Assessing Recommender System Architectures: A Comparative Study of Collaborative Filtering, Graph-Based, and Hypergraph-Based Models

A Major Research Paper
presented to the Toronto Metropolitan University

in partial fulfillment of the requirements for the degree of
Master of Science in Data Science and Analytics
Department of Engineering

MRP Supervisor:
Dr. Pawel Pralat

Second Reader:
Dr. Ceni Babaoglu

Candidate:
Nadia Salou Doudou,
BSc Computer Science, University of North Carolina at Greensboro, 2021

2023 - 2024

Author's Declaration for Electronic Submission of a Major Research Project (MRP)

Abstract

Recommender systems play a critical role in enhancing user experience on digital platforms by predicting user preferences. This study undertakes a comparative analysis of three prominent models: Conventional Collaborative Filtering (CF), Graph-based models, and Hypergraph-based models. Each model presents unique strengths in terms of accuracy, computational complexity, scalability, and the ability to handle data sparsity. By thoroughly evaluating these aspects, this research aims to identify the optimal recommender system architecture for varied operational environments. The findings provide valuable insights into the strengths and weaknesses of each model, guiding the development of effective, scalable, and robust recommendation systems that significantly enhance digital services.

Keywords:
Recommender Systems, Collaborative Filtering, Graph Convolutional Network, Graph Attention Network, GraphSAGE, HyperGCN, Node2Vec

Acknowledgments

I would like to express my profound gratitude to my MRP supervisor, Dr. Pawel Pralat, for his unwavering support, insightful guidance, and constant encouragement throughout the duration of this research project. His expertise and advice have been invaluable, and his dedication to my academic and professional growth has greatly contributed to the successful completion of this work.

Thank you,
Professor Pawel Pralat

TABLE OF CONTENTS

List of Tables

List of Figures

Introduction

This document provides a comprehensive overview of my Major Research Project (MRP) titled “Assessing Recommender System Architectures: A Comparative Study of Collaborative Filtering, Graph-Based, and Hypergraph-Based Models.” The study commences with an introduction to the domain, topic, and datasets involved, followed by a precise articulation of the research problem and question. A thorough literature review is then presented, alongside an extensive exploratory analysis of the datasets. The subsequent sections detail the methodologies and experimental frameworks employed to compare the different recommender system architectures. The findings from these experiments are meticulously analyzed and discussed. The paper concludes with strategic recommendations for future research endeavors in this domain.

1.1 Background

In recent years, recommender systems have become integral to enhancing user experiences across various digital platforms, with significant applications in the entertainment industry. This research focuses on movie recommendation systems, utilizing the MovieLens 100k dataset, which contains 100,000 ratings from 943 users on 1,682 movies. The MovieLens dataset is a benchmark in the field, providing a robust foundation for evaluating different recommender system architectures. By leveraging this dataset, the study aims to assess and compare the efficacy of collaborative filtering, graph-based, and hypergraph-based models in generating accurate and personalized movie recommendations. This background sets the stage for a comprehensive exploration of the strengths and limitations of these models, contributing to the advancement of recommender system technologies.

1.2 Research Questions

The objective of this study is to determine which recommender system architecture collaborative filtering, graph-based systems, or hypergraph-based systems delivers the most accurate and relevant movie recommendations by evaluating their performance using MSE, RMSE, MAE, Precision@10, and Recall@10 metrics on the MovieLens 100k dataset. Additionally, this research aims to investigate how effectively each model addresses the cold-start problem with new users or items, and to assess which model delivers the most reliable recommendations when faced with sparse data environments. This comprehensive evaluation will provide insights into the optimal design and implementation of recommendation systems in practical applications.

Literature Review

In the rapidly evolving field of recommender systems, several innovative approaches have been developed to enhance the precision and relevance of recommendations across various domains. This literature review synthesizes findings from fifteen recent studies, beginning with advancements in collaborative filtering, progressing to graph-based systems, and culminating in hypergraph-based methods.

2.1 Conventional Collaborative Filtering

Collaborative filtering (CF) has long been a cornerstone of recommender systems, renowned for its simplicity and effectiveness in leveraging user-item interaction data to provide personalized recommendations. One significant enhancement to traditional matrix factorization techniques is the incorporation of temporal features, which address the static nature of earlier models. By integrating time-dependent user and item features, these improved models adapt to changing user preferences and item popularity over time. For instance, a study on the Movielens dataset demonstrated that this temporal adaptation led to an improvement of over 1% in recommendation accuracy compared to static models [1]. This incremental enhancement, though seemingly modest, can translate into substantial improvements in user satisfaction and engagement over time.

Further refining CF approaches, another framework leverages social networks to boost the precision and relevance of recommendations. By combining user ratings with social connections, this method calculates user similarity through a weighted combination of these factors [2]. This approach not only enhances accuracy but also provides more diverse recommendations and effectively tackles cold-start and sparsity issues. Evaluations on movie rating datasets confirmed its superiority over traditional CF methods [2]. The integration of social network data allows the system to better understand user preferences through their social circles, thus providing recommendations that are not only relevant but also socially validated.

In addition to these approaches, another significant development in CF is the hybrid recommendation system, which combines content-based filtering with collaborative filtering using artificial neural networks. This system leverages the strengths of both methods to enhance recommendation accuracy. The hybrid model utilizes Singular Value Decomposition (SVD) for collaborative filtering and TF-IDF algorithms for content-based filtering, integrating these with a neural network model to refine recommendations further. Evaluations demonstrated that the hybrid system outperformed traditional CF methods, especially in terms of precision and recall [5].

2.2 Graph Based Systems

Graph neural networks (GNNs) represent a significant shift from traditional CF techniques, offering enhanced capabilities for modeling complex user-item interactions. A notable application of GNNs in recommendation systems is link prediction, where social networks are represented as graphs, and GNNs are employed to predict future links between users and items. Studies using datasets from platforms like Facebook, Twitter, and Google+ have shown that GNNs can achieve high accuracy in predicting user preferences [3]. The careful tuning of hyperparameters and feature selection further boosts performance, making GNNs a powerful tool for modern recommendation systems. These models leverage the structural properties of graphs to uncover latent connections and patterns that traditional methods might miss.

A collaborative filtering movie recommendation system based on Graph Neural Networks (GNNs) has also been proposed, focusing on improving recommendation accuracy through graph representation. This system models user-item interactions as graphs and employs link prediction techniques within these graphs to suggest relevant items to users. Results demonstrated that the model achieved high precision and recall, highlighting its effectiveness in capturing user behavior patterns [4].

A study using LightGCN demonstrated that aggregating information from neighboring nodes in a graph significantly improves the quality of user and item embeddings [4]. This approach not only captures direct user-item interactions but also the influence of related users and items, leading to more nuanced recommendations. The results showed remarkable stability and improvement in precision and recall with increased training iterations, underscoring the robustness of this approach [4].

Another innovative graph-based method incorporates social network analysis to enrich recommendation models. This approach combines collaborative filtering with social network data to capture complex user-item relationships [2]. The integration of these data sources enables the model to provide more personalized recommendations by considering the influence of social connections on user preferences [2]. By understanding the social dynamics and leveraging them in the recommendation process, these models can offer suggestions that are not only relevant but also socially informed, enhancing user trust and engagement.

The utilization of knowledge graphs in recommendation systems represents another leap forward. By embedding knowledge graphs into low-dimensional spaces, these systems can capture semantic relationships between entities, such as users, items, and attributes. One study introduced a knowledge graph-based recommendation system enhanced by neural collaborative filtering, which fine-tunes pre-trained embeddings without adding extra layer weights [9]. This method demonstrated substantial improvements in recall, precision, and F1-score compared to state-of-the-art approaches, highlighting the efficacy of integrating knowledge graph embeddings with collaborative filtering techniques [9]. The use of knowledge graphs allows for the incorporation of rich contextual information, which can significantly enhance the quality of recommendations by providing deeper insights into user preferences and item characteristics.

2.3 Hypergraph Based Systems

Hypergraph-based systems take the complexity of graph-based methods to a new level by modeling high-order relationships between users, items, and additional attributes. These systems connect multiple vertices through hyperedges, allowing for the representation of more complex interactions. One prominent hypergraph-based method combines predictions from various models to enhance recommendation accuracy. By using hypergraph ranking, this ensemble approach outperforms individual models and weighted hybrid methods, as evidenced by experiments on datasets from domains such as movies, music, and news [11]. The ability to model high-order interactions allows these systems to capture more nuanced relationships and dependencies, leading to more accurate and diverse recommendations.

A particularly innovative study proposed a multi-objective recommendation framework using a User-Item-Attribute-Context (UIAC) data model and improved hypergraph ranking. This model effectively handles diverse user requirements and improves recommendation accuracy for both single and multi-objective recommendations [14]. Empirical tests with Yelp data demonstrated the model's superior performance, suggesting its potential for broader applications in e-commerce and other fields [14].

Hypergraph-based systems excel in handling complex recommendation scenarios that involve multiple types of relationships and interactions. For instance, by modeling interactions through hyperedges, these systems can consider user-item-context relations simultaneously, offering a more holistic approach to recommendation. This capability is particularly useful in domains where user preferences are influenced by multiple factors, such as location, time, and social context. The flexibility of hypergraph-based models makes them well-suited for applications requiring nuanced and multi-dimensional recommendations.

Exploratory Data Analysis (EDA)

In the exploratory data analysis (EDA) conducted, we assessed the structure and completeness of multiple datasets that collectively form the movie dataset used in this project.

3.1 Data Acquisition

The datasets used in this project were obtained from MovieLens, accessible at grouplens.org/datasets/movielens and adhere to the standards required for the MRP, being both open-source and compliant.

3.2 Data Source and Data Files

The datasets used in this project, specifically 'Links', 'Movies', 'Ratings', and 'Tags', were obtained as CSV files from the MovieLens website, hosted by grouplens.org. These files provide comprehensive data that is essential for developing and testing the movie recommendation system.

3.3 Data Structure Analysis

Field	**Description**
movieId	A unique identifier for each movie within the dataset. This ID is used to link and reference movies across different tables or files in the database.
imdbId	The unique identifier corresponding to the Internet Movie Database (IMDB) entry for the movie. This ID can be used to fetch more detailed information about the movie from IMDB.
tmdbId	The unique identifier corresponding to The Movie Database (TMDb) entry for the movie. Similar to the IMDB ID, this ID can be used to access detailed movie data from TMDb.
Title	The official title of the movie as recognized in media databases.
Genre	A list of genres associated with the movie. Genres describe the movie's category or style, such as Comedy, Drama, Thriller, etc.
userId	A unique identifier for each user who has rated or tagged movies in the dataset.
rating	A numerical value assigned by a user to a movie, which typically ranges on a scale from 1 to 5. This rating reflects the user's liking or disliking toward the movie.
tag	A user-generated label or keyword assigned to a movie. Tags help describe the movie beyond its official genres.
timestamp	The specific date and time when a rating or tag was assigned by a user. This field is stored as a UNIX timestamp.

