Surveying Representation Models for Collaborative Filtering Recommendation Systems

Jonathan Samuel
Department of Computer Science
Virginia Polytechnic Institute and State University
samueljon17@vt.edu

Abstract

Recommendation systems have gained significant importance in recent years, enabling online platforms to provide personalized recommendations to users based on their past behavior and preferences, thus improving user engagement and satisfaction. While content-based filtering has been a popular approach, the emergence of big data and advancements in machine learning has led to the prominence of various other approaches, such as collaborative-based filtering, matrix factorization, and deep learning-based models. This paper presents a comprehensive survey of various recommendation systems, including their underlying techniques, advantages, and limitations. Specifically, this study experiments with different collaborative filtering models using implicit and explicit data, such as matrix factorization with alternating least square (MF-ALS), neural collaborative filtering (NCF), and light graph convolutional networks (LightGCN), to recommend the top K items for users. The results show that MF-ALS performs comparably to some state-of-the-art deep learning-based approaches. Moreover, LightGCN, leveraging the structure of the graph, outperforms the baseline, NCF, and MF-ALS in recall@K. The findings highlight the potential of collaborative filtering approaches in recommendation systems and provide insights into the strengths and weaknesses of different models.

1 Introduction

In recent years, there has been a significant increase in the usage of information filtering systems in various online platforms. These systems aim to reduce the information overload by removing redundant or unwanted information from the information stream before it is presented to the user. Among active information filtering systems, recommendation systems are particularly popular, and are widely used in e-commerce, social media, and online content platforms [8]. For instance, Netflix utilizes sophisticated recommendation systems that analyze user behavior data, including viewing history, search queries, and contextual information, to personalize content offerings and provide tailored recommendations to its users. Given the challenges posed by the abundance of information, recommendation systems have become a critical tool for users to manage information overload. Therefore, building effective recommendation systems is crucial, and requires accurate modeling of users' preferences to ensure enhanced user satisfaction and engagement.

One approach for recommendation systems are content-based approaches, where items are recommended based on the similarities with items that the user has historically preferred [4]. In contrast, another approach is to recommend items via collaborative filtering (CF) [6, 2]. These methods construct personalized suggestions by leveraging user behavior data to identify individuals with similar preferences, and subsequently recommend items that were favored by those users in the past.

Under CF, there lies two main frameworks: neighborhood-based methods and representation learning or latent factor analysis models [4]. Neighborhood methods rely on the assumption that users prefer items that are similar to the ones they've liked before. Similarity can be defined in many ways such

as cosine similarity metrics, Pearson similarity, or Jaccard similarity. Also, similarity can be defined as the minimum distance between a data point and another point in space as used by K-Nearrest Neighbor (KNN) models. In contrast, representation learning aims to learn a low-dimensional representation of users and items that captures their latent features and relationships. Some examples include matrix factorization and deep learning models [3, 1]. Early neighborhood methods were successful due to their simplicity, efficiency, and effectiveness in directly recommending items to users [3]. However, representation learning has become increasingly more popular as they've been proven more accurate to classical neighborhood methods.

Most of the data in recommendation systems have a graph structure where user-item interactions can be represented by a bipartite graph between user and item nodes, with observed interactions represented by links. Therefore, geometric deep learning methods, in particular graph neural networks (GNNs), is a rapidly evolving field that has great potential to outperform traditional structured models [8]. GNNs have the ability to naturally and explicitly encode the crucial collaborative signal (i.e., topological structure) to improve user and item representations unlike traditional methods that only implicitly capture collaborative signals.

2 Problem Definition and Motivation

When selecting a model or framework for recommendation systems, several factors related to the data should be considered. These include: explicit or implicit feedback, size, degree of sparsity, diversity and temporal dynamics [8]. The aim of this study is to propose several methods for recommending books to users based on their historical user-item implicit and explicit data. The task involves predicting the top K unobserved items, denoted by i_i, for a given user, denoted by u_i, using various neighborhood-based, model-based, and graphical-based collaborative filtering techniques. The primary objective of this work is to understand the technical landscape within representational models for recommendation systems.

3 Related Work

3.1 Neighborhood Based Models

3.1.1 KNN

One type of collaborative filtering, neighborhood based models are based on the K-Nearest Neighbor (KNN) algorithm. KNN aims to identify items that are similar to those preferred by a user by analyzing the preferences of other users. This is achieved by identifying K other neighbors that satisfy the minimum distance metric criterion. The KNN algorithm applies a majority vote system to label a given data point. For instance, in a study Sarwar et al. [5], a KNN-based algorithm was used to recommend music to users based on their listening history. The authors found that the algorithm was able to effectively capture users' preferences and generate accurate recommendations. Despite the KNN algorithm's usefulness, this approach does not leverage the implicit characteristics or attributes of a data point that are not explicitly provided, which representation models can utilize.

3.2 Representation Based Models

3.2.1 Matrix Factorization

Matrix factorization is a popular technique in recommendation systems. This approach aims to factorize the user-item rating matrix into two lower-dimensional matrices, representing the latent user and item features [3]. This enables the identification of latent user preferences and item attributes that can be used to generate accurate recommendations.

