend popular items in some scenarios, as a recommendation algorithm based on item similarity, but it demonstrates excellent capability in recommending long-tail items, which is evident across all datasets. However, for the AMF model, its performance in this aspect is also generally inferior to other MF algorithms and Simple-HGN in most cases.

6.6 Model sensitivity analysis

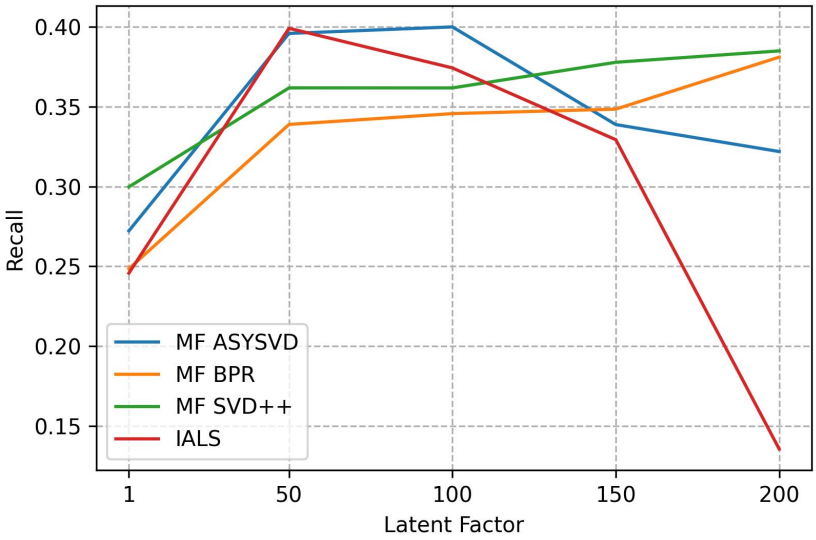


Figure 6.17: The effect of the latent factor dimension on the performance of MF algorithms on dataset MovieLens, taking Recall with cutoff at 20 as the measure metric.

In this section, we start by analyzing the performance of four commonly used MF algorithms and their variation characteristics as the latent factors increase from small to large. This provides us with an initial understanding of the sensitivity of the models to latent factors. Subsequently, we focus on the analysis of the three latent factors in AMF.

To begin with, Figure 6.17 shows that the Recall performance of all algorithms is influenced by the latent factor dimension, as the Recall either increases or decreases with the increase of latent factors. This suggests that latent factor dimension has an impact on their performance. Additionally, the four algorithms can be classified into two groups based on the trend of Recall. One group is MF BPR and MF SVD++, The Recall of them increases continuously as the latent factor dimension increases, while the other group is MF ASYSVD and IALS, Their trends of Recall first increase and then decrease. By examining the change value of Recall in different intervals presented in Table 6.5, it can be concluded that IALS is the most sensitive to changes in latent factor dimension due to its largest change values of Recall in most ranges among the four algorithms. In contrast, MF SVD++ is the least sensitive. Furthermore, MF ASYSVD is the most sensitive in the 100-150 range, while MF BPR is the least sensitive.

When analyzing the result of AMF, which is shown in Figure 6.18, it is obvious that compared to the above MF algorithms, the changes in Recall vary significantly with the

Interval	MF ASYSVD	MF BPR	MF SVD++	IALS
1-50	0.1237	0.0904	0.0620	**0.1534**
50-100	0.0041	0.0068	0.0000	**-0.0248**
100-150	**-0.0612**	0.0028	0.0161	-0.0450
150-200	-0.0169	0.0326	0.0071	**-0.1938**

Table 6.5: Comparison of RECALL values of MF algorithms with cutoff at 20 in different latent factor intervals, the bold value indicates the maximum value of change in each interval, and the underlined value indicates the minimum value of change in each interval.

three different latent factor dimensions.