Table 3.1: Data Structure Analysis

Datasets Structure

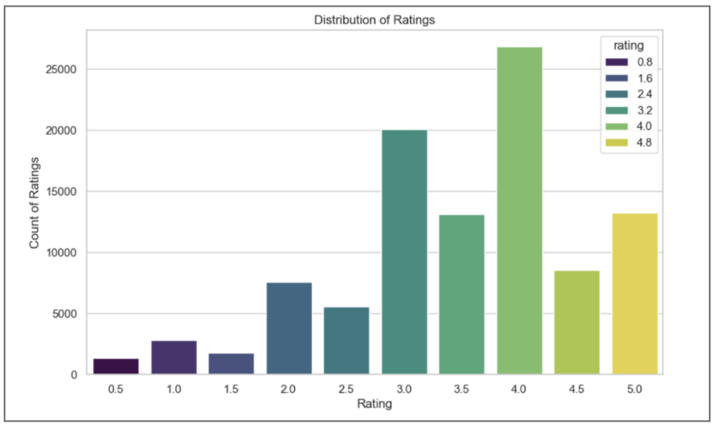
- **Links Dataset**: 9,742 entries, 3 columns.
- **Movies Dataset**: 9,742 entries, 3 columns.
- **Ratings Dataset**: 100,836 entries, 4 columns.
- **Tags Dataset**: 3,683 entries, 4 columns.

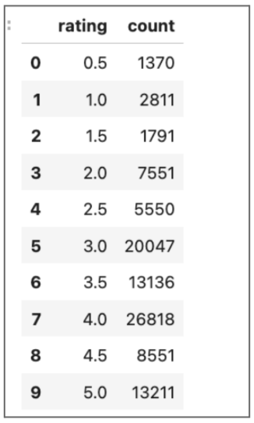
Missing Value Analysis

- **Links Dataset**: 8 missing values in ‘tmdbId’.
- **Movies Dataset**: No missing values.
- **Ratings Dataset**: No missing values.
- **Tags Dataset**: No missing values.

3.4 Data Analysis and Statistics

Figure 3.1 displays the distribution of ratings from 0.5 to 5.0, where the most common ratings are 4.0 and 3.0, indicating a generally positive response. Ratings of 0.5 are the least common, showing fewer low evaluations.





	rating	count
0	0.5	1370
1	1.0	2811
2	1.5	1791
3	2.0	7551
4	2.5	5550
5	3.0	20047
6	3.5	13136
7	4.0	26818
8	4.5	8551
9	5.0	13211

Figure 3.1: Ratings Distribution

Figure 3.2 shows that the most common average ratings for movies are around 3.0 and 4.0, indicating that most movies are rated as “good” by users. The distribution is slightly right-skewed, with fewer movies having very low or very high average ratings. This suggests that users generally rate movies favorably, with a positive bias towards higher ratings. The bimodal peaks at 3.0-3.5 and 4.0-4.5 highlight distinct groups of movies rated differently, reflecting varying levels of quality or popularity. The histogram uses a bin size of 0.167. Which is appropriate because it provides a good balance between detail and clarity, allowing us to observe meaningful patterns without introducing excessive noise. The additional line drawn represents a kernel density estimate, which smooths the distribution to better highlight the underlying pattern in the data.

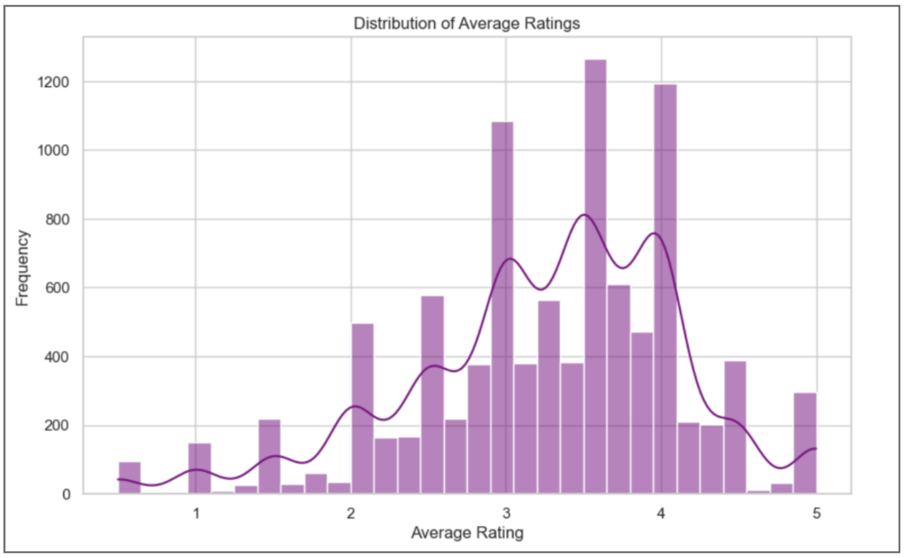
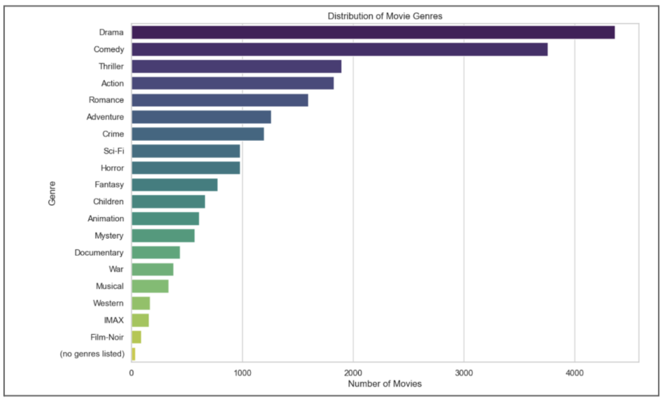
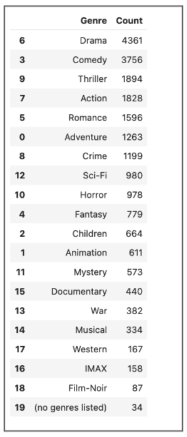


Figure 3.2: Average Ratings Distribution

Figure 3.3 shows the distribution of movie genres based on the number of movies in each genre. Drama is the most prevalent genre with 4,361 movies, followed by Comedy with 3,756 movies. Thriller and Action genres also have significant representations. On the lower end, genres like Western, IMAX, and Film-Noir have fewer movies, indicating less popularity or production volume in these categories. This distribution highlights the dominance of Drama and Comedy in the dataset.





	Genre	Count
6	Drama	4361
3	Comedy	3756
9	Thriller	1894
7	Action	1828
5	Romance	1596
0	Adventure	1263
8	Crime	1199
12	Sci-Fi	980
10	Horror	978
4	Fantasy	779
2	Children	664
1	Animation	611
11	Mystery	573
15	Documentary	440
13	War	382
14	Musical	334
17	Western	167
16	IMAX	158
18	Film-Noir	87
19	(no genres listed)	34

Figure 3.3: Movies Genre Distribution

In Figure 3.4, the word cloud captures a wide array of tags used to describe content, with words like “thought-provoking,” “superhero,” and “surreal” appearing prominently. The varied sizes indicate the relative frequency of each tag, with larger words being more common. This suggests a broad range of genres and themes, from “crime” to “Disney,” reflecting diverse viewer interests.

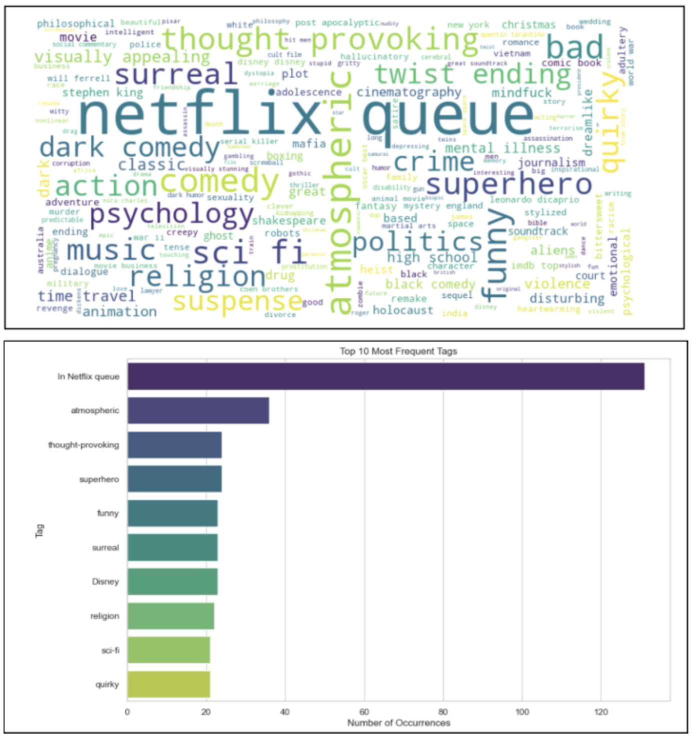
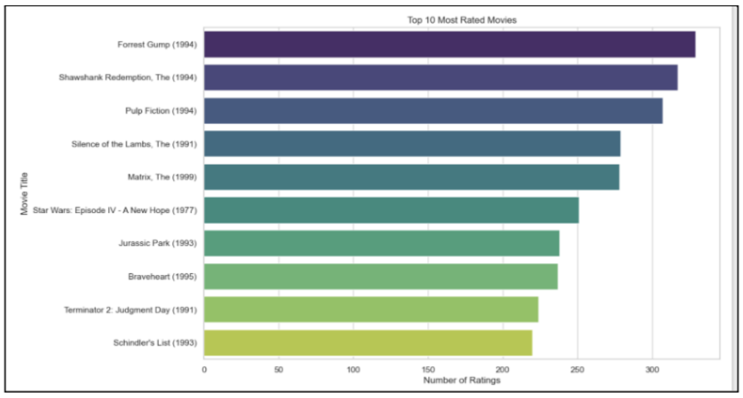
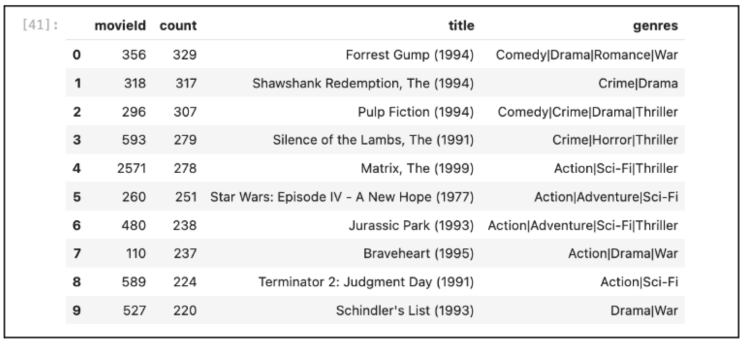


Figure 3.4: Word Cloud of Movie Tags

Figure 3.5 shows the number of ratings each movie has received. “Forrest Gump” (1994) leads with the highest number of ratings, followed closely by “The Shawshank Redemption” (1994) and “Pulp Fiction” (1994). Additionally, This aligns with earlier observations that drama is the most frequent movie genre overall as we can see, the top movies include drama as one of their genres.