Funk matrix factorization and SVD++ are notable examples that have been shown to achieve state-of-the-art performance in various domains [4]. Funk matrix factorization, proposed by Funk (2006), is the *vanilla* or basic version of matrix factorization that uses stochastic gradient descent to optimize the user-item rating matrix factorization. There are various other extensions to the model such as SVD++, proposed by Koren et al. [3], to address some of the limitations of traditional matrix factorization by introducing bias terms and incorporating implicit feedback information.

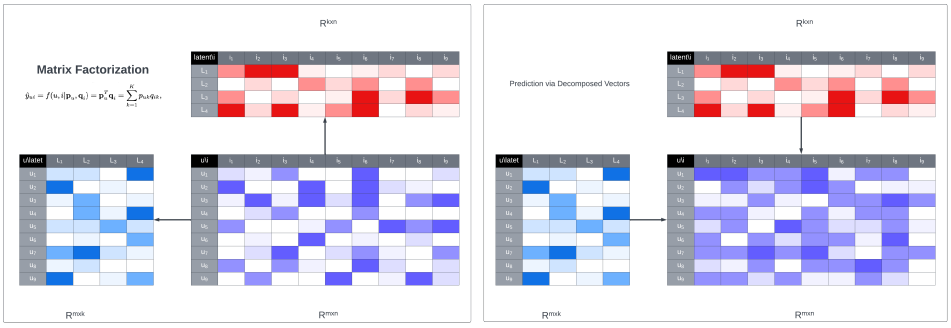


Figure 1: Matrix Factorization Concept

$$\min_{P,Q} \sum_{(u,i)} ||R_{u,i} - \widehat{R}_{u,i}||^2 + \lambda(||P||_F^2 + ||Q||_F^2) \tag{1}$$

The basic matrix factorization is expressed as follows: given the user-item interaction, $\widehat{R}_{u,i} = PQ^T$ will be factorized into a user latent matrix $P \in R^{m \times k}$ and an item latent matrix $Q \in R^{n \times k}$, where u denotes a specific users and i denotes a specific items, and the values of $\widehat{R}_{u,i}$ represent explicit ratings, then the goal for matrix factorization is to minimize the mean square loss between the actual rating of $R_{u,i}$ and predicted rating $\widehat{R}_{u,i}$ (Eq. 1). λ denotes the regularization rate and is used to prevent over fitting the MF model.

3.2.2 Deep Learning

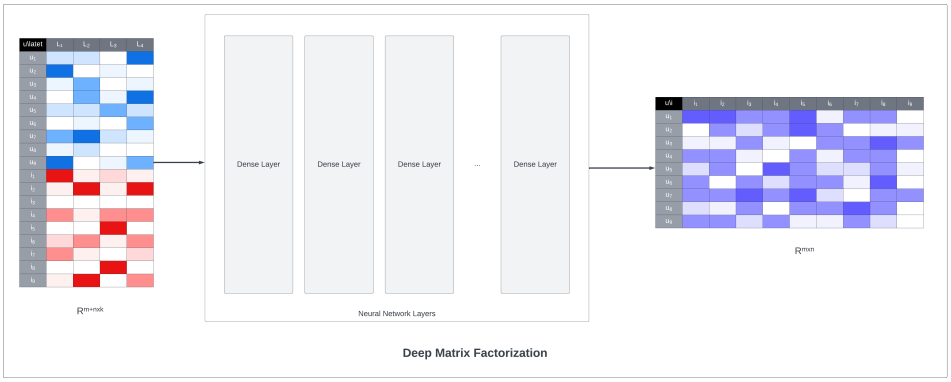


Figure 2: Basic architecture of Deep Learning CF

More recently, deep neural networks (DNNs) have also been proposed for CF tasks. DNNs have the ability to learn complex and nonlinear representations of user-item interactions. Compared to traditional CF models, which rely on matrix factorization techniques to generate user and item embeddings, DNNs can learn hierarchical representations that capture both local and global dependencies between users and items, resulting in more accurate predictions [4]. For instance, neural collaborative filtering (NCF) [1] introduced a neural network-based approach that can capture non-linear user-item interactions. The model employs a multi-layer perceptron (MLP) to learn the user and item embeddings and combines them using a dot product to predict the user-item ratings. Other works have extended this approach, for example, by incorporating attention mechanisms or by using convolutional neural networks (CNNs) to model the user-item interactions. Given a vector that specifies the identity a given user and item, v_u^U and v_i^I respectfully, and the model parameters Θ_f, the predicted rating, $\hat{y}_{ui}$ is as follows:

$$\hat{y}_{ui} = f(P^T v_u^U, Q^T v_i^I | P, Q, \Theta_f) = \phi_{out}(\phi_x(...\phi_2(\phi_1(P^T v_u^U, Q^T v_i^I))...)) \tag{2}$$

where ϕ_{out} and ϕx respectively denote the mapping function for the output layer and x-th neural collaborative filtering (CF) layer.