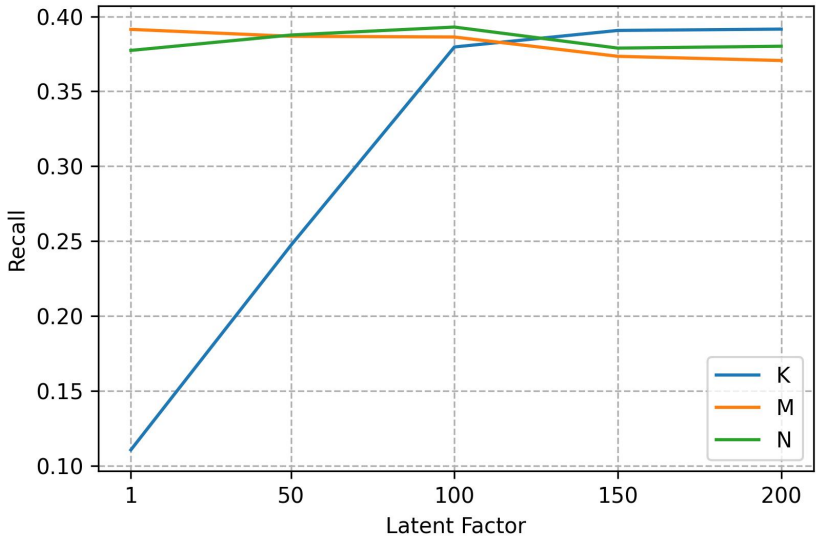


Figure 6.18: The effect of the latent factor dimensions on the performance of AMF on dataset MovieLens, taking Recall with cutoff at 20 as the measure metric. K is the latent factor of user-item weight matrix, M is the latent factor of the MF that calculates attention weights for user similarities, and N is the latent factor of the MF that calculates attention weights for item similarities.

In AMF, under the influence of M and N, the overall trend of Recall is stable without major fluctuations. Specifically, the Recall change trend of M has been declining slowly from the beginning, and the difference in Recall between the smallest and largest dimension is only about 0.02. Meanwhile, the trend of N initially rises before declining, with the difference in Recall between the minimum and maximum values also being only about 0.02. Therefore, it can be concluded that the sensitivity of AMF to the dimension of these two latent factors is quite low. On the other hand, as the latent factor of the first MF

in the algorithm, K is more sensitive to changes in latent factor dimension, the Recall continuous improves with the increase of dimension, and the fastest growing range is the dimension from 1 to 100, where the change value between the beginning and the end is about 0.14. However, the speed of growing in the range of 100-200 drops significantly, with a change value of only about 0.01.

Chapter 7: Conclusion

In this paper, we introduce Attention-based Matrix Factorization (AMF), which is primarily based on the MF technique. It calculates scores based on the user-item weight matrix formed by MF and combines user embeddings from the first weight matrix to compute the similarity between users, multiplied by attention weights obtained from a set of graph-based embeddings assigned to these similarities. Similarly, it combines item embeddings from the same weight matrix to compute the similarity between items, multiplied by attention weights obtained from another set of graph-based embeddings assigned to these similarities.

From the experimental results, we observed a significant gap in the recommendation performance between the AMF model and Simple-HGN in all cases, indicating the crucial role of the HGNN model in the performance of Simple-HGN. Additionally, we noticed that the AMF model, developed primarily using MF as a technique, exhibits similar recommendation quality to other mainstream MF algorithms in most cases. However, it overlooks the item coverage and diversity of the recommended results. In the context of real-world application scenarios, we conducted carousel analysis. Under the condition of using the TopPop algorithm as the existing carousel layout, in terms of relative performance among all the algorithms, AMF shows no significant difference from its individual performance in most cases, lagging behind Simple-HGN and other MF models. Then, an interesting point to note is that Simple-HGN's performance drops significantly because it generated trivial recommendations on MovieLens. Additionally, conducting a popularity analysis revealed that AMF did not bring any surprising results in terms of mitigating popularity bias and enabling exploration of long-tail items. Regrettably, AMF consistently performed inferior to Simple-HGN and other MF models in these aspects.

Furthermore, in addition to the three aforementioned analyses, we conducted preliminary exploration of the characteristics of the AMF model. We primarily analyzed the sensitivity of the model to the sizes of latent factors in the three MF methods. The results indicated that the performance of the AMF model is sensitive to the sizes of latent factors in the user-item weight matrix, while being insensitive to the sizes of the two sets of latent factors used for decomposing and generating graph-based embeddings.

