Goodreads Recommender System

Arda Baris Basaran, Louis Yann Nicolas Duval
Electrical and Electronics Engineering Department, Neuro-X Department
EPFL
Lausanne, Switzerland

I. Introduction

Goodreads is a popular social cataloging website that allows users to search for books, write reviews, and curate personal reading lists. A notable feature of Goodreads is its capability to recommend books based on user preferences. This report explores the development of a recommender system to enhance the Goodreads user experience. By leveraging the Goodreads public dataset, which includes millions of ratings and extensive metadata, we will examine a variety of recommendation techniques. These techniques include content-based filtering, collaborative filtering, matrix factorization methods, neural collaborative filtering, and graph-based algorithms. Our study evaluates the performance of these methods using metrics such as RMSE, R-squared, MAE, and NDCG. Additionally, we consider different train/test split strategies to address real-world challenges, particularly in the context of few-shot learning.

II. Dataset

The dataset utilized for this study is derived from the Goodreads public dataset [1], encompassing a diverse array of data related to books and user interactions. This dataset includes six million ratings for the ten thousand most popular books, defined by the number of ratings they have received. It provides a comprehensive view of user-book interactions through multiple files: ‘ratings.csv‘ contains user ratings for books, with ratings ranging from one to five stars and user and book IDs ranging from 1 to 53424 and 1 to 10000, respectively.

Additional files enrich the dataset with various aspects of user behavior and book metadata. The ‘to_read.csv‘ file lists books marked as ”to read” by users, presenting nearly a million such user book pairs. The ‘books.csv‘ file offers metadata for each book, including titles, authors, average ratings, and publication years.

Graph Data Analysis : Due to computational resource constraints and the need to maintain manageable inference time, we selected 10% of the users with more than 20 ratings. This resulted in a graph with 529,887 edges and 12,396 nodes, including 5,342 user nodes and 7,834 book nodes. The edges represent the ratings given by users to books. Users have an average degree of 99.2, meaning they rate around 99 books on average, while books have an average degree of 67.6, with each book receiving 67.6 ratings on average. The user-book assortativity coefficient is -0.13, suggesting a slight tendency for highly active users to rate books that are not the most popular, while less active users tend to rate more popular books. The average degree centrality values for users and books are 0.015 and 0.010, respectively, while the graph density is 0.006 and the sparsity level is 0.012. These values highlight that only a small proportion of potential connections are made and that only a small fraction of possible user-book pairs are present. This is typical for large-scale datasets, demonstrating a high level of sparsity.

III. Methods

The recommender system is implemented using a variety of approaches, each leveraging different techniques to optimize the recommendation process. These methods include content filtering, collaborative filtering, matrix factorization techniques, and neural networks.

A. Content Filtering

Content-based filtering is a recommendation technique that leverages the intrinsic properties of items to generate recommendations. Each book in the dataset is represented by a set of descriptive features, including textual information such as titles and authors, numerical data like average ratings and rating counts, and categorical data such as genres and languages. These features are converted into feature vectors, enabling the computation of similarities between items. By analyzing these feature vectors, the system can identify and recommend books that are similar to those a user has already shown interest in, based on their content.

To achieve this, the textual features, such as book titles and authors, are tokenized and transformed into vector representations using Word2Vec, a technique for generating vector embeddings of words. Other features, like language, are one-hot encoded, and numerical attributes are scaled appropriately. Genres are treated using a MultiLabelBinarizer to create a binary matrix that indicates the presence of each genre for every book. Combining these diverse feature vectors into a single sparse matrix format allows for the computation of cosine similarity between books. We implemented two different methods for content filtering:

User Profile Vector method generates personalized book recommendations by creating a user profile based on their past ratings. This user profile is calculated by first identifying the books rated by the user from the training set and retrieving their feature vectors from a precomputed sparse matrix. These feature vectors are then weighted by the user’s ratings to compute a weighted average feature vector called the user profile vector, which effectively represents the user's preferences.