3.2.3 Graph Based CF

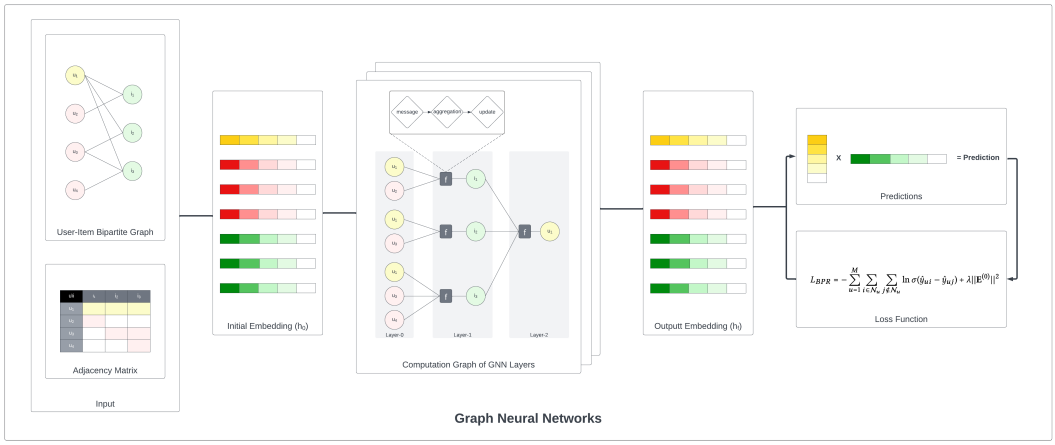


Figure 3: Basic architecture of Graph Neural Networks

The field of recommendation systems has witnessed a significant evolution over the years, as we've witnessed neighborhood based models, matrix factorization, and deep learning models so far. Despite their success in several domains, these methods have limitations in modeling complex relationships between users and items, as they often do not account for higher-order interactions. Graph neural networks (GNNs) have recently emerged as a promising technique for recommendation systems due to their ability to model complex interactions and relationships between entities [8, 6, 2]. GNNs leverage graph theory to capture the dependencies and interactions between users, items, and other entities in a graph structure allowing for more accurate and efficient recommendations.

Figure 3 shows the basic structure of a GNN. The basic structure consists of a series of graph convolutional layers that update the node representations by aggregating and transforming information from neighboring nodes. Each layer involves two steps: message passing and message aggregation. During message passing, each node sends a message to its neighbors that summarizes its own features. The messages are then aggregated and transformed by a learnable function that updates each node's representation. The process is repeated for multiple layers, allowing the network to capture higher-order interactions between nodes. The final node representations can then be used for various downstream tasks, such as node classification or link prediction.

To design and develop effective collaborative filtering based GNNs, there are five items to consider:

- Graph construction: How to structure the graph and which neighbors to sample for computation efficiency. For example, using a bipartite graph or enriching the bipartite graph (Figure 3) with additional edges that represent K-hop interactions.
- Messaging: Given a prior hidden layer for node u, $h_u^{(l-1)}$, how do we transform the prior message from neighboring nodes such that $m_u^{(l)} = MSG^{(l)}(h_u^{(l-1)})$. An example includes a linear transformation on the prior hidden layer, $m_u^{(l)} = W^{(l)}(h_u^{(l-1)})$ where $W^{(l)}$ is a trainable weight matrix.
- Neighbor aggregation: Given a set of messages for node v's neighbors, N_v, how do we compress the information from neighboring nodes and account for the importance of each neighbor such that $h_v^{(l)} = AGG^{(l)}(\{m_u^{(l)}, u \in N_v\})$. An example includes sum(*) such that $h_v^{(l)} = Sum(\{m_u^{(l)}, u \in N_v\})$, mean(*), max(*), etc.
- Information update: How to integrate the central node's representation with the aggregated representation of its neighbors. For example, a central nodes representation can be dropped or added within the message updating procedure.

- Final node representation: Given all layers for a given node u, how do we represent the final layer? For example the final layer, h_u^*, can just be the last representation layer, $h_u^* = h_u^{(L)}$ or a combination of the node representation layers such as a summation-pool: $h_u^* = \sum_{l=0}^{L} h_u^{(l)}$, mean-pool: $h_u^* = \frac{1}{L+1} \sum_{l=0}^{L} h_u^{(l)}$, concatenation: $h_u^* = h_u^{(0)} \oplus h_u^{(1)} \oplus ... \oplus h_u^{(L)}$, etc.

4 Dataset

4.1 Statistics

	500K Subset		Full Dataset	
	Pre-Processed	Post-Processed	Pre-Processed	Post-Processed
# Unique Interactions	500,000	31,519	228,648,342	13,240,977
# Unique Users	1,034	398	876,145	190,435
# Unique Books	181,946	24,542	2,360,650	2,066,636
Sparsity	>99%	>99%	>99%	>99%

Table 1: GoodReads User-Item dataset details

The present study utilizes the GoodReads *shelves* dataset, comprising over 229 million user-item interactions. The dataset provides detailed information on users and books interactions, including `user_id & book_id`, the user's rating of a given book (`ratings`), and two binary fields (`is_read & is_reviewed`). The binary fields are hypothesized to indicate whether a user has declared having read and reviewed a book within the GoodReads application. This dataset comprises both explicit and implicit data, including user-item rankings and whether a user has reviewed a book (0 otherwise).