[41]:

	movieId	count	title	genres
0	356	329	Forrest Gump (1994)	Comedy|Drama|Romance|War
1	318	317	Shawshank Redemption, The (1994)	Crime|Drama
2	296	307	Pulp Fiction (1994)	Comedy|Crime|Drama|Thriller
3	593	279	Silence of the Lambs, The (1991)	Crime|Horror|Thriller
4	2571	278	Matrix, The (1999)	Action|Sci-Fi|Thriller
5	260	251	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi
6	480	238	Jurassic Park (1993)	Action|Adventure|Sci-Fi|Thriller
7	110	237	Braveheart (1995)	Action|Drama|War
8	589	224	Terminator 2: Judgment Day (1991)	Action|Sci-Fi
9	527	220	Schindler's List (1993)	Drama|War

Figure 3.5: Top 10 Most Rated Movies

Figure 3.6 shows the top 10 movies with the highest average ratings, all of which have received at least 50 ratings. The movies, including “The Shawshank Redemption”, “The Godfather”, and “Fight Club”, span several decades and various genres, reflecting a broad consensus on their quality. The consistently high ratings indicate these films are widely regarded as some of the best in cinematic history.

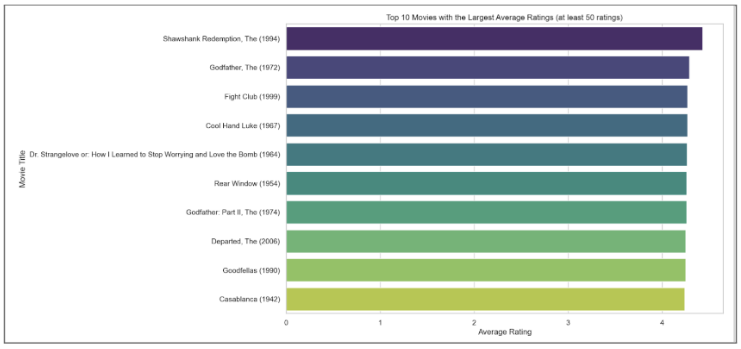


Figure 3.6: Top 10 Movies With Highest Average Rating

Figure 3.7 illustrates the distribution of the number of ratings provided by users, with the x-axis representing the number of ratings and the y-axis indicating the frequency of users. The distribution is heavily right-skewed, revealing that the majority of users contribute a relatively small number of ratings, predominantly fewer than 500. A notable peak is observed around users who have given fewer than 100 ratings. Although there are outliers with up to 2500 ratings, they are relatively few, and the frequency sharply declines as the number of ratings increases. This indicates that while a small fraction of users are highly active, most users are less engaged. The extra line shown is a kernel density estimate, which smooths out the distribution to more clearly reveal the underlying trends in the data.

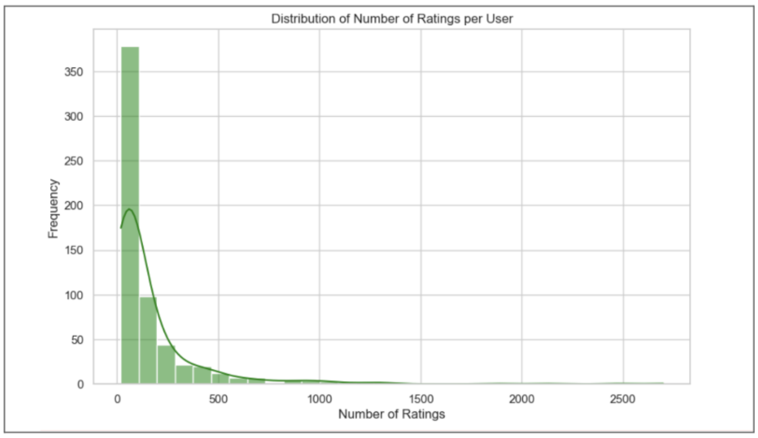


Figure 3.7: Distribution of Number of Rating per User

Figure 3.8 shows the distribution of the average rating given by each user, where the x-axis denotes the average rating and the y-axis represents the frequency of users with that average rating. This distribution closely resembles a normal distribution, centered around an average rating of approximately 3.5 to 4.0, suggesting that most users tend to give ratings on the higher end of the scale. The slight skewness towards higher ratings points to a possible leniency or positivity bias in user evaluations. Similar to the previous figures, the additional line represents a kernel density estimate, which smooths the distribution to more effectively highlight the underlying patterns in the data.

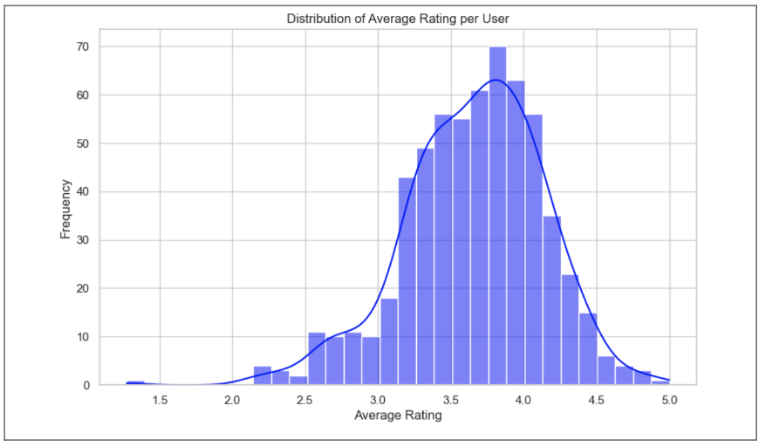


Figure 3.8: Distribution of Average Rating per User

Figure 3.9 is highlighting the top 10 most active users based on the number of ratings they have provided, with user IDs on the x-axis and the number of ratings on the y-axis. User 414 emerges as the most active user, having provided over 2500 ratings. There is a noticeable drop in activity from user 414 to user 599, demonstrating a steep decline even among the top 10 users. This high level of activity is consistent with the average ratings centered around 3.5 to 4.0, indicating that the most active users are also likely to be giving favorable ratings. This consistency underscores the need for balancing the influence of these active users in the recommendation system to ensure a representative and fair output.

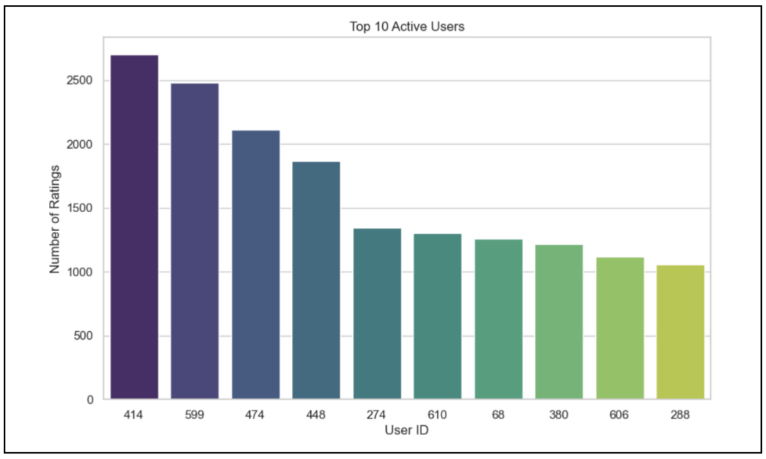
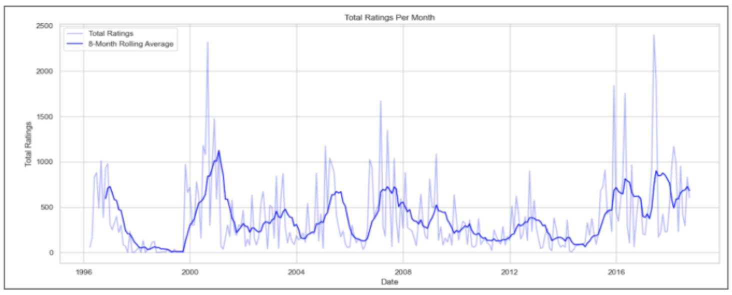


Figure 3.9: Top 10 Most Active Users

In Figure 3.10, the first chart, “Total Ratings Per Month”, shows fluctuations in user activity from 1996 to 2017, with notable peaks around 2000 and 2016, reaching over 2000 ratings in some months. The 8-month rolling average smooths these variations, indicating cyclical trends in user engagement.
The second chart, “Average Rating Per Month”, depicts the average rating stability over the same period. Despite individual rating variations between 2.75 and 4.5, the 8-month rolling average shows a consistent trend around 3.5 to 4.0, highlighting overall rating consistency.