To ensure consistent similarity calculations, the user profile vector is normalized to unit length. The similarity between each book's feature vector in the user's test set and the user's profile vector is calculated and used to predict the rating for each book by weighting it with the book's average rating. This method ensures that books similar to those the user liked have a higher influence on the predicted rating.

Aggregate Similarity Scoring method for book recommendation leverages the direct relationships between the books a user has rated and the books being evaluated for recommendation. Unlike methods that construct a single user profile vector, this approach focuses on the aggregate similarity between rated books and all other books in the dataset.

In this method, the books rated by the user in the training set are first identified, and their feature vectors are retrieved from a precomputed sparse matrix. These feature vectors are used to compute cosine similarity scores between all books in the dataset and the books rated by the user in the training set. Each similarity score is then weighted by the corresponding user rating for the books in the training set, resulting in a weighted prediction score for each book.

B. Collaborative Filtering

Collaborative filtering techniques are based on the principle that users who have agreed on items in the past will continue to agree in the future. Two main types of collaborative filtering are employed in recommender systems: User-User Collaborative Filtering and Item-Item Collaborative Filtering.

User-User Collaborative Filtering identifies users with similar preferences and uses their ratings to generate recommendations. The process begins by creating a sparse user-item matrix from the user ratings data, where each row represents a user, and each column represents a book. Entries in this matrix denote the ratings given by users to books. Cosine similarity is computed based on their rating patterns to determine the similarity between users.

Once the similarity scores between all user pairs are calculated, these scores are used as weights to compute weighted sums of ratings. For a given user, the algorithm identifies the n closest neighbors—those users with the highest similarity scores. The ratings from these closest neighbors are weighted by their respective similarity scores, resulting in a weighted rating vector for the given user. This vector represents the estimated ratings for all books for the user. The predicted ratings are also normalized. This approach of using the nearest neighbors is why the algorithm is also referred to as k-nearest Neighbors (k-NN) User Collaborative Filtering.

Item-Item Collaborative Filtering works similarly to User-User Collaborative Filtering but focuses on finding similarities between items rather than users. The process begins by creating a sparse user-item matrix from the user ratings data, where each row represents a user, and each column represents a book. Entries in this matrix denote the ratings given by users to books. In this approach, instead of computing the similarity between users, the algorithm computes the similarity between items based on the users' rating patterns. Cosine similarity is used to determine how similar two books are based on how users have rated them. Once the similarity scores between all item pairs are calculated, these scores are used as weights to compute weighted sums of ratings. For a given item, the algorithm identifies the n closest neighbors—those items with the highest similarity scores. The ratings for the target item are predicted by taking into account the ratings of these similar items, weighted by their respective similarity scores. This results in a weighted rating vector for the given item, which represents the estimated ratings from all users for that item.

C. Neural Collaborative Filtering (NCF)

Neural Collaborative Filtering (NCF) [2] is a deep learning approach designed to enhance collaborative filtering by addressing the limitations of traditional matrix factorization methods. Traditional matrix factorization methods, such as MF, use the inner product to model the interaction between user and item latent features. However, this approach is limited by its linearity and inability to capture complex patterns in user-item interactions. NCF replaces the inner product with a neural network, allowing for the modeling of an arbitrary function that can capture more intricate patterns in the data.

The NCF framework employs a multi-layer perceptron to learn the interaction function between users and items. It begins with an input layer that represents users and items using one-hot encoding. These inputs are then transformed into dense vectors in the embedding layer, which serve as the latent factors for users and items. These embeddings are passed through multiple hidden layers of the neural network, which can capture various levels of abstraction and complex user-item interaction patterns. The final output layer produces a prediction score indicating the likelihood of user interaction with an item. The training process involves minimizing a pointwise loss function, often using techniques such as stochastic gradient descent. By leveraging deep learning, NCF can model non-linear relationships and provide more accurate recommendations.