A second dataset, *books*, is employed for manual evaluation purposes. This dataset contains comprehensive meta-data information such as book titles, authors, descriptions, publication years, and 20+ additional fields. This dataset is solely used for manual evaluation, whereby predicted books are randomly sampled and compared with existing books that users have read within the training set. A full description of both datasets can be found in the footnotes [1].

4.2 Pre-Processing

Due to limited computational resources, the original dataset was trimmed to the first 500,000 user-item interactions. As suggested, `is_read & is_reviewed` are interpreted therefore only data that has true for both fields were considered which allots to 10% of the data as shown in Figure 5a. To develop a rich user-item bipartite graph representing user's interest towards a given book, user-item interactions with ratings greater than or equal to 3 are considered representing 90% of the data shown in Figure 5b. Lastly, to handle the cold-start problem, I adopt the 10-core setting [2, 6] to ensure that each user and item have at least ten interaction (Figure 5c). The dataset is partitioned into a training set and a test set by randomly sampling 80% of the historical interactions of each user as the training set, while the remaining interactions are reserved for the test set. Each observed user-item interaction is treated as a positive instance and is subjected to a negative sampling strategy, whereby it is paired with one item that the user has not previously consumed. This process generates negative instances that are used for model training and evaluation.

5 Methods

5.1 Popularity

For a baseline assessment, I developed a model that randomly selects the top K books for a given user by having the random selection be proportional to the book normalized popularity / item-degree. In this case, normalized popularity is defined as the count of users that have ranked a given book divided the total number of users, U.

[1] UCSD GoodReads Dataset `https://sites.google.com/eng.ucsd.edu/ucsdbookgraph`

$$\frac{1}{U}\sum_{u=0}^{U}(1 : R_{u,i}; 0 : else) \tag{3}$$

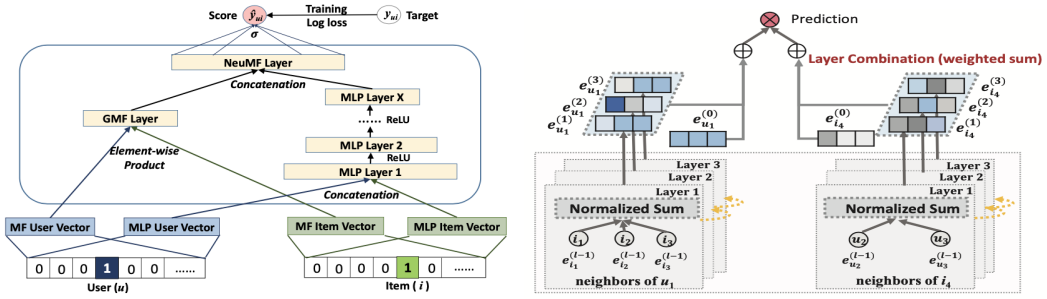
5.2 Matrix Factorization

This model uses the optimization objective function described in section 3.2.1, specifically Eq. 1. Using the user-item interaction processed dataset, I can directly optimize the user and item latent vectors. One approach is to apply gradient descent but requires finding the gradient for both vectors separately, which can be computationally expensive. An alternative method known as alternating least squares with weighted regularization (ALS-WR), can be used where one vector is treated as a constant while the gradient is solved with respect to the other vector. ALS-WR has a closed-form solution (Eq. 4 & 5) for updating the factors, which eliminates the need for iterative optimization and can lead to faster convergence.

$$\frac{\partial L}{\partial P_u} = R_u Q(Q^T Q + \lambda I)^{-1} \tag{4}$$

$$\frac{\partial L}{\partial Q_i} = R_i P(P^T P + \lambda I)^{-1} \tag{5}$$

5.3 Neural Graph Collaborative Filtering



(a) Neural Collaborative Filtering Architecture [1]

(b) LightGCN Convolutional Layer [2]

Figure 4: DNN and GNN Architectures

Amongst all the DNNs, Neural Collaborative Filtering (NCF) is a popular deep learning-based approach for collaborative filtering [1]. NCF is combines the strengths of Generalized Matrix Factorization (GMF) and Multi-Layer Perceptron (MLP) to model the latent structures of users and items, integrating the linearity of MF with the non-linearity of MLP. With σ_{out} representing an activation function and $\odot$ as element-wise product, GMF can be represented as:

$$GMF = \hat{R}_{ui} = \sigma_{out}(h^T(Q_i \odot P_u)) \tag{6}$$

The MLP is split amongst it's input, hidden and output layer. With W_l, b_l denoting the weight and bias matrix/vector for layer l respectfully, the MLP model can be represented as:

$$InputLayer : z_1 = \phi_1(P_u, Q_i) \tag{7}$$

$$HiddenLayers : \phi_l(z_l) = \sigma(W_l^T z_l + b_l) \forall l \in [2, ..., L-1] \tag{8}$$

$$OutputLayers : \hat{R}_{ui} = \sigma(h^T \phi(z_{L-1})) \tag{9}$$

Thus, the NCF model trains separate latent factor matrices for both GMF and MLP and combines the final representation through concatenation to achieve the final form as shown in Figure 4a

$$\hat{R}_{ui} = \sigma(h^T \begin{bmatrix} P_u^{GMF} \odot Q_i^{GMF} \\ \phi(z_{L-1}) \end{bmatrix}) \tag{10}$$

I used the packaged version Microsoft[2] developed to employ an NCF model on the user-item implicit interaction dataset (stripping the explicit rating information).