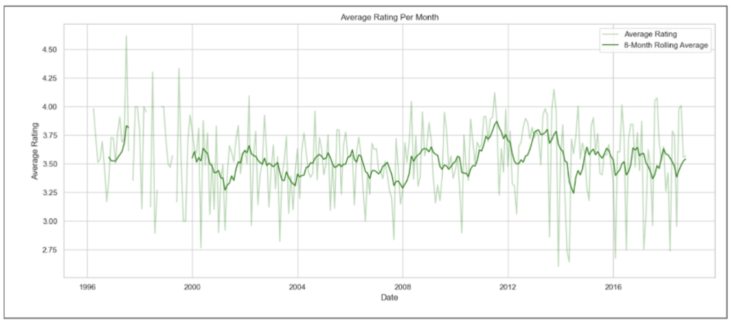


Figure 3.10: Total Ratings per Month & Average Ratings per Month

Overview of implemented Algorithms

4.1 Collaborative Filtering Recommender Systems

4.1.1 KNNBasic (K-Nearest Neighbors)

The KNNBasic algorithm leverages the k-nearest neighbors technique to predict user ratings based on the weighted average of ratings from similar users or items. This approach involves several key steps:

1. **Similarity Calculation**:

   Calculate the similarity between users (or items) using various similarity measures. The choice of similarity measure can significantly impact the performance of the algorithm. There are several similarity measures but in this paper we focus on three similarity measures, described as follows:

   - **Pearson Similarity**:
     This measure computes the linear correlation between two vectors, accounting for differences in their rating scales. It is useful for identifying the strength and direction of a linear relationship between user or item ratings. The formula for Pearson similarity between two vectors A and B is:

     $$\text{Pearson Similarity} = \frac{\sum(A_i - \bar{A})(B_i - \bar{B})}{\sqrt{\sum(A_i - \bar{A})^2}\sqrt{\sum(B_i - \bar{B})^2}}$$

     where $\bar{A}$ is the mean of the elements in vector A and $\bar{B}$ is the mean of the elements in vector B.
   - **Pearson Baseline Similarity**:
     This measure extends Pearson similarity by normalizing the ratings based on user and item biases. It adjusts for user preferences and item popularity, providing a more accurate similarity measure. The formula for Pearson baseline similarity between two vectors A and B is:

     $$\text{Pearson Baseline Similarity} = \frac{\sum(A_i - b_{A_i})(B_i - b_{B_i})}{\sqrt{\sum(A_i - b_{A_i})^2}\sqrt{\sum(B_i - b_{B_i})^2}}$$

     where b_{A_i} and b_{B_i} are the baseline estimates for user and item biases.
   - **Mean Squared Difference (MSD)**:
     This measure calculates the average squared difference between corresponding ratings, highlighting the dissimilarity between two vectors. It is useful for emphasizing larger differences. The formula for MSD between two vectors A and B of length n is:

     $$\text{MSD} = \frac{1}{n}\sum(A_i - B_i)^2$$

     where n is the number of elements (or ratings in this case) in each vector, A_i is the i-th rating in vector A, and B_i is the i-th rating in vector B.

2. **Finding Nearest Neighbors**:

   - Identify the k-nearest neighbors for each user or item based on the computed similarity scores. The value of k is a hyperparameter that can be tuned to optimize performance.

3. **Prediction**:

- Predict the rating for a target user-item pair by computing the weighted average of ratings from the k-nearest neighbors. The weights are determined by the similarity scores, giving higher importance to more similar neighbors.

4.1.2 Singular Value Decomposition (SVD)

SVD is a matrix factorization technique that decomposes the user-item rating matrix into latent factors, enabling the prediction of ratings through these latent factors. The algorithm's process is broken down into the following steps:

1. **Matrix Decomposition**:
   - Decompose the user-item rating matrix into three matrices: U (user factors), Σ (diagonal matrix of singular values), and V^T (item factors). This step transforms the original high-dimensional data into lower-dimensional latent factors.
2. **Latent Factor Calculation**:
   - Utilize the decomposed matrices to derive latent factors for both users and items. These latent factors capture underlying patterns and relationships in the data that are not immediately apparent.
3. **Rating Prediction**:
   - Reconstruct the user-item rating matrix by multiplying the decomposed matrices (U, Σ, and V^T). The predicted ratings are then obtained by computing the dot product of the user and item latent factors for each user-item pair.

4.1.3 CoClustering

CoClustering: CoClustering simultaneously clusters users and items to uncover hidden relationships in the data, facilitating more accurate rating predictions. The algorithm involves several critical steps:

1. **Initialization and Clustering**:
   - Initialize the clusters for users and items. This step involves partitioning the user-item rating matrix into co-clusters, where each co-cluster represents a subset of users and items that share similar rating patterns.
2. **Simultaneous Clustering**:
   - Perform iterative co-clustering to refine the user and item clusters. During this process, users and items are reassigned to different clusters to maximize the similarity within each co-cluster.
3. **Prediction**:
   - Predict ratings by averaging the observed ratings within the corresponding co-cluster. This method leverages the inherent relationships discovered during the clustering process to make informed predictions.

4.2 Graph-based Recommender Systems

4.2.1 LightGCN

The basic idea of GCN is to learn representation for nodes by smoothing features over the graph. To achieve this, it performs graph convolution iteratively, i.e., aggregating the features of neighbors as the new representation of a target node. Such neighborhood aggregation can be abstracted as:

$$\mathbf{e}_u^{(k+1)} = \mathrm{AGG}(\mathbf{e}_u^{(k)}, \{\mathbf{e}_i^{(k)} : i \in \mathcal{N}_u\}).$$

The AGG is an aggregation function — the core of graph convolution- that considers the k-th layer's representation of the target node and its neighbor nodes, according to He et al., 2020. However, in LightGCN, only the normalized sum of neighbor embeddings is performed towards next layer; other operations like self-connection, feature transformation, and nonlinear activation are all removed, which largely simplifies GCNs [15]. In Layer Combination, we sum over the embeddings at each layer to obtain the final representations.

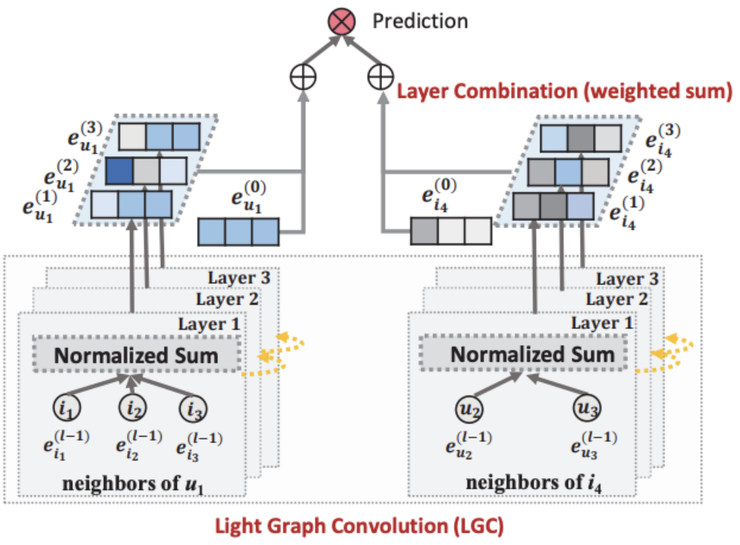


Figure 4.1: LightGCN Architecture (Adapted from He et al., 2020 [15])

4.2.2 Graph Attention Network

Graph Attention Networks (GATs) are a type of neural network designed for processing graph-structured data. GATs leverage attention mechanisms to weigh the importance of neighboring nodes when aggregating information for each node. Unlike traditional graph convolutional networks (GCNs) that use a fixed weighting scheme, GATs dynamically assign different weights to different edges in the graph based on their relevance to the task at hand. This adaptive approach allows GATs to more effectively capture the underlying structure of the graph and improve the accuracy of predictions in tasks such as node classification, link prediction, and graph classification [22].

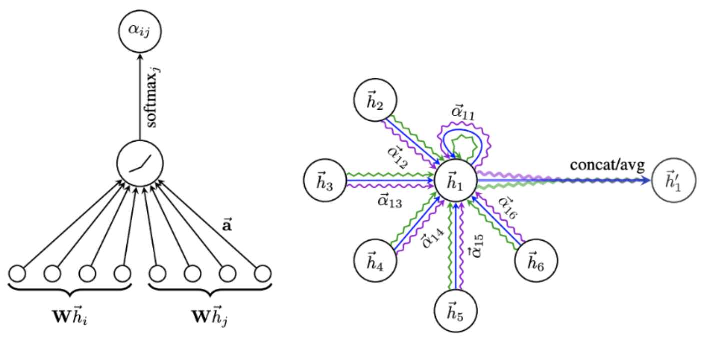


Figure 1: **Left:** The attention mechanism $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by our model, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

Figure 4.2: Graph Attention Network Mechanism (Adapted from Veličković et al., 2018 [16])

4.2.3 GraphSAGE

The idea behind GraphSAGE (SAmple and aggreGatE) is to generate node embeddings for graphs in an inductive manner by leveraging node features and the graph's topological structure. Unlike traditional embedding methods that require training a distinct embedding vector for each node, GraphSAGE trains a set of aggregator functions to aggregate feature information from a node's local neighborhood. This allows the model to generate embeddings for previously unseen nodes during inference by applying the learned aggregation functions. The key innovation in GraphSAGE is its ability to generalize to new nodes, making it suitable for dynamic graphs where nodes can frequently be added or removed.
GraphSAGE's process involves sampling a neighborhood, aggregating feature information from neighbors, and then using this aggregated information to predict the graph context and label. The following images illustrate this process:

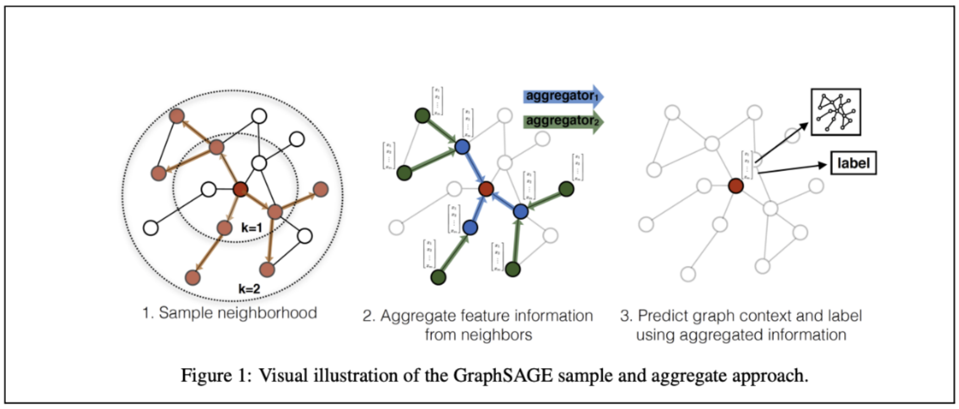

k=1
k=2
1. Sample neighborhood
aggregator1
aggregator2
2. Aggregate feature information from neighbors
label
3. Predict graph context and label using aggregated information
Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

Figure 4.3: GraphSAGE Architecture (Adapted from Hamilton et al., 2017 [17])

4.3 Hypergraph-based Recommender Systems

4.3.1 HyperGCN

HyperGCN utilizes a hypergraph structure to represent relationships between users and items. The model propagates information through the hypergraph to learn embeddings for each node, which are then used for rating prediction.