Future research on the AMF model can be expanded in two aspects. Firstly, the current method for calculating similarity is based on the Pearson coefficient. We can explore the variations in model performance by using a simpler method like direct calculation through vector multiplication. Secondly, in the carousel analysis, we conducted the study without considering user navigation behaviors on the layout. To better reflect the real situation, a more in-depth analysis of the performance of various algorithms should be conducted by incorporating actual user navigation behaviors.

Bibliography

[1] P. Aditya, I. Budi, and Q. Munajat. A comparative analysis of memory-based and model-based collaborative filtering on the implementation of recommender system for e-commerce in indonesia: A case study pt x. Mar. 2017. doi: 10.1109/ICACSIS.2016.7872755.

[2] Y. A. Ali, E. M. Awwad, M. Al-Razgan, and A. Maarouf. Hyperparameter search for machine learning algorithms for optimizing the computational complexity. *Processes*, 11(2), 2023. ISSN 2227-9717. doi: 10.3390/pr11020349. URL `https://www.mdpi.com/2227-9717/11/2/349`.

[3] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith. Cython: The best of both worlds. 13(2), 2011. ISSN 1521-9615. doi: 10.1109/MCSE.2010.118. URL `https://doi.org/10.1109/MCSE.2010.118`.

[4] Bluepi. Classifying different types of recommender systems. https://www.bluepiit.com/blog/classifying-recommender-systems/.

[5] R. Burke. Hybrid web recommender systems. *The adaptive web: methods and strategies of web personalization*, pages 377–408, 2007.

[6] P. Castells, N. Hurley, and S. Vargas. Novelty and diversity in recommender systems. In *Recommender systems handbook*, pages 603–646. Springer, 2021.

[7] A. Castrounis. Ai for people and business. a framework for better human experiences and business success. (ebook). 2019.

[8] M. Chen and P. Liu. Performance evaluation of recommender systems. *International Journal of Performability Engineering*, 13(8):1246, 2017.

[9] D. Chong. Deep dive into netflix's recommender system. https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48.

[10] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. pages 39–46, 09 2010. doi: 10.1145/1864708.1864721.

[11] M. F. Dacrema, S. Boglio, P. Cremonesi, and D. Jannach. A troubling analysis of reproducibility and progress in recommender systems research. *CoRR*, abs/1911.07698, 2019. URL `http://arxiv.org/abs/1911.07698`.

[12] Z. Deutschman. Recommender systems: Machine learning metrics and business metrics. https://neptune.ai/blog/recommender-systems-metrics.

[13] Z. Fayyaz, M. Ebrahimian, D. Nawara, A. Ibrahim, and R. Kashef. Recommendation systems: Algorithms, challenges, metrics, and business opportunities. *Applied Sciences*, 10, 2020.

[14] N. Felicioni, M. Ferrari Dacrema, and P. Cremonesi. A methodology for the offline evaluation of recommender systems in a user interface with multiple carousels. In J. Masthoff, E. Herder, N. Tintarev, and M. Tkalcic, editors, *Adjunct Publication of the 29th ACM Conference on User Modeling, Adaptation and Personalization, UMAP 2021, Utrecht, The Netherlands, June 21-25, 2021*, pages 10–15. ACM, 2021. doi: 10.1145/3450614.3461680. URL `https://doi.org/10.1145/3450614.3461680`.

[15] M. Ferrari Dacrema. Demonstrating the equivalence of list based and aggregate metrics to measure the diversity of recommendations (student abstract). 35:15779–15780, May 2021. doi: 10.1609/aaai.v35i18.17886. URL `https://ojs.aaai.org/index.php/AAAI/article/view/17886`.

[16] M. Ferrari Dacrema, P. Cremonesi, and D. Jannach. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In T. Bogers, A. Said, P. Brusilovsky, and D. Tikk, editors, *Proceedings of the 13th ACM Conference on Recommender Systems, RecSys 2019, Copenhagen, Denmark, September 16-20, 2019*, pages 101–109. ACM, 2019. doi: 10.1145/3298689.3347058. URL `https://doi.org/10.1145/3298689.3347058`.

[17] M. Ferrari Dacrema, F. Parroni, P. Cremonesi, and D. Jannach. Critically examining the claimed value of convolutions over user-item embedding maps for recommender systems. 07 2020.