5.4 LightGCN

I developed a GNN following a LightGCN architecture [2]. The LightGCN architecture only trains the 0-th layer embeddings $h_u^{(0)}$ and $h_i^{(0)}$ for users and items, respectively. In addition, LightGCN found that removing non-linear activation functions has little effect on the model's performance, resulting in a more straightforward update process and improved computational efficiency compared to other GNN architectures. As described in section 3.2.3, there are five items to consider when developing a GNN: graph construction, messaging, neighborhood aggregation, information update and final node representation. For LightGCN, the architecture described in Figure 4b consists of:

- Graph construction: Bipartite Graph
- Messaging: Degree normalization
  - User Message: $m_u^{(l)} = \frac{1}{\sqrt{|N_u|}\sqrt{|N_i|}} h_i^{(l)}$
  - Item Message: $m_i^{(l)} = \frac{1}{\sqrt{|N_u|}\sqrt{|N_i|}} h_u^{(l)}$
- Neighbor aggregation: Weighted Sum
  - User Message: $h_u^{(l+1)} = \sum_{i \in N_u} m_u^{(l)}$
  - Item Message: $h_i^{(l+1)} = \sum_{u \in N_i} m_i^{(l)}$
- Information update: Drop the central node's representation and only consider the neighbors
- Final node representation: Weighted-pooling
  - User Message: $h_u^* = \sum_{k=0}^{K} \alpha_k h_u^{(k)}; s.t. \alpha_k \geq 0$
  - Item Message: $h_i^* = \sum_{k=0}^{K} \alpha_k h_i^{(k)}; s.t. \alpha_k \geq 0$

In, He et al. [2], suggest setting $\alpha_k = 1/(K+1)$ as it generally performs well. For this experiment, I manually developed a LightGCN GNN using PyTorch and compared it with Microsoft's version of LightGCN within their Python Recommender package mentioned earlier.

6 Experimental Settings

Method	Loss Function	Optimization Strategy
Popularity	—	—
MF-ALS-WR	Mean Square Error (MSE)	ALS-WR
NCF-Microsoft	Binary Cross Entropy Loss (BCE)	Stochastic Gradient Descent (SGD)
LightGCN	Bayesian Personalized Ranking (BPR)	Adam Optimizer w. Mini Batch
LightGCN-Microsoft	Bayesian Personalized Ranking (BPR)	Adam Optimizer w. Mini Batch

Table 2: Method's loss and optimization strategy

As discussed in section 3.2.1, specifically Eq. 1, the loss function for MF uses the MSE with an ALS-WR approach to solve the optimization problem. The NCF model uses a binary cross entropy loss 11 to measure the difference between the true binary labels and the predicted probabilities for each example in the dataset. The loss is calculated as the negative log-likelihood of the observed data given the predicted probabilities.

$$L_{BCE} = \sum_{(u,i) \in R \cup R^-} R_{ui} log \hat{R}_{ui} + (1 - R_{ui})(1 - \hat{R}_{ui}) \tag{11}$$

[2]Microsoft Recommenders `https://github.com/microsoft/recommenders`

Method	Latent Factors	Layers	Iter/Epochs	Batch Size	λ
Popularity	—	—	—	—	—
MF-ALS-WR	36	—	20	—	1E-2
NCF-Microsoft	4	16	50	1024	5E-3
LightGCN	64	3	50	1024	5E-3
LightGCN-Microsoft	64	3	50	1024	5E-3

Table 3: Hyper parameters

Both the LightGCN that I developed and the one Microsoft developed use the Bayesian personalized ranking loss 12 [7] since it is designed to optimize the ranking of positive items over negative items for a given user. This is done by learning the loss the user and item embeddings that maximize the probability of a positive item being ranked higher than a negative item. The BPR also incorporates a regularization term, λ, on the only learnable parameter the 0-th layer, i.e. initial embedding $h^{(0)}$.

$$L_{BPR} = \sum_{u=1}^{M} \sum_{i \in N_u} \sum_{j \notin N_u} ln\sigma(\hat{R}_{ui} - \hat{R}_{uj}) + \lambda ||h^{(0)}||^2 \tag{12}$$

To assess the effectiveness of top-K recommendation and preference ranking, I employ two standard evaluation metrics: recall@K and precision@K. Specifically, all unobserved items for each user in the test set are considered negative examples. I calculate the predicted preference scores for all such items using the trained models, excluding the positive examples from the training set. The scores are computed for each user, where K represents the *top* number of recommended items and averaged scores across all users in the test set are reported.

