Barcelona School of Economics

Data Science Methodology Program

21D009 Networks: Concepts and Algorithms

Movie recommendations using networks

Authors:
Codd, Jonny
Chen, Joshua
Gallegos, Rafael
Pérez, Carlos

Professors:
Milán, Pau
Komander, Björn

December 20^{th}, 2023

Contents

List of Figures

List of Tables

1 Introduction

With the increase in choice across various domains, from entertainment to online social networks, recommendation systems have become increasingly crucial in alleviating information overload and shaping experience in every-day life. This has fueled demand for ever more sophisticated methods to delivery personalised recommendations as the variety of choice continues to expand. Historically, personal recommendations were primarily sourced from one's social circles, operating on the premise that friends or acquaintances with similar interests or tastes could offer valuable suggestions. This traditional approach, while effective to a degree, was inherently limited by the scope of one's immediate social network. However, modern recommendations systems now operate and build on a similar premise, leveraging network and network-adjacent concepts to understand the connections between people and tailor recommendations to individuals preference.

Currently, as we see recommendation systems utilize advancements in machine/deep learning to become more powerful and precise, we are furthering conversations on the increasingly central role that these recommendations systems take on shaping people's preferences. In some sense, recommendation algorithms have gotten so good that, instead of trying their best to predict consumer preferences, they are now creating consumer preferences. Consider how TikTok in news articles is often paired with the word "Addictive" and how it has reshaped how, for example, the music industry interacts with its consumers. Another example is the criticism against Spotify who has faced criticisms that their algorithm is biased towards popular songs/artists and "buries" smaller or independent artists.

Our project is two-fold. Using the concepts learned in this course, we will first analyse the properties of a network build from the MovieLens dataset. We will then review and build recommendation systems using a variety of network techniques commonly used in the literature. By using historic consumer preference data, one can build a matrix of user-to-item scores that can be thought of as the adjacency matrix of a bipartite network - with the users representing one set of nodes and the items representing the other. The edges are represented either by binary or scalar relationships between users and items, implying unweighted or weighted graphs respectively. Furthermore, we will consider how these systems impact consumer tastes over time.

2 Literature Review

The exploration and application of network analysis in the context of building recommender systems is a widely studied topic. This began with the development of the Collaborative Filtering (CF) in the 1990s - a technique that pioneered the construction of user preference networks to tailor recommendations. CF has been applied and studied in many scenarios using a wide variety of datasets, for example, Amazon first began using collaborative filtering in the late 1990s to recommend users products [**collab_f**]. While, according to [**survey_reccomendation**], Collaborative Filtering is the most widely studied recommendation model, accounting for over 41% of all papers, research on the topic has begun to decline since 2014.

Recent trends in recommendation system research have built on the foundational work of CF and shifted towards other techniques such random walk processes [**prob_s**] and, more prominently, Graphical Neural Networks (GNN). GNNs can encode the complex relationships within user-item interaction graphs into dense, low-dimensional vector representations, uncovering subtle relationships and patterns overlooked by traditional CF models . As a result, GNNs have been used in a wide variety of practical applications including designing web-scale reccomedation systems based on networks with billions of nodes [**GNNwebscale**]. This said, they are harder to build and interpret than traditional CF models, and come at greater computational cost [**collab_f**].

Cold start is a common problem with recommendation systems. Collaborative filtering can be user-based or item-based. For the former, the system will recommend items enjoyed by people similar to you. For the latter, the system will recommend items similar to the items that you enjoyed. The former uses user to user similarity and the second uses user to user similarity. This issue arises for new users and new items that have few or no edges (existing histories). Therefore, the system is unable to provide recommendations [**cold_start**]. An example of this is the aforementioned criticism against Spotify.

3 Dataset

MovieLens is popular movie recommender system dataset developed by GroupLens, a computer science research lab at the University of Minnesota. The goal of this challenge is to recommend movies to its users based on their movie ratings. Group Lens offers datasets of different sizes and their datasets are widely used in research and teaching contexts.

The selected dataset consists mainly on two files: movies.csv and ratings.csv. Movies dataset has 9,742 unique films and a column indicating the genres of the film. All possible genres are: 'Romance', 'Musical', 'Children', 'Documentary', 'Sci-Fi', 'Film-Noir', '(no genres listed)', 'Crime', 'Mystery', 'Drama', 'Western', 'Fantasy', 'Animation', 'Thriller', 'War', 'Action', 'Adventure', 'IMAX', 'Comedy', 'Horror'. The number of movies per genre is represented in Figure 3.2.

Ratings dataset consists of 100,836 ratings with 610 unique users that rated 9,724 movies. As it can be observed in Figure **??**, the ratings from users are right-skewed, which suggests that users tend to enter their rating on movies that they probably have liked. Ratings from users have been registered from 1996-03-29 until 2018-09-24. The most popular movies among users have been: Shawshank Redemption, The (1994), Godfather, The (1972), Fight Club (1999), Godfather: Part II, The (1974) and Goodfellas (1990).

The median user has rated 70 films, whereas the user with the lowest number of watched films was 20 movies and the user with the highest number of rated films is 2698.

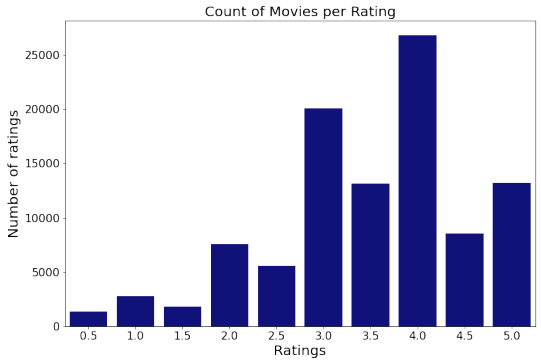


Figure 3.1: Count of Movies per Rating

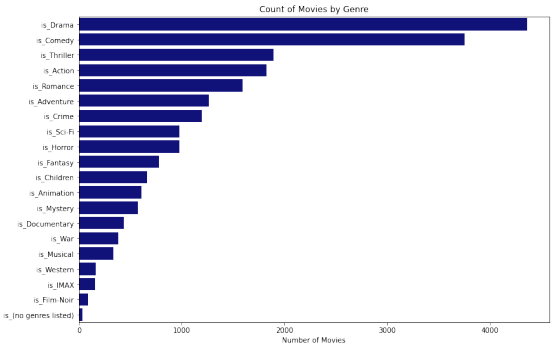


Figure 3.2: Count of Movies per Genre

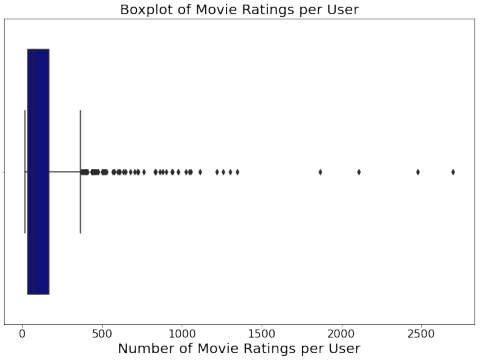


Figure 3.3: Count of Movies per User

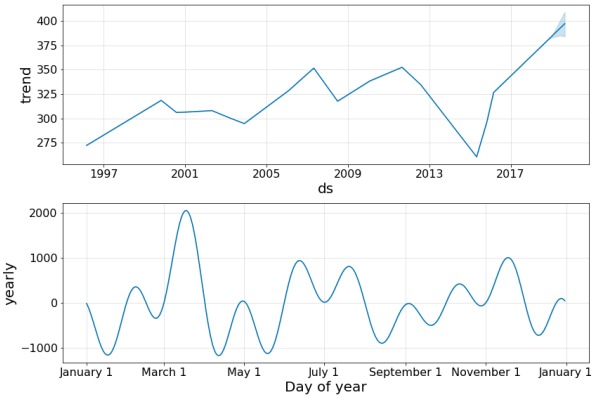


Figure 3.4: Time Series Analysis of Ratings with Prophet

4 Network analysis

With the MovieLens dataset, we created 4 different networks. The first, the user-movie network, is the aforementioned user-item network that will serve as the foundation for both the other networks and the recommendation systems that we build. From this network, we also constructed a user-to-user network - a symmetric unipartite network capturing the similarity between users. Similarly, we did this for the movies as well. Finally, we created another bipartite network that, instead of capturing relationships between users and specific movies, it describes the "fan score" between a user and a genre of movie.

4.1 User and movie bipartite network

To create the user/movie bipartite network, we used the Networkx package in Python. We selected the unique User Ids as one set of nodes and the movie titles as another. Edges were then added if a user has seen a movie. Note that this is a undirected, unweighted graph. Ratings are not considered here.

Despite an incredibly sparse matrix, understandably with close to 10,000 movies - this results in a connected graph - meaning there is only one component.

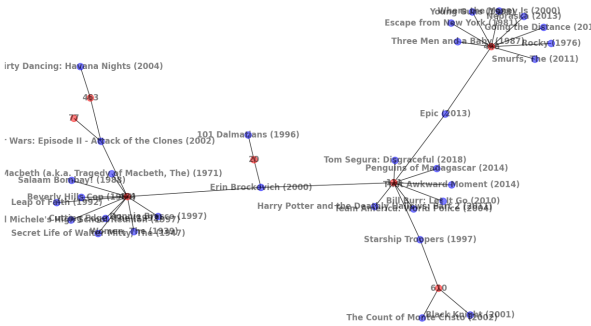


Figure 4.1: Subset of User to Movie Network

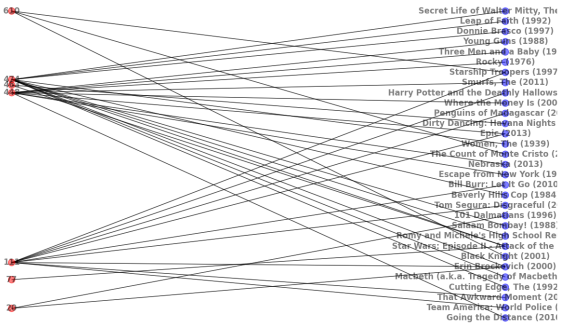


Figure 4.2: BipartiteLayout

To better understand that users and movies, we calculated the centrality for each of them: With this graph - the degree centrality for users and movies are aligned with how many reviews they have. The centrality score for users is much higher than that for movies as there are many more reviews per user than there are per movie.

However, this may not be the best way to actually capture centrality as highly central movies and users might not just be how many direct nodes that they have. Therefore, we also calculated the betweenness and closeness centralities. Take, for example, The Avengers (2012). Regardless of how many reviews it may have, it will probably be an important bridge (betweeness) between the Marvel movies. It is relationships like these that we are hoping to capture. However, there seems to be very little difference in the rankings of these centrality metrics as the top scores keep generally the same order. A couple near the bottom were pushed down but there was no large shuffling.

It is important to note already that the most highly centralized movies are all produced before 2000 (in fact, this extends far beyond just the top 10), despite most of the reviews coming after the 2020s. This is understandable as movies that have been around longer are more likely to have more reviews.

4.2 User-to-user network

To construct the user-to-user network, we calculated the similarity between all users as our edges. We first pivoted the Ratings dataframe on the *UserId*. Our resulting dataframe has the UserId as the index, the Movies as the columns, and any user ratings inside. This is exactly the adjacency matrix for the User to Movie network that we created in the previous section; however, just weighted edges based on ratings.

We used cosine similarity as our node embedding technique to map user vectors into Euclidean space. Cosine similarity is a common space because it ignores magnitude and focuses only on directions in space. For example, A harsh critic might rate an average movie a 2 but a more generous critic might consider average to be 3. This would not affect at all the

Figure 4.3: Network Centrality Measures and Comparative Metrics for Movies

User Id	Degree Centrality	Betweeness Centrality	Closeness Centrality
599	4.067	0.141 (1)	0.406 (2)
414	4.429	0.131 (2)	0.413 (1)
474	3.460	0.120 (3)	0.395 (3)
448	3.061	0.110 (4)	0.388 (4)
274	2.210	-	0.373 (5)
610	2.136	0.060 (5)	0.372 (6)
68	2.067	-	0.371 (7)
380	2.000	0.034 (10)	0.370 (8)
606	1.829	0.050 (6)	0.367 (9)
288	1.731	-	0.365 (10)

Movie Title	Degree Centrality	Betweeness Centrality	Closeness Centrality
Forrest Gump (1994)	0.0339	0.0064	0.4824
Pulp Fiction (1994)	0.0316	0.0050	0.4650
Matrix, The (1999)	0.0286	0.0048	0.4694
The Silence of the Lambs (1991)	0.0287	0.0046	0.4624
Shawshank Redemption (1994)	0.0326	0.0042	-
Star Wars: Episode IV (1977)	0.0258	0.0041	0.4613
Jurassic Park (1993)	0.0245	-	-
Braveheart (1995)	0.0244	-	-
Terminator 2 (1991)	0.0231	-	-
Schindler's List (1993)	0.0226	-	-

cosine similarity (if we kept the ratings positive). A similarity measure such as Euclidean distance would fail here because the more movies you add can only possibly add distance. Therefore people are punished for actually having seen more of the same films.

Because this is an incredibly sparse matrix, in order to calculate similarity scores, we only used the films that people had shared reviews for. However, this could result in the case where two people are assigned similarities despite having very little in common. For example, if two people who have wildly different preferences watched one random movie and both gave it 4 stars, these two people would be assigned a perfect similarity when, clearly, this should not be the case. To mitigate this, we only kept similarity scores for people that have rated 10 or more movies in common.

We actually calculated two similarity scores. Because ratings are all positive, all vectors will only be in the positive space. Therefore, cosine similarities are limited to only positive values as any two vectors cannot exceed more than a right angle. We constructed a similarity matrix using this framework which will henceforth in the paper be referred to as simply the User Network.

However, we also constructed another network where we subtracted 2.5 from all ratings.

This allows for negative cosine similarities but has two important implications. First, in the example above regarding the harsh vs the generous critic, this is no longer true as now, their vectors are directly opposite (-0.5 vs 0.5). This model will be less able to adjust for user biases within their personal ratings. However, a large benefit of this approach is that it also captures *dissimilarity* as well. For example, if two users have seen the same movie but have rated it 0.5 and 5.0, while these ratings are clearly diametrically opposed, this still add similarity towards their score. Now, when we subtract 2.5, essentially set 2.5 as the "neutral" threshold. Anything below is considered negative and anything above is considered positive. Users who share negative or positive scores will be rewarded with similarity and users who have disagreements will negative similarity. This network will henceforth be referred to as the Midrange-Adjusted User Network.

Again, these similarities can be conceived of as an adjacency matrix - this for a unipartite, weighted, bidirectional graph (i.e. the matrix is symmetric as the similarity from user a to user b is the same as user b to user a). From these graphs we made an unweighted graph by setting a threshold of 0.9 for the User Network and 0.5 for the Midrange-Adjusted User Network. These thresholds were chose as they both return similar numbers of edges (around 68,000). If the similarity score is greater than the threshold, an edge is created.

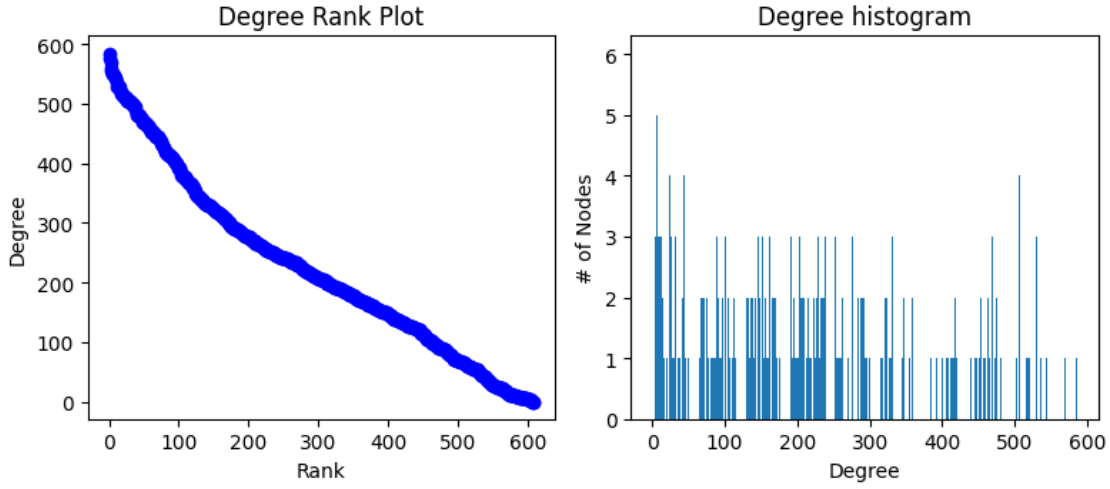


Figure 4.4: Degree Information for User Network

4.3 Movie-to-Movie network

4.4 User-to-Genre network

Many recommendation systems, including Random Walks and Collaborative Filtering which we will discuss later, only take into account user and item interactions. The content of the films is irrelevant and no relationship between the items is captured beyond what was described in the matrices above. Clearly, if we were able to capture these relationships, we could then build much stronger recommendations. One simple ubiquitous covariate is the genre of the film. It wouldn't feel quite right to sit down for family movie night and be

recommended The Shining because you watched The Lion King last week - despite both movies probably having somewhat similar cosine similarity as both movies are generally well regarded. Item-to-item recommendations should consistently stay within the same category and, often, user-recommendations are filtered by said categories.

Therefore, we took a network approach to deepen our understanding between users and genres. Our goal is to quantitatively capture relationships between users and genres to provide better recommendations. In order to do this, we first classified out movies. The movie dataset contains a string of genres separated by a |. We split up the string and then filtered each movie into their respective genres. Movies with no listed genres were dropped. We then calculated the average rating per person as well as the average rating per genre per person. We also counted, the total number of reviews, as well as the total number of reviews per genre.

For each genre, we then had to create a formula to determine whether or not someone was a fan of a genre. We settled on the following formula:

$$\mathrm{F}_{gi} = \tanh\left(\frac{P_{gi} R_{gi} \ln(C_{gi})}{3}\right)$$

P_g is the percentage of user reviews within genre g. Naturally, if a larger percentage of a person's movies belong to a single genre, this should be rewarded. Similarly, R_g represents the average review for a user within a genre. The higher the reviews, the greater the score. Finally, we wanted to factor in the number of films within the genre watched. While this should be somewhat accounted for in P_g, this may be helpful for identifying super fans or "influencers". This is reward logarithmically and we divide by 3 as a scaling constant. This is then passed through tanh to give us fan scores between 0 and 1 (as this score can never be negative unless we adjust with midrange which then could capture a possible "hater score").

Then we arbitrarily chose 0.8 as the threshold. For users with over a 0.8, they were considered a fan.

Once again, we can create an unweight bipartite adjacency matrix from this with, the users as one set of nodes and the movies as the other (visualized in Figure 4.6).

Here, the degree represents how the number of genres for which that user is considered a fan. With the degree histogram, we see a strong fit of a binomial distribution with $n = 19$ (the number of genres) and $p = \frac{4.5}{19}$

However, looking at Figure 4.5, we see an issue. It appears here that we are just measure how many movies there are per genre. Our formula, while it could be effective for large relatively balanced sets, fails in this context. Therefore, we will attempt to create a definition of "fan" that is more accounts for movie imbalances.

$$F^*_{gi} = \left(\frac{R_{gi} - R_i}{R_g}\right) \mathbb{1}\{C_g > 10\}$$

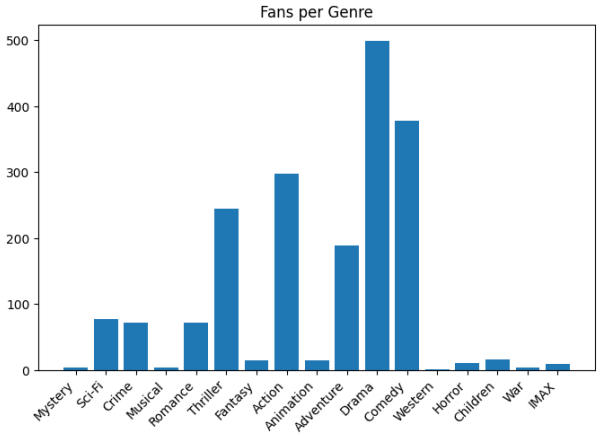


Figure 4.5: Fans per Genre

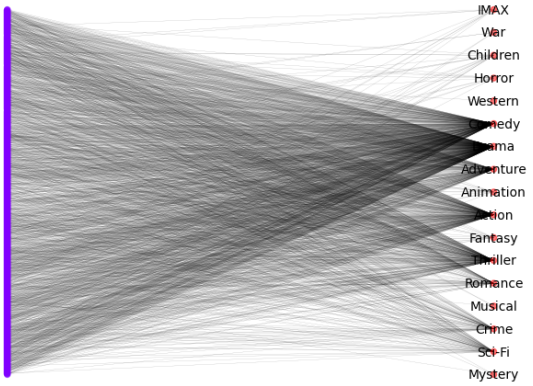


Figure 4.6:
Users - Genre Bipartite Graph

This simplified version finds the difference between a user's score per genre with the average rating per genre and normalizes it by the average rating of that user. The threshold for being a fan here was set to 0.5. Here are the results:

The distribution of fans across genres is now much more even and the degree distribution among fans aligns closely a Poisson with mean 5.5.

5 Random Walk Recommendation Systems

In this section, we explore recommendation systems that directly leverage the network structure of the data. Specifically, we focus on the Probabilistic Spreading (ProbS) and Heat Spreading (HeatS) algorithms, which employ a random walk process within a binary user-item bipartite network to provide a ranking to movies that a user has not seen.

For a given user i , the ProbS algorithm initialises by giving a unit amount of resource to all items that they are connected to. The algorithm then employs a two-step random walk process to redistribute these resources, aiming to accentuate items preferred by users with similar tastes.

In the first step, each item's allocated resource is evenly distributed among users connected to it. This step effectively maps out the extent of shared interests between users based on the items they are associated with. Subsequently, the second step redistributes the resources accumulated by each user back to the items they are connected to, but now the distribution is equal among all such items. Through this reciprocal resource exchange, the algorithm iteratively refines the weight or importance of each item based on the density and depth of shared user preferences. Finally, the scores of items already connected with user i are set to zero so as to not recommend items already connected with the user. A graphical representation of this process is shown in Figure 5.1a.

HeatS works in a very similar way to ProbS except a nodes score in the first and second steps

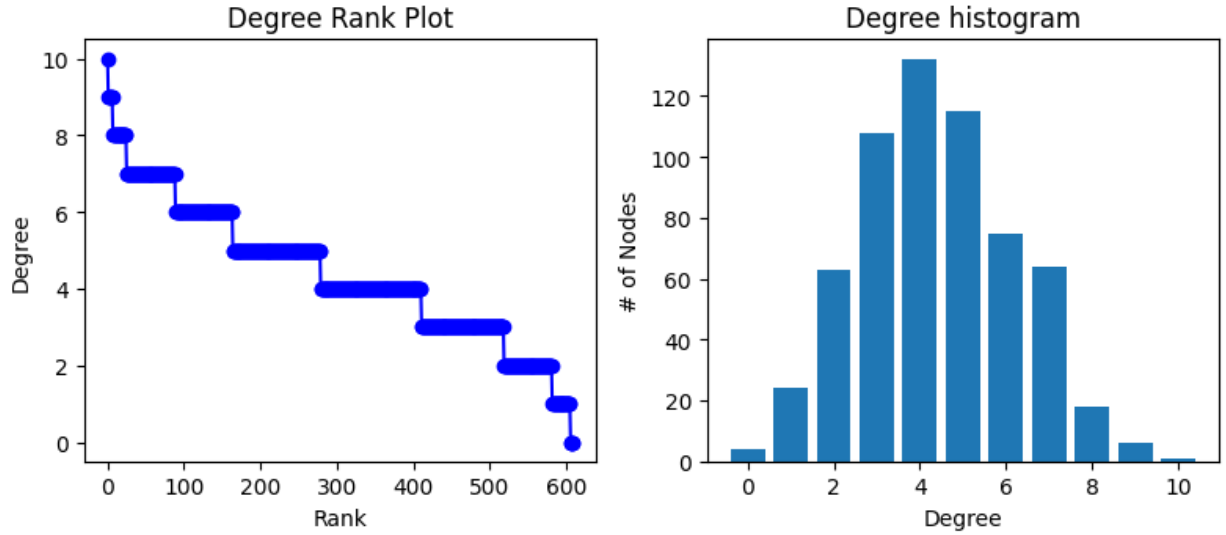
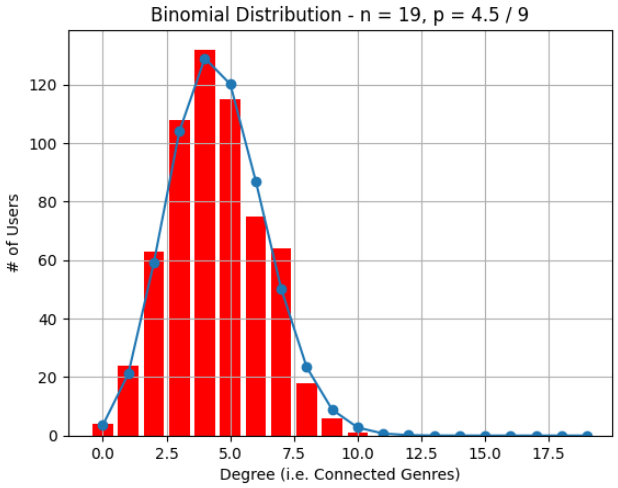


Figure 4.7: User Degrees



are calculate as a simple average of the nodes it is connected to. A graphical representation of this process is shown in Figure 5.1b.

Although the distinction between the two algorithms is subtle, there is a stark difference between the recommendations they produce. The ProbS method exhibits a stronger preference for popular items as the process is cumulative; therefore, an item enhances its likelihood of achieving a high score by accumulating numerous links. In contrast, the HeatS approach tends to favour less popular items. This preference is due to the averaging nature of the HeatS algorithm, where an item can increase its potential for a high score by having a limited number of links to users who possess a significant amount of resource value. This distinction between the two methods highlights their unique approaches to leveraging network dynamics, where ProbS capitalizes on the popularity and widespread connectivity of items, whereas HeatS leverages the principle of scarcity and targeted endorsements from highly resourced users.

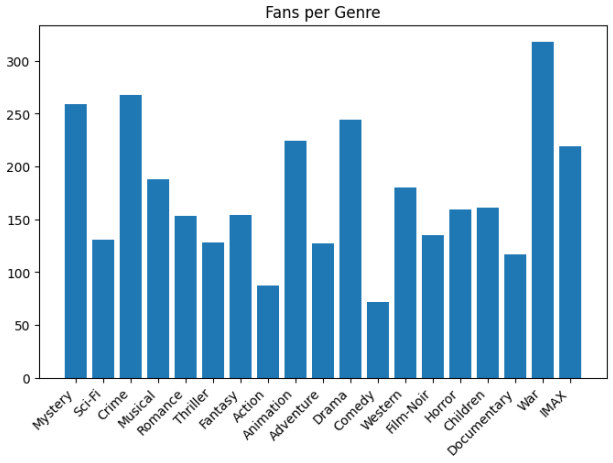


Figure 4.8: New Fans per Genre

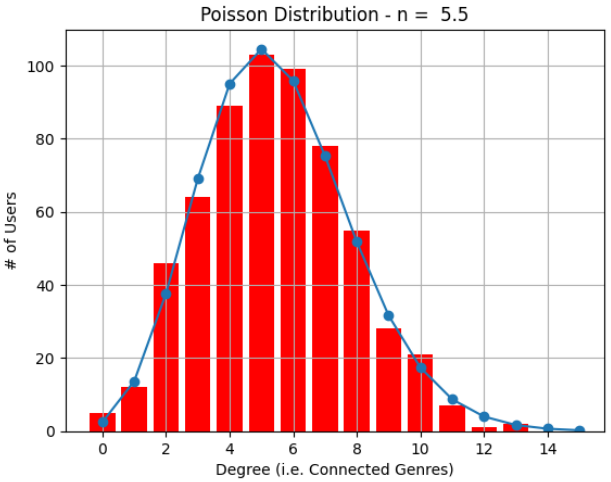
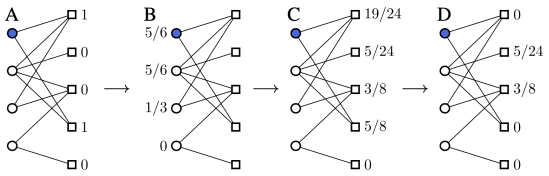
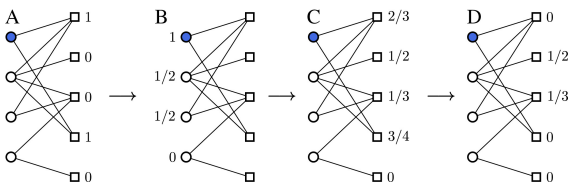


Figure 4.9: Poisson fit with $n = 5.5$



(a) ProbS Algorithm



(b) HeatS Algorithm

Figure 5.1: Illustrations of the Probabilistic Spreading (ProbS) and Heat Spreading (HeatS) algorithms. In both diagrams, circles and squares represent users and items, respectively. The color-marked circle signifies the target user for whom the recommendations are being computed. Figures adapted from [**prob_s**].

As an extension of these algorithm, we also experiment with initialising weights proportional to the rating a user gave each movie. By doing so, we hope to provide prioritise movies that the user enjoyed more, potentially leading to more tailored recommendations.

5.1 Temporal Analysis

To gain deeper insights into the long-term effects of these recommendation algorithms, we examine the evolving dynamics of the network with iterative applications of the algorithm. In each cycle, we posit that the user selects and watches the top-recommended movie, subsequently leading to an update of the binary bipartite network to reflect this new user-item interaction. The recommendation algorithm is then reapplied, taking into account the updated state of the network. This process is repeated, allowing us to observe how the network—and thus the recommendations—might evolve over time with continuous user engagement.

Figure 5.2 shows the recommendation counts after 1, 10 and 100 iterations of the binary ProbS algorithm, and the effect this has on the degree of each user. The recommendation

counts clearly illustrates a power-law distribution, indicating a strong preference by the algorithm for particular items. As a result, certain movies become highly connected, reaching a saturation point where nearly all users have watched with them. This saturation compels the algorithm to diversify its recommendations, thereby incrementally expanding the range of movies suggested to users.

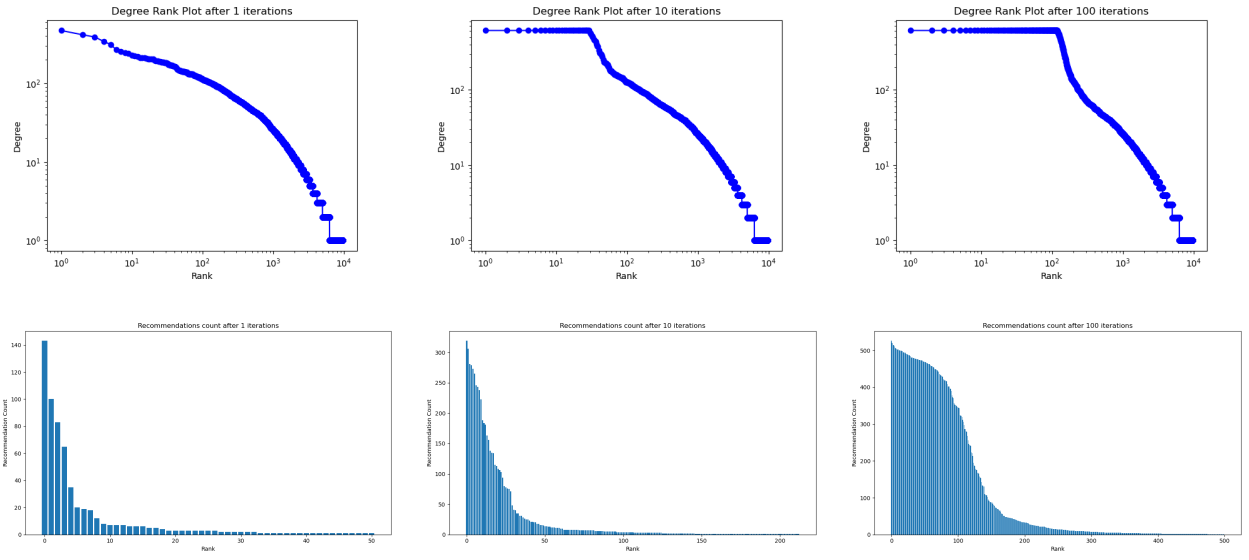


Figure 5.2: ProbS - Recommendation Dynamics

Figure 5.3 shows the temporal analysis of the binary HeatS algorithm. Unlike ProbS, HeatS distributes its recommendations more evenly among items, as evidenced by the flatter distribution of recommendation counts over iterations. Items less connected in the user-item bipartite network are given relatively higher visibility, shown by the degree of lesser connected movies increasing while that of more connected films stays relatively unchanged. This prevents the dominance of a few movies, giving users recommendations from a broader array of content.

5.2 Out of Sample Performance

In order to evaluate the effectiveness of these recommender system, we perform a train-test split by randomly removing 20% of each users ratings and subsequently calculate the Area Under the Curve (AUC) and average precision (i.e. percentage of recommendations that were actually liked by the user). The AUC is a performance metric used to evaluate the quality of binary classification models, which seemed a natural way to assess the performance of these algorithms since we can easily normalise the recommendation score given to be between 0 and 1. It represents tradeoff between the true and false positive rates at various threshold settings. The AUC ranges from 0 to 1, where a model with an AUC of 1 perfectly distinguishes between the two classes, and an AUC of 0.5 indicates no discriminative power, equivalent to random guessing. For the evaluation, a movie in the test set is deemed "liked" by a user if its rating exceeds the user's average rating across all movies in the training set,

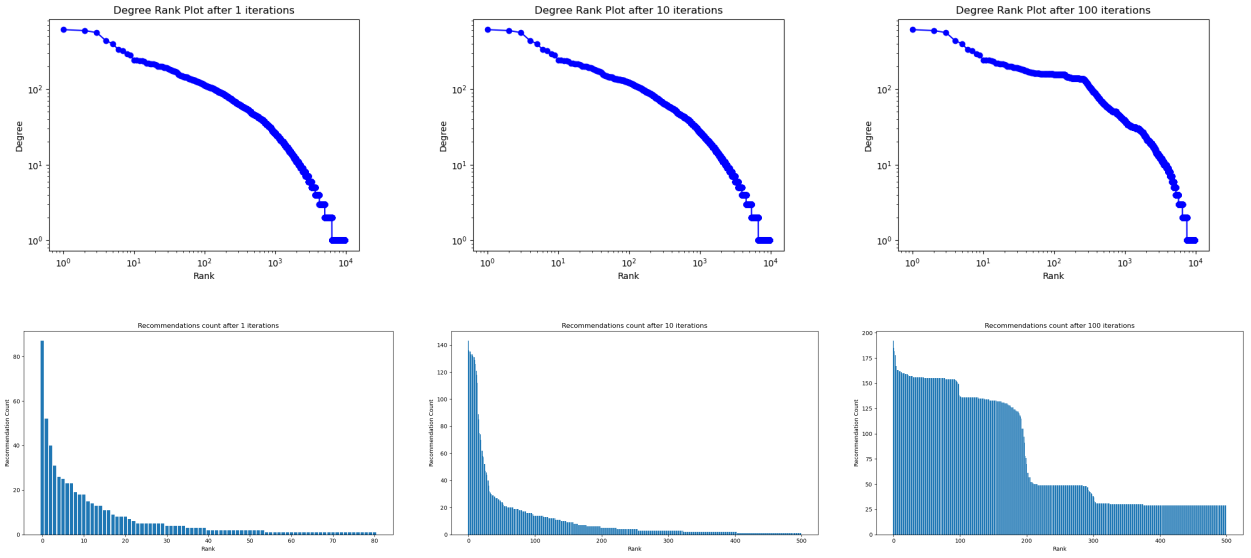


Figure 5.3: HeatS - Recommendation Dynamics

and thus we classify it as a good movie for the algorithm to recommend. To benchmark the results, we also calculate the average precision and AUC for a recommender system simply based on the average rating for each movie.

Model	**AUC**	**Average Precision**
Using Average Rating	0.67	0.67
Prob S - uniform weight	0.57	0.62
Prob S - using ratings	0.59	0.62
Heat S - uniform weight	0.51	0.54
Heat S - using ratings	0.54	0.57

Figure 5.4: ProbS - Model performance

As shown in Figure 5.4, both the ProbS and HeatS models perform worse than the recommendations from using a movies average rating, with the incorporations of ratings only resulting in a small improvement. Furthermore, HeatS only performs slightly better than random. This underperformance could be due to data sparsity and the "cold-start problem", where the algorithms struggles to make accurate predictions because of insufficient user-item interaction data. In contrast, average ratings don't rely on user interaction and can recommend popular items effectively.

This said, the poor performance of the random walk models in terms of AUC does not necessarily mean that they are of no use. One key limitation is that our assessment is based solely on the movies held out in test that a user has already watched. These models might excel at suggesting new and diverse content that aligns with user preferences, yet this aspect remains unmeasured in this setting. This is likely to be particularly true of HeatS, which offers a more diverse array of suggests than ProbS or HeatS, which will often recommend items a user is already aware of.

6 Collaborative Filtering

For comparison, we implemented a basic user-based Collaborative Filtering system. Using the user similarity network and user-to-movie reviews, we are able to calculate predicted scores using the following formula:

$$\hat{R}_j = \frac{\sum_{i=1}^{n} S_{ij} R_i}{\sum_{i=1}^{n} S_{ij}}$$

R_j is the predicted score for user j and S_{ij} is the similarity score between user i and user j. Esentially, we are taking an average of all other reviews weighting by similarity score.

One issue we quickly found is that there were a couple movies that were always getting predicted repeatedly at 5.0. We then realized is that this was because these movies only had one or two perfect reviews. Therefore we only apply this formula to movies that user j hasn't seen with at least 5 people in common (at least 5 similarities).

Therefore, to pick a recommendation for a user, we cycle through all movies that the user j hasn't seen and calculate predicted scores. We then select recommendations based off of highest predicted ratings.

User Id: 314

	movieId	title	genres	clean_title	predictedRating
6	176	Living in Oblivion (1995)	Comedy	Living in Oblivion 1995	4.301551
4	162	Crumb (1994)	Documentary	Crumb 1994	4.281168
7	246	Hoop Dreams (1994)	Documentary	Hoop Dreams 1994	4.278520
12	475	In the Name of the Father (1993)	Drama	In the Name of the Father 1993	4.266497
0	28	Persuasion (1995)	Drama|Romance	Persuasion 1995	4.225406
10	306	Three Colors: Red (Trois couleurs: Rouge) (1994)	Drama	Three Colors Red Trois couleurs Rouge 1994	4.114517
13	541	Blade Runner (1982)	Action|Sci-Fi|Thriller	Blade Runner 1982	4.109619
14	608	Fargo (1996)	Comedy|Crime|Drama|Thriller	Fargo 1996	4.100769
8	280	Murder in the First (1995)	Drama|Thriller	Murder in the First 1995	4.076580
2	94	Beautiful Girls (1996)	Comedy|Drama|Romance	Beautiful Girls 1996	4.057643
5	175	Kids (1995)	Drama	Kids 1995	4.049988
11	308	Three Colors: White (Trzy kolory: Bialy) (1994)	Comedy|Drama	Three Colors White Trzy kolory Bialy 1994	4.035003
9	290	Once Were Warriors (1994)	Crime|Drama	Once Were Warriors 1994	4.034105
3	111	Taxi Driver (1976)	Crime|Drama|Thriller	Taxi Driver 1976	4.021900
1	58	Postman, The (Postino, Il) (1994)	Comedy|Drama|Romance	Postman The Postino Il 1994	4.021390

Figure 6.1: Recomendations for User 314

After running this for every single user, we see a pattern for the top reviews recommend. Figure 6.2 shows that top 10 top-recommended movies. We see the cold start problem. All top recommended movies are movies that are generally highly rated and with a lot of reviews. All top 10 movies were released before 2000 despite this being not at all rerpresentative of the total population of movies.

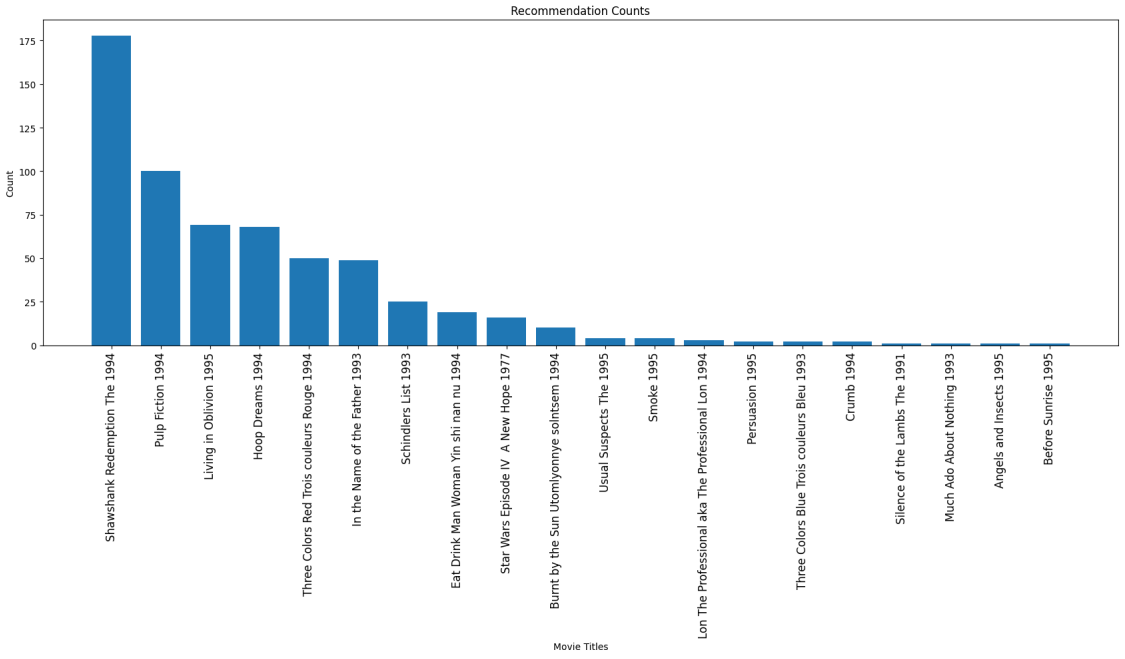


Figure 6.2: Top Recommendation of Movies

6.1 Validation

6.1.1 Mean Square Error

Because this model is able to produce predictied ratings, we are able to use mean square error to calculate model performance. Because it would too computationally expensive to test every single movie, we take a Monte Carlo approach towards this. We randomly select a user and from the users films, we randomly select a movie they have seen. We calculate the expected rating given our model and calculate the mean squared error. We do this for 1000 movies.

After conducting this 20 times, we get an average Mean Squared Error of 0.9694. For reference, by just predicting the average rating, following the same process, we get an average Mean Squared Error of 1.079. Therefore, it's slightly better than average but not by much.

While the ability to output interpretable predict ratings is a nice benefit of this simple collaborative filtering system, it's not necessarily that useful. When watching recommended movies, while it might be nice to get a 5.0 movies instead of a 4.8 movie, the difference is generally indstinguishable and either one would be considered a successs. Generally, the only thing that matters is whether or not a movie should or shouldn't have been recommended. If you lliked the movie, then the recommendation worked. If you didn't then the recommendation did not work. Therefore this becomes a binary classification problem.

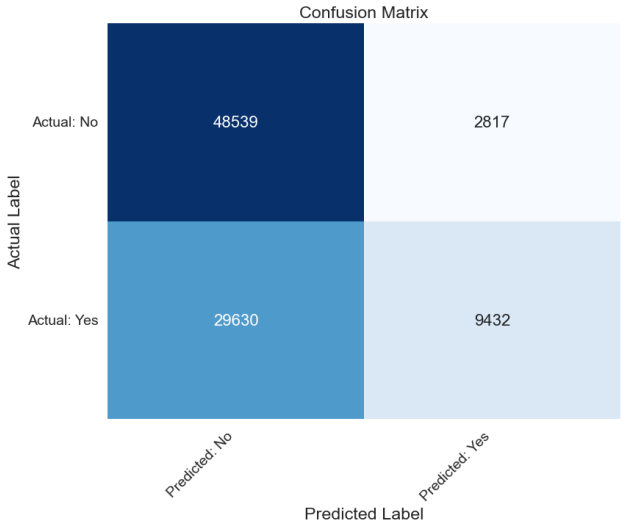


Figure 6.3: Confusion Matrix with Collaborative Filtering

6.1.2 Binary Classification

We will evalutate peformance based on Precisinon, Recall, and Accuracy, That way, we can also compare the peformance of the ProbS and HeatS models. For binary classification, we consider a rating to be positive if it is 4 or above. From our samples of reviews, we have a pretty balanced dataset with 48580 positive reviews, and 52256 negative reviews.

Figure 6.3 is the Confusion Matrix after running this with 90418 reviews. What we see is a huge inbalance towards preditcing negative reviews. This results in an accuracy of 0.546, recall of 0.241, and precision of 0.770.

However, the most important metric here is precision. When discussing with an industry professional who worked on a system for a e-commerce website, he said that they mostly looked at recall. In that situation, it make snse that the "error" of recommending an item that a user doesn't want is not as bad as not recommending something the user would've bought. One is a negligible annoyance and the other is missed revenue. However, in our case, it is much more important to recommend ONLY movies that people enjoy moreso than recdommedning all movies that a person would enjoy.

Therefore, we see that even this simple Collaborative Filtering system beats ProbS, HeatS, and the baseline.

7 Graphical Neural Networks

Finally, we implement a Graphical Neural Networks (GNN) as recommendation system. In particular we use the architecture called LightGCN. This GNN is a simplified version of A Neural Graph Collaborative Filtering (NGCF), which adapts Graph Convolutional Networks in recommendation systems and employs one convolutional layer to exploit the direct connections between users and items [**lightgcn**].

That is, LightGCN employs a simplified graph convolution that directly aggregates the embeddings of neighboring nodes without the use of transformation weights or non-linear activations, thus preserving the collaborative signal in the embeddings.

The embedding propagation can be recursively defined as:

$$e_u^{(k+1)} = \sigma\left(W_1 e_u^{(k)} + \sum_{i \in N_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} W_2(e_i^{(k)} \cdot e_u^{(k)})\right),$$
$$e_i^{(k+1)} = \sigma\left(W_1 e_i^{(k)} + \sum_{u \in N_i} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} W_2(e_u^{(k)} \cdot e_i^{(k)})\right),$$

where W_1 and W_2 are trainable weight matrices, σ denotes a non-linearity such as LeakyReLU, and $\mathcal{N}_u$ and $\mathcal{N}_i$ represent the sets of first-hop neighbors for users and items, respectively.

The user's final preference for an item is calculated using the dot product of the final user and item embeddings after K layers of propagation, which are combined to obtain the final representations:

$$\hat{y}_{ui} = (e_u^{(*)})^T e_i^{(*)}.$$

7.1 Model Architecture

LightGCN simplifies the NGCF by removing feature transformation and non-linear activation steps. The architecture involves stacking multiple layers of simple weighted sum aggregators to capture the collaborative signal within the high-order connectivities.

The graph convolution operation in LightGCN is depicted as follows:

$$e_u^{(k+1)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} e_i^{(k)} \tag{7.1}$$

$$e_i^{(k+1)} = \sum_{u \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_u|}} e_u^{(k)} \tag{7.2}$$

where $e_u^{(k)}$ and $e_i^{(k)}$ are the embeddings of user u and item i at the k-th layer, and $\mathcal{N}_u$ and $\mathcal{N}_i$ are the first-hop neighbors of user u and item i, respectively.

The final embeddings are obtained by combining the embeddings from all layers. The user's preference for an item is predicted by the dot product of the user and item embeddings:

$$\hat{y}_{ui} = (e_u^{(*)})^\top e_i^{(*)} \tag{7.3}$$

The model employs Bayesian Personalized Ranking (BPR) loss, which is formulated as:

$$L_{\text{BPR}} = -\sum_{u=1}^{M} \sum_{i \in \mathcal{N}_u} \sum_{j \notin \mathcal{N}_u} \ln \sigma(\hat{y}_{ui} - \hat{y}_{uj}) + \lambda ||e^{(0)}||^2 \tag{7.4}$$

where σ denotes the sigmoid function and λ is the regularization parameter. The model utilizes the Adam optimizer for training in a mini-batches.

7.2 Implementation

The implementation of the network is using an open source from Microsoft Recommenders (https://github.com/microsoft/recommenders), that is based in Pytorch. The parameters that we used was 3 layers, batch size of 1024, 1,000 epochs, a learning rate of 0.001 and we look for the top 10 most recommended movies.

After training, we note that the networks converges as can be observed in the figure 7.1.

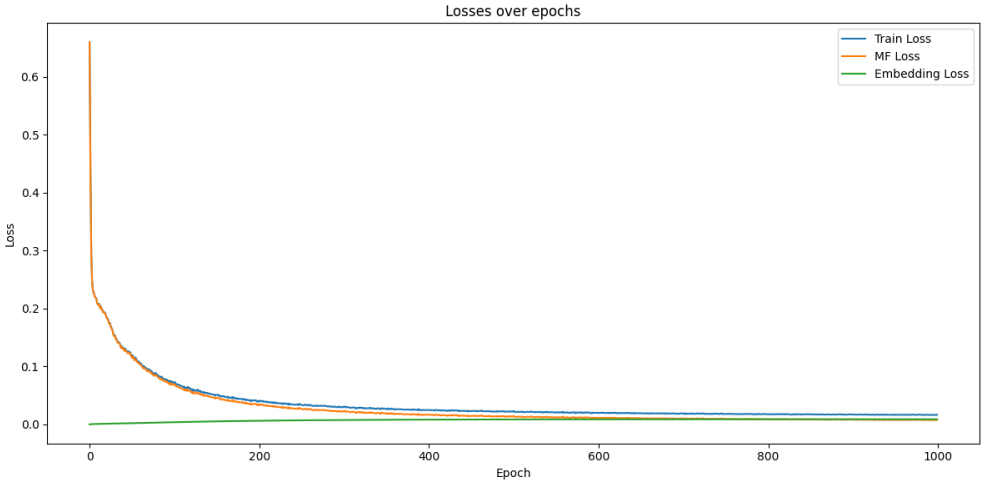


Figure 7.1: Convergence of the GNN.

The code predicts the ranking of movies for every user instead of the rating itself like the last exercises. This ranking predicted is shown in the figure 7.2.

	userID	itemID	prediction
0	1	593	9.765488
1	1	1387	9.508644
2	1	1923	9.385398
3	1	1287	9.346621
4	1	924	9.339483

Figure 7.2: Predicted Ranking for the user 1

With the top 10 scores predicted by the model, we evaluate how LightGCN performs on this test set. The metrics are the followings:

1. Precision = 0.277049 (proportion of recommended items that are relevant)
2. Recall = 0.162039 (proportion of relevant items that are recommended)

3. Mean Average Precision = 0.203541 (how well the predicted items for a user are ranked based on relevance)
4. Normalized Discounted Cumulative Gain = 0.324148 (evaluates how well the predicted items for a user are ranked based on relevance)

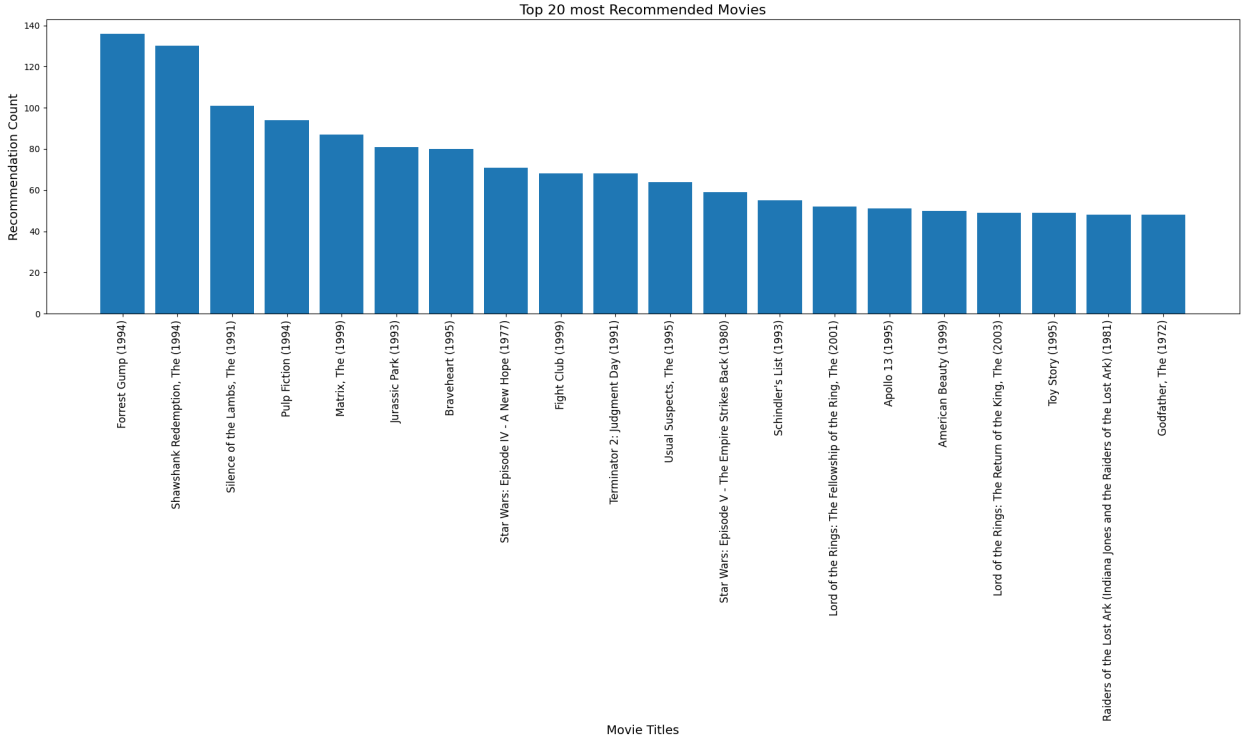


Figure 7.3: Illustration of user-item interaction and high-order connectivity.

The figure 7.3 shows top 20 most recommend movies for all users. This time, we get very famous movies, such as, Forest Gum, Matrix, Terminator, lord of the rings and Star Wars movies, that make sense. Then, we can assume that the metrics used by the code indicates a good enough performance.

8 Conclusions