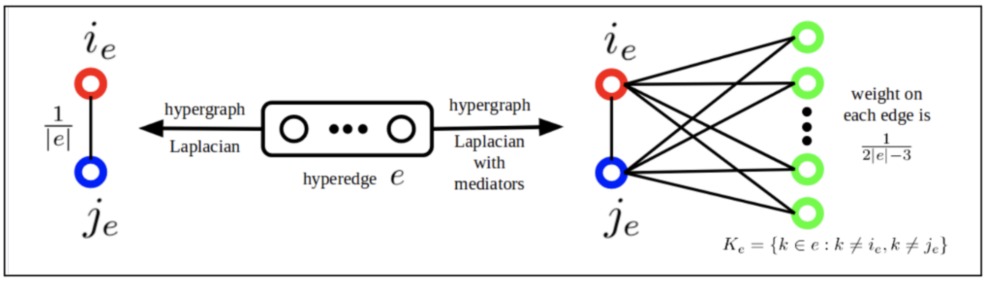

i_e
$\frac{1}{|e|}$
j_e
hypergraph Laplacian
hyperedge e
hypergraph Laplacian with mediators
weight on each edge is $\frac{1}{2|e|-3}$
$K_e = \{k \in e : k \neq i_e, k \neq j_e\}$

Figure 4.4: HyperGCN Architecture (Adapted from Yadati et al., 2019 [18])

4.3.2 Node2Vec

Node2Vec is adapted to work with hypergraphs by generating random walks from the hypergraph structure. These walks are used to learn embeddings for nodes, which can be used to predict interactions between users and items.

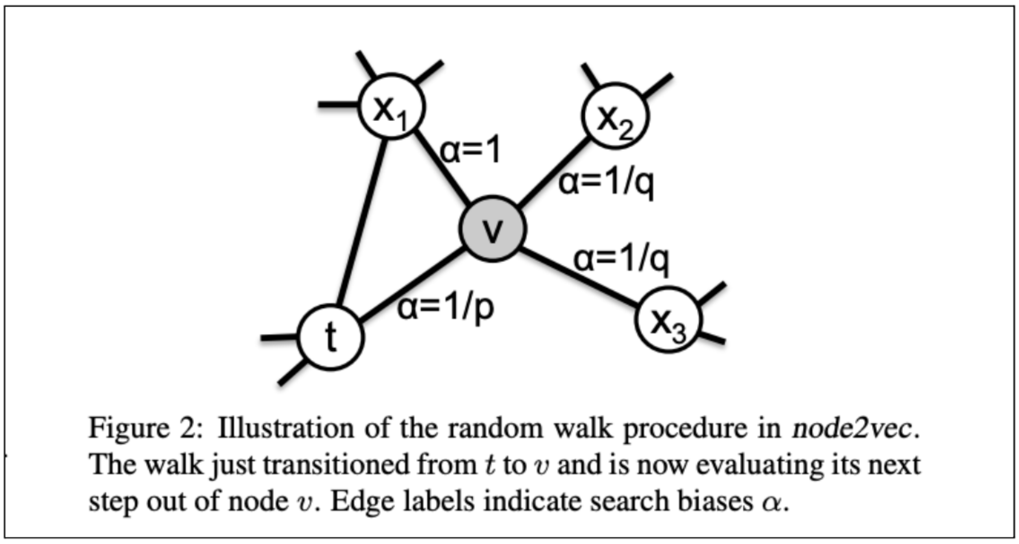


Figure 2: Illustration of the random walk procedure in *node2vec*. The walk just transitioned from t to v and is now evaluating its next step out of node v. Edge labels indicate search biases α.

Figure 4.5: Random Walk in Node2Vec illustration (Adapted from Grover et al., 2016 [19])

Methodology and Experiments

Objectives

This study seeks to conduct a thorough comparative analysis of these three models, focusing on their performance with regards to accuracy, computational complexity, scalability, and their effectiveness in handling data sparsity and dynamically changing environments. By evaluating these aspects, the research aims to illuminate the operational strengths and weaknesses of each model, providing clear insights that could guide the development and deployment of future recommender systems. Through this comparative framework, we aspire to answer which model, under what conditions, provides the most reliable and robust recommendations, thereby significantly contributing to the optimization of digital services.

5.1 Data Collection and Preprocessing

The data for this project is sourced from the MovieLens group and we choose the 100k ratings dataset, which includes files such as links.csv, movies.csv, ratings.csv, and tags.csv. The preprocessing steps begin with loading the dataset and encoding the user and movie IDs to ensure they are in a numerical format suitable for model training. The dataset is then split into training and testing sets, with 80% of the data allocated for training and the remaining 20% reserved for testing.
To address the challenges of data sparsity and the cold start problem, we generated specific subsets of the training data to simulate these scenarios:
Data Sparsity Simulation: To evaluate the model's performance under sparse data conditions, the model is trained on only 10% of the initial training dataset. The model's accuracy is then tested on the original 20% test dataset. This approach helps us understand how well the model can perform when trained with limited data.
Cold Start Simulation: To simulate the cold start scenario, we selected a subset of 10% of users who have provided only a few ratings from the original training set. Using the same 20% test set, the model is then evaluated on its ability to generate accurate recommendations for these new or less active users, which is a common challenge in real-world recommendation systems.
These scenarios are designed to mimic real-world conditions where data may be incomplete or new users are introduced, allowing us to rigorously assess the robustness and adaptability of the recommendation algorithms.

5.2 Models implementation and training

Collaborative Filtering:

For this part, we have used the Surprise Library to implement the algorithms for the Conventional Collaborative Filtering. It offers a diverse range of algorithms, including KNN-based methods, SVD (Singular Value Decomposition), and CoClustering, which were used to assess the performance of different recommendation strategies. Surprise supports various similarity measures and provides robust implementations of these algorithms, making it ideal for experimenting with different approaches and comparing their effectiveness within our study.

LightGCN:

The LightGCN model was implemented using TensorFlow, with users and items represented as embeddings in a high-dimensional space. The model was trained to predict user ratings by learning the interactions between these user and item embeddings. To ensure the model's robustness, early stopping and model check-pointing techniques were employed during training, allowing the process to be monitored closely and the model to be saved at its best performance point to prevent overfitting. This approach was consistently applied across all scenarios, including the sparse data and new user (cold start) situations, to evaluate the model's effectiveness under different conditions.

Graph Attention Network (GAT):

The GAT model was implemented using TensorFlow, with users and items represented as embeddings, and attention layers applied to highlight the most important connections within the data.To ensure that the GAT model could learn from the user-item interactions effectively, an adjacency matrix was constructed to represent these connections. The model was trained to predict user ratings by learning the interactions between user and item embeddings through the attention mechanism. Like the LightGCN implementation, early stopping and model check-pointing techniques were employed during training to monitor the process and save the model at its best performance point, thus preventing overfitting.

Graph Sample and Aggregate (GraphSAGE):

The model was implemented using TensorFlow, where users and items were represented as embeddings, and a graph structure was utilized to model the interactions between them. The GraphSAGE model employs an aggregation mechanism to combine the features of each node (user or item) with those of its neighbors, allowing the model to learn more contextual and relevant embeddings. To facilitate this, an adjacency matrix was constructed to represent the connections between users and items based on their interactions in the dataset. This matrix was then converted into a TensorFlow SparseTensor to efficiently handle the graph structure within the model. The GraphSAGE layer was designed to aggregate information from neighboring nodes, and the model was trained to predict user ratings by learning the relationships between user and item embeddings. Similar to the other models, early stopping and model checkpointing were employed during training to prevent overfitting and ensure that the best-performing model was saved.

HyperGCN:

The model was built using TensorFlow, where users and items were represented as nodes in a hypergraph, and the HyperGCN model was designed to learn embeddings for these nodes by aggregating information from the hypergraph structure. The model architecture included an embedding layer for nodes, followed by multiple dense layers to capture complex interactions. A dropout layer was included to prevent overfitting and ensure that the model could generalize well to unseen data. The model was trained using a pairwise hinge loss function, which helps in learning the relative preferences of users by comparing positive user-item interactions against negative samples. To train the model effectively, negative samples were generated by selecting user-movie pairs that were not present in the training data. This allowed the model to learn to distinguish between relevant and irrelevant items for each user. The model was trained for multiple epochs, with the optimization process guided by an Adam optimizer. In addition to the normal scenario, where the model was trained on the full dataset, separate hypergraphs were created from the corresponding training sets for the sparse data and new user (cold start) scenarios. This approach enabled the HyperGCN model to be tested under different conditions, evaluating its performance when data is limited or when new users are introduced.

Node2vec:

The Node2Vec model was built using TensorFlow, where the model aimed to learn embeddings for nodes (users and movies) by generating random walks within the hypergraph and using these walks to train the embedding model. The process began with constructing a hypergraph from the movie ratings data, where nodes represented users and movies, and edges represented the interactions between them. Random walks were then generated within this hypergraph, simulating sequences of node interactions. These walks were used to create context-target pairs for training the Node2Vec model, which learns embeddings by predicting the target node (movie or user) given the context (sequence of nodes). The Node2Vec model architecture consisted of an embedding layer, followed by several dense layers with dropout and batch normalization to prevent overfitting and ensure robust learning. The model was trained using an exponential decay learning rate schedule to optimize the embedding representations over multiple epochs. Separate hypergraphs were created from the corresponding training sets for both the sparse data and new user (cold start) scenarios. This allowed the Node2Vec model to be tested under different conditions, assessing its ability to generalize and perform well when the available data is limited or when new users are introduced.

5.3 Models Evaluation

In this section, we discuss the various evaluation metrics we have used to assess the performance of the recommendation systems.