D. Stochastic Gradient Descent (SGD) Based Matrix Factorization (SVD)

SGD-based matrix factorization [3] is an optimization technique that refines Singular Value Decomposition (SVD) by iteratively updating the latent factors to minimize the prediction error, thereby enhancing recommendation accuracy.

Instead of applying SVD directly to the user-item rating matrix R, we estimate user and item latent feature vectors, p_u and q_i, respectively. Initially, these vectors are randomly initialized. The predicted rating $\hat{r}_{ui}$ for user $\boldsymbol{u}$ and item i is computed as:

$$\hat{r}_{ui} = q_i^T p_u$$

where p_u corresponds to the row vectors from U and q_i corresponds to the row vectors from V. To find the optimal latent factors, we minimize the regularized squared error:

$$\min_{p_*, q_*} \sum_{(u,i)\in\kappa} \left(r_{ui} - q_i^T p_u\right)^2 + \lambda \left(\|q_i\|^2 + \|p_u\|^2\right)$$

where κ is the set of known ratings and λ is the regularization parameter. Using Stochastic Gradient Descent (SGD), we iteratively update the parameters based on the prediction error $e_{ui} = r_{ui} - q_i^T p_u$:

$$q_i \leftarrow q_i + \gamma \left(e_{ui} p_u - \lambda q_i\right)$$
$$p_u \leftarrow p_u + \gamma \left(e_{ui} q_i - \lambda p_u\right)$$

Here, γ is the learning rate, controlling the step size of the updates. Note that the algorithm given below randomly initializes q_i and p_u vectors and uses the Adam optimizer instead of traditional SGD.

E. Alternating Least Squares (ALS) Based Matrix Factorization (SVD)

Alternating Least Squares (ALS) [4] is an optimization technique commonly used for matrix factorization in collaborative filtering recommender systems. ALS is particularly well-suited for handling large-scale and sparse data, such as user-item rating matrices.

ALS begins by initializing the user and item factor matrices U and V with small random values. Each user u is represented by a k-dimensional vector p_u (a row of U), and each item i is represented by a k-dimensional vector q_i (a row of V).

The goal is to minimize the regularized squared error on the observed ratings:

$$\min_{p_*, q_*} \sum_{(u,i)\in\kappa} \left(r_{ui} - q_i^T p_u\right)^2 + \lambda \left(\|q_i\|^2 + \|p_u\|^2\right)$$

where κ is the set of known ratings, and λ is the regularization parameter.

ALS optimizes the user and item latent factors iteratively by alternating between fixing one set of factors and solving for the other:

- Fix V, Update U : When V is fixed, each user's factors p_u are updated by solving a leastsquares problem. For each user u :

$$p_u \leftarrow \left(V^T V + \lambda I\right)^{-1} V^T r_u$$

where r_u is the vector of observed ratings for user u.

- Fix U, Update V : When U is fixed, each item's factors q_i are updated by solving a least-squares problem. For each item i :

$$q_i \leftarrow \left(U^T U + \lambda I\right)^{-1} U^T r_i$$

where r_i is the vector of observed ratings for item i.

This process is repeated, alternating between updating U and V, until the changes in U and V are sufficiently small (convergence) or for a fixed number of iterations. This iterative approach allows for the efficient handling of large and sparse datasets, making ALS a robust method for matrix factorization in recommender systems.

F. Graph Based Algorithms

GraphSage (Graph Sample and Aggregate) is an inductive framework for generating node embeddings, suitable for dynamic graphs like those in book recommender systems. GraphSAGE [5] addresses the cold-start problem by generating embeddings for new users based on initial interactions. It samples and aggregates features from a node's local neighborhood using trainable aggregators (mean, LSTM, or pooling), then passes these combined features through fully connected layers with non-linear activation functions to generate the final embeddings. The model trains by optimizing a loss function that encourages similar embeddings for connected nodes using stochastic gradient descent and backpropagation. This allows GraphSAGE to generalize to unseen nodes, potentially providing high-quality recommendations for new users.