7 Results & Discussion

Method	Training Time (sec)	precision@20	recall@20
Popularity	—	9.52E-4	5.47E-3
MF-ALS-WR	50.8	9.84E-3	4.50E-2
NCF-Microsoft	161.1	5.71E-3	2.43E-2
LightGCN	161.0	9.37E-3	4.77E-2
LightGCN-Microsoft	73.3	1.60E-4	6.35E-4

Table 4: Evaluation for all models

Figure 6 illustrates the training loss of all models except NCF. After approximately 5 iterations, the MF-ALS-WR model 6a appears to have converged to an equilibrium for the decomposed latent factors of users and items. The LightGCN manual 6b and Microsoft packaged 6c models exhibit a difference of two orders of magnitude in training loss. It's currently unclear why there shows a significant discrepancy amongst the two models since they should be approximately equivalent.

Figure 7 and Table 4 present the precision@K and recall@K for all models, including the baseline model. Notably, the LightGCN model provided by Microsoft exhibited the poorest precision@20, while displaying a similar recall@20 value to the baseline model. This discrepancy could potentially be attributed to inconsistencies in the evaluation metrics utilized. To ensure standardized evaluation, Microsoft's precision and recall tools were employed for all models, which may have inadvertently introduced confounding variables that could have influenced the final results. Despite LightGCN-Microsoft's suboptimal precision@20 and recall@20 performances compared to the baseline, Figure 8 illustrates all models' ability to predict the top 20 items for userID 3, and whether those predictions were present in the training or testing set. Notably, LightGCN-Microsoft generated higher rankings for items already present in the user's training set, relative to my model's implementation. Additionally, a key finding was that my model's implementation proposed fewer distinct books for all users, indicating shared embeddings across users.

The performance of three models, namely MF-ALS-WR, NCF-Microsoft, and my implementation of LightGCN, are comparable in terms of the evaluation metrics, with the highest precision@20 and recall@20 achieved by my LightGCN and MF-ALS-WR models. Among them, MF-ALS-WR has a slight edge in terms of training time, taking nearly 100 seconds less to train than my LightGCN on a dataset with approximately 31K entries. This time advantage could potentially save significant resources if the model needs to be retrained in real-time on large user-item datasets.

8 Future Work

In future work, I aim to address several areas to ensure consistency and accuracy in evaluating the performance of collaborative filtering (CF) models. Firstly, I plan to retrain all models using the complete GoodReads user-item interaction dataset to obtain a more realistic representation of the models' performance in a real-world application. Additionally, I would like to address the precision@K and recall@K metrics to ensure uniformity in evaluating the utility of each model, while also ensuring consistency in evaluating my implementation of LightGCN. As my understanding of the technical landscape for recommendation systems grows, I aim to expand my repertoire of models and conduct comparative analyses to gain insights into their respective strengths and weaknesses.

Project Github: `https://github.com/SamuelJon17/AML_finalProject.git`

9 Contribution

Jonathan Samuel contributed 100% of this project.

References

[1] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering. *Proceedings of the 26th International Conference on World Wide Web*, 2017. doi: 10.1145/3038912.3052569.

[2] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020. doi: 10.1145/3397271.3401063.

[3] 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.

[4] OpenAI. Chatgpt3: A large-scale generative language model. `https://chat.openai.com/`, June 2020. Accessed: May 2, 2023.

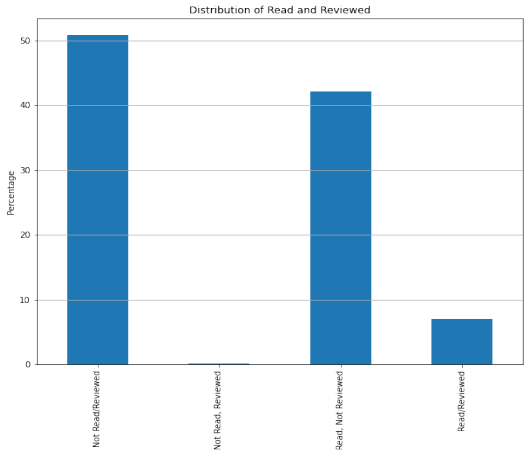
[5] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. *International Journal of Innovative Research in Computer Science & Technology*, 8(4):285–295, 2001. doi: 10.21276/ijircst.2020.8.4.2.

[6] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua. Neural collaborative filtering. *Proceedings of the 26th International Conference on World Wide Web*, 2019. doi: 10.1145/3331184.3331267.

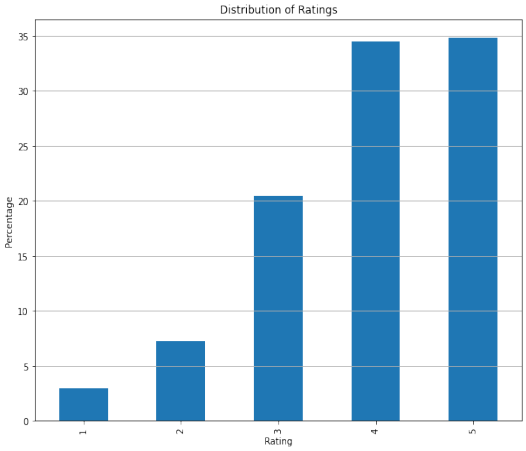
[7] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui. Bpr: Bayesian personalized ranking from implicit feedback. *25th Conference on Uncertainty in Artificial Intelligence*, 2019. doi: 10.48550/arXiv.1205.2618.

[8] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui. Graph neural networks in recommender systems: A survey. *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 423–445, 2023. doi: 10.1007/978-981-16-6054-2_19.

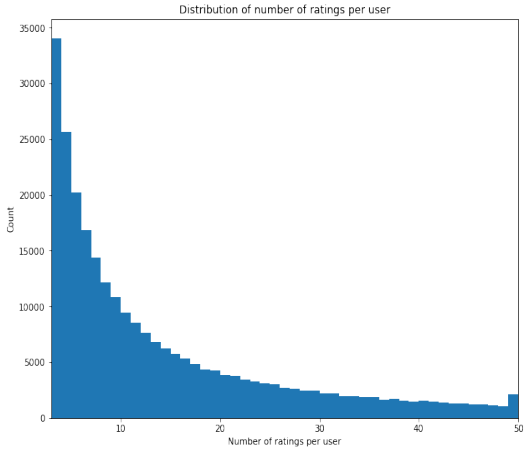
A Appendix



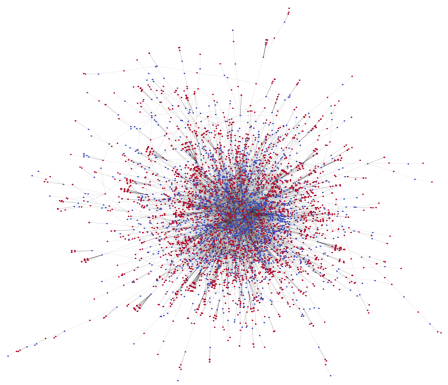
(a) Distribution of *read* & *reviewed* fields



(b) Distribution of ratings per item

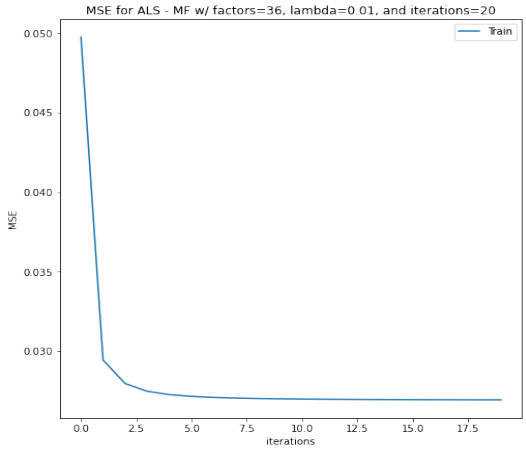


(c) Distribution of ratings per user

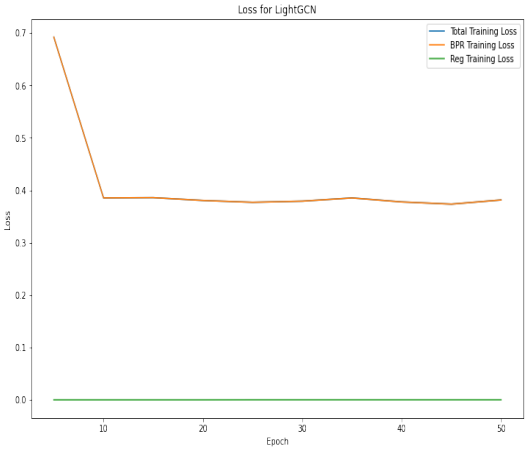


(d) 50K Subset of Bipartite Graph

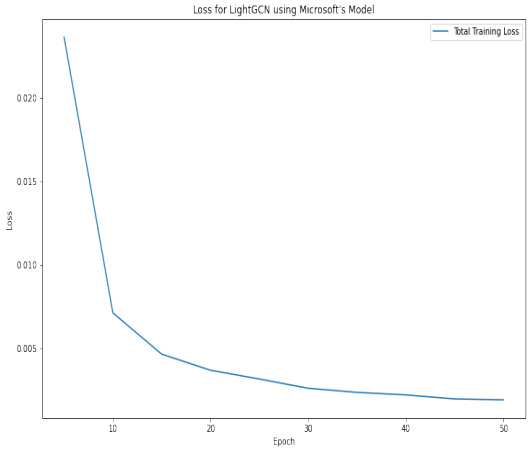
Figure 5: GoodReads dataset statistics



(a) MF-ALS-WR

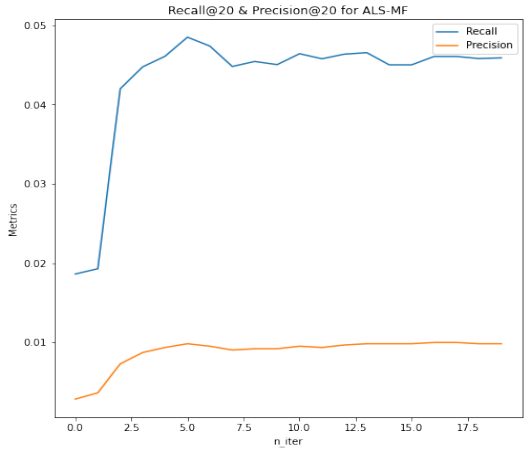


(b) LightGCN

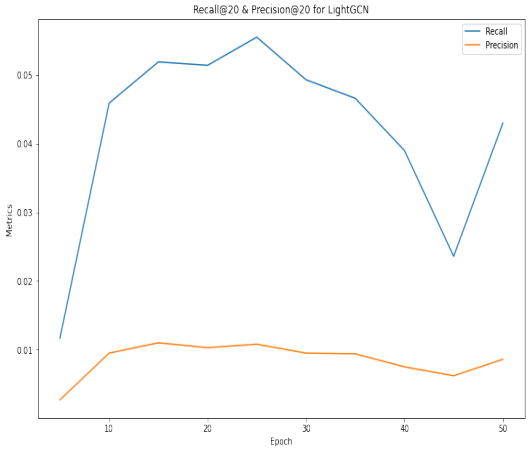


(c) LightGCN-Microsoft

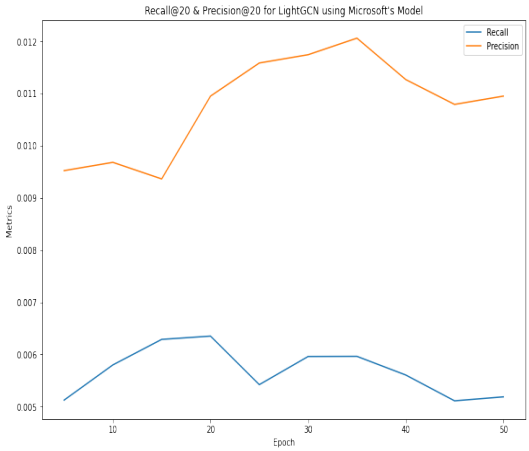
Figure 6: Training Loss for all models except NCF



(a) MF-ALS-WR



(b) LightGCN



(c) LightGCN-Microsoft

Figure 7: Evaluation for all models except NCF

	itemID	relevance_score	inTrain	inTest
0	595	5.096090	True	False
1	1194	5.048410	True	False
2	785	5.045375	True	False
3	1192	5.013458	True	False
4	1456	5.012570	True	False
5	1457	5.005933	True	False
6	1226	5.002216	True	False
7	1588	4.981640	True	False
8	2605	4.980178	True	False
9	2252	4.980178	True	False
10	2482	4.975610	True	False
11	2674	4.967356	True	False
12	2775	4.964586	True	False
13	1193	4.964586	True	False
14	2703	4.964586	True	False
15	1296	4.964586	True	False
16	1297	4.964586	True	False
17	2975	4.958096	True	False
18	2607	4.957761	True	False
19	608	4.957180	True	False

(a) MF-ALS-WR

	userID	itemID	prediction	inTrain	inTest
0	3	1275	0.999999	False	True
1	3	3071	0.999989	False	False
2	3	3323	0.999989	False	False
3	3	2844	0.999985	False	False
4	3	2858	0.999976	False	False
5	3	6439	0.999887	False	False
6	3	1343	0.999835	False	False
7	3	1549	0.999686	False	False
8	3	2530	0.999213	False	False
9	3	15130	0.998712	False	False
10	3	3239	0.998532	False	False
11	3	2220	0.997701	False	False
12	3	3312	0.997371	False	True
13	3	2888	0.997198	False	False
14	3	3399	0.993831	False	False
15	3	4090	0.992791	False	False
16	3	1360	0.991973	False	False
17	3	1133	0.990922	False	False
18	3	802	0.988050	False	False
19	3	1612	0.987822	False	False

(b) NCF

	rank	score	itemID	True?
0	0	4.344011	595	True
1	1	3.688305	267	False
2	2	3.600038	173	False
3	3	3.397644	1866	False
4	4	3.396638	672	False
5	5	3.370777	567	False
6	6	3.369359	565	False
7	7	3.349603	3536	False
8	8	3.342956	14288	False
9	9	3.319023	1179	False
10	10	3.306475	18126	False
11	11	3.292820	530	False
12	12	3.281635	675	False
13	13	3.226487	811	False
14	14	3.215848	673	False
15	15	3.208455	559	False
16	16	3.184030	2647	False
17	17	3.135139	2366	False
18	18	3.133282	60	False
19	19	3.120928	327	False

(c) LightGCN

	userID	itemID	prediction	inTrain	inTest
1340	4	3501	10.467842	True	False
1341	4	3464	9.882442	True	False
1342	4	3500	9.790209	True	False
1343	4	3485	9.519238	True	False
1344	4	3504	9.399839	True	False
1345	4	3502	8.370725	True	False
1346	4	3446	8.339659	True	False
1347	4	999	8.289755	True	False
1348	4	173	7.653675	True	False
1349	4	40	7.276745	True	False
1350	4	393	7.003430	True	False
1351	4	689	6.582014	True	False
1352	4	9011	4.712164	False	False
1353	4	4942	4.700366	False	False
1354	4	17574	4.636837	False	False
1355	4	10947	4.628959	False	False
1356	4	5478	4.596860	False	False
1357	4	8436	4.586622	False	False
1358	4	10394	4.448208	False	False
1359	4	471	4.319985	False	False

(d) LightGCN-Microsoft

Figure 8: UserID-3's Top 20 Recommendations for all models