Evaluation Metrics:

- **Mean Squared Error (MSE)**: Measures the average squared differences between predicted and actual ratings.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- **Root Mean Squared Error (RMSE)**: Measures the square root of the average squared differences between predicted and actual ratings.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **Mean Absolute Error (MAE)**: Measures the average absolute differences between predicted and actual ratings.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Precision and **recall** are binary metrics traditionally used to evaluate models with binary outputs. In the context of recommender systems, where ratings are typically numerical (from 1 to 5), these metrics can still be applied by translating the numerical ratings into a binary classification of **relevant** and **not relevant** items [20]. To achieve this translation, we define a threshold rating value. If an item's predicted rating for a user exceeds this threshold, the item is considered "relevant" for that user. Conversely, if the predicted rating is below the threshold, the item is deemed "not relevant." For instance, in our case, a threshold of **3.5** is used. An item with a predicted rating of 3.5 or higher is considered a good recommendation (relevant), while an item with a rating below 3.5 is considered not

relevant. This binary classification allows the application of precision and recall metrics in the evaluation of recommender systems, enabling us to measure how well the model identifies and ranks relevant items for users.
After calculating the Precision@k and Recall@k for each user, the average precision and recall across all users were computed. This final step ensures that the overall performance of the recommendation system is summarized by a single Precision@k and Recall@k value.

- **Precision@k**: Measures the proportion of recommended items in the top-k list that are relevant.

$$\text{Precision@k} = \frac{|\{\#\text{ of relevant items}\} \cap \{\ \#\text{ of recommended items@k}\}|}{k}$$

- **Recall@k**: Measures the proportion of relevant items that are recommended in the top-k list.

$$\text{Recall@k} = \frac{|\{\#\text{ of relevant items}\} \cap \{\ \#\text{ of recommended items@k}\}|}{|\{\text{total }\#\text{ of relevant items}\}|}$$

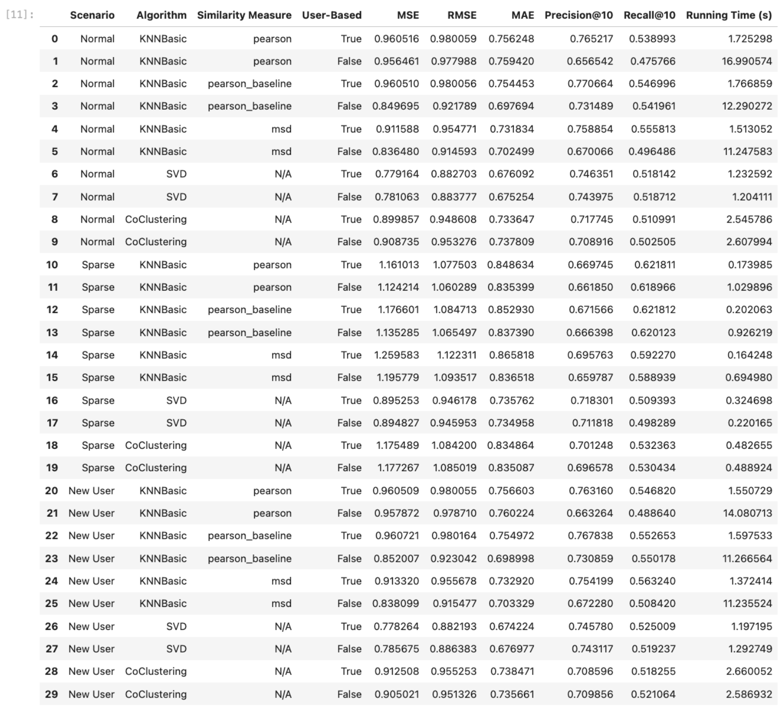
Results and Discussion

6.1 Results

In this section, we present the evaluation results of the various recommender systems that we have implemented on the MovieLens dataset. Our primary objective was to compare the performance of these recommendation algorithms under three scenarios: on a normal dataset with no missing values, on a sparse dataset with the majority of ratings missing, and on a dataset with a lot of new users, in other words there is no information about these users in the systems. The algorithms tested include KNNBasic, SVD, and CoClustering for the conventional collaborative filtering; LightGCN, GAT, and GraphSAGE for the graph-based systems; HyperGCN, and Node2Vec for the hypergraph-based systems. Each model was evaluated using key metrics such as MSE, RMSE, MAE, Precision@10, Recall@10, and training time. The results provide a comprehensive understanding of the strengths and weaknesses of each algorithm, shedding light on their suitability for different recommendation scenarios.

6.1.1 Collaborative Filtering

The results presented in figure 6.1 indicate that user-based collaborative filtering algorithms generally perform better than item-based algorithms across various scenarios on the MovieLens 100k dataset. Specifically, user-based methods consistently achieve lower error metrics (MSE, RMSE, MAE) and higher precision and recall scores, suggesting better predictive accuracy and relevance in recommendations. For clarity and brevity, we highlight the best-performing algorithm under each scenario. To determine the best-performing algorithm, we group the results by algorithm and user-based configuration, calculating the average RMSE for each group. The algorithm with the lowest average RMSE was identified as the best performer. We choose to focus on RMSE as our primary evaluation metric throughout the study for several reasons. Firstly, RMSE offers a balanced compromise MAE and MSE, as it is more sensitive to outliers than MAE but less so than MSE [21]. This characteristic makes RMSE particularly useful in scenarios where it's important to penalize large errors, but not to the extent that MSE does. Secondly, RMSE retains the same unit as the actual and predicted values, which makes it more interpretable and directly comparable to the original data. This combination of properties makes RMSE a reliable and intuitive metric for assessing the overall accuracy of the recommendation algorithms, thereby guiding us in selecting the best-performing algorithm across different scenarios. Table 6.1 presents the results of the best-performing algorithm in the normal, sparse, and new user conditions.



[11]:

	Scenario	Algorithm	Similarity Measure	User-Based	MSE	RMSE	MAE	Precision@10	Recall@10	Running Time (s)
0	Normal	KNNBasic	pearson	True	0.960516	0.980059	0.756248	0.765217	0.538993	1.725298
1	Normal	KNNBasic	pearson	False	0.956461	0.977988	0.759420	0.656542	0.475766	16.990574
2	Normal	KNNBasic	pearson_baseline	True	0.960510	0.980056	0.754453	0.770664	0.546996	1.766859
3	Normal	KNNBasic	pearson_baseline	False	0.849695	0.921789	0.697694	0.731489	0.541961	12.290272
4	Normal	KNNBasic	msd	True	0.911588	0.954771	0.731834	0.758854	0.555813	1.513052
5	Normal	KNNBasic	msd	False	0.836480	0.914593	0.702499	0.670066	0.496486	11.247583
6	Normal	SVD	N/A	True	0.779164	0.882703	0.676092	0.746351	0.518142	1.232592
7	Normal	SVD	N/A	False	0.781063	0.883777	0.675254	0.743975	0.518712	1.204111
8	Normal	CoClustering	N/A	True	0.899857	0.948608	0.733647	0.717745	0.510991	2.545786
9	Normal	CoClustering	N/A	False	0.908735	0.953276	0.737809	0.708916	0.502505	2.607994
10	Sparse	KNNBasic	pearson	True	1.161013	1.077503	0.848634	0.669745	0.621811	0.173985
11	Sparse	KNNBasic	pearson	False	1.124214	1.060289	0.835399	0.661850	0.618966	1.029896
12	Sparse	KNNBasic	pearson_baseline	True	1.176601	1.084713	0.852930	0.671566	0.621812	0.202063
13	Sparse	KNNBasic	pearson_baseline	False	1.135285	1.065497	0.837390	0.666398	0.620123	0.926219
14	Sparse	KNNBasic	msd	True	1.259583	1.122311	0.865818	0.695763	0.592270	0.164248
15	Sparse	KNNBasic	msd	False	1.195779	1.093517	0.836518	0.659787	0.588939	0.694980
16	Sparse	SVD	N/A	True	0.895253	0.946178	0.735762	0.718301	0.509393	0.324698
17	Sparse	SVD	N/A	False	0.894827	0.945953	0.734958	0.711818	0.498289	0.220165
18	Sparse	CoClustering	N/A	True	1.175489	1.084200	0.834864	0.701248	0.532363	0.482655
19	Sparse	CoClustering	N/A	False	1.177267	1.085019	0.835087	0.696578	0.530434	0.488924
20	New User	KNNBasic	pearson	True	0.960509	0.980055	0.756603	0.763160	0.546820	1.550729
21	New User	KNNBasic	pearson	False	0.957872	0.978710	0.760224	0.663264	0.488640	14.080713
22	New User	KNNBasic	pearson_baseline	True	0.960721	0.980164	0.754972	0.767838	0.552653	1.597533
23	New User	KNNBasic	pearson_baseline	False	0.852007	0.923042	0.698998	0.730859	0.550178	11.266564
24	New User	KNNBasic	msd	True	0.913320	0.955678	0.732920	0.754199	0.563240	1.372414
25	New User	KNNBasic	msd	False	0.838099	0.915477	0.703329	0.672280	0.508420	11.235524
26	New User	SVD	N/A	True	0.778264	0.882193	0.674224	0.745780	0.525009	1.197195
27	New User	SVD	N/A	False	0.785675	0.886383	0.676977	0.743117	0.519237	1.292749
28	New User	CoClustering	N/A	True	0.912508	0.955253	0.738471	0.708596	0.518255	2.660052
29	New User	CoClustering	N/A	False	0.905021	0.951326	0.735661	0.709856	0.521064	2.586932

Figure 6.1: Collaborative Filtering Performance

Our analysis has revealed that SVD is the best performing algorithm having the lowest average RMSE score across the 3 scenarios. Table 6.1 shows that SVD consistently delivered strong results across all scenarios, achieving some of the lowest RMSE and MAE values, which were particularly evident in the normal and new user scenarios. These low error metrics were accompanied by relatively high Precision@10 and Recall@10 scores, indicating that SVD not only makes accurate predictions but also effectively identifies relevant recommendations. Notably, in the sparse data scenario, where user-item interactions are less frequent, the RMSE increases to 0.9462. This suggests that the performance of SVD slightly deteriorates when the data is sparse. The increase in RMSE reflects the challenges the algorithm faces in making accurate predictions with limited data. For the new user or cold start scenario, the RMSE remains at 0.8822, which is nearly identical to the RMSE in the normal scenario. This indicates that SVD is robust in handling cold start situations, where new users with little to no prior interaction data are introduced.

Scenario	**Algorithm**	**MSE**	**RMSE**	**MAE**	**P@10**	**Re@10**	**Time**
Normal	SVD	0.779164	0.882703	0.676092	0.746351	0.518142	1.232592
Sparse	SVD	0.895253	0.946178	0.735762	0.718301	0.509393	0.324698
New User	SVD	0.778264	0.882193	0.674224	0.745780	0.525009	1.197195

Table 6.1: Best Performing Algorithm in Collaborative Filtering

6.1.2 Graph-based Models

LightGCN

Scenario	Algorithm	MSE	RMSE	MAE	P@10	R@10	Time
Normal	LightGCN	1.300472	1.140382	0.814420	0.749278	0.503184	16.588426
Sparse	LightGCN	1.353399	1.163357	0.837602	0.750687	0.504528	2.137099
New User	LightGCN	1.352309	1.162888	0.837188	0.751206	0.504964	0.539947

Table 6.2: Performance Metrics for LightGCN Algorithm

LightGCN exhibits relatively low RMSE (1.140382) and MAE (0.814420) with high Precision@10 (0.749278) and moderate Recall@10 (0.503184) in the normal scenario, though it comes with a higher running time of 16.588426 seconds. In the sparse scenario, RMSE (1.163357) and MAE (0.837602) slightly increase, indicating a marginal decline in accuracy, but Precision@10 and Recall@10 remain stable, with a significantly reduced running time of 2.137099 seconds which is expected since the amount of training data has significantly decreased. In the new user scenario, LightGCN maintains similar performance with an RMSE of 1.162888, MAE of 0.837188, and stable Precision@10 (0.751206) and Recall@10 (0.504964), while its running time decreases further to just 0.539947 seconds, showcasing its robustness in handling cold start situations efficiently. Looking at table 6.2 Precision@10 and Recall@10 tend to be more stable across different scenarios because these measures focus on the accuracy of the top 10 recommendations rather than the entire set of predictions. However, we noticed that while the errors increased in the sparse and new users scenarios, the precision and recall increased instead of decreasing. This can be possibly be explained by a bias toward popular items. When data is sparse or when dealing with new users, the model might lean more heavily on popular items that are generally well-received. Popular items are likely to be relevant to a larger number of users, which could lead to higher precision and recall, even though the prediction of ratings (which impacts RMSE and MAE) might not be as accurate.

Graph Attention Network

Scenario	Algorithm	MSE	RMSE	MAE	P@10	R@10	Time
Normal	GAT	5.279527	2.297722	1.813445	0.640868	0.339237	76.762282
Sparse	GAT	9.903407	3.146968	2.508598	0.665226	0.375002	9.373171
New User	GAT	8.366129	2.892426	2.294709	0.656693	0.358051	2.174327

Table 6.3: Performance Metrics for Graph Attention Network Algorithm

In table 6.3 GAT shows significantly higher RMSE and MAE in the normal scenario, indicating lower accuracy. The Precision@10 and Recall@10 are also relatively low, suggesting that the recommendations are less relevant and less effective. Additionally, the running time is notably long at 76.76 seconds, highlighting inefficiency in this scenario. In the sparse scenario, the errors worsen with RMSE increasing to 3.14 and MAE to 2.50, although Precision@10 and Recall@10 slightly improve, possibly due to the algorithm relying more on generalized patterns in the sparse data. The running time, however, drops significantly to 9.37 seconds. In the new user scenario, GAT continues to exhibit high RMSE and MAE, but Precision@10 and Recall@10 remain relatively stable compared to the sparse scenario. The running time decreases further, indicating quicker processing but still subpar performance in terms of accuracy and relevance in cold start situations.

Scenario	Algorithm	MSE	RMSE	MAE	P@10	R@10	Time
Normal	GraphSAGE	0.995274	0.997634	0.765923	0.737043	0.525224	71.149196
Sparse	GraphSAGE	1.191938	1.091759	0.853637	0.729891	0.473050	8.101512
New User	GraphSAGE	2.352740	1.533864	1.183540	0.732416	0.497194	1.240817

Table 6.4: Performance Metrics for GraphSAGE Algorithm

From table 6.4 we can see that GraphSAGE performs well in the normal scenario, with a low RMSE and MAE indicating high accuracy. The Precision@10 and Recall@10 are also strong, suggesting that GraphSAGE provides relatively good recommendations. However, this accuracy comes at the cost of a relatively long running time of 71.149196 seconds, pointing to a compromise between accuracy and processing time.
In the sparse scenario, GraphSAGE's RMSE increases to 1.09 from 0.99 and MAE to 0.85 from 0.76, indicating a slight decline in accuracy as data sparsity increases. Precision@10 and Recall@10 see a small decrease, but they remain competitive. Notably, the running time significantly decreases to 8.101512 seconds, indicating improved efficiency when handling sparse data.
In the new user scenario, GraphSAGE experiences a more pronounced decline in accuracy, with RMSE rising to 1.53 and MAE to 1.18. Despite the higher error rates, Precision@10 0.73 and Recall@10 0.49 remain relatively stable, suggesting that GraphSAGE still delivers fairly relevant recommendations even in cold start situations. The running time drops considerably to 1.240817 seconds, showing that GraphSAGE adapts efficiently when new users are introduced, balancing performance and computational demands effectively in these challenging scenarios.

6.1.3 Hypergraph-based Models

HyperGCN

Scenario	Algorithm	MSE	RMSE	MAE	P@10	R@10	Time
Normal	HyperGCN	5.608074	2.368137	2.050150	0.116924	0.009341	86.924503
Sparse	HyperGCN	13.239963	3.638676	3.480585	0.000000	0.000000	11.528216
New User	HyperGCN	5.691181	2.385620	2.063831	0.105558	0.008382	88.792880

Table 6.5: Performance Metrics for HyperGCN Algorithm

HyperGCN demonstrates notably poor performance across all scenarios. In the normal scenario, it has a high RMSE and MAE, indicating low prediction accuracy. Precision@10 and Recall@10 are extremely low, suggesting that the recommendations are largely irrelevant to users. Additionally, the algorithm has a very long running time of 86.92 seconds, which further emphasizes its inefficiency.
In the sparse scenario from table 6.5, HyperGCN's performance deteriorates even further, with RMSE increasing to 3.63 from 2.36 and MAE to 3.48 from 2.05. Both Precision@10 and Recall@10 drop to 0, indicating that the algorithm fails to make any useful recommendations in this scenario. This situation of precision and recall being 0 can be explained by the fact that since the amount of training data has significantly decreased, the intersection between the number of relevant movies and the number of recommended(based on predicted rating) movies is 0, so the two sets do not have any movie in commun. The running time, while reduced to 11.52 seconds, still reflects inefficiency relative to the lack of relevant recommendations.
In the new user scenario, HyperGCN maintains similarly poor performance as in the normal scenario, with RMSE and MAE remaining high, looking at table 6.5. Precision@10 and Recall@10

continue to be very low, showing that the algorithm struggles to make relevant recommendations for new users. The running time remains long, indicating that HyperGCN is not only inaccurate but also computationally expensive across all scenarios.

Node2Vec

Scenario	Algorithm	MSE	RMSE	MAE	P@10	R@10	Time
Normal	Node2Vec	47.267650	6.875147	5.017519	0.473556	0.104442	57.877078
Sparse	Node2Vec	48.482455	6.962934	5.095941	0.407286	0.099925	38.807848
New User	Node2Vec	47.357681	6.881692	4.988233	0.442582	0.097093	59.448123

Table 6.6: Performance Metrics for Node2Vec Algorithm

Node2Vec, as shown in table 6.6, exhibits poor performance across all scenarios, with consistently high error rates. In the normal scenario, it has an extremely high RMSE and MAE , indicating a significant lack of prediction accuracy. While the Precision@10 is somewhat moderate, the Recall@10 is low, meaning that while some relevant items are recommended, many are missed. The running time is also substantial at 57.87 seconds, reflecting a significant computational cost for relatively low performance.
In the sparse scenario, Node2Vec's both RMSE and MAE increase slightly, and MAE, showing a further decline in accuracy as data becomes sparser. Precision@10 and recall@10 decrease, indicating that the relevance of recommendations diminishes further. The running time decreases to 38.80 seconds, which means the efficiency is slightly improved but still at a high computational cost for the accuracy achieved.
In the new user scenario, Node2Vec's RMSE and MAE are similar to the normal scenario, indicating that the algorithm does not adapt well to new user conditions. Precision@10 and Recall@10 remain low, showing limited relevance in recommendations for new users. The running time is the highest among the scenarios at 59.44 seconds, highlighting inefficiency in handling cold start situations.

Figure 6.2 presents a line graph that visually compares the performance of the algorithms side by side, focusing specifically on RMSE as the key metric. This graphical representation provides a clear and concise way to evaluate and contrast the effectiveness of each algorithm, following our earlier discussion on the importance of RMSE as a key performance measure.

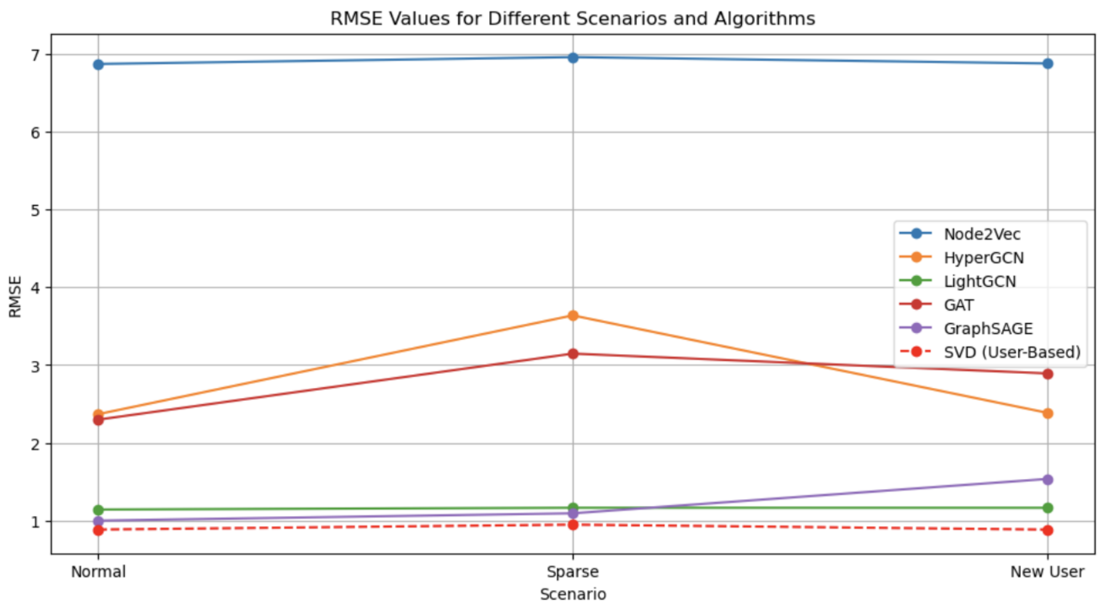


Figure 6.2: Comparison of RMSE Across Different Scenarios for Recommender System Algorithms

6.2 Discussion

The evaluation results underscore the varying strengths and weaknesses of different recommendation algorithms, revealing important insights for selecting the most appropriate method depending on the scenario. SVD emerged as the most consistent and reliable performer across all scenarios, balancing accuracy and computational efficiency effectively. Its ability to maintain low RMSE values, around 0.88 (see figure 6.2), and deliver relevant recommendations makes it an excellent choice for our recommendation tasks using the MovieLens 100k dataset.
In figure 6.2 we observed that GraphSAGE demonstrated superior performance in both normal and sparse data scenarios, achieving lower RMSE values compared to LightGCN. This indicates that GraphSAGE is particularly effective when the dataset is either complete or sparse, making it a strong contender when accuracy is prioritized over computational efficiency. LightGCN, while slightly outperformed by GraphSAGE in these scenarios, still maintained robust performance across all conditions, particularly excelling in handling new user (cold start) situations with low RMSE and efficient running times (see table 6.2). This makes LightGCN a reliable all-around performer, especially when dealing with varied data conditions. On the other hand, Node2Vec and HyperGCN consistently underperformed, with high RMSE values across all scenarios. Node2Vec, in particular, exhibited the poorest accuracy, with RMSE values around 6.87 (see table 6.6), indicating significant challenges in adapting to different data conditions. In table 6.5 we can see that HyperGCN also struggled, especially in the sparse scenario where it failed to produce relevant recommendations, evidenced by Precision@10 and Recall@10 dropping to zero. GAT performed better than Node2Vec and HyperGCN (see figure 6.2) but still showed relatively high RMSE values, particularly in the sparse scenario, suggesting that it is less effective when data is limited. However, our expectation that hypergraph models like HyperGCN would outperform other algorithms based on their theoretical advantages in capturing higher-order relationships in complex data was not met. One potential reason for this underperformance could be that the MovieLens dataset used in this evaluation may not be complex enough to fully leverage the strengths of hypergraph models. Hypergraph models are particularly powerful when dealing with data that involves intricate, multi-dimensional relationships, such as in domains where items are connected in more complex ways than simple pairwise interactions. The results suggest that while some algorithms like SVD and LightGCN offer a balance of accuracy and efficiency, others like GraphSAGE provide better accuracy under specific conditions, such as normal and sparse data scenarios. In contrast, algorithms like Node2Vec and HyperGCN may need to be reconsidered or even disregarded for our recommendation systems implementation, especially in sparse data or cold start conditions.

Conclusion and Future Works

7.1 Conclusion

This study set out to assess and compare the efficacy of collaborative filtering, graph-based, and hypergraph-based models in generating accurate and personalized movie recommendations using the MovieLens 100k dataset. The comprehensive evaluation of various algorithms across different scenarios—normal, sparse, and new user (cold start)—revealed significant insights into the strengths and weaknesses of each approach, providing valuable guidance for the design and implementation of recommender systems in practical applications.
Singular Value Decomposition (SVD) emerged as the most consistent and reliable performer, demonstrating a remarkable balance between accuracy and computational efficiency. Its ability to maintain low RMSE values across all scenarios, while also delivering relevant recommendations, underscores its robustness as a go-to algorithm our recommendation tasks. SVD's versatility in handling different data conditions, including cold start situations and sparse data, further solidifies its position as an optimal choice our movie recommendation systems.
GraphSAGE also proved to be a strong contender, particularly excelling in normal and sparse data scenarios where it outperformed LightGCN in terms of RMSE. This suggests that GraphSAGE is particularly effective in scenarios where data is either complete or sparse, making it a preferred option when accuracy is the primary concern. However, its slightly lower performance in handling new users, coupled with longer running times, suggests that while GraphSAGE is highly accurate, it may require more computational resources and may not be as versatile as SVD in all scenarios.
LightGCN, despite being slightly outperformed by GraphSAGE in certain scenarios, demonstrated robust all-around performance. Its ability to maintain low RMSE values, particularly in new user scenarios, along with its efficient running times, makes it a reliable choice for environments with varied data conditions. LightGCN's consistent performance across all scenarios, combined with its computational efficiency, suggests that it is a strong option for systems that need to balance accuracy and scalability.
In contrast, **Node2Vec** and **HyperGCN** consistently underperformed, with high RMSE values and lower relevance in recommendations across all scenarios. Node2Vec, in particular, struggled with adapting to different data conditions, making it a less suitable choice for recommendation tasks. Hypergraphs, while theoretically promising due to its ability to capture higher-order relationships, failed to deliver the expected results. This underperformance is likely due to the relatively simple nature of the MovieLens dataset, which may not have provided the complexity needed to fully leverage the strengths of hypergraph models. As a result, these algorithms may need to be reconsidered or even disregarded for tasks requiring high accuracy and relevance, particularly in sparse data or cold start conditions when the dataset is not complex enough.

7.2 Future Works

Building on the findings of this study, several avenues for future research can be pursued:

- **Parameter Optimization:** Further tuning of hyperparameters for graph-based and hypergraph-based models could potentially improve their accuracy and reduce error metrics, enhancing their practical applicability.

- **Scalability Studies:** Investigating the scalability of these models with larger datasets and real-world applications will provide insights into their performance and practicality in different scenarios.

- **Design Space for Recommender Systems:** Developing a design space framework for recommender systems could be a valuable tool. Such a system would quickly analyze the data and suggest the most suitable algorithms based on the dataset characteristics, thereby streamlining the selection process and improving recommendation accuracy.

By addressing these areas, future research can continue to advance the field of recommender systems, providing more accurate, efficient, and personalized recommendations across various applications.

Appendix A

GitHub Repository

https://github.com/diatt17/mrp_masters

Data Files

https://grouplens.org/datasets/movielens/

Bibliography

[1] Behera, G., & Nain, N. (2023). Collaborative filtering with temporal features for movie recommendation system. Procedia Computer Science, 218, 1366-1373. https://www.sciencedirect.com/science/article/pii/S1877050923001151

[2] Fareed, A., Hassan, S., Belhaouari, S. B., & Halim, Z. (2023). A collaborative filtering recommendation framework utilizing social networks. Machine Learning with Applications, 14, 100495. https://www.sciencedirect.com/science/article/pii/S2666827023000488

[3] Safae, H., Mohamed, L., Chehri, A., El Alami Yasser, E. M., & Saadane, R. (2023). Link prediction using graph neural networks for recommendation systems. Procedia Computer Science, 225, 4284-4294. https://www.sciencedirect.com/science/article/pii/S1877050923015831

[4] Nesmaoui, R., Louhichi, M., & Lazaar, M. (2023). A collaborative filtering movies recommendation system based on graph neural network. Procedia Computer Science, 220, 456-461. https://www.sciencedirect.com/science/article/pii/S1877050923005938

[5] Afoudi, Y., Lazaar, M., & Al Achhab, M. (2021). Hybrid recommendation system combined content-based filtering and collaborative prediction using artificial neural network. Simulation Modelling Practice and Theory, 113, 102375. https://www.sciencedirect.com/science/article/pii/S1569190X21000836

[6] Singh, N., Bhokare, S., & Shah, M. (2023). Movie Recommendation System. International Journal of Novel Research and Development (IJNRD), 8(4). Guide: Isha Sood. https://www.ijnrd.org/papers/IJNRD2304674.pdf

[7] Afoudi, Y., Lazaar, M., & Hmaidi, S. (2023). An enhanced recommender system based on heterogeneous graph link prediction. Engineering Applications of Artificial Intelligence, 124, 106553. https://www.sciencedirect.com/science/article/pii/S0952197623007376

[8] Elahi, M., Khosh Kholgh, D., Kiarostami, M. S., Oussalah, M., & Saghari, S. (2023). Hybrid recommendation by incorporating the sentiment of product reviews. Information Sciences, 625, 738-756. https://www.sciencedirect.com/science/article/pii/S0020025523000518

[9] Shokrzadeh, Z., Feizi-Derakhshi, M.-R., Balafar, M.-A., & Bagherzadeh Mohasefi, J. (2024). Knowledge graph-based recommendation system enhanced by neural collaborative filtering and knowledge graph embedding. Ain Shams Engineering Journal, 15(1), 102263. https://www.sciencedirect.com/science/article/pii/S2090447923001521

[10] Zamanzadeh Darban, Z., & Valipour, M. H. (2022). GHRS: Graph-based hybrid recommendation system with application to movie recommendation. Expert Systems with Applications, 200, 116850. https://www.sciencedirect.com/science/article/pii/S0957417422003025

[11] Gharahighehi, A., Vens, C., & Pliakos, K. (2023). HypeRS: Building a hypergraph-driven ensemble recommender system. arXiv. https://arxiv.org/pdf/2306.12800

[12] Tewari, A. S. (2020). Generating items recommendations by fusing content and user-item based collaborative filtering. Procedia Computer Science, 167, 1934-1940. https://www.sciencedirect.com/science/article/pii/S1877050920306803

[13] Bertram, N., Dunkel, J., & Hermoso, R. (2023). I am all EARS: Using open data and knowledge graph embeddings for music recommendations. Expert Systems with Applications, 229, 120347. https://www.sciencedirect.com/science/article/pii/S0957417423008497

[14] Gharahighehi, A., Vens, C., & Pliakos, K. (2021). Fair multi-stakeholder news recommender system with hypergraph ranking. Information Processing & Management, 58(5), 102663. https://www.sciencedirect.com/science/article/pii/S030645732100146X

[15] He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020). LightGCN: Simplifying and powering graph convolution network for recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (pp. 639-648). https://arxiv.org/pdf/2002.02126

[16] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. In International Conference on Learning Representations (ICLR). https://arxiv.org/pdf/1710.10903

[17] Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS 2017). https://cs.stanford.edu/people/jure/pubs/graphsage-nips17.pdf

[18] Yadati, N., Nimishakavi, M., Yadav, P., Nitin, V., Louis, A., & Talukdar, P. (2019). HyperGCN: A new method of training graph convolutional networks on hypergraphs. In Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). https://papers.nips.cc/paper_files/paper/2019/file/1efa39bcaec6f3900149160693694536-Paper.pdf

[19] Grover, A., & Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 855-864). https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf

[20] Malaeb,M. (2017). Recall and Precision at k for Recommender Systems.https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54

[21] Matalonga, H. (2023). Choosing between MAE, MSE and RMSE. https://hmatalonga.com/blog/choosing-between-mae-mse-and-rmse/

[22] Hristov, H. (2024). Graph Attention Networks. https://baeldung.com/cs/graph-attention-networks