LightGCN (Light Graph Convolution Network) is tailored for collaborative filtering in recommendation systems. LightGCN [6] simplifies traditional Graph Convolution Networks by focusing on neighborhood aggregation, omitting feature transformation, and nonlinear activation to enhance performance. Each user and book node in LightGCN is initially represented by an ID embedding. The model propagates these embeddings across the user-item interaction graph through multiple layers, each aggregating features from neighboring nodes. The final embedding for a node is a weighted sum of its embeddings from all layers, capturing various levels of neighborhood information. LightGCN is efficient in handling large-scale and sparse data, providing robust recommendations even for cold-start users.

NGCF Neural Graph Collaborative Filtering integrates user-item interactions directly into the embedding process to enhance collaborative filtering. NGCF [7] captures high-order connectivity in this bipartite graph by recursively propagating and aggregating embeddings through multiple layers. Each layer refines user and book embeddings by incorporating features from their neighbors, explicitly modeling collaborative signals. The model consists of an embedding layer, multiple embedding propagation layers, and a prediction layer. The propagation layers aggregate messages from neighbors to refine node embeddings, effectively encoding the collaborative filtering effect. By stacking these layers, NGCF captures complex user-item interactions, enhancing recommendation accuracy.

IV. Results

A. Train/Test Split Configurations

To evaluate the performance of the recommender system, we experimented with various strategies for splitting the dataset into training and test sets. These configurations are designed to mimic different real-world scenarios and challenges, particularly in few-shot learning, where the amount of training data is limited.

The method used to split the ratings dataset accommodates four distinct strategies:

Random Few-Shot: In this method, a random subset of user ratings is chosen for the training set based on a specified parameter 'shot.' The randomness ensures that the training data is diverse and representative of the entire dataset. This strategy is particularly effective in few-shot learning, where training examples are limited and randomly selected to reflect the unpredictability of real-world data availability.

Highest Ratings Few-Shot: For this configuration, the top k ratings provided by each user are selected for the training set. By leveraging the most positive feedback from users, this method helps the model learn from strong user preferences. By focusing on the items users rated the highest, the model can more accurately predict other items that users are likely to rate highly, improving recommendation accuracy in few-shot scenarios.

Lowest Ratings Few-Shot: Contrary to the highest ratings split, this strategy involves selecting the lowest k ratings for the training set. This approach aids the model in understanding user dislikes, which is essential for avoiding poor recommendations. In few-shot learning, grasping negative preferences is as crucial as positive ones, ensuring a balanced recommendation system.

Percentage Split: A certain percentage of each user's ratings is allocated to the test set, with the remaining ratings used for training. This proportional split ensures that both the training and test sets are balanced and representative of the user's overall rating behavior.

By comparing the performance across these configurations, we can determine the most effective approach for building robust and accurate recommendation models.

B. *Performance Metrics*

To comprehensively assess the performance of the recommender system, we utilized several metrics for both rating prediction and ranking quality. For rating prediction, we employed Root Mean Square Error (RMSE), R-squared (R^2), and Mean Absolute Error (MAE).

To evaluate ranking performance, we used a combination of metrics: Normalized Discounted Cumulative Gain (NDCG), Precision, Recall, and Mean Average Precision (MAP). NDCG is a measure of ranking quality that accounts for the position of relevant items in the recommended list, giving higher importance to items appearing at the top.

By utilizing these metrics, we ensure a well-rounded evaluation of the recommender system's capabilities in accurately predicting user ratings and effectively ranking items according to user preferences.

V. Conclusion

A. *Interpretation of Rating and Ranking Task Results I II*

In the random few-shot configuration, SGD-SVD and Item-Item Collaborative Filtering (CF) were the top performers for rating tasks. SGD-SVD excelled at $k = 10$ and $k = 20$, demonstrating robustness in predicting ratings accurately with limited training data due to its iterative update of latent factors using stochastic gradient descent. Item-Item CF showed strong performance at $k = 5$, leveraging item relationships to generate accurate recommendations with a small number of ratings. For ranking tasks, SGD-SVD and ALS-SVD emerged as the leading methods, with ALS-SVD performing exceptionally well at higher k values due to its effective optimization of user-item interactions.