[18] M. Ferrari Dacrema, N. Felicioni, and P. Cremonesi. Offline evaluation of recommender systems in a user interface with multiple carousels. *Frontiers Big Data*, 5: 21 pages, 2022. ISSN 2624-909X. doi: 10.3389/fdata.2022.910030. URL `https://doi.org/10.3389/fdata.2022.910030`.

[19] S. Gong, H. Ye, and H. Tan. Combining memory-based and model-based collaborative filtering in recommender system. In *2009 Pacific-Asia Conference on Circuits, Communications and Systems*, pages 690–693, 2009. doi: 10.1109/PACCS.2009.66.

[20] A. Gunawardana and G. Shani. A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research*, 10(12), 2009.

[21] X. He, H. Zhang, M. Kan, and T. Chua. Fast matrix factorization for online recommendation with implicit feedback. *CoRR*, abs/1708.05024, 2017.

[22] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[23] K. Holewa. We know what you like! perks of recommendation systems in business. https://www.miquido.com/blog/perks-of-recommendation-systems-in-business/.

[24] G. Karypis and X. Ning. Slim: Sparse linear methods for top-n recommender systems. In *2013 IEEE 13th International Conference on Data Mining*, pages 497–506, Los Alamitos, CA, USA, dec 2011. IEEE Computer Society. doi: 10.1109/ICDM.2011.134. URL `https://doi.ieeecomputersociety.org/10.1109/ICDM.2011.134`.

[25] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL `http://arxiv.org/abs/1609.02907`.

[26] J. A. Konstan and J. Riedl. Recommender systems: from algorithms to user experience. *User modeling and user-adapted interaction*, 22:101–123, 2012.

[27] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. doi: 10.1109/MC.2009.263.

[28] D. Kotkov, J. Veijalainen, and S. Wang. How does serendipity affect diversity in recommender systems? a serendipity-oriented greedy algorithm. *Computing*, 102 (2):393–411, Feb. 2020. ISSN 0144-3097. doi: 10.1007/s00607-018-0687-5. URL `http://link.springer.com/10.1007/s00607-018-0687-5`.

[29] T. J. Lakshmi and S. D. Bhavani. Link prediction approach to recommender systems. *arXiv preprint arXiv:2102.09185*, 2021.

[30] X. Li and H. Chen. Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach. *Decision Support Systems*, 54(2): 880–890, 2013. ISSN 0167-9236. doi: https://doi.org/10.1016/j.dss.2012.09.019. URL `https://www.sciencedirect.com/science/article/pii/S0167923612002540`.

[31] J. W.-B. Lin, H. Aizenman, E. Manette Cartas Espinel, K. Gunnerson, and J. Liu. *An Introduction to Python Programming for Scientists and Engineers*. Cambridge University Press, 2022. doi: 10.1017/9781108571531.

[32] P. Lops, M. De Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. *Recommender systems handbook*, pages 73–105, 2011.

[33] Q. Lv, M. Ding, Q. Liu, Y. Chen, W. Feng, S. He, C. Zhou, J. Jiang, Y. Dong, and J. Tang. Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 1150–1160, 2021.

[34] C. Pan and W. Li. Research paper recommendation with topic analysis. In *2010 International Conference On Computer Design and Applications*, volume 4, pages V4–264. IEEE, 2010.

[35] C. Paolo. *Recommender Systems*. 2021.

[36] F. B. Pérez Maurera, M. Ferrari Dacrema, and P. Cremonesi. An evaluation study of generative adversarial networks for collaborative filtering. In M. Hagen, S. Verberne, C. Macdonald, C. Seifert, K. Balog, K. Nørvåg, and V. Setty, editors, *Advances in Information Retrieval - 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10-14, 2022, Proceedings, Part I*, volume 13185 of *Lecture Notes in Computer Science*, pages 671–685. Springer, 2022. doi: 10.1007/978-3-030-99736-6_45. URL https://doi.org/10.1007/978-3-030-99736-6_45.

[37] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[38] S. Rendle, W. Krichene, L. Zhang, and J. R. Anderson. Neural collaborative filtering vs. matrix factorization revisited. In R. L. T. Santos, L. B. Marinho, E. M. Daly, L. Chen, K. Falk, N. Koenigstein, and E. S. de Moura, editors, *RecSys 2020: Fourteenth ACM Conference on Recommender Systems, Virtual Event, Brazil, September 22-26, 2020*, pages 240–248. ACM, 2020. doi: 10.1145/3383313.3412488. URL https://doi.org/10.1145/3383313.3412488.

[39] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. *Collaborative Filtering Recommender Systems*, pages 291–324. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-72079-9. doi: 10.1007/978-3-540-72079-9_9. URL https://doi.org/10.1007/978-3-540-72079-9_9.

[40] G. Schröder, M. Thiele, and W. Lehner. Setting goals and choosing metrics for recommender system evaluations. In *UCERSTI2 workshop at the 5th ACM conference on recommender systems, Chicago, USA*, volume 23, page 53, 2011.

[41] L. Semage. Recommender systems with random walks: A survey. 11 2017.

[42] B. Shams and S. Haratizadeh. Graph-based collaborative ranking. *Expert Systems with Applications*, 67:59–70, 2017.

[43] G. Shani and A. Gunawardana. *Evaluating Recommendation Systems*, pages 257–297. Springer US, Boston, MA, 2011. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_8. URL https://doi.org/10.1007/978-0-387-85820-3_8.

[44] G. Shani and A. Gunawardana. Evaluating recommendation systems. *Recommender systems handbook*, pages 257–297, 2011.

[45] C. Shi. *Heterogeneous Graph Neural Networks*, pages 351–369. Springer Nature Singapore, Singapore, 2022. ISBN 978-981-16-6054-2. doi: 10.1007/978-981-16-6054-2_16. URL https://doi.org/10.1007/978-981-16-6054-2_16.

[46] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms, 2012.

[47] H. Steck. Embarrassingly shallow autoencoders for sparse data. In L. Liu, R. W. White, A. Mantrach, F. Silvestri, J. J. McAuley, R. Baeza-Yates, and L. Zia, editors, *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 3251–3257. ACM, 2019. doi: 10.1145/3308558.3313710. URL https://doi.org/10.1145/3308558.3313710.

[48] Y. Sun and J. Han. Mining heterogeneous information networks: a structural analysis approach. *Acm Sigkdd Explorations Newsletter*, 14(2):20–28, 2013.

[49] F. Vahedian, R. Burke, and B. Mobasher. Weighted random walk sampling for multi-relational recommendation. In *Proceedings of the 25th conference on user modeling, adaptation and personalization*, pages 230–237, 2017.

[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

[51] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[52] J. Wang, A. P. De Vries, and M. J. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 501–508, 2006.

[53] G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu, and Z. Chen. Scalable

collaborative filtering using cluster-based smoothing. In R. A. Baeza-Yates, N. Ziviani, G. Marchionini, A. Moffat, and J. Tait, editors, *SIGIR*, pages 114–121. ACM, 2005. ISBN 1-59593-034-5. URL http://dblp.uni-trier.de/db/conf/sigir/sigir2005.html#XueLYXZYC05.

[54] X. Yang, M. Yan, S. Pan, X. Ye, and D. Fan. Simple and efficient heterogeneous graph neural network, 2022. URL https://arxiv.org/abs/2207.02547.

[55] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 793–803, 2019.

[56] E. Zhang and Y. Zhang. *Average Precision*, pages 192–193. Springer US, Boston, MA, 2009. ISBN 978-0-387-39940-9. doi: 10.1007/978-0-387-39940-9_482. URL https://doi.org/10.1007/978-0-387-39940-9_482.

[57] J. Zhang, L. Tan, X. Tao, T. Pham, and B. Chen. Relational intelligence recognition in online social networks — a survey. *Computer Science Review*, 35:100221, 2020. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2019.100221. URL https://www.sciencedirect.com/science/article/pii/S1574013718303575.

[58] Y. Zhu, J. Lin, S. He, B. Wang, Z. Guan, H. Liu, and D. Cai. Addressing the item cold-start problem by attribute-driven active learning. *IEEE Transactions on Knowledge & Data Engineering*, pages 631–644, 2020.