In the highest ratings few-shot configuration, Aggregate Similarity Scoring and Item-Item CF were the best for rating tasks. Aggregate Similarity Scoring achieved optimal results at $k = 10$ and $k = 20$ by utilizing book embeddings based on features like titles, authors, genres, and metadata. This method captures strong user preferences and provides accurate predictions for books a user might rate highly. Item-Item CF excelled at $k = 5$. For ranking tasks, ALS-SVD and LightGCN performed well at $k = 5$, while User-User CF surpassed them at $k = 10$ and $k = 20$, benefiting from more data to gauge user preferences accurately. Item-Item CF was highly efficient at $k = 20$ by leveraging item similarities.

In the lowest ratings few-shot configuration, Aggregate Similarity Scoring consistently outperformed other methods across all k values in rating tasks. Its success is due to its use of book embeddings, which helps avoid poor recommendations by finding similarities between book embeddings and weighting them with the user's ratings. For ranking tasks, ALS-SVD, SGD-SVD, and LightGCN were closely matched at lower k values, but ALS-SVD was the best at $k = 20$.

For the random split configuration, where 20% of the data was used for testing and 80% for training, SGD-SVD showed superior performance in rating tasks due to its ability to iteratively optimize user and item latent factors, capturing complex user-item interactions effectively. In ranking tasks, Item-Item CF performed best, leveraging substantial training data to determine item relationships accurately.

B. *General Conclusion*

Our evaluation reveals that different recommendation methods excel under different conditions. **SGD-SVD** and **ALS-SVD** show strong performance due to their effective optimization of user-item interactions in many scenarios. **Aggregate Similarity Scoring** is highly effective in the rating task with ordered few-shot scenarios with sparse data, utilizing book embeddings to accurately capture user preferences. **Item-Item Collaborative Filtering** performs well in many scenarios by leveraging item relationships. **User-User CF** shows strong performance at the highest ratings few-shot configuration, where it benefits from a larger number of ratings as it leverages the similarities between users to make recommendations. While graph-based methods like **LightGCN** and **NGCF** show promise in the ranking task, they did not outperform other methods. Further fine-tuning or improvements and larger datasets may help these methods realize better performance. We could also suppose that by making hybrid models, using feature augmentation, or creating combinations of the methods we described previously for Collaborative Filtering and Content-Based Filtering, we would be able to address both tasks and potentially yield better results with graph methods.

Appendix

TABLE I: RMSE, MAE, R^2 for Various Methods and Train/Test Split Configurations in rating prediction task

Method	**Random Few Shot**			**Highest Few Shot**			**Lowest Few Shot**			**Random Split (20% Test 80% Train)**
	k=5	**k=10**	**k=20**	**k=5**	**k=10**	**k=20**	**k=5**	**k=10**	**k=20**	
User Profile Vector	1.74, 1.49, -2.08	1.63, 1.39, -1.70	1.57, 1.33, -1.49	1.80, 1.52, -2.31	1.64, 1.38, -1.83	1.48, 1.23, -1.41	1.89, 1.63, -3.47, 0.24	1.79, 1.55, -3.63	1.78, 1.55, -4.49	1.51, 1.28, -1.39
Aggregate Similarity Scoring	1.22, 0.97, -0.50	1.18, 0.95, -0.42	1.17, 0.93, -0.38	1.28, 1.02, -0.68	**1.22, 0.97, -0.56**	**1.18, 0.94, -0.53**	**1.25, 1.02, -0.97**	**1.17, 0.95, -0.98**	**1.12, 0.90, -1.16**	1.14, 0.91, -0.35
User-User CF	3.44, 3.40, -46.47	3.47, 3.42, -47.14	3.47, 3.41, -47.12	3.68, 3.64, -53.13	4.06, 4.03, -64.84	4.30, 4.27, -73.03	2.98, 2.90, -34.45	2.43, 2.35, -22.66	2.34, 2.27, -20.94	3.44, 3.38, -46.44
Item-Item CF	**1.04, 0.79, -0.10**	1.10, 0.81, -0.22	1.13, 0.84, -0.31	**1.16, 0.85, -0.38**	1.35, 1.04, -0.90	1.50, 1.19, -1.43	1.43, 1.13, -1.52	1.74, 1.46, -3.31	1.76, 1.52, -4.39	1.05, 0.79, -0.13
NCF	1.06, 0.83, -0.14	1.06, 0.82, -0.13	1.06, 0.83, -0.15	1.35, 1.02, -0.87	1.43, 1.10, -1.12	1.48, 1.17, -1.38	2.14, 1.91, -4.69	1.96, 1.74, -4.49	1.77, 1.56, -4.45	0.96, 0.73, 0.06
SGD-SVD	1.27, 1.01, -0.64	**1.04, 0.81, -0.09**	**0.91, 0.70, 0.17**	1.25, 0.94, -0.59	1.30, 1.00, -0.77	1.42, 1.12, -1.20	2.23, 2.05, -5.14	2.02, 1.84, -4.81	1.73, 1.54, -4.18	**0.84, 0.65, 0.28**
ALS-SVD	2.67, 2.40, -6.20	2.36, 2.04, -4.65	1.71, 1.36, -1.97	1.79, 1.43, -2.27	1.58, 1.24, -1.60	1.30, 1.02, -0.83	2.65, 2.42, -7.70	2.67, 2.45, -9.12	2.29, 2.04, -8.07	1.10, 0.82, -0.24

TABLE II: NDCG, Precision, Recall, and Map for Various Methods and Train/Test Split Configurations in ranking task

Method	**Random Few Shot**			**Highest Few Shot**			**Lowest Few Shot**			**Random Split (20% Test 80% Train)**
	k=5	**k=10**	**k=20**	**k=5**	**k=10**	**k=20**	**k=5**	**k=10**	**k=20**	
User Profile Vector	0.26, 0.08, 0.01, 0.19	0.29, 0.09, 0.01, 0.20	0.29, 0.09, 0.01, 0.21	0.31, 0.10, 0.01, 0.24	0.29, 0.08, 0.01, 0.22	0.23, 0.06, 0.01, 0.16	0.24, 0.07, 0.01, 0.18	0.27, 0.08, 0.01, 0.20	0.25, 0.07, 0.01, 0.19	0.12, 0.03, 0.01, 0.08
Aggregate Similarity Scoring	0.26, 0.08, 0.01, 0.19	0.28, 0.08, 0.01, 0.20	0.28, 0.08, 0.01, 0.20	0.31, 0.09, 0.01, 0.24	0.30, 0.08, 0.01, 0.23	0.23, 0.06, 0.01, 0.17	0.23, 0.06, 0.01, 0.17	0.27, 0.08, 0.01, 0.20	0.23, 0.07, 0.01, 0.17	0.12, 0.03, 0.01, 0.08
User-User CF	0.35, 0.11, 0.01, 0.25	0.57, 0.23, 0.03, 0.44	0.67, 0.33, 0.05, 0.53	0.52, 0.21, 0.03, 0.41	**0.66, 0.33, 0.05, 0.53**	**0.66, 0.35, 0.05, 0.53**	0.04, 0.01, 0.00, 0.03	0.30, 0.11, 0.02, 0.21	0.60, 0.29, 0.05, 0.47	0.61, 0.35, 0.05, 0.53
Item-Item CF	0.23, 0.08, 0.01, 0.16	0.23, 0.08, 0.01, 0.20	0.42, 0.18, 0.03, 0.32	0.32, 0.12, 0.02, 0.23	0.57, 0.27, 0.03, 0.45	**0.66, 0.37, 0.05, 0.54**	0.10, 0.03, 0.01, 0.06	0.26, 0.09, 0.01, 0.21	0.50, 0.23, 0.03, 0.39	**0.65, 0.37, 0.05, 0.52**
NCF	0.11, 0.03, 0.00, 0.07	0.16, 0.06, 0.01, 0.10	0.26, 0.11, 0.01, 0.17	0.22, 0.07, 0.01, 0.15	0.33, 0.14, 0.02, 0.23	0.30, 0.13, 0.02, 0.21	0.13, 0.04, 0.00, 0.09	0.25, 0.09, 0.01, 0.17	0.32, 0.14, 0.02, 0.22	0.44, 0.16, 0.08, 0.32
SGD-SVD	**0.49**, **0.21**, **0.02**, **0.37**	0.60, 0.33, 0.04, 0.47	0.66, 0.37, 0.05, 0.53	0.51, 0.23, 0.03, 0.40	0.60, 0.33, 0.04, 0.47	0.55, 0.29, 0.04, 0.43	**0.32, 0.12, 0.01, 0.23**	**0.49, 0.23, 0.03, 0.36**	0.53, 0.27, 0.04, 0.39	0.48, 0.18, 0.09, 0.36
ALS-SVD	0.47, 0.20, 0.02, 0.36	**0.68**, **0.38**, **0.05**, **0.54**	**0.71**, **0.44**, **0.06**, **0.58**	**0.53, 0.25, 0.03, 0.42**	0.63, 0.34, 0.04, 0.50	0.61, 0.33, 0.05, 0.48	0.17, 0.05, 0.01, 0.12	**0.50, 0.23, 0.03, 0.38**	**0.64, 0.35, 0.05, 0.50**	0.48, 0.18, 0.09, 0.35
GraphSage	0.33, 0.13, 0.014, 0.25	0.50, 0.26, 0.031, 0.38	0.56, 0.28, 0.037, 0.42	0.36, 0.13, 0.016, 0.27	0.48, 0.23, 0.027, 0.37	0.49, 0.24, 0.030, 0.37	0.25, 0.09, 0.010, 0.18	0.38, 0.17, 0.021, 0.28	0.44, 0.21, 0.027, 0.32	0.40, 0.14, 0.069, 0.29
LightGCN	0.45, 0.21, 0.024, 0.34	0.62, 0.35, 0.042, 0.49	0.60, 0.34, 0.045, 0.48	**0.53, 0.24, 0.030, 0.42**	0.58, 0.31, 0.038, 0.46	0.53, 0.27, 0.036, 0.41	**0.32, 0.11, 0.014, 0.24**	0.47, 0.22, 0.027, 0.34	0.47, 0.22, 0.030, 0.34	0.46, 0.16, 0.081, 0.35
NGCF	0.30, 0.11, 0.014, 0.21	0.52, 0.25, 0.031, 0.39	0.63, 0.34, 0.047, 0.49	0.30, 0.11, 0.014, 0.21	0.47, 0.22, 0.028, 0.35	0.57, 0.28, 0.039, 0.43	0.23, 0.09, 0.010, 0.16	0.44, 0.20, 0.025, 0.32	0.56, .28, 0.038, 0.42	0.47, 0.17, 0.087, 0.35

References

[1] Z. Zajac, "Goodbooks-10k: a new dataset for book recommendations," *FastML*, 2017.

[2] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural collaborative filtering," *CoRR*, vol. abs/1708.05031, 2017. [Online]. Available: http://arxiv.org/abs/1708.05031

[3] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[4] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," 12 2008, pp. 263–272.

[5] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017. [Online]. Available: http://arxiv.org/abs/1706.02216

[6] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," 2020. [Online]. Available: http://arxiv.org/abs/2002.02126

[7] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," 2019. [Online]. Available: http://arxiv.org/abs/1